단체법에서 기저역행렬과 입력자료의 보관방법과 자료구조

김우제**, 안재근*, 서용원*, 성명기*, 박순달*

** : 대진대학교 산업공학과, * : 서울대학교 산업공학과

요 약 문

본 연구에서는 기저역행렬의 계산방법에 따른 효율적인 자료구조를 실험적으로 검토하고, 입력자료방식과 효율화 방법을 제안하여 구현하였다. 기저역행렬의 계산방법에 따른 효율적인 자료구조는 명시형에서는 연결리스트 방식이 유리하였으며, 상하분해형에서는 연결 리스트 방법과 Gustavson 방법이 비슷한 효율을 보였다. 새로운 비영요소의 도입이 많은 경우일수록 연결 리스트가 효율적인 자료구조인 것으로 분석된다. 그리고 MPS자료의 입력방식과 효율화 방안에서는 각 열별로 행 정렬을 실시하고 해싱(Hashing)함수를 도입하여 효율화를 기하였다.

Keyword : 기저역행렬, 연결리스트, Gustavson 자료구조, 입력방식

1. 서론

단체법은 연립 일차 방정식의 특성에 기반을 두고 발전된 해법으로, 가장 널리 활용되고 있는 선형계획법 해법이다. 단체법은 초기해를 구하는 과정, 진입변수와 탈락변수를 선택하는 과정, 그리고 기저수정 과정 등으로 구성된다[1,8,9]. 이 과정들은 모두 기저역행렬을 이용해서 계산하는 과정이므로, 단체법의 수행 시간은 기저역행렬의 처리방법에 따라 크게 영향받는다. 기저역행렬을 계산하는 방법으로는 명시형(Explicit form), 적산형(Product form), 상하분해형(LU Decomposition)등이 있으며, 상하분해형은 다시 기저수정방법에 따라 Bartels-Golub 방법[3], Forrest-Tomlin 방법[7], Reid 방법[10,11] 등이 연구되었다. 이 방법들을 컴퓨터 프로그램으로 구현할 때, 우선 문제시되는 것은 기저역행렬을 보관하는 자료구조이다.

일반적으로 행렬을 보관하는 방법에는 행렬 전체를 모두 관리하는 방법과 행렬 내에서 비영요소만을 관리하는 방법이 있다. 전자는 행렬의 밀집도(Density)가 높은 경우에 사용되며, 후자의 방법은 희소행렬(Sparse matrix)인 경우에 사용되는 방법이다. 그런데 단체법으로 풀게 될 대부분의 선형계획문제는 희소행렬을 가지게 되므로, 단체법 프로그램은 비영요소만 보관하는 것이 효율적이다. 비영요소만 보관하는 방법에 대한 자료구조는 연결 리스트(Linked list)를 이용하는 방법과 희소벡터의 집합으로 표현하는 Gustavson 자료구조가 있다[5]. 따라서 기저역행렬을 비영요소만 보관하는 전략 하에서 기저역행렬의 계산방법에 따라 효율적인 자료구조에 대한 연구는 단체법 프로그램의 효율화에 대한 기여도가 높을 것으로 기대된다.

한편, 단체법을 컴퓨터 프로그램으로 수행할 때, 입력자료를 효율적으로 읽어들이는 것이 중요하다. 선형계획문제의 입력자료를 보관하는 방법의 대표적인 방법으로 IBM에서 제안한 MPS 보관방식이 있다[8]. 이는 MINOS, CPLEX, LINDO등 대부분의 상용 패키지들에서 표준적으로 사용되고 있으며, 선형계획법 프로그램의 실험 자료로 널리 쓰이고 있는 NETLIB 자료도 MPS형으로 되어 있다. 그런데 이 자료를 선형계획법 프로그램의 내부 자료 구조로 읽어들이는 방식을 어떻게 설계하는가에 따라 자료 입력에 걸리는 시간에 큰 차이가 나게 된다. 따라서 MPS형 자료를 읽어들이는 과정에서 나타나는 문제점들을 분석하고, 효율적으로 읽어들일 수 있는 방법에 대한 연구도 필요하다.

본 연구의 목적은 기저역행렬의 계산방법에 따른 효율적인 자료구조를 실험적으로 검토하고, 입력자료방식과 효율화 방법을 제안하고자 한다.

