韓國 軍事運營分析 學會誌
第 11 券 第 2 號, 1985.12.

小型電算機를 이용한 線型計劃 解法의 效率性 提高에 관한 硏究

함 주 호*
박 순 달*

요 약 문

이 논문은 소형개인용전산기를 이용하여 많은 변수를 가진 선형계획법을 풀려고 할 때 사용될 수 있는 몇가지 기법을 소개한다. 이 논문에서는 이런 목적에 알맞는 單體法 計算技法, 파일의 종류, 그리고 주기억장치의 配分을 다루고 있다. 그리고 여러가지 형태의 문제에 대해 이들 결과의 타당성을 소형전산기로 실험하였다.

1. 序 論

소형전산기 특히 개인용 소형 전산기의 이용은 날로 증가하고 있다. 특히 최근 개인용 소형전산기도 주기억용량이 커짐에 따라 일반행정경영의 의사결정에 활용되고 있어 그 이용도가 더욱 높아가고 있다.

본 논문은 현재 우리나라에서 널리 활용되고 있는 개인용 소형전산기인 APPLE을 이용하여 일반행정·경영의 의사 결정에 널리 사용되고 있는 선형계획법의 활용도를 높이고자 한다.

앞으로 상당한 기간동안 우리나라에서 사용될 주력기종은 APPLE과 IBM PC가 아닐까 한다. IBM PC는 주기억용량이 비교적 크기 때문에 선형계획법 사용에 별로 제한을 받지 않지만 APPLE을 사용할 경우 주기억용량의 제한으로 APPLE을 통한 선형계획법의 실용도는 아주 낮다. 보통 APPLE을 이용하여 선형계획법을 풀 때 그 크기는 제약식이 50개에도 미치지 못한다. 그래서 본 논문에서는 APPLE (64K)에다 보조기억장치 (디스크)를 이용하여 제약식이 100 단위가 되는 실제 문제의 선형계획법을 풀 수 있게 하고저 하는 것이다.

이 문제를 해결하기 위해서는 먼저

첫째, 단체법 중에서 어떤 방법이 적절한가?

둘째, 디스크에 file을 저장할 때 file

*서울大學校 工科大學

형태는 어떤 것이 적절한가? 그리고 필요한 array종류와 量은?

세째, 주기억장치의 관리는 어떻게 배분하는 것이 좋은가?

를 해결하여야 한다. 그래서 앞으로 다루게 될 단체법 중 두가지의 계산 방법을 정의한 후 제 2 절에서는 디스크를 이용할 때 각 file 종류와 array를 결정하며 제 3 절에서는 주기억용량의 배분문제를 다루며 마지막으로 제 4 절에서는 실험결과를 토의하기로 한다.

여기서 다루게 될 선형계획법은 다음 식 (1)과 같이 일반한계 선형계획법문제

$$\text{Maximize } CX$$
$$\text{s.t. } AX = \mathbf{b}$$
$$\mathbf{l} \leq X \leq \mathbf{u} \quad \cdots\cdots\cdots\cdots\cdots (1)$$

단, $\mathbf{u}$는 上限

$\mathbf{l}$은 下限

(linear programming problem with bounded variables) 인데 이런 선형계획법 문제는 보통 수정단체법 (revised simplex method)가 사용되고 있으며 수정단체법의 흐름과 각 부분의 副프로그램(subroutine)의 이름은 다음과 같다.

단계 1 초기해 (INITIAL)

여유변수, 잉여변수 또는 인공변수로 초기해를 구한다. 이때 최초의 기저行列의 逆行列 B^{-1}는 I로 둔다.

단계 2 목적함수계수 (FORMC)

2 단계법 (2－phase method)를 사용함에 따라 각 단계에 따른 목적함수의 係數를 적절하게 취한다.

단계 3 단체승수 (BTRAN)

$\pi = C_B B^{-1}$를 구한다.

단계 4 진입변수결정 (PRICE)

(1) 할인

할인가 (reduced price)

$\overline{c_j} = \pi A \cdot j - c_j$를 구한다.

단 $A \cdot j$는 A의 j 列

(2) Min $\overline{c_j} = \overline{c_k}$를 구하여 만일 $\overline{c_k}$가 非陰이면 최적이다. 만일 아니면 x_k를 진입변수로 결정한다.

단계 5 進入列의 수정 (FTRAN) $\overline{A} \cdot k = B^{-1} A \cdot k$

進入列 $A \cdot k$를 수정한다. 즉 $\overline{A} \cdot k = B^{-1} A \cdot k$

단계 6 탈락변수의 결정 (CHUZR)

(1) 우변상수 수정

먼저 우변상수 $\mathbf{l}\mathbf{b}$를 수정한다. 즉, $\overline{\mathbf{b}} = B^{-1}\mathbf{b}$

(2) 탈락변수 결정

비율검정으로 탈락변수 x_r을 결정한다.

단계 7 기저수정 (UPBI)

기저 行列 B의 逆行列 B^{-1}을 수정한다.

단계 8 해의 수정 (UPSOL)

기저의 변화에 따른 해를 수정한다.

수정단체법은 잘 알려져 있는 바와 같이 基底行列의 逆行列을 어떻게 보관하고 수정하느냐에 따라 明示型 (explicit form of the inverse), 積算型 (product form of the inverse), 上下分解型 (LU factorization)으로 나누어 진다. 그러나 여기서 上下分解型은 프로그램이 복잡해지고 더욱이 이 방법은 대형문제에 사용하기 위한 방법이기 때문에 여기서 제외하였다.

