線型計劃의 解法에 관하여
— On Algorithms of Linear Programming —

梁 光 敏*

《目 次》

Ⅰ. 序 論

一般 線型計劃(linear programming)問題는 단지히(George B. Danzig)에 의해 1947年 처음으로 그 體系的인 解法(單體法 ; simplex 方法)이 考案되고 그 後 5年後인 1952年 正月에 美國 標準局(National Bureau of Standards)의 SEAC 컴퓨터로 最初로 電算處理된 이래 지금까지 30餘年間 많은 연구와 발전이 있었다.

數理計劃(mathematical programming)이 經營科學에서 主된 영역을 차지하는 것과 같이 線型計劃은 數理計劃에서 그 每핕에 해당한다고 볼 수 있다. 이는 現在의 整數計劃(integer programming)을 포함한 모든 非線型計劃의 解法이 선형계획을 根幹으로 하여 開發되었다는 點에서 보더라도 선형계획 解法의 발전은 곧 非線型計劃의 발전에 寄與한다고 보는 것이다. 이에 본 硏究에서는 그 對象을 一般 선형계획문제의 해법에 局限하여 다루고자 한다.

* 중앙대학교 경영대학 부교수

Ⅱ. 問題의 提起

線型計劃문제의 解를 구함은 制約式으로 이루어지는 볼록體(convex polytope)의 모든 極點(extreme points) 中 目的函數의 값을 가장 좋게(최대 또는 최소로)하는 것을 찾는 것이라고 볼 수 있다.[1] 위의 方法에 의해 선형계획의 解를 구한다고 할 경우 우선 두 가지의 문제點이 抬頭된다. 첫째로 볼록體의 모든 極點을 찾는 일이고, 둘째로는 이 極點의 數가 문제에 따라서는 너무 커서 실제로 다루기에 어려울 수가 있다는 점이다. 만약 m개의 制約式(=型)과 n개의 變數가 있는 선형계획의 경우 基底解(basic solutions)의 數는 $\binom{m}{n}$개가 있으므로 m과 n이 커짐에 따라 極點의 數가 급속히 증가할 수가 있다.

위의 두가지 短點을 補完하기 위한 Danzig의 單體法은 體系的으로 極點을 구하는 方法과 모든 極點을 구하여 目的函數를 評價하는 대신 부분적인 극점들만을 평가한 後 목적함수를 最適으로 하는 極點을 찾을 수 있도록 考案된 方法인 것이다. 即, 單體法은 한 極點에서 다른 극점으로 옮겨가는 方法이며 이때 목적함수의 값이 결코 먼저번 극점의 것보다 나쁘지 않도록 하고, 목적함수의 값이 더 이상 좋아질 수 없는 극점(最適解)에 도달할 경우이를 感知할 수 있는 方法인 것이다.

單體法이 모든 극점을 평가하여 최적해를 찾는 방법이 아니라는 點에서는 커다란 발전이나 單體法에서도 문제의 크기가 커짐에 따라 評價해야 할 극점의 數는 빠른 속도로 증가한다. (實際計算 경험에 의하면 최적해에 도달하는 시간은 보통 제약식의 數에 따라 線型으로 증가한다고 하나 문제에 따라서는 指數的으로 증가하는 경우도 있다.[10])

單體法에 관한 많은 연구가 있었음에도 不拘하고 최적해를 찾는 데 필요한 極點數(pivot 數 또는 iteration 數)의 下限이 전혀 알려져 있지 않기 때문에 자연히 硏究는 pivot 當 計算努力을 줄이려는 데 重點이 두어졌으며 그 結果의 하나가 修正單體法(revised simplex method)이다.

여기서 잠시 單體法에 대한 論議를 접추고 선형계획 문제에 관한 전혀 다른 解法을 논의해 보기로 한다. 선형계획의 해법에 관한 根源的인 연구(單體法 이외의)는 실제로 많이 행해졌으리라 보아지나 모두 이렇다 할 結實이 없었던 관계로 문헌에 보고된 것이 많지 않았다고

1) empty feasible set, multiple optima, unbounded solution 등 pathological cases를 除外함.

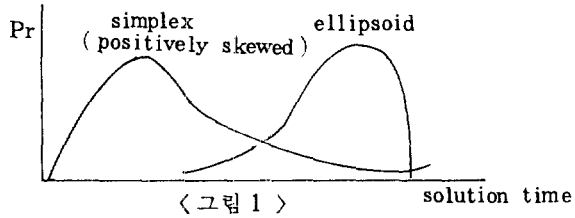
생각된다.

1979年 L. G. Khachiyan은 楕圓體(ellipsoid)方法에 의해 선형계획 문제를 풀 수 있음을 보였다. Khachiyan의 硏究의 중요한 結果는 오랜동안 이 분야의 難題였던 선형계획 문제의 計算難易性(computational complexity)에 관한 아주 중요한 理論的인 문제에 解答을 주었다는 점이다. 즉 선형계획 문제가 多項的時間(polynomial-time) 알리고즘에 의해 풀 수 있는 문제들에 屬한다는 것이다.

多項的時間 알리고즘에 의해 풀 수 있는 선형계획 문제를 單體法은 多項的制限時間(polynomially-bounded time)에 풀지 못한다. 反面 楕圓體法은 多項的時間 알고리즘이다. 따라서 타원체法이 單體法에 比해 우월하다고 볼 수도 있겠으나 實質的인 面에서는 楕圓體法이 卓越하다고 보기는 어렵다.

