0－1 배낭問題의 Algorithm 開発에 관한 研究

(The Development of Algorithm Method for 0－1 Knapsack Problem)

申 鉉 宰*

Abstract

Many methods have been developed to get a good Computation steps. I think that almost methods of them have been solved by using a theory of 〔Vj〕.

But I have thought that it Can be solved by an other method. This method is a way to get a Computations steps by using 〔Aj〕 instead of 〔Vj〕. It requires less Computation time than 〔Vj〕.

So I think that method is an efficient Algorithm about "the Development of Algorithm method for the 0－1 Knapsack problem."

1. 序 論

한정된 資源의 制約條件下에서 목적하고자 하는것을 最大化하려고 할 때 意思決變數를 선택하는 것은 0－1 計劃法의 배낭問題(0－1 Knapsack Problem)에 의해서도 決定된다.

지금까지 0－1 배낭문제는 여러 사람에 의해 研究되어 왔다. 이를테면 Greenberg and Hegerich 의 GH 法〔1〕이 있고 Korsh and Ingaragiola 〔2〕는 GH 法의 계산시간을 단축하기 위하여 매우 효과적인 KI 法을 개발하였고 Geoffrion〔3〕은 Lagrange 완화 개념을 도입(LGR法)하여 活用하였다. 또한 Horowitz and Sahni 〔4〕은 GH 法보다 우수한 分枝限界法(HS 法)을 개발하였으며, Nauss 〔5,6〕 and Ross and Soland 〔7〕은 일반화된 할당문제 설비능력 입지문제의 解에서 각각 Subprogram 을 개발하였고 Kochenberger and McCarl and Wyman〔8〕, Toyoda〔10〕, Loulou and Michaelides〔9〕, 朴淳達(PH 法) 등은 계산시간과 解의 精密度를 높이기 위해 多次元의 배낭문제(Knapsack Problem)을 研究하였다.

0－1 Knapsack Problem의 일반적인 모형은 크게 두 가지로 나누어진다. 즉 制約條件式에서 $i \leqq 1$ 과 $i > 1$의 次元이다. Greedy 方法은 일반식에서 $i = 1$일때 적용이 가능하고 $i > 1$일 때는 이용불가능〔12〕 하다.

본인은 $i \geqq 1$ (多次元問題) 일 때 Greedy法의 기여율과 벌과금을 혼합적용하여 解의 精密度를 적정선으로 유지하면서 계산시간을 단축하려고 할 때 효과적인 方法을 모색하고자 한다.

2. 0－1 배낭問題의 Model

0－1 Knapsck Problem 은 m개의 制約條件과 n個의 意思決定變數가 있을 때 最適意思決定變數를 선택하려면 다음과 같은 一般式에 의해 다루어진다.

$$(P) \quad \max \sum_{j=1}^{m} C_j x_j$$

$$S.t \sum_{j=1}^{n} w_{ij} x_j \leqq b_i \ (i=1, 2, \cdots\cdots m)$$

$$x_j = 0 \text{ 또는 } 1 \quad \forall_j$$

$$w_{ij} \geqq 0, \ b_i \geqq 0, \ C_j \geqq 0, \quad \forall_j$$

위의 式은 $S.t$ 양변을 b_i 로 나누어 표시하기도 한다.

$$(PI) \quad \max \sum_{j=1}^{n} C_j x_j$$

$$S.t \sum_{j=1}^{n} r_{ij} x_j \leqq 1 \ (i=1, 2, \cdots\cdots m)$$

$$x_j = 0 \text{ 또는 } 1, \ \forall_j$$

$$C_j \geqq 0, \ r_{ij} \geqq 0, \ \forall_j$$

* 仁川大學 工業經營學科

2·1 $i \leqq 1$ 일때

(P) 式은 다음과 같이 변한다.