그리고 선형계획법을 풀 때 단체표(simplex tableau)를 이용하는 방법(이하 單體法이라 부름)은 보편적으로 사용하지 않지만 만일 보조기억장치를 이용할 경우 수

정단체법과 효율면에서 얼마나 차이가 있을 것인지 분석하기 위하여 포함 시키기로 하였다. 그리고 明示型이 積算型보다 기억용량을 더 많이 차지하기 때문에 이 연구에 포함되는 선형계획법으로서는 다음 두가지, 즉,

單體法 : 單體表를 이용하는 方法

積算型 : 積算型 修正單體法

을 사용하기로 하였다.

이 두가지 方法의 기본적인 차이를 요약하면 다음 표 1 과 같다.

기법 / 단계	단체법	적산형
INITIAL	초기해	초기해
FORMC	2단계법에 따라 목적함수 결정	左 同
BTRAN	단체표를 수정함.	$\pi = C_B E_t \cdots E_1$ 단, $B^{-1} = E_t \cdots E_1$
CBAR		$\bar{c}_j = \pi A \cdot j - c_j$
PRICE		$\text{Min } \bar{c}_j = c_s$
FTRAN		$\bar{A} \cdot s = E_t \cdots E_1 \cdot A_s$
CHUZR	비율검정에 의해 기저탈락자 결정	(1) $\bar{\mathbf{b}} = E_t \cdots E_1 \cdot \mathbf{b}$ 구함 (2) 左 同
UPBI	上記 단체표에서 구해짐	새로운 기본 행렬 E_{t+1} 추가. 즉, B^{-1} 를 $E_{t+1}, E_t, \cdots\cdots E_1$ 으로 보관
UPSOL		$\bar{\mathbf{b}} = E_{t+1} \cdots\cdots E_1 \mathbf{b}$ 구함

표 1. 각 단체법의 비교

그래서 이 연구는 결국 上記와 같은 기본 두가지 단체법을 가지고 보조기억장치를 이용할 때와 하지 않을 때의 계산속도를 비교분석하기로 한다.

그리고 여기에 사용할 프로그램은 BASIC을 사용하기로 하였다. BASIC 을 사용하면 연산속도가 느리지만 이 연구의 목적이 소형전산기를 사용하고 보조기억장치의 효용도를 분석하는 것이기 때문에 사용하기 편리한 BASIC 언어를 interpreter로 사용키로 하였다.

2. 파일의 종류와 Array

보조기억장치를 이용할 때 얼마나큰선형계획법 문제를 풀 수 있는가? 이것은 결국 보조기억장치를 많이 이용하면 할수록 시간이 많이 걸리기 때문에 시간과 문제의 크기를 어느 정도 절충하지 않을 수 없다. 보기기억장치의 의존도와 문제해결시간, 문제크기의 관계는 그림 1 과 같이 나타나는데, 이 연구에서는 수백개의 制約式을 가진 선형계획법 문제를 수 시간대에서 해결한다는 목표를 가지고 연구를 진행하도록 한다.

$$\text{보조기억장치 의존도} = \frac{\text{변수보관에 사용된 보조기억용량}}{\text{변수보관에 사용된 주기억용량} + \text{변수보관에 사용된 보조기억용량}}$$

그런데 보조기억장치로써 디스크를 이용할 때는 자료를 디스크에 보관할 때 파일 (file) 단위로 보관하게 되고 이 파일을 다시 이용하고자 할 때는 다음 3 단계를 거쳐야 한다.

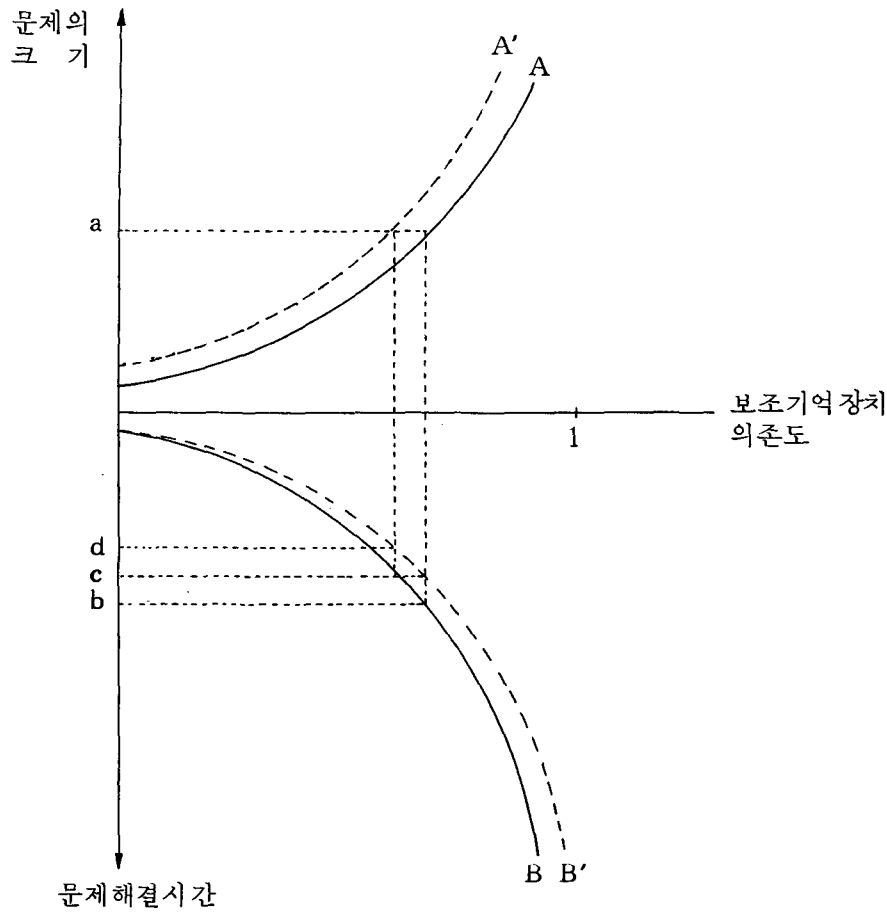
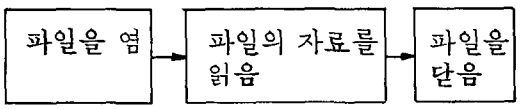


그림 1. 보조기억장치 의존도와 문제해결시간



이 파일을 여닫는 것은 대개의 소형전산기가 주기억장치에 이들 file로부터 읽어 들인 자료를 보관하는 Buffer 지역이 있고 이 buffer 지역은 대개 3개 파일의 자료를 읽어 들일 수 있게 해 두고 있다. 더 많이 파일로부터의 자료를 주기억장치에 보관하려면 보통 5개까지 되지만 그러면 주기억장치의 사용지역이 줄어 들게 된다.