첫째로 타원체법이 單體法에 比해 어느 경우나 最適解에 빨리 收斂하지 않는다. 最惡경우 限界(worst-case bound)는 單體法의 경우 實際의 性能을 나타내는 아주 나쁜 指標가 되는 反面 最惡限界는 楕圓體法의 경우 아주 良好한 指針이라고 보여진다[2]. 다시 말하면 最惡경우(worst-case) 楕圓體法이 우월하나 平均경우(average-case)를 비교하면 單體法이 훨씬 優勢하며 實質的인 面에서 平均경우의 性能비교가 最惡경우의 성능비교보다 合理的이라고 보는 것이다. 둘째로 타원체법은 대부분의 선형계획 문제가 가지고 있는 성김(sparseness)의 利點을 알고리즘에서 活用하지 않는다는 點이다. 따라서 타원체법이 最惡경우의 反復(iteration)數는 相當히 줄일 수 있으나 아주 크고 성긴 선형계획 문제를(平均的으로) 單體法과 같이 효율적으로 푸는 데는 문제가 있다고 보여진다. 楕圓體法은 위의 최적해에의 收斂속도 이외에도 問題表現(encoding)을 위한 情密度를 維持하는데 따른 實際履行(implementation) 上의 문제도 있다.

위에서 살펴 본 바와 같이 線型計劃문제 自體는 最惡의 경우 多項的時間(polynomially-bounded time)에 풀 수 있는 문제에 屬함에도 不拘하고 현재 효율적인 解法으로 널리 쓰여지고 있는 單體法은 선형계획 문제를 多項的時間에 풀 수 없다는 것이 逆說的으로 보일 수 있으나 解를 얻기 위한 計算時間의 期待値로선 最善의 方法인 것으로 보인다(그림 1 參照).



〈그림 1〉

앞으로 單體法의 경우 pivot 회수가 문제 크기의 多項式으로 表示될 수 있는 pivot 法이 發見되거나(不可能한 것으로 보아지나 아직 證明되지 않음), 楕圓體法의 경우 平均경우 計算時間의 현저한 발전이 없는 限, 선형계획 해법의 발전은 從來의 연구方向인 pivot 當계산을 어떻게 효율적으로 遂行할 수 있겠는가에 歸着된다. 이는 선형계획의 特殊形態인 仲介地經由 수송문제(transshipment problems)가 효율적인 基底變換(basis updating)과 컴퓨터科學의 發展으로 問題크기가 大規模일 경우에도 거의 實時間으로 풀릴 수 있다는 것과 脈絡을 같이 한다[3]. 즉 極點의 個數는 문제가 대규모化함에 따라 急히 증가하나(實際계산경험에 의하면 평가해야 할 極點의 數는 제약식의 數에 관해 線型으로 증가[2]) 極點에서 극점으로 옮겨가는데 要하는 計算量(즉 pivot에 必要한 계산)을 줄임으로써 상대적으로 커다란 문제를 풀 수 있도록 하려는 意圖인 것이다.

Ⅲ. 解　　法

1. 序

앞에서 언급한 바와 같이 pivot 회수를 줄이고자 하는 努力으로서의 crashing이라든가 進入變數(entering variable)의 선택을 통해 보다 效果的으로 最適解에 도달하려는 multiple pricing (또는 block pricing)등 최적해에 도달하는 經路(trajectory)에 관한 여러가지 構想은 그 試圖가 이론적으로 餘他 pricing 方法(most negative rule 또는 first negative rule)에 비해 일반적으로 우월함을 보일 수 없으므로 여기서는 논의하지 않기로 한다.

본章에서는 線型計劃 一般에 걸쳐 효율적인 解를 구하는 데 고려해야 할 項目들을 차례로 論議하고자 한다. 第 2 節은 元來 單體法(古典的 單體法)이 列基底(column basis)를 사용하는 데 反해 行基底(row basis)를 사용하는 Graves 의 알고리즘을 紹介한다. 선형계획의 解法을 다룬 거의 모든 敎材·文獻이 이 古典的 單體法에 準하여 이론을 展開·說明하므로 筆者를 포함한 많은 사람이 이에 대한 一種의 固定觀念이 들어있으나 이 Graves 의 行基底에 의한 접근은 문제를 定型된 그대로 보려는 것이므로 보다 自然的인 접근方法이라고 볼 수 있다. 이 Graves 의 解法을 기초로 하여 第 3 節에서는 退化에 관한 組織的인 해결方案을 다루고 있다. 많은 文獻들이 退化에 관해 언급하면서도 實際 문제에서 別로 問題視 되고 있지 않는다고 하여 가볍게 넘어가나 문제에 따라서는(scheduling 等) 이 퇴화문제의 효율적인 해결

방안이 문제를 풀 수 있는 關鍵이 되기도 한다.

컴퓨터의 하드·웨어 값이 每年 25～30 %程度로 下落하므로[15] 컴퓨터의 성능 및 용량은 상대적으로 增加한다. 그러나 大規模문제(대략 2000 × 3000 이상)를 푸는 데는 아직도 限界가 있으므로 資料構造(data structure) 및 알고리즘 上의 여러가지 裝置가 필요하다. 第4節에서는 이러한 문제를 記述的으로 다룬다. 第5節에서는 線型計劃 模型化의 適切性 문제에 대한 한가지 代案을 提示하고자 한다. 즉 各制約式의 制約性의 정도에 따른 加重値를 考慮하여 목적함수에 反映하려는 意圖에서 試圖된 彈力計劃法(elastic programming)에 관한 것이다. 第6節은 線型計劃 모형이 大規模化함에 따라 不可避하게 된 資料管理시스템에 관한 것을 다룬다. matrix generator, report writer 및 會話型(interactive) 보고서 作成技法 등이다.