$$\max \sum_{j=1}^{n} C_j x_j$$

$$\text{S.t} \sum_{j=1}^{n} w_j x_j \leqq b$$

$$x_j = 0 \text{ 또는 } 1, \quad \forall j$$

$$w_j \geqq 0, \ b \geqq 0, \ \forall j$$

위의 方法은 일반적인 Knapsack Problem 으로 알려져 왔다. Balas and Zemel [11] 은 (C_j / w_j) 를 계산하여 큰 것부터 $x_j = 1$로 두어 意思決定變數의 주변을 잘 처리하면 우수한 解를 얻을 수 있다고 하였다. 또한 Nauss [6] 은 $(C_1/w_1 \geqq C_2/w_2 \geqq \cdots\cdots \geqq C_n/w_n)$의 큰 것부터 最適解 $\bar{x}$ 를 구하여 目的函數($\bar{P}$)를 구할 때에 Dembo가 導入한 최적쌍대승수인 Lagrange 개념인 $\bar{\lambda}$ 를 다음과 같이 活用하였다.

For $\forall j = 1, 2, \cdots\cdots, r-1$, *if* $V(\bar{P}) - C_i + \bar{\lambda} w_i \leqq z^*$, $x_i = 1$,

For $\forall j = r+1, \cdots\cdots n$, *if* $V(\bar{P}) + C_i - \bar{\lambda} w_i \leqq z^*$, $z_i = 0$.

(z^* = 하한 한계)

2·2 $i > 1$일 때

여러개의 制約條件이 있을 때 각 制約條件을 만족시키면서 目的函數의 값을 最大化하는 方法으로 앞의 (P) 式을 (PI) 式으로 바꾸어 Penalty Factor (V_{ij})를 活用하기도 한다. 이는 C_j를 Penalty Factor 로 나누어 그 값중 큰 것부터 制約條件이 만족될 때까지 x_j 를 1로 놓아 계산하였다.

(1) Toyoda [10] 은 다음 식에 의해 적게 사용된 資源을 많이 사용하도록 고려하는 方法을 내놓았다.

$$V_j = IR_j \cdot (IR / \| IR \|)$$

$$= \sum_{j=1}^{m} r_{ij} \cdot d_i^0 / \left(\sum_{i=1}^{m} d_i^{02} \right)^{\frac{1}{2}}$$

$$d_i^0 = \sum_{i=1}^{m} r_{ij} x_j : \text{資源 } i \text{의 使用된 量}$$

(2) Kochenberger and McCarl and Wyman [8] 은 앞의 (P) 문제점에서 다음과 같은 공식을 만들어, 현재 남은 資源量의 消費量에 대한 比率을 消費量이 적은 것에 우선적으로 선택되도록하는 方法이다.

$$V_j = \sum_{i=1}^{m} a_{ij} / b_i^*$$

$$b_i^* = b_i - \sum_{j=1}^{n} a_{ij} x_j$$

(3) 朴淳達 [12] 은 (P) 문제에서 다음과 같은 공식을 유도하여 資源消費率이 가장 큰 것에 더 많은 比重을 두어 부족한 資源을 더욱 강력하게 保護하고자 하였다.

$$V_j = \sum_{i=1}^{m} (a_{ij} / b_i^*) + \alpha \max (a_{ij} / b_i^*)$$

$$\alpha \geqq 0.$$

3. Algorithm 開發의 節次

最適Knapsack Problem에서 目的函數를 決定할 때에 두가지 관점을 생각할 수 있다. 優先加重値가 주어지는 경우와 絶對的 優先이 주어지는 경우이다.

3·1 優先加重値가 주어지는 경우

우선가중치가 주어지는 경우에는 앞서 열거된 $i \leqq 1$의 方法과 $i > 1$의 方法을 혼합 형태이다. 즉 $i \geqq 1$에 적용이 가능하다. 여기서 目的函數를 最大化시키기 위한 方法으로 Penalty Factor (V_j) 대신 기여율($Aij = Cj/wij$) 을 利益의 最大에 반영하였다.