디스크를 이용할 때는 주기억장치의 자료를 이용하는 것보다 이 세단계의 자료를 읽는 것이 훨씬 많은 시간이 걸린다. 그래서 디스크를 이용할 때는 파일을 여는데 시간이 걸리고, 파일의 자료를 읽는데 시간이 걸리며, 파일을 닫는데 시간이 걸린다. 그래서 가능한 디스크의 파일을 이용하는 횟수를 줄이며 뿐만 아니라 파일을 가능한 여닫지 않도록 하며 파일의 자료를 읽을때는 가능한 빨리 읽을 수 있도록 하여야 한다.

그래서 보조기억장치로써 디스크를 이용할 때는 다음의 작업, 즉

1. 파일의 자료를 빨리 읽을 수 있도록 함.
2. 보조기억장치의 이용횟수를 줄임.
3. 파일의 여닫는 횟수를 줄임.

이 중요하다. 첫째, 작업은 파일을 어떤 종류로 선택하느냐의 문제이고, 둘째는 디스크에 보관할 array의 갯수를 줄이는 것이고 세째는 파일을 열면 가능한 닫지말고 오래 열어 두도록 하는 것이다.

결국 작업 1은 문제의 크기와 보조기억장치의 의존도에 상관없이 문제해결속도를 줄이는 것이다. 그림 1에서 보면 곡선 B를 B'로 개선시키는 것이다. 작업 2는 array의 갯수를 줄임으로써 보조기억장치의 의존도를 낮추어 주는 역할을 하는 것인데 그림 1에서 보면 곡선 A를 A'으로, 곡선 B를 B'으로 개선시켜 주는 것이다. 작업 3은 작업 1과 비슷한 효과를 가져다 준다. 작업 1과 3은 이 절에서, 그리고 작업 2는 다음 절에서 처리하기로 한다.

대체적으로 선형계획법을 풀기 위해서는 20여개의 변수 array가 사용된다. 이 20여개의 array를 모두 주기억장치에 보관하면 자연히 각 array의 길이가 줄어지고 그래서 풀 수 있는 문제의 크기가 작아진다. 그러나 이 많은 array도 프로그램을 여러 단위로 나누어 차례로 수행하면 한 단위에서 필요로 하는 array는 수개로 줄게 되며, 따라서 한 array가 갖을 수 있는 길이는 증가하며 따라서 문제의 크기도 증가하게 된다. 그래서 분해하는 단위를 작게하면 작게 할수록 해결가능한 문제의 크기는 커진다. 그러나 단위를 작게 분해하면 할수록 시간이 증가하기 때문에 단위를 무조건 작게 할 수는 없다.

분해단위는 일반적으로 subroutine단위, Do-loop 단위, 연산단위 등으로 생각해 볼 수 있으나 연산단위는 보조기억장치의 존가 1에 가까우며, 시간이 막대하므로 고려가치가 없다. 그리고 subroutine 단위는 사용 array갯수의 감소량이 충분치 않아 결국 Do-loop단위로 분해하기로 하였다. 분해된 단위를 블럭이라 부르기로 한다.

분해단위를 Do-loop단위로 하기로 한다면 단체법의 각 단계에 Do-loop이 몇개 있고 각 Do-loop마다 어떤 변수가 있으며 각 변수의 종류와 길이가 얼마이며 만일 이 모든 변수의 값을 디스크에 파일로 저장한다고 가정했을 때 이 파일의 이용상태를 우선 알아야 한다. 그래서 LPSIMPB와 LPREPOB를 Do-loop 단위로 알아보면 그림 2과 그림 3와 같다. 각 변수의 뜻은 다음과 같다.

M : Constraint 갯수
N : 변수의 갯수
L : less than (≦) Constraint 갯수
E : equal (=) Constraint 갯수
G : greater than (≧) Constraint 갯수
$N_1 = N+G$
$N_3 = N+G+L+E+G$
A(I,J): Constraint Coefficient I = 1 to M, J = 1 to N
B(I): Right Hand Side (RHS) I = 1 to M
C(J): Objective Coefficient J = 1 to N
LB(J): Variable J의 lower bound J = 1 to N_3

UB(J): Variable J의 Upper bound J
= 1 to N_3

BS(I): I번째 Basic Variable I = 1
to M

NB(J): J번째 Nonbasic Variable
J = 1 to N_1

SG(J): Variable J의 Status
= -1: below lower bound
0 : between bounds
1 : above upper bound
2 : at lower bound
3 : at upper bound

	실 수								정 수		
길이 / sub 및 Do-loop 변수	M	M* N3	M	N	N3	N3	N3	N3	M	N1	N3
	BB AR	C1	AR	C	C22	C21	LB	UB	BS	NB	SG
INITIAL		X									
									X	X	X
	X	X					X				
							X	X			
FORMC	X						X	X	X		X
		X			X	/				X	
PRICE					X	/				X	X
CHUZR	X		X				X	X	X		X
									X	X	X
UP · SOL	X		X								
		X	X		X						
MAIN	X			X					X		
				X			X	X		X	X

그림2. LPSIMPB의 분석결과

※ 굵은 선은 디스크에 보관된 해당 변수 array (파일)의 사용을 나타냄. 4각형의 맨위에 표시되어 있는 것은 읽는것 (READ)를 뜻하고 아래에 표시되어 있는 것은 쓰는 것 (WRITE)를 뜻함.