2. Graves의 解法

본節에서는 pivot 當 計算努力을 줄일 수 있는 方案에 관해서 다루기로 한다.

單體法에서 修正單體法으로 발전한 것을 돌이켜 본다면 이는 한 極點에서 다른 극점으로 옮겨 가는 데 필요한 情報는 基底와 pricing에 필요한 目的함수 行 및 右項(right-hand side)뿐이지 單體表(simplex tableau)전체가 아니라는 데 着眼한 것이다. 만약 m이 제약식의 數를 나타낸다고 하면(原問題와 双對問題의 제약식數중 작은 쪽을 表示한다고 하자) 基底行列은 그 크기가 m × m이다. 따라서 수정單體法에서 어떤 方法으로든 基底행렬의 크기를 줄이지 않는 限 計算量을 줄일 수 있는 方案은 없을 듯 하다.

特定 線型計劃 문제의 최적해를 구했을 경우 제약식 모두가 結束(binding)하는 경우란 實際의 문제에서 거의 없고 그 反對로 많은 數의 제약식이 '들러리'(redundant)일 경우가 普遍的이다. 古典的單體法에서는 列基底(column basis)를 사용하므로 이 '들러리'도 基底에 포함되어 基底의 크기가 제약식의 數와 같았던 것이다. 따라서 만약 이 들러리 제약식을 除去하고 남는 제약식만으로 구성되는 基底를 變換(update)할 수 있다면 보다 효율적이라는 데 착안해 볼수 있다. 그러나 古典的 單體法은 單體法을 시작할 때 餘裕(또는 剩餘) 變數를 導入하여 모든 제약식을 等式化하여 基底의 크기를 처음부터 固定시킨다. 이러한 點을 克服하기 위해 Graves는 行基底(row basis)와 不等式을 그대로 사용한다.

다음에 一般 線型計劃 문제에 관한 Graves의 解法을 簡略히 다루어 본다[7]. (附錄의 原問題 알고리즘의 例 參照)

다음의 선형계획 문제를 생각하자.

$$(P)\quad \begin{aligned} &\text{Min} \quad wy \\ &\text{Subject to} \quad Ay \leq r, \\ &\qquad\qquad -Iy \leq 0, \end{aligned}$$

여기서 A는 $m \times n$.

Graves의 解法에서는 n개의 線型獨立인 제약식으로 이루어지는 行基底를 사용한다(古典單體法은 m개의 列基底(column basis)를 사용함). 行基底를 B라고 하고 非(行)基底를 D라고 하면

$$B = \begin{pmatrix} A_{11} & A_{12} \\ O & -I \end{pmatrix} \qquad D = \begin{pmatrix} -I & O \\ A_{21} & A_{22} \end{pmatrix}$$ 로

표시되고

여기서 $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ 이다.

(P)를 基本 tableau 型으로 表示하면

$$\left(\begin{array}{cc:c} A_{11} & A_{12} & r_1 \\ A_{21} & A_{22} & r_2 \\ \hdashline w_1 & w_2 & o \end{array}\right)$$

여기서 $r = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$ $w = (w_1 \quad w_2)$ 이다.

pivot은 단순히 B와 D사이의 제약식의 交換이며 일반적으로 어느 pivot 단계든 다음과 같이 tableau로 表記할 수 있다.

$$
\begin{array}{c|cc:c}
 & (h) & (n-h) & (1) \\
\hline
(h) & A_{11}^{-1} & A_{11}^{-1}A_{12} & A_{11}^{-1}r_1 \\
(m-h) & -A_{21}A_{11}^{-1} & A_{22}-A_{21}A_{11}^{-1}A_{12} & r_2-A_{21}A_{11}^{-1}r_1 \\
\hdashline
(1) & -w_1A_{11}^{-1} & w_2-w_1A_{11}^{-1}A_{12} & -w_1A_{11}^{-1}r_1
\end{array}
$$

( )는 크기를 表示함.

위의 tableau는 原問題에 conjugate basis $P = -B^{-1} = \begin{pmatrix} -A_{11}^{-1} & -A_{11}^{-1}A_{12} \\ O & I \end{pmatrix}$를 post-multiply하여 얻을 수도 있다 (또는 A_{11} submatrix를 pivot element로 block pivot[2] 하여 얻을 수도 있다). 위의 tableau에서 보는 바와 같이 基底變換(basis update)에 필요로 하는 情報는 A_{11}^{-1}와 맨 아래 行 및 右項 뿐이며 그 밖의 情報는 恒時 計算하지 않고 필요時마다 A_{11}^{-1} 및 原問題로부터 隨時 計算하여 사용한다. 이렇게 함으로써 i) pivot當 計算量의 감소, ii) 필요한 계산기 메모리 감소 및 iii) 再逆行列을 구할 때 數値不安定 현상의 감소를 이룰 수 있다. 여기서 A_{11}^{-1}의 크기는 $h \times h$ ($h \leq m$)이고 反復(iteration)時마다 그 크기가 變化한다. 勿論 최적해의 基底제약식이 모두 結束(binding)할 경우 $h = m$이나 그 밖의 경우 $h < m$이므로 修正單體法과 비교해 $(h/m)^2$의 비율로 계산량이 감소된다. 예로서 70%의 제약식만이 結束할 경우 수정단체법에 비해 두배, 50%의 경우 4배나 빠르게 계산할 수 있다. 實驗的으로 0-1 就數를 發生시켜 無作爲로 set packing 문제를 만들어보아도 90%정도[3]의 제약식만이 結束制約式이었다. 實際 선형계획 모형을 작성함에 많은 數의 제약식이 redundant하여[12] 商業用 code는 이들 제약식을 찾기 위한 前處理(preprocess)에 많은 시간을 소모하며, 또한 最適解가 주어지지 않는 限 모든 非結束제약식을 가려낼 수 없는 短點이 있다.