Cj/w_{ij} 이면 wij의 한 單位當 期待利益率이 계산된다. 이는 $i = 1$일 때 $i \leqq 1$의 方法과 같다. 여기서 wij의 單位當 期待利益率中 xi에 대한 利益率의 合 ($\sum_{i=1}^{m} Aij$) 을 계산할 수 있다. 또한 여기에다가 x_j의 列 Vector 중 wij가 큰 것은 그만큼 bi에 할당부담량을 많이 주게되어 bi에 대한 기여도가 높아지게 되므로 Aij는 적게 된다. wij의 單位當 利益率에 bi에 기여율이 큰 것을 다시 반영하면 wij가 큰 것은 그만큼 선택의 폭이 낮아지게 되기 때문에 資源의 制限量을 單位當 利益率에 따라 保護되며, 解의 精密度를 보다 높일 수 있게 된다. 그러나 여기에서 고려할 것은 wij의 계수가 bi의 量이 감소함에 따라 比重을 달리하고 있어 Aij의 큰 값을 반영하는 것이 아니고 Vj의 큰 값을 갖는 항의 Aij를 반영하여야 한다는 것이다.

이를 整理하면 다음과 같이 되고, Cj를 優先加重値로 보아서 계산하면 問題가 해결된다.

$$Aj = \sum_{i=1}^{m} Cj / wij + Cj / wij$$

$$(= \max_{i} wij / bi^* \text{의 항})$$

$$= \sum_{i=1}^{m} Aij + Aij \ (= max \ \ wij / \overset{*}{bi} \text{의 항})$$

for all j.

3·2 絶対的 優先이 주어지는 경우

일반적인 상황에 따라 어떤 意思決定變數를 꼭 선택하여야 할 경우가 발생할 때, 制約條件에서는 意思決定變數를 제외시켜 계산을 별도로 하였으나 意思決定變數를 선택할 때, 目標計劃法의 개념을 도입〔13〕하면 효과적으로 처리할 수가 있다. 필수적으로 選擇하고자 하는 의사결정변수는 $P_1 \gg P_2 \cdots\cdots \gg P_m$의 P_1에 해당되기 때문에 P_1은 꼭 選擇되어야 한다. 따라서 P_1은 Cj의 값에 영향을 받지 않도록 하기 위하여 선택되고자 하는 意思決定變數의 Cj값에 앞의 優先加重値가 아닌 Big M의 개념을 도입시켜 계산하면, 絶對的 優先이 주어지는 것은 Aij가 무한히 커지게 되어 자동적으로 선택이 되고 制約條件에 따라 그 다음 變數들은 優先加重値로 決定되게 된다.

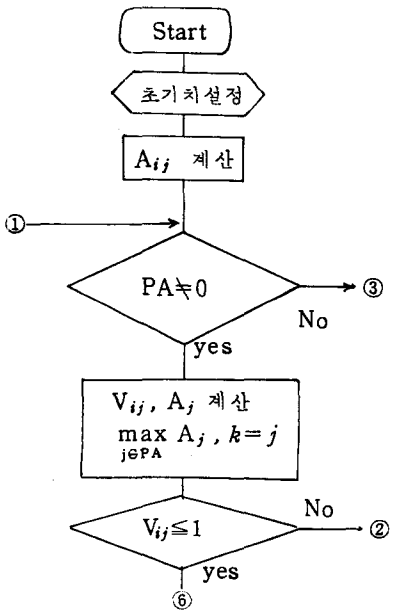
이때 目的函數는 다음과 같다.

$$max \ \sum_{j=1}^{n} Cj \ xj$$

($Cj = M$: 절대적 우선인 xj의 Cj값)

3·3 計算節次

Algorithm 開發의 흐름도는 그림 1과 같고 계산절차는 다음과 같다.