BI (I,J): B^{-1} matrix I = 1 to M, J = 1 to M

PI (I) : πi I = 1 to M

CBAR(J): relative objective coefficient J = 1 to N_3

AR(I) : entering variable의 updated column I = 1 to M

RV : B^{-1}의 update 횟수

EC(RV): RV번째 entering variable

LR(RV): RV번째 dropping variable

	실수											정수				
길이 / sub 변수 및 Do-loop	M*N	M	M	M	M*M	M	M	N3	N3	N3	N3	M	N1	N3	N3	N3
	A	BB AR	PI	AR	BI	TR	TE	C1	CB AR	LB	UB	BS	NB	SG	EC	LR
INITIALIZE												X	X	X		
	X	X								X				X		
										X	X					
FORMC								X				X	X	X		
BTRAN			X		X			X				X				X
CBAR	X		X					X	X				X			
PRICE													X	X		
FTRAN				X	X		X									X
CHUZR		X		X						X	X	X	X	X		
UP · SOL		X		X					X							
												X	X	X	X	X
UP BI				X	X											
MAIN								X								
		X						X				X				
								X		X	X		X	X		X
REINV.							X									X
							X					X			X	X
					X	X	X								X	X

그림 3 . LPREPOB의 분석결과

2.1 사용 파일의 종류

소형컴퓨터에서 사용하는 파일은 크게 다음 두가지가 있다. 즉,

- sequertial file
- random access file

sequential file은 모든 자료를 入力 순서대로 보관하고 있으며 특정 자료에 대한 지수가 없으므로 이를 찾는데 불편하다. 그러나 모든 자료를 읽어 내는 시간은 random access file보다 적다. random access file은 특정 자료를 찾는데 용이하나, 모든 자료를 읽어내는 시간은 sequential file보다 길다. 따라서 각 변수가 사용되는 성질에 따라 보관 파일의 종류를 결정하여야 한다.

LPSIMPB와 LPREPOB에서 사용되는 각 변수의 대체적인 사용형태는 표2에서와 같이 간략히 살펴볼 수 있다.

표2. 각 프로그램에서의 변수 array 사용 특성

次 元	1 次 元 array	2 次 元 array
변수 array 이름	BBAR, CBAR, LB, UB, 등	단체표, A·BI 등
LPSIMPB	분해단위別로 일괄적인 사용	단체표—列단위 사용
LPREPOB		A, BI—列단위 사용

표2에서 변수의 일괄적인 사용이란, 분해단위를 Do-loop으로 한 경우, 분해단위내에서 사용되는 변수가 array 형태를 갖을 때, 이 중 몇개 특정 요소만이 필요한 경우는 거의 없으며 대부분 변수 array에 보관된 모든 값이 필요한 것을 말한다. 따라서 이 경우는 array 전체의 값을 한꺼번에 디스크로부터 들여오거나, 디스크로 내보내야 하는 경우이므로 sequential file이 적합하다. 2次元 array 변수의 경우 대부분 列 또는 行단위로 사용된다. 즉 한 Do-loop內에서 특정 列 또는 行만이 사용되거나 또는 내용 전체가 列 또는 行단위로 사용된다. 따라서 이 경우는 특정 列이나 行만이 필요한 경우이므로, 이를 찾아쓰기 용이하게 하기 위하여서는 한 列이나 行을 한 record단위로 하여 보관하는 것이 좋다. 이 경우 이 변수를 列단위로 보관할 것이냐, 行단위로 보관할 것인가를 결정해야 한다. LPSIMPB나 LPREPOB에서는 2次元 array 변수의 사용이 항상 列단위로 이루어지므로 아무이의 없이 한 列을 한 record로 하여 파일을 구성하는 것이 좋다.

표3. 각 프로그램에서 각 변수들이 보조기억장치에서 사용하는 File의 종류

프로그램	사용 File의 종류	
	Random Access File	Sequential File
LPSIMPB	CI(단체표)	BBAR, PI, AR, CBAR,
	A	LB, UB, BS, NB, SG
LPREPOB	A	BBAR, PI, AR, CI, CBAR
	BI (B^{-1})	LB, UB, BS, NB, SG, EC, LR

위의 사항을 고려하여 각 변수들이 사용하는 파일의 종류를 결정하면, 1次元 array 변수는 sequential file에, 2次元 array변수는 한 列을 1 record로 하여 random access file에 보관하기로 하였다(표3)

2.2 Array의 설정

소형전산기에는 보통 다음과 같이 3종류의 array가 있다.

- 실수 변수 array
- 정수 변수 array
- string variable array

또 각 array는 각각의 次元과 길이를 갖는다.

대체적으로 실수 array는 1개 요소에 4~5 바이트, 정수 array는 2바이트 사용한다. 여기서 주로 이 두 array만 사용하기 때문에 string variable array는 제거 한다. 결국 전술한 바와 같이 주어진 주기억용량에 큰 문제를 짧은 시간에 풀려면 보관할 array의 종류를 줄여야 한다. 그런데 이때 고려하여야 하는 것은 주기억장치에 array보관장소를 만들어 두었을때 긴 array는 더 짧은 길이를 가진 변수 array가 사용할 수 있으나 짧은 시간에 풀려면 보관할 array의 종류를 줄여야 한다. 그런데 이때 고려하여야 하는 것은 주기억장치에 array 보관장소를 만들어 두었을 때 긴 array는 더 짧은 길이를 가진 변수 array가 사용할 수 있으나 짧은 array는 더 긴 길이를 갖은 변수 array가 사용할 수 없다는 점과 실수 array에는 정수 array가 사용될 수 있으나 반대는 불가능하다는 점이다.