---

2) 古典單體法에서 通常 사용하는 pivot과 달리 in-place pivot인 Bordering method에 의한 pivot임. 附錄의 例題 參照.

3) 密度 5~8%이고 크기 50×500인 문제行列 20개에 대한 統計値임.

3. 退　　化

線型計劃 理論과 解法이 개발되던 初期段階는 勿論이고 현재까지도 많은 教科書·文獻들이 退化(degeneracy ; blocking [4]) 문제를 소홀히 취급하고 있다. 退化問題自體의 어려움은 인정하지만 이는 실제계산상의 重要性은 별로 느끼지 않은 게 사실이다. 初期에는 아주 작은 單純한 經濟問題 정도가 선형계획의 對象이었기 때문에 이러한 觀點이 있을 수 있었다고 본다. 그러나 현재의 선형계획 코드는 아주 복잡한 非線型, 整數計劃 또는 複雜한 分解技法의 一部로서 사용되므로 실제 계산에 退化가 문제되지 않는다는 생각은 올바른 것이 아니다. 하나의 非線型計劃 또는 整數計劃 문제를 풀기 위해 보통 數千個의 線型計劃 문제를 풀어야 한다. 현재 對象이 되는 大規模數理計劃 문제는 數千個의 0-1 變數를 포함하여 이러한 문제들은 자주 甚하게 퇴화하는 경우가 많다. 이러한 경우 이론적으로 基底는 循環(cycling)할 수 있고 最適解에의 收斂을 防止한다는 사실을 모두 認識하면서도 이러한 경우가 실제의 문제에서는 일어나지 않으므로 문제視되지 않는다고 대부분의 教材·文獻이 유행처럼 구실을 달아 넘어가고 있다. 이러한 이유로 週期가 數百 또는 數千 pivot이 되는 대규모문제에서 循環이 전혀 밝혀지지 않는 경우가 실제로 있다. 대규모 문제를 풀지 못하는 이유로 단순히 數値誤差 (round-off error)라고 못 박고 退化의 役割에 대해서는 전혀 調査해 보지 않는 것이 보통이다. set partitioning 類(대체로 scheduling 문제들)의 문제에서 퇴화가 極甚함은 쉽게 설명될 수 있으며 이러한 문제를 풀기 위해서는 퇴화문제가 先決되어야 함을 再論할 필요가 없다.

퇴화문제의 해결에서 중요한 것은 이들 해결방법의 效率이다. 원래 攝動法(perturbation method) 또는 編輯順序法 (lexicographic method)은 단지 循環을 防止하려는 것만이 그 目標였다. 왜냐하면 이론展開를 위해서는 순환을 방지할 수 있어야만 有限단계에 최적해에의 수렴을 보일 수 있기 때문이다. 여러가지 編輯순서法의 효율성의 문제가 선형계획 문헌에서 主題로 다루어진 예는 전혀 없다. 현재 사용되고 있는 많은 컴퓨터 LP코드도 퇴화에 對處하기 위한 節次를 사용하지 않고 그저 遇然에 의하거나, 退化防止節次를 가지고 있는 코드도 거의 모두 任意의 편집順序에 의한다. 勿論 어느 방법이건 基底를 바꾸는 것이 非循環的이기만 하면 선형계획 문제를 有限단계에 푸는 데는 충분하다. 初期의 小規模 경제문제의 경우 이러한 任意의 非循環 基底方法이 적합하였으나 현재의 대

4) degeneracy 보다 blocking이 문제의 正確한 表現이다.

규모문제에서와 같이 基底가 클 경우에는 효율적인 退化해결方案이 必須的이다.

Graves의 退化에 대한 해결方法〔7〕은 다른 方法에서와 같은 任意性이 없다는 점이 특징이다. 紙面上 간략히 요약해 보기로 한다. 만약 原문제 알고리즘을 쓰는 가운데 退化가 일어나면 双對문제알고리즘으로 옮겨가서 풀고, 반대로 双對문제를 푸는 가운데 퇴화가 일어나면 原문제 알고리즘으로 돌아가는 것이다. 이와같이 原·双對 알고리즘을 原來문제(original problem)의 副問題(subproblems)에 交互로 적용하면 退化를 해결하거나 最惡의 경우 縮小된 퇴화문제가 연속되거나 이다. 한 알고리즘에서 다른 알고리즘으로 옮길때 마다 문제의 축소(strict contraction)는 보장되므로 이 副問題에 原問題 알고리즘과 双對문제 알고리즘을 교호로 적용하여 有限단계에 수렴할 수 있게 하는 것이다. 이와같은 Graves의 方法을 효과적으로 수행하기 위해서는 原問題 알고리즘과 双對문제 알고리즘을 課外의 負擔없이 쉽게 交代로 행할 수 있어야 한다. 이를 위해 Graves의 알고리즘은 基本表(basic tableau) 하나로 二重表現을 가능하게 하도록 原問題와 双對문제를 選擇하고 있다. (알고리즘上 자세한 것은 Graves〔7〕參照)

4. 大規模 선형계획(Large-scale LP)

대규모 선형계획 문제라 함은 어떤 絶對的인 基準이 있어 區分되는 것이 아니라 대략 제약식 2,000개 變數 3,000개 程度이상으로 구성되는 선형계획 문제를 지칭한다〔15〕.