이를 위해 본 연구에서는 다음과 같은 사항을 연구 목표로 한다. 첫째, 명시형에서 연결 리스트 자료구조와 Gustavson 자료구조를 설계하고 실험결과를 분석하여 효율성을 검토한다. 둘째, 상하분해형에서 Bartels-Golub 방법과 Reid 방법에 대해 연결 리스트 자료구조와 Gustavson 자료구조를 설계하고 실험결과를 분석하여 효율성을 검토한다. 마지막으로 입력자료의 방식과 효율화 방안을 설계하고 실험결과를 분석한다.

2. 단체법과 자료구조

(1) 입력 방식과 자료구조

MPS형 자료는 NAME, ROWS, COLUMNS, RHS, RANGES, BOUNDS, ENDATA등 7개의 영역으로 구성된다. NAME은 문제의 이름을 기록하는 영역으로, 1-4열은 NAME이라는 명령어를 쓰고 문제의 이름은 15-22열에 기록한다. ROWS는 선형계획문제의 각 제약식의 이름을 등록하는 영역이다. MPS형 자료에서는 목적함수도 제약식과 비슷하게 취급하여 하나의 행으로 간주한다. ROWS 영역은 1-4열에 명령어를 쓰고 제약식의 형태(2-3열)와 행의 이름(5-12열)을 연속적으로 나열하게 된다. COLUMNS 영역은 목적함수 및 제약식 계수행렬의 비영요소를 기록한다. 형식은 1-7열에 COLUMNS라고 쓰고, 변수의 이름(5-12열)과 목적함수 및 제약식 계수행렬의 이름(15-22열, 40-47열)과 비영요소의 값(25-36열, 50-61열)을 연속적으로 나열하게 된다. RHS는 비영의 우변상수를 기록하는 영역으로, 1-3열에 RHS라고 쓰고, 각 제약식의 우변 상수 중 비영인 것들을 기록한다. RANGE 영역은 우변상수의 범위를 기록하는 영역으로 1-6열에 RANGES라고 쓰고 COLUMNS 영역과 비슷한 방법으로 값의 범위를 기록한다. BOUNDS는 변수의 상·하한을 기록하는 영역으로, 1-6열에 BOUNDS라고 쓰고 각 변수의 상·하한을 기록한다. RANGES와 BOUND는 문제에 따라 생략될 수 있다. ENDATA는 MPS형 자료의 입력이 끝났음을 알리는 것으로 1-6열에 ENDATA라고 쓴다.

한편, 입력 자료를 보관하는 자료구조는 크게 행렬 전체(Full matrix)를 보관하는 방법과 행렬의 비영요소만 보관하는 방법으로 나누어볼 수 있다. 특히, 계수 행렬은 양이 많고 비영요소의 희소도(Sparsity)가 높으므로, 비영요소만 보관하는 방법을 쓰게 되는데, 압축된 비영요소 보관 형태(Packed form)가 많이 사용된다. 이것은 각 열에서 비영인 요소들의 값과 행번호를 일차원 배열에 저장하고 각 열의 시작위치와 비영요소의 갯수를 보관하는 방식이다. 연산 과정의 효율을 위해 각 열에서 자료들은 행번호 순서로 미리 정렬해 두기도 한다. 다음 예제 행렬 A를 압축된 비영요소 보관 형태로 표현하면 다음과 같다.

$$A = \begin{bmatrix} 0 & 1 & 2 & 4 \\ 13 & 0 & 0 & 15 \\ 0 & 0 & 10 & 0 \\ 9 & 0 & 0 & 3 \end{bmatrix}$$

배열 주소	0	1	2	3	4	5	6	7
비영요소	13	9	1	2	10	4	15	3
행번호	2	4	1	1	3	1	2	4
열 시작위치	1	3	4	6				
각 열 비영요소 갯수	2	1	2	3				

압축된 비영요소 보관 형태는 기존 비영요소의 삭제나 새로운 비영요소의 추가 작업이 불편하기 때문에, 입력 자료와 같이 한번 입력하면 수정이 필요 없는 자료의 보관에 주로 사용된다[2].