위의 사항을 고려하여 모든 단계가 수행되는 데 필요한 array 들의 최소 갯수를 구해보자.

지금 길이가 다른 실수 array가 m개, 정수 array가 n개 있다고 하자. 이들 길이를 l_i, $i=1, \cdots\cdots, m+n$로 두자. 처음 m개는 실수, 나중 n개는 정수이고 각각 길이가 큰 것이 먼저 나열되어 있다고 하자. 그러면 최소 필요한 i종류의 array갯수 q_i^*는 다음 식(1)에 의해 구해진다.

$$q_1^* = \max_k \{q_1^k\},\ k=1, \cdots\cdots, s$$

$$q_i^* = \max_k \left\{ \sum_{j=1}^{i} q_j^k \right\} - \sum_{j=1}^{i-1} q_j^*,\ k = 1, \cdots\cdots, s,\ i=2, \cdots\cdots, m$$

$$q_I^* = \max_k \left\{ \sum_{i \in T} q_i^k \right\} - \sum_{i \notin T} q_i^*,\ k=1, \cdots\cdots, s,\ I = m+1, \cdots\cdots, m+n$$

단, $T = \{j : l_j \geqq l_I,\ j = 1, \cdots\cdots, m+n\}$

여기서 s는 분해된 블럭의 총갯수를 나타내고 q_i^k는 i번째 변수가 k블럭에서 필요한 array갯수를 표현한다.

LPS IMPB에서 각 종류(길이)의 변수들이 각 단계에서 사용되는 갯수 (q_i^k)는 표4와 같다. 여기서 式(1)에 의해 LPSIMPB의 q_i^*를 구하면 다음과 같다. (실수 : $i=1,2,3$, 정수 : $i=4,5,6$)

$q_1^*=2$, $q_2^*=1$, $q_3^*=1$,
$q_4^*=1$, $q_5^*=1$, $q_6^*=0$

즉 실수 array는 길이가 N_3인 것이 2개, 길이가 N인 것이 1개, 길이가 M인 것이 1개 필요하며, 정수 array는 길이가 N_3인 것이 1개, 길이가 N_1인 것이 1개가 필요하다.

표 4. **LPSIMPB**의 각 단계에 필요한 **array** 갯수

종 류		실수			정수		
블럭 (k)	array종류 / subroutine	M	N	N3	M	N1	N3
1	INITIAL	1	0	0	0	0	0
2		0	0	0	1	1	1
3		2	0	1	0	0	0
4		0	0	2	0	0	0
5	FORMC	1	0	2	1	0	1
6		1	0	1	0	1	0
7	PRICE	0	0	1	0	1	1
8	CHUZR	2	0	2	1	0	1
9		0	0	0	1	1	1
10	UPSOL	2	0	0	0	0	0
11	UPBI	2	0	1	0	0	0
12	MAIN	1	1	0	1	0	0
13		0	1	2	0	1	1

단, $M \leqq N \leqq N1 \leqq N3$

비슷한 방법으로 LPREPO에 대해서도 구하여 종합하면 표5와 같이 된다. 이 표를 보면 필요한 array 종류가 절반이상 줄어들었음을 알 수 있다.

그래서 주기억장치에서 자료보관에 사용되는 양이 10k bytes라 할 경우 처음에는 30×30 정도의 문제만이 해결 가능하나, 보조기억장치 이용시에는 200×200 정도의 문제를 풀 수 있음을 알 수 있다.

3. 기억공간의 배분

여기서 다루려고 하는 문제는 앞에서 결정된 array 종류와 갯수를 가지고 가능한 효과적으로 활용하자는 것이다. 이 효과적 활용은 2가지 방법에 의해 가능하다.

첫째는, 주기억장치에 설정해 둔 array에, 예로써 Array K (길이는 K)에 가능한 많은 양의 자료를 보관하자는 것이다.

다음은 주기억장치에 설정해 둔 array에 가능한 같은 변수를 오래토록 사용하게 하는 것이다. 만일 주기억장치에 3개의

표 5. 각 프로그램의 **Array** 사용갯수 (원래사용갯수/최소설정갯수)

종 류	실 수				정 수				합 계
길 이	M	N	N1	N3	M	N	N1	N3	
LPSIMPB	4 / 1	1 / 1	0 / 0	4 / 2	1 / 0	0 / 0	1 / 1	1 / 1	12 / 6
LPREPOB	6 / 1	1 / 0	0 / 0	4 / 3	1 / 0	0 / 0	1 / 1	3 / 2	16 / 7

* disk access를 고려하여, 2 — dimension array는 column의 길이에 대응하는 1 — dimension array로 고려하였음.

array를 설정해 두었는데 디스크에 보관한 자료의 종류, 즉 파일갯수가 이보다 많으면 결국 필요할 때 디스크의 파일을열어 자료를 읽은 다음 주기억장치의 array 에 옮긴 후 파일을 닫는 일을 반복하여야 한다. 이는 많은 시간을 소요하기 때문에 가능한 주기억장치에 설정된 array를 한 변수가 오래 사용할 수 있도록 하여야 한다. 이렇게 함으로써 디스크의 파일 여닫는 시간 및 자료를 옮기는 시간을 줄일 수 있기 때문이다.