선형계획 문제를 有限단계에 풀 수 있는 알고리즘이 존재함에도 또 컴퓨터의 性能이 계속 발전함에도 不拘하고 보다 나은 解法을 硏究함은 주어진 문제를 빠른 시간에 풀려고 하는 努力이라고도 볼 수 있으나 그 보다도 더욱 큰 問題를 풀 수 있도록 하는 노력이라고 보아야 한다. 위의 두가지 문제는 같은 문제인 듯 하나 後者는 時間 및 空間(time and space complexity) 모두를 다루어야 하므로 더욱 어렵다.

현재 技術水準으로 다룰 수 있는 대규모 선형계획 문제의 크기는 매우 制限되나 實際模型에서 볼 수 있는 몇가지 特性을 利用하면 위에서 定義한 대규모 선형계획의 범주에 屬하는 문제를 許容된 자원의 범위內에서 풀 수 있다.

實際 대규모 선형계획 문제가 가지고 있는 特性이란 i) 성김(sparseness), ii) 一般化上限제약식(generalized upper bound(GUB) constraints)의 존재 및 iii) 서로 다른

實數(unique real numbers)로 된 係數의 數가 限定돼 있다는 것 등이다.

問題行列(problem matrix)의 성긴 程度(sparseness)는 전체행열의 元의 數에 대한 非零(non-zero)元의 比인 密度(density)로 나타내며 特殊한 경우 (例로서 power supply system)를 제외하고는 1～5%미만이며 대규모 문제의 경우 0.1～1% 미만인 경우가 일반적이다[12]. 대규모선형계획 문제는 여러 계획기간에 따른 문제 (multi-period), 여러 조직 또는 地域間에 걸친 문제(muti-departmental), 또는 多段階(multi-stage) 문제 등이 포함되므로 자연히 성긴 문제행열을 갖기가 쉽다. 이러한 성긴 행열(supersparse matrix)을 다루는 技法은 여러가지가 있다[11]. 대부분의 기법이 非零元素(non-zero elements)와 관련된 情報만을 저장하므로 時·空兩面에 모두 有利하나 行 및 列의 兩方向으로의 資料檢索이 가능하도록 하는 資料構造(例로서 doubly-linked lists)를 構想해야 한다. 알고리즘을 效率的으로 수행하는 데는 原問題에 대한 資料構造와 더불어 어떻게 基底를 나타내고 update하느냐 하는 點이 더욱 重要하다. Graves의 알고리즘에서 kernel(A_{II}^{-1})은 그 크기가 反復時마다 變하므로 kernel自體는 挿入(insertion) 및 削除(deletion)를 容易하게 할 수 있는 자료구조를 가져야 함에 留意해야 한다. 結局 어떤 자료구조를 사용하여 kernel을 表示하는가에 따라 전체 알고리즘의 효율이 결정된다는 점이다.

위에서 설명한 바와 같이 問題行列의 非零元素만을 저장한다고 할 경우라도 문제의 크기가 커짐에 따라 必要로 하는 記憶容量이 커진다. 지금 記憶해야 할 정보는 行 또는 列番號, pointer 및 元素自體의 값이다. 여기서 具體的으로 어떤 list 構造를 가질 것인가에 따라 필요한 pointer의 數, 行번호와 列번호 또는 그 중 하나만이 필요한가가 결정되나 어느 경우든 元素의 값이 필요한 것에는 變함이 없다. 行번호, 列번호, 또는 pointer는 모두 整數이므로 IBM의 경우 2 bytes면 대부분의 경우($n \leq 2^{16}-1$)충분하나 元素의 값은 實數(floating point)이므로 적어도 4 bytes가 필요하다. 이 點에 着案한 것이 實數 pool(real numbers pool)이다. 즉 實務에서 當面하는 線型計劃 문제의 係數를 調査하여 보면 獨特한 實數(unigue real numbers)는 매우 制限되어서 大規模 선형계획 문제의 경우도 몇千個 程度에 지나지 않는다는 사실이다[8]. 많은 係數가 重復되어 쓰여진다는 뜻이다. 따라서 이 獨特한 實數만으로 實數 pool을 만들고 문제行列의 list에서는 實數自體 代身에 이 實數 pool에의 pointer만을 維持하면 4 bytes의 折

半인 2 bytes로 充分히 어느 實數든 間接的으로 表示할 수 있다는 생각이다. 勿論 實數 pool을 만들기 위한 hasing函數 등이 필요한 點, pointer 使用에 따른 알고리즘 수행時 間接 addressing에 필요한 經微한 시간損失등 短點이 있으나 필요한 記憶용량을 줄일 수 있다는 點은 大規模 선형계획 문제에는 必須的인 사항이다.