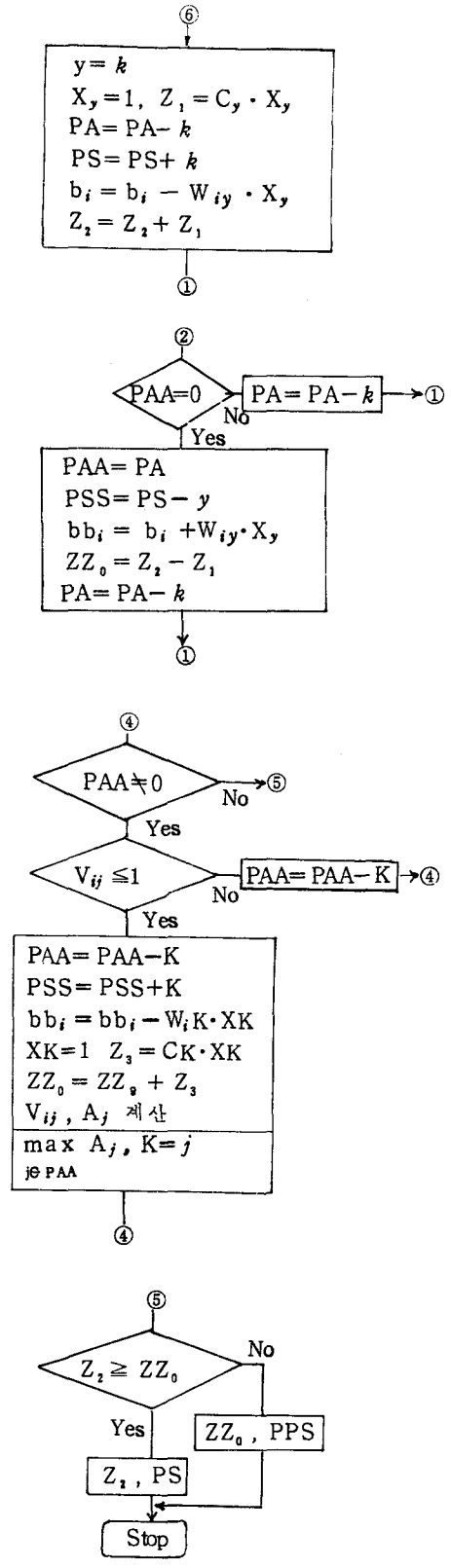


그림 1, **Algorithm** 개발을 위한 흐름도

단계 1 : 초기치를 모든 變數에 대해 설정한다.

단계 2 : 意思決定變數의 Cj를 wij로 나누어 單位當 利益率을 계산하고 xj에 대한 $\sum_{j=1}^{m} Aij$를 구한다.

단계 3 : 選擇可能變數의 集合〔$PA=0$〕이면 比較過程으로 간고 $PA \neq 0$ 이면 단계 4로 간다.

단계 4 : Vij (wij / bi)를 구하여 각 xj에 대해 $max\ Vij$에 해당되는 단위당 이익률의 값의 Aij를 찾아 Aj〔$= \sum_{i=1}^{m} Aij + Aij\ (= \max_{i} Vij)$〕를 계산하고 선택가능변수중 $\max_{i} Aj$의 j를 결정하여 $k=j$로 x_k를 구한다.

단계 5 : Vij가 1보다 크면 능력을 벗어나므로 나중에 비교하기 위한 Data 저장실로 가서 단계 3으로 가고 그렇지 않으면 단계 6으로 간다.

단계 6. : k를 y에 저장시키고 z_1의 값(임시목적함수값)을 계산하고, PA에서 선택된 것을 감해주고, PS(선택된 변수의 집합)에 더해주고, 새로운 bi의 값을 계산하고, 누적목적함수값을 기억시키고 단계 3으로 간다.

단계 7 : Vij가 1보다 클 때 선택가능변수들과 그때(n 회)의 선택결정변수와의 비교를 위한 결산절차가 진행된다. 이는 첫번째에 한해서만 입력자료가 기억되고 그 다음부터는 PA에서 k만 감해준다.

단계 8 : 단계 7에서 기억되었던 값들을 꺼내 단계 6에서 결정되었던 방법외에 다른 변수들의 조합으로 최대값을 찾는다.

단계 9 : 단계 6과 단계 8에서 구한 z_2 zz_0 를 비교하여 큰 것을 목적함수의 값으로 결정하며 그때의 Ps를 기록한다.