예로써 그림 6을 보자. 빈 곳에 각각 변수 x_i가 지정되어 있다. BBAR의 경우 단계 5, 6 에서는 사용치 않다가 단계 7 에서 사용된다. 그리고 다시 단계 8은 사용치 않다가 단계 9 에서 사용된다. 이곳을 x_1, x_2 라고 두자. 여기서 문제는 x_1, x_2 곳에 BBAR을 계속 CPU에 보관할 것인지 아니면 사용할 때만 disk 에서 읽어 사용후 다시 그 파일을 닫을 것인지 하는 것이다. 그래서 여기서 우리 목표는 가능한 $x_i = 1$로 두어 파일을 한번 읽었으면 그대로 CPU에 보관코저 하는 것이다. $x_i = 0$은 사용치 않는 것을 뜻하기로 한다.

그런데 x_i를 가능한 1로 만들고저 하지만 각종 array의 갯수 q_i^*가 이미 한정되어 있어 이 제약을 만족시켜야 한다. 그림 6의 경우 이 제약식은 다음과 같다.

단계 5 :

$x_7 + x_8 \leqq 1$

$x_1 + x_4 + x_7 + x_8 \leqq 2$

$x_7 + x_8 + x_{12} \leqq 2$

$x_1 + x_4 + x_7 + x_8 + x_9 + x_{12} \leqq 3$

단계 6 :

$x_7 + x_8 \leqq 1$

$x_1 + x_3 + x_4 + x_7 + x_8 + x_9 \leqq 3$

단계 7 :

$x_6 \leqq 0$

단계	sub 및 Do-loop / 변수명 / 길이	실수								정수		
		BB AR	C1	AR	C	C22	C21	LB	UB	BS	NB	SG
		M	M	M	N	N3	N3	N3	N3	M	N1	N3
5	FORMC	X_1		X_4				X_7	X_8	X_9		
6	PRICE											
7	CHUZR		X_3								X_{10}	
8		X_2		X_5		X_6						
9	UPSOL										X_{11}	X_{12}
10	PIVOT											

그림 6. LPSIMPD의 主 routine에서의 x_i

※ 굵은 선은 그림 3 에서와 同一한 뜻

$x_8 + x_6 \leqq 0$

$x_3 + x_6 + x_{10} \leqq 0$

단계 8 :

$x_6 + x_7 + x_8 \leqq 2$

$x_2 + x_3 + x_5 + x_6 + x_7 + x_8 \leqq 4$

단계 9 :

$x_6 + x_7 + x_8 \leqq 2$

$x_3 + x_6 + x_7 + x_8 \leqq 2$

$x_6 + x_7 + x_8 + x_{12} \leqq 3$

$x_6 + x_7 + x_8 + x_{11} + x_{12} \leqq 4$

$x_6 + x_7 + x_8 + x_9 + x_{11} + x_{12} \leqq 4$

단계 10 :

$x_7 + x_8 \leqq 2$

$x_1 + x_7 + x_8 \leqq 1$

$x_7 + x_8 + x_{12} \leqq 2$

$x_7 + x_8 + x_{11} + x_{12} \leqq 3$

$x_1 + x_7 + x_8 + x_9 + x_9 + x_{11} + x_{12} \leqq 3$

이상과 같은 제약을 만족시키면서 목표하는 바는 각 변수 x_i 부분에 속해 있는 변수 array를 읽고 기록하는 필요한 시간 C_i을 최대화하고저 한다. 이 시간을 최대화하면 결국 필요없이 파일을 읽고 기록하는 시간을 절약하게 된다.

이런 시간은 기종에 따라 또 문제의 크기에 따라 달라지나 여기서는 APPLE Ⅱ plus에서 APPLE DOS를 사용하고 문제의 크기가 200×200 (M=200, N=200), N1=400, N3=600을 기준으로 구하기로 한다. 우선 길이가 200, 400, 600인 array의 자료를 sequential file에 READ, WRITE하는 시간을 측정하여 보자.

길이 200: READ 6.64초 WRITE 8.17초

길이 400: READ 12.14초 WRITE 14.17초

길이 600: READ 17.74초 WRITE 20.17초

위의 시간과 그림 6에서의 굵은 선을 이용하여 C_i를 구하면 다음과 같다.

$C_1 = 14.81$ $C_7 = 17.74$

$C_2 = 6.64$ $C_8 = 17.74$

$C_3 = 6.64$ $C_9 = 12.14$

$C_4 = 0$ $C_{10} = 12.14$

$C_5 = 14.81$ $C_{11} = 26.31$

$C_6 = 37.91$ $C_{12} = 37.91$

앞에서의 제약식과 C_i를 이용하여 선형계획법으로 풀면 $x_2, x_5, x_7, x_9, x_{11}, x_{12}$ 는 1이 되며 나머지는 0이다. 또 목적함수의 값은 115.55이 된다. $\sum_{i=1}^{12} C_i = 204.79$ 이었던 것을 생각하면 디스크사용시간이 약 50 % 줄어듬을 알 수 있다.

이렇게 主 routine에 대해 배분이 끝났으면 部外 routine을 고려해야 한다. 部外 routine 중 INITIAL은 FORMC 앞에 들어가며 MAIN은 PRICE가 끝난뒤 수행되고 다시 PRICE로 들어간다. 이 위치를 고려하여 部外 routine의 기억용량 배분한다. 그 배분한 결과는 그림 7과 같다.

LPREPOD에서도 기억용량의 배분방법은 LPSIMPD와 거의 유사하나 主 routine은 FORMC → BTRAN → CBAR → PRICE → FTRAN → CHUZR → UPSOL → UPBI → FORMC이며 部外 routine으로 INITIAL은 FORMC 앞에, MAIN은 PRICE 다음에 수행된 뒤 CBAR로 들어가고, REINV는 CHUZR과 UPSOL 사이에 들어간다. 이에 따라 기억용량을 배분하면 그림 8과 같다.