MPS형 자료를 압축된 비영요소 보관 형태로 읽어들이는 경우 각 비영요소의 행·열 이름의 검색, 분산되어 있는 자료의 각 열별 정리, 각 열의 비영요소를 행번호 순서로 정렬하는 과정 등이 필요하다. 그런데 이들 과정을 설계하는 방법에 따라 자료 입력 시간에 큰 차이가 나게 되므로 효율적인 자료 입력 과정을 설계하는 것이 중요하다.

(2) 기저역행렬 계산방법과 자료구조

① 기저역행렬 계산방법

기저역행렬을 만드는 방법은 명시형, 적산형 그리고 상하분해형이 있다. 이 중 명시형과 상하분해형은 다음과 같다.

· 명시형

명시형에서 주어진 기저역행렬을 수정하는 방법은 다음과 같다.

$B=[\ B_1, B_2, B_3, \cdots, B_n\ ]$ (단, B_i는 B의 i열)이라 할 때, B_r과 $A_{.s}$를 교환하면 다음과 같다.

$\overline{B}=B[\ e_1, e_2, \cdots, e_{r-1}, \overline{A}_{.s}, e_{r+1}, \cdots, e_n] = B \cdot F$ (단, $\overline{A}_{.s}=B^{-1}A_{.s}$, $e_i=(0,\cdots,0,1,0,\cdots,0)^T$)

따라서 $\overline{B}^{-1}$는 다음과 같다.

$$\overline{B}^{-1}=F^{-1}B^{-1}$$

단, $F^{-1}=\begin{bmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & \eta_s & & \\ & & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \end{bmatrix}$ 단, $\eta_s=\begin{pmatrix} -\overline{a}_{is}/\overline{a}_{rs} & , & i \neq r \text{ 일때} \\ 1/\overline{a}_{rs} & , & i=r \text{ 일때} \end{pmatrix}$

· 상하분해형

상하분해방법은 $Bx=b$ (B : n×n 행렬, x, b : n×1 벡터)를 $LUx=b$로 분해하여 $Ux=c$ 와 $Lc=b$ 형태의 치환연산(Substitution)으로 푸는 방법이다.

상·하삼각행렬을 U와 L^{-1}형태로 보관할 경우 이를 수정하는 방법으로는 Bartels-Golub 방법, Forrest-Tomlin 방법 그리고 Reid 방법 등이 있는데, 이 중 Bartels-Golub 방법과 Reid 방법은 다음과 같다.

- Bartels-Golub 방법

새로 진입되는 열을 $A_{.j}$라고 하면 U의 탈락열 자리에 $L^{-1}A_{.j}$가 들어가게 되는데 이 열을 맨 뒤로 보낸 뒤 상삼각행렬을 만들어 주기 위해 대각요소 아래 부분이 0이 되도록 기본행연산(Elementary row operation)을 실시한다. 이때 수치 안정성을 고려하여 U와 L^{-1}를 수정한다.

- Reid 방법

Bartels-Golub 방법과 달리 U의 희소성을 이용한 방법으로, U의 아래 쪽에 있는 비영요소들을 가급적 U로 흡수하여 연산량과 비영요소의 갯수를 줄이는 방법이다. Reid 방법에서는 새로운 비영요소의 도입을 억제하기 위해 여러 번의 치환과정을 거쳐 U의 아래 쪽에 있는 비영요소를 줄여 기본행연산의 횟수를 줄인다. 여기서도 Bartels-Golub 방법과 같이 안정성을 고려한다.

② 자료구조

행렬을 컴퓨터 메모리에 저장할 때 비영요소의 수가 많으면 모든 요소를 저장하는 것이 가능하지만 행렬이 크고 또 비영요소의 수가 매우 적은 희소행렬의 경우는 비영요소만 보관하는 방법이 기억장소 및 계산 시간 측면에서 효율적이다. 행렬의 비영요소만을 보관하는 방법은 여러 가지가 있는데 이 중 연결 리스트(Linked list)로 보관하는 방법과 Gustavson 구조로 보관하는 방법은 다음과 같다.

· 연결 리스트 구조

연결 리스트 구조는 비영요소들을 행과 열의 링크(Link)로 연결시켜 놓은 구조이다.

행렬이 수정되면 요소들의 값이 변하게 되므로 비영요소의 삽입, 삭제가 필요하고 또 같은 행·열에 있는 비영요소를 검색할 필요가 있기 때문에 비영요소의 값, 행·열 번호, 다음 비영요소의 주소, 각 행·열의 시작 주소를 보관하는 배열 등이 필요하다[2].