대규모 선형계획 문제가 一般的으로 성긴 문제行列을 갖는 것과 같이 많은 部分의 제약식이 generalized upper bound (GUB) 型이라는 點이다. GUB型 제약식의 特殊型으로 整數計劃에서 보는 選多型(multiple choice) 제약식은 그 代表的인 例이다. GUB型 제약식은 自然的으로 모형에 삽입되기도 하고 아니면 모형이 작성된 後 scaling (row and column scaling)에 의해 事前處理(preprocessing)되어 2次的으로 轉換되기도 한다. 이러한 GUB型 제약식이 경우에 따라서는 총제약식중 절반에 해당하기도 하며〔9〕, 따라서 이들 GUB型 제약식이 많이 존재함을 이용해 考案된 것이 Graves와 Mc Bride〔9〕의 因數分解(factorization) 方法이다. 이는 Danzig와 Van Slyke〔4〕보다 일반적인 것으로 모든 GUB型 제약식을 默示的(implicitly)으로 다루므로 基底變換에 전혀 計算負擔을 주지 않는다. 따라서 GUB型 제약식은 아무리 많아도 컴퓨터 記憶容量面이나 計算時間에 직접적인 영향을 주지않아 相對的으로 대규모선형계획문제를 풀 수 있게 해 준다.

古典單體法이나 修正單待法의 基底대신 kernel을 사용함은 기억해야 할 資料量의 감소, 또 그에 따른 계산量의 감소와 더불어 대규모선형계획의 경우 반복(iteration)회수가 증가함에 따라 발생하는 數値不安定(numerical instability) 문제를 다소나마 解決해 주고 定期的인 再逆行列(periodic reinversion)을 구하기가 보다 容易하다. 행렬의 크기가 커짐에 따라 數値不安定에 의한 fill-in 可能性은 보다 커지고 따라서 계산량이 증가함에 留意해야 한다.

分解技法(decomposition method)에 比해 特殊한 경우(subproblems이 trivial한 경우)를 除外하고는 因數分解方法이 대규모선형계획 문제에 대한 보다 一般的이고 효율적인 接近方法으로 보인다. 分解技法은 主問題(master problem)와 副問題(subproblems) 사이의 媒介변수(parameters)의 選定이 局地的이라는 알고리즘上의 문제와 主·副問題를 交互로 풀어야 하는 데서 오는 기억장치 사용의 非效率性, 모형의 定型化문제 등을 효과적으로 해결해야만 하는 短點이 있다.

5. 彈力計劃法

通常 線型計劃 모형이 갖는 의미는 제약식이 만족하는 범위내에서 目的함수가 最適이 되도록 하는 feasibility-optimality의 二段階로 구별해 볼 수 있다. 즉 第1次的인 目標(objective)가 제약식을 만족하는 것이고 第2次的인 目標가 목적함수를 만족(最適化)하는 2개의 優先順位(preemptive priority structure)를 갖고 있는 문제라고 볼 수 있다. 여기서 1次優先(priority level one)인 제약식의 만족은 絶對目標(absolute objective)이어서 違反을 許容치 않으며 2次優先은 하나의 목적함수를 만족(최적化)하는 일인 非絶對目標(nonabsolute objective)이다.

본節에서는 線型計劃의 제약식이 모두 絶對目標에 포함되어야 하는가 또는 하나의 우선순위(priority level one)에 동일하게 취급되어야 하는가에 대한 吟味와 方案의 模索이 그 대상이다.

어느 특정문제를 일단 선형계획 문제로 定型化한 後에는 單體法은 모든 제약식을 同一比重(하나의 優先順位內)으로 어느 제약식도 위반하지 않는 解(絶對目標)를 구한다. 그러나 實際問題를 定型化함에 있어서 模型作成者가 제약식이 갖기를 원하는 의미는 그렇지 않을 수가 많다. 예컨대 어느 제약식(例로서 構造的인 것)은 반드시 만족해야 하는 反面 다른 제약식(예로써 豫測値와 關聯된 것)은 어느 정도의 融通性을 가질 수 있는 것이다. 그러나 通常的인 선형계획에는 이러한 각각의 제약식에 대한 絶對目標에서의 偏倚정도를 配慮할 수 있는 裝置가 마련되어 있지 않다.

이에 대한 方案으로 解가 特定 제약식을 한單位 違反함에 대한 罰金(penalty)을 정하여 이를 목적함수에 反映하여 trade-off를 계산하고자 意圖에서 考案된 것이 彈力計劃法(elastic programming)이다. 즉 이 탄력계획법은 제약식의 優先順位를 다르게 함이 아니라 同一順位內에서의 加重値(weighting factor)에 의해 模型作成者의 의사에 接近하려는 방안으로 通常 선형계획의 脆弱點의 하나인 제약식이 절대목표인 점만을 緩和한 형태인 것이다.

이 彈力計劃法에서 사용하는 罰金은 事前双對變數값(à priori dual value)의 性格이며 이의 適正値를 정하는 어려움은 존재하나 이 計劃法이 'hard' 및 'soft' 제약식을 區別하여 보다 實際에 가까운 模型을 작성할 수 있게 해 준다는 점에서 장점이 된다. 또 실제 계산경험에 의하면 最適解에의 收斂이 빠르다는 點(反復回數의 감소)을 들고 있으나 이의 理論的인 뒷받침은 없다.

이 彈力計劃法은 從來의 目的計劃法 (goal programming) 과는 그 出發動機가 다르나 비교해 본다면 同一우선순위반을 허용하고 원래의 선형계획의 제약식마다 一方加重値를 사용한 것이라고 보면 된다. 單一優先順位이므로 우선순위의 槪念이 필요없고 모든 제약식의 제약위반과 목적함수는 同一단위로 計量할 수 있다는 (commensurable) 것이다. 원래의 선형계획의 틀에 목적계획법의 特殊型을 끼운 형태라고 볼 수 있다.

6. 資料管理시스템

線型計劃 해법의 발달과 컴퓨터科學의 발달에 힘입어 20年前의 30 × 30程度의 線型計劃問題에서 현재는 2000 × 8000의 문제를 常例로 취급하게 되었다〔15〕.