〈例〉 $max \quad 15x_1 + 6x_2 + 12x_3 + 12x_4 + 30x_5$

$3x_1 + x_2 + 6x_3 + 4x_4 + 10x_5 \leqq 15$

$7x_1 + 3x_2 + 3x_3 + 5x_4 + 7x_5 \leqq 15$

$xj = 0$ or 1

$\forall j$ ($j = 1.\ 2.\ 3.\ 4.\ 5$)

4. 結 論

많은 制約條件式을 갖는 배낭문제(0－1 Knapsack Proble)에서는 앞에 열거한 $i > 1$의 方法이 使用되어 왔다. 즉, 資源制限量의 變動에 따라 Cj / Vij와 資源消耗量이 큰 것을 반영하는 過程을 매 단계마다 반복계산하고 또한 選擇된 變數와 選擇可能變數의 값들을 매단계마다 計算하여 큰 것을 선택하는 것이다.

본 論文에서는 單位當 利益率(Aij)를 한번만 계산하고 解의 精密度를 높이기 위해 매단계마다 單位當 利益에 대해 資源消耗率이 많은 것(각 j에 대한 i) 을 일률적으로 더하면 資源消耗率이 낮은 것이 選擇의 폭이 커지게 됨을 이용한 것이다. 最適解의 比較는 資源의 制限量에 制約이 있을 때(n 단계) 까지 계속적으로 意思決定變數들을 선택한 후 制約이 걸리면 ($n-1$) 단계에서 選擇된 變數와 選擇可能變數를 比較하므로 比較過程은 마지막단계 가까이에서 발생하게 되어 많은 計算時間을 단축하게 되며 最適解에서도 차이가 없다.

例에서 다른 方法과 본 論文을 比較하면 반복계산 단계가 단축됨을 알 수 있고(j 가 많을수록 효과적) 最適解도 같게나와 매우 效率的인 방법이라 생각한다.

參 考 文 獻

1) Greenberg, H. and Hegerich, R., "A Branch Search Algorithm for the Knapsack Problem," *Management Science*, Vol. 16 No. 5 (January, 1970), pp. 327～332.

2) Ingarogiola, G. P. and Korsh, J. F. "Reduction Algorithm for Zero-one Single Knapsack Problems," *Management Science*, Vol. 20 No. 4 (December, 1973), pp. 460～463.

3) Geoffrion, A. M., "Lagrangean Relaxation for Integer Programming," *Mathematical Programming Study* 2 (1974), pp. 82～114.

4) Horowitz, E. and Sahni, S., "Computing Partitions with Applications to the Knapsack Problem," *Journal of the Association for Computing Machinery*, Vol. 21 No. 2 (April, 1974), pp. 277～292.

5) Nauss, R. M., "Parametric Integer Programming," Ph. D. Dissertation, University of Califonia, Los Angeles, 1974.

6) Nauss, R. M., "An Efficient Algorithm for The 0 - 1 Knapsack Problem," *Management Science*, Vol. 23 No. 1 (September, 1976), pp. 27～31.

7) Ross, G. T. and Soland, R. M., "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Mathematical Programming*, Vol. 8 No. 1 (1975), pp. 91～103.

8) Kochenberger, G. A., McCarl, B. A. and Wyman, F. P., "A Heuristic for General Integer Programming," *Decision Science* 5 (1974), pp. 36~44.

9) Loulou, R. and Michealides, E., "New Geedy-like Heuristics for the Multidimensional 0-1 Knapsack Problem," *Operations Research* 26 (1978), pp. 1101~1114.

10) Toyoda, Y., "A Simplified Algorithm for Obtaining Approximate Solutions to Zero-one Programming Problems," *Management Science* 21 (1975), pp. 1417~1427.

11) Balas, E. and Zemel, E., "An Algorithm for Large Zero-one Knapsack Problem," *Operations Research* 28 (1980), pp. 1130~1150.

12) 朴淳達, 「多次元배낭문제의 새로운 解法」, 대한산업공학회지 Vol. 9 No. 1 (1983), pp. 3~6.

13) 李相文·白淙鉉, 어퍼레이션스 리써어치, 서울 : 經文堂, 1982, pp. 142.