여유기억용량을 최적으로 배분하여 감소된 disk access 횟수는 표 6과 같다. 여기서 主 routine은 매 iteration마다 지

(○표한 곳이 여유기억용량이 배분된 곳임)

	실 수								정 수		
길이 / sub / 변수 / 및 Do-loop	M	M* N3	M	N	N3	N3	N3	N3	M	N1	N3
	BB AR	C1	AR	C	C22	C21	LB	UB	BS	NB	SG
INITIAL		⊠									
									⊠	⊠	⊠
	⊠	⊠					⊠		○	○	○
	○						⊠	⊠	○	○	○
FORMC	⊠				○	◌	⊠	⊠	⊠		⊠
		⊠			⊠	⊠	○	○	○	⊠	○
PRICE					⊠	⊠	○		○	⊠	⊠
CHUZR	⊠		⊠				⊠	⊠	⊠		⊠
	○		○				○		⊠	⊠	⊠
UPSOL	⊠		⊠				○			○	○
PIVOT		⊠	⊠		⊠		○			○	○
MAIN	⊠			⊠			○		⊠	○	○
				⊠			⊠	⊠	○	⊠	⊠

그림 7. LPSIMPD를 위한 기억용량의 배분 결과

나가는 routine 이며 部外 routine은 프로그램 전체에 걸쳐 1번만 지나가는 routine 이며 Reinversion은 Reinversion이 필요한 경우만 지나가는 routine 이다.

따라서 UPSIMPD에서 disk access 횟수의 전체감소량은

12 × iteration 횟수 + 12 이며

LPREPOD에서 disk access 횟수의 전체감소량은 10 × iteration 횟수 + 8 + Reinversion 횟수 × 2 이다.

	실 수											정 수				
길이 / sub 변수 / 및 Do-loop	M* N	M	M	M	M* M	M	M	N3	N3	N3	N3	M	N1	N3	N3	N3
	A	BB AR	PI	AR	BI	TR	TE	C1	CB AR	LB	UB	BS	NB	SG	EC	LR
INITIAL												X	X	X		
	X	X								X		O	O	X		
										X	X	O	O	O		
FORMC								X				X	X	X		O
BTRN					X			X				X	O	O		X
CBAR	X		X					X	X				X	O		O
PRICE													X	X		O
FTRAN				X	X		X						O	O		X
CHUZR		X		X						X	X	X	X	X		
UPSOL		X		X					X			O	O	O		
				O								X	X	X	X	X
UPBI				X	X								O	O		O
MAIN								X					O	O		O
		X						X				X	O	O		O
								X		X	X	O	X	X		X
REINV							X					O		O		X
							X					X		O	X	X
					X	X	X					O		O	X	X

그림 8. LPREPOD를 위한 기억용량의 배분 결과

표6. **Memory Allocation**에 의한 **Disk Access** 횟수의 감소량

	LPSIMPD		LPREPOD		
	主 Routine	部 外 Routine	主 Routine	部 外 Routine	Reinversion
Memory Allocation	23+2N1	15	29+N1+2RV	16	2M+3
Memory Allocation	11+2N1	5	17+N1+2RV	8	2M+1
감 소 횟 수	12	10	12	8	2

4. 실험결과 및 분석

실험하는 문제는 일반한계문제로서

Maximize CX

Subject to AX ≦ **b**

l ≦X≦ **u** A:m×n

이다. 실험에 사용한 기종은 APPLE Ⅱ Plus이며 APPLE DOS上에서 실행하였다. 또 **b**를 제외한 모든 숫자는 2자리수 random number를 사용하였으며 random number generator로는 APPLE Ⅱ Plus에 내장된 RND function을 사용하였다. 또, 모든 문제는 feasible한 것을 사용하였다. 각 문제에서 A matrix의 sparcity는 거의 0 (zero)이다.

LPSIMPD, LPREPOD가 풀 수 있는 문제의 크기는 대개 400×400정도이나, 보조기억장치로 floppy disk를 사용하는 관계로 말미암은 보조기억용량의 제약으로 80×80까지의 문제에 한정하였고, 10×10부터 8단계로 나누어 각 크기에 대해 5번씩 시행하였다. 계산결과는 그림10과 같다. 이 시간은 BASIC으로 원프로그램을 Interpreter 그대로 사용한 결과이다. 이 프로그램을 compile 하면 훨씬 빨라질 것이지만 이 연구의 목적이 가능한 한 큰 문제를 푸는데 필요한 연구를 수행하는 것이기 때문에 comple을 하지 않고 사용하였다.

그림10에서 보면 40×40까지는 LPSIMPD가 좋으나 50×50부터는 LPREPOD의 수행속도가 빠르다. 또 이는 문제의 크기가 커질수록 더욱 차이가 많이 나게 된다. 또 기억용량의 배분을 한 경우와 하지 않은 경우에 대해서도 문제의 크기가 커짐에 따라 차이가 점점 더 커짐을 알 수 있다.

따라서 LPREPOD가 일정크기 이상의 문제에서는 LPSIMPD보다 좋음을 알 수 있다. 그러나 현 실험에서 사용한 제약식은 모두 ≦(less than)이다. 이것이 = (equal)로 바뀌면 일반적으로 iteration 수가 늘어나게 되며 따라서 LEREPOD에서는 reinversion의 횟수가 증가하게 되어 달라질 것이다.