問題가 大型化함에 따라 從前의 단 한가지 關心事이었던 解를 만들어 내는 optimizer 에서 문제를 optimizer에 효율적으로 供給하는 matrx generator, optimizer의 出力을 原問題와 관련하여 解釋하고 樣式에 맞추어 報告하는 보고서작성기능, 이러한 資料·機能들을 管理하는 시스템 (DBMS) 이 보다 중요하게 抬頭됐다. 이러한 현상은 數理計劃技法 適用의 全體的인 효율向上이라는 觀點에서 當然한 歸結라고 보아 진다. 이는 두가지 點에서 설명될 수 있다고 본다. i) 이들 소프트·웨어는 經營科學者의 업무 부담을 덜어주고 管理者와의 효과적인 意思疏通의 tool을 提供하고 ii) 費用을 節減한다. 例로 하드·웨어는 매년 25 – 30 %씩 감소하는 데 비해 分析家의 人件費는 每年 9 %씩 增加한다〔15〕는 點이다. 여기서 LP資料管理 시스템과 관련된 문제는 어떤 資料관리 시스템을 導入하여 高價의 分析家 (management scientists) 의 生産性을 높일 수 있을 것인가 이다. 즉 最小의 努力으로 최대의 效果를 보고자 하는 것이다. 이에 대한 代替案은 i) 凡用言語의 사용 (例로 COBOL, FORTRAN. 과 PL/I) 과 ii) LP用 特殊言語 (DATAFORM, CFMS, MGG, GAMMA, ALPS, OMNI 등) 의 사용이다. 두개의 代替案중 Falk와 Tumbusch〔6〕는 凡用語를, Hirshfeld, Orchard-Hays 및 Halliburton〔15〕은 特殊目的言語를 選好하나 이는 사용자인 각 데이터·센터의 性格·人的構成과 관련되어 결정되어 질 性質의 것이라고 보며 여기서 중요한 사항은 어느 LP자료관리 시스템이건 optimizer와 더불어 전체 효율향상에 필수적인 요소라는 점이다.

現在 컴퓨터科學의 발달로 많은 應用分野가 實時間會話型으로 이루어지고 있다. 그러나 선형계획 문제는 反復的 解法을 이용하는 관계로 解를 實時間에 얻기 어렵고 따라서 一括處理方式 (batch process) 에 의한 것이 一般的이다. 이는 大規模線型計劃 문제와

整數計劃 문제를 고려하면 더욱 그러하다. 그러나 이는 단지 最適化段階 (optimizer) 에 적용되는 문제일 뿐이지 matrix generation 및 report writing에는 該當치 않는다. 이런 觀點에서 O'Neill 〔14〕은 보고서작성 部門에 사용하는 會話型檢索言語 (PERUSE) 를 試圖했다.

앞으로 이와같은 관리자와 經營科學者의 interface를 원활히 수행할 수 있는 領域의 소프트·웨어의 開發이 大規模선형계획의 常例化와 더불어 바람직하다고 본다.

Ⅳ. 結　　論

古典單體法과 Graves의 解法과는 그 接近에서 根本的인 차이가 있는 것이 아니고 問題의 表現 및 알고리즘上의 보다 發展된 形態라고 보는 것이 정확하다.

古典單體法은 Graves의 해법에 비해 다음과 같은 點에서 상이 하다. 즉 1) 退化문제에 대한 효율적인 해결방안, 2) 等式型 제약식과 自由變數 (free variable) 의 취급方法, 3) 餘裕·剩餘·人工變數등 많은 課外變數가 필요하 점, 4) 單位行列로 이루어지는 最初解가 필요한 것, 5) 많은 컴퓨터 기억용량이 필요하 점, 6) 대규모선형계획 문제에 대한 未備點등이다.

위의 대부분의 古典單體法의 制約은 보다 自然的인 行基底의 사용과 제약식을 있는 그대로 等式 또는 不等式 (≤) 으로 混用하는 것으로 많이 解消된다. Bordering方法에 의한 Pivot, 單一基本表에 의한 原·双對문제의 2重表現, kerhel 概念 등은 時間 및 記憶裝置의 節約을 가져 오는 알고리즘 上의 特記點이다.

如何間의 현재의 기술수준은 3000 ~ 6000個의 제약식을 가진 일반선형계획 문제를 別負擔없이 常例로 풀 수 있게 하고 있다. 앞으로 계속 컴퓨터의 성능은 向上되고 價格은 低下할 것이 分明하다. 本研究의 대부분은 선형계획 문제를 효율적으로 풀수있는 알고리즘에 관한 것이었으나 實相은 문제를 빨리 풀 수 있는 것과 함께 큰 문제를 풀 수 있게 하는 데 力點이 있었던 것이다. 아마도 현재 大規模모형이 많이 活用되지 않는 이유는 우리가 이의 解를 效率的으로 얻을 수 없어서가 아니라 커다란 문제를 理解하고 定型化하고, 그 結果를 실제 문제에 實行하는 효율적인 管理能力의 未洽인 것으로 생각된다. 그러나 대규모문제를 효율적 (時間的, 費用面에서) 으로 다룰 수 있는 알고리즘의 考案과 컴퓨터科學의 발달은 궁

극적으로 큰 문제의 관리능력을 향상시키는 方便이 될 것이다.

資料構造를 생각지 않고 考案된 알고리즘이 효율적인 것이 될 수 없듯이 현대의 數理計劃도 原來의 수리계획과 컴퓨터과학과의 學際的인 接近(interdisciplinary approach)이 이루어져야 한다고 본다. 本稿에서는 컴퓨터科學 쪽 技術(hashing, 자료구조, virtual memory 效果, 基底表現 등)과 關聯한 구체적인 알고리즘의 履行(implementation)은 다음 기회로 미루고 다루지 않았으나 이의 重要性은 거듭 强調되어야 할 것으로 본다.

참 고 문 헌

Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974, pp. 232-242.

Robert G. Bland, Donald Goldfarb and Michael J. Todd, "The Ellipsoid Method: A Survey," *Operations Research*, Vol. 29, No. 6 (November-December 1981), pp. 1039-1091.

Gordon H. Bradley, Gerald G. Brown and Glenn W. Graves, "Design and Implementation of Large Scale Primal Transshipment Algorthms," *Management Science*, Vol. 24, No. 1 (September 1977), pp. 1-34.

G B. Danzig and R. M. Van Slyke, "Generalized Upper Bounding Techniques for Linear Programming", *Journal of Computer and System Sciences*, Vol. 1, No. 3 (1967), pp. 213-226.

The Editor, "A Polynomial-Time Algorithm for Solving Linear Programs", *Mathematics of Operations Research*, Vol. 5, No. 1 (February 1980), p. iv.

Patrick G. Falk and James J. Tumbusch, "User Evaluation of MPSX/370", *SIGMAP Bulletin*, No. 23 (April 1978), pp 41-54.

G. W. Graves, *Mathematical Programming*, forthcoming.

_____, private communication.

G.W. Graves and R.D. McBride, "The Factorization Approach to Large-Scale Linear Programming," *Mathematical Programming*, Vol. 10 (1976), pp. 91-110.

V. Klee and G. L. Minty, "How Good is the Simplex Algorithm?", O. Shisha(ed.), *Inequalities III*, Academic press, 1972, pp. 159-175.

Donald E. Knuth, "*The Art of Computer Programming*, Vol. 1, Addison-Wesley, 1973.

C. B. Krabek, R. J. Sjoquist and D. C. Sommer, "The Apex Systems: Past and Future," *SIGMAP Bulletin*, No. 29 (April 1980), pp. 3-23.

David G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, 1973.

Richard P. O'Neill, "An Interactive Query System for MPS Solution Information," *SIGMAP Bulletin*, No. 23 (April 1978), pp. 31-40.

James S. Welch, "Answers Delayed Are Answers Denied," *SIGMAP Bulletin*, No. 27 (July 1979), pp. 16-33.

附　　錄

(P)　　Subject to

$$
\begin{aligned}
y_2 &= 2 \\
-y_1 - y_3 &= -2 \\
-y_1 - 3y_3 &\leq 1 \\
-y_2 + y_3 &\leq -1 \\
-y_2 &\leq 0 \\
-y_3 &\leq 0
\end{aligned}
$$

$$min\ 6y_1 + 3y_2 + 4y_3$$

(P)에 대한 雙對問題는

Subject to

$$
\begin{aligned}
(D)\qquad & -x_2 - x_3 = 6 \\
& x_1 - x_4 \leq 3 \\
& -x_2 - 3x_3 + x_4 \leq 4 \\
& x_3 \leq 0 \\
& x_4 \leq 0 \\
& max\ 2x_1 - 2x_2 + x_3 - x_4
\end{aligned}
$$

다음은 (P)에 대한 primal 알고리즘의 解이다.

初期表(initial tableau)는

FV1	V2	V3		
0	1	0	2	E1 ←
-1	0	-1	-2	E2
-1	0	-3	1	I3
0	(-1)	1	-1	I4
6	3	4	0	
	↑			

여기서 FVj은 free variable,

Vj 은 nonnegative variable,

Ei 은 equation,

Ii 은 inequality를 각각 表示한다.

FV1	I4	V3		
0	1	1	1	E1
-1	0	-1	-2	E2
-1	0	-3	1	I3
0	-1	-1	1	V2
6	3	7	-3	

(再 配 列 前)

I4	FV1	V3		
-1	0	-1	1	V2
①	0	1	1	E1 ←
0	-1	-1	-2	E2
0	-1	-3	1	I3
3	6	7	-3	

↑ (再 配 列 後)

E1	FV1	V3		
1	0	0	2	V2
1	0	1	1	I4
0	-1	-1	-2	E2
0	-1	-3	1	I3
-3	6	4	-6	

(再 配 列 前)

E1	FV1	V3		
1	0	0	2	V2
0	⊖1	-1	-2	E2 ←
1	0	1	1	I4
0	-1	-3	1	I3
-3	6	4	-6	

↑

(再 配 列 後)

E1	E2	V3		
1	0	0	2	V2
0	-1	1	2	FV1
1	0	1	1	I4
0	-1	-2	3	I3
-3	6	-2	-18	

(再 配 列 前)

E1	E2	V3		
0	-1	1	2	FV1
1	0	0	2	V2
1	0	①	1	I4
0	-1	-2	3	I3
-3	6	-2	-18	←

↑

(再 配 列 後)

E1	E2	I4		
-1	-1	-1	1	FV1
1	0	0	2	V2
1	0	1	1	V3
2	-1	2	5	I3
-1	6	2	-16	

최적解는 $y_1^* = 1$, $y_2^* = 2$, $y_3^* = 1$, $y_0^* = 16$

$x_1^* = 1$, $x_2^* = -6$, $x_3^* = 0$, $x_4^* = -2$, $x_0^* = 16$