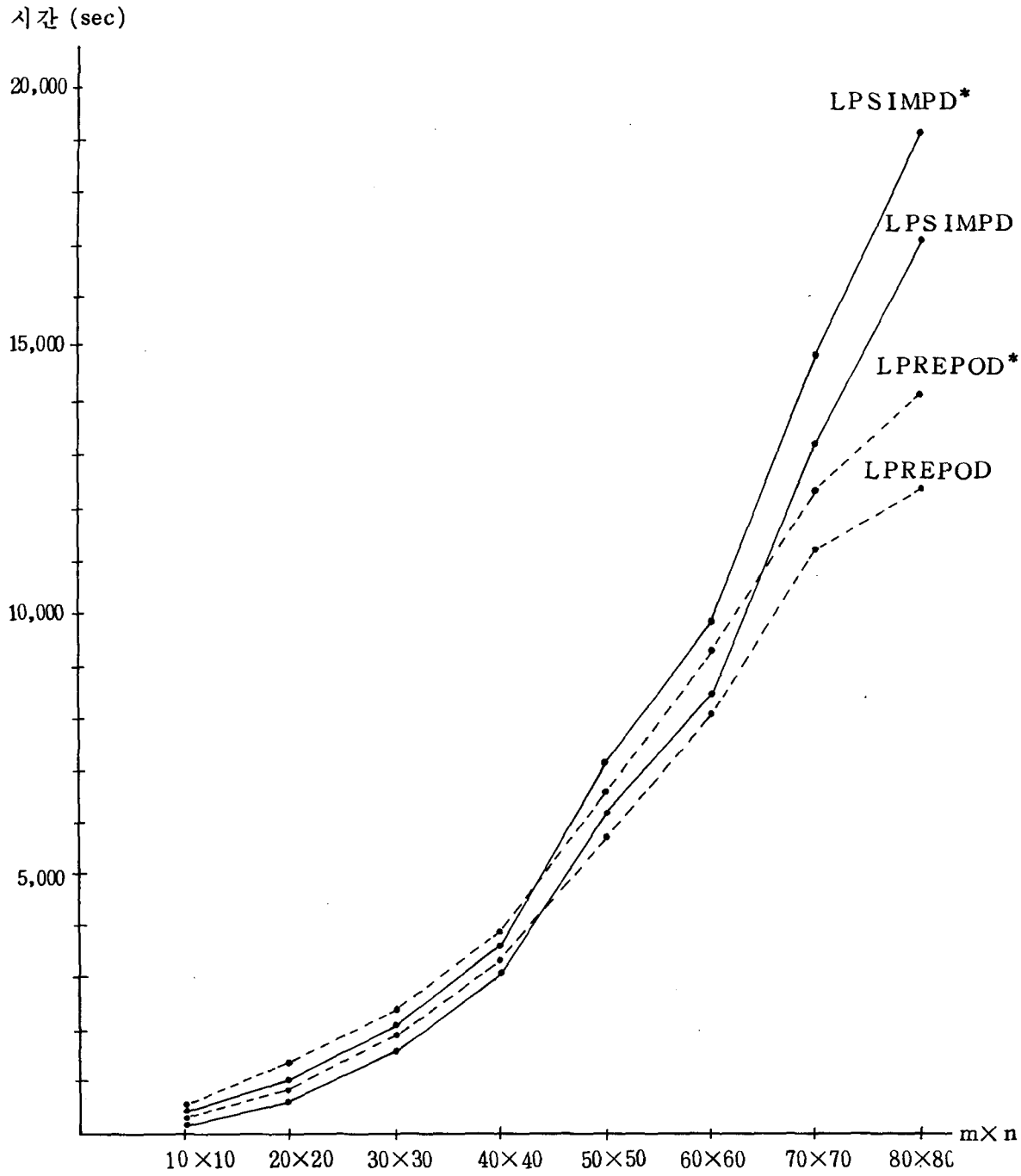


그림 **10. Disk Access** 를 하는 경우 각 프로그램의 수행시간

(LPSIMD*; LPSIMPD 에서 memory allocation 을 하지 않은 Program
LPREPOD*; LPREPOD에서 memory allocation 을 하지 않은 Program)

이 연구에서는 64KB이하의 주기억용량을 가진 소형전산기에서 100단위 이상의 선형계획법을 풀기 위해 보조기억장치를 효율적으로 사용하여 적정시간내에 해를 구하는 방법에 대해 연구하였다.

이 연구에서는 선형계획 프로그램을 여러 단계로 분해하여 각 단계에서 필요로 하는 변수를 파악하고, 모든 단계에서필요로 하는 array를 최소갯수와 길이로 설정하여 각 array의 길이를 최대화 함으로써 풀 수 있는 문제의 크기를 최대화 하였다. 그 결과 소형전산기에서 수백 단위의 LP를 적정시간에 풀 수 있게 되었다.

또 각 단계에서 필요로 하지 않는 변수를 보조기억장치에 보관하는 경우 각 변수에 적합한 file의 선택으로, 사용을 용이하게 하며, 사용시간을 단축하였으며, 매 단계에서 사용하지 않는 주기억용량은 보관가치가 높은 변수의 보관에 이용하여 보조기억장치의 이용시간을 단축시켰다. 그 결과 주기억용량의 잔여분을 이용하지 않은 경우보다 적어도 10 % 이상의 시간이 단축되었다.

參 考 文 獻

1) 朴淳達, 線型계획법, 大英社, 1983.
2) 〃 , OR프로그램집, 大英社, 1983.
3) 〃 , OR프로그램집 Ⅱ, 大英社, 1983
4) 〃 , 경영자를 위한 BASIC프로그램집, 大英社, 1984.
5) 洪永植, 嚴基賢, 자료구조, 正益社, 1982
6) Bruce A Murtagh, Advanced Linear Programming, McGraw Hill, 1981.
7) Aho, Hopcroft, ullman, The Design and Analysis of Computer Algorithm, Addson Wesley, 1976.
8) Rodnay Zaks, Programming the 6502, Sybex, 1983.
9) Apple Computer Co., APPLE II Refeence Manual, 1979.
10) Apple Computer Co., APPLE II Dos Manual, 1979.
11) Apple Computer Co., Apple Soft Monual, 1979.