예제 행렬 A를 연결 리스트로 표현하면 다음과 같다..

배열 주소	0	1	2	3	4	5	6	7	8	9	10
비영요소	9	1	*	10	3	4	2	15	*	13	*
행 번 호	4	1	0	3	4	1	1	2	0	2	0
열 번 호	1	2	*	3	4	4	3	4	*	1	*
행방향 링크	4	5	*	#	#	#	1	9	*	#	*
열방향 링크	9	#	*	6	7	#	#	5	*	#	*

행 시작주소	6	7	3	0
열 시작주소	0	1	3	4

* : 임의의 값, # : link의 끝임을 알려 주는 특정한 값

· Gustavson 구조

Gustavson 자료구조는 행렬의 비영요소만을 행별로 보관한다. 즉, 비영요소를 보관하는 배열과 각 비영요소의 열 번호를 보관하는 배열이 있고 각 행의 시작번지와 비영요소 갯수를 보관하는 배열이 있다. 또 각 열의 비영요소의 행번호를 알 필요가 있기 때문에 각 열의 비영요소의 행번호를 보관하는 배열이 있고, 이 배열에서 각 열의 시작번지와 비영요소 갯수를 알려주는 배열이 있다.

예제 행렬 A를 Gustavson 구조로 나타내면 다음과 같다.

배열 주소	0	1	2	3	4	5	6	7	8	9	10	11	12
비영요소	1	4	2	*	*	3	9	13	15	*	10	*	*
열번호	2	4	3	0	0	4	1	1	4	0	3	0	0

행 시작주소	0	7	10	5
행 길이	3	2	1	2

행번호	2	1	4	1	0	0	4	2	0	1	3	0	0

열 시작주소	6	3	9	0
열 길이	2	1	2	3

* : 임의의 값

Gustavson 구조에서는 같은 행(또는 같은 열)에 있는 비영요소는 연속해서 나타나야 한다. 위의 예에서 4행에 새로운 비영요소가 생기는 경우에는 4행의 마지막 요소 뒤에 여유공간이 없으므로 그 행 전체를 맨 뒤로 옮긴 다음 새로운 비영요소를 넣는다[5].

새로운 비영요소가 많이 생겨서 더 이상의 여유공간이 없게 되면 공간을 제거해주는 압축(Compression)과정을 통하여 기억용량을 효율적으로 사용한다.

Reid에 의하면, 선형계획법 프로그램인 LA05의 경우, 초기(LA03)에는 기저역행렬(정확히는 상삼각행

렬)의 보관에 연결 리스트를 사용하였으나, 당시로서는 정수로 링크를 나타낼 경우 약 32000개 이상의 비영요소를 사용할 수 없고, 페이징(Paging)시의 문제점들 때문에 Gustavson 자료구조로 전환하였다고 한다[10]. 그러나 최근의 컴퓨터들에서는 4바이트 정수를 사용하고 페이징 기법에서도 많은 개선이 이루어졌기 때문에, 연결 리스트를 좋은 대안으로 생각할 수 있어 이들에 대한 새로운 비교 검토가 필요하다.

3. 입력자료구조와 효율화

MPS형 자료를 읽어들일 때에는 자료의 정렬방법 및 행과 열의 검색과정과 보관방식이 효율화에 영향을 미친다. 입력자료를 보관하는 방식에는 모든 요소를 보관하는 자료구조와 비영요소만을 보관하는 자료구조가 있다. 특히, 비영요소만을 관리하는 자료구조에서는 자료의 정렬방법과 행, 열의 검색과정에 따라 선형계획법의 수행에 큰 영향을 미친다.

행과 열의 번호를 기준으로 자료를 정렬(Sorting)하는 방법에는 Bubble sort, Selection sort, Insertion sort, Quick sort, Recursion을 제거한 Quick sort 등이 있다. 앞의 세 가지는 $O(n^2)$, 뒤의 두 가지는 평균 $O(n\log n)$의 시간에 정렬을 수행하는 방법이다[12].

행과 열을 검색하는 방법으로는 선형검색(Linear Search)방법과 해싱(Hashing)방법 등이 있다. 선형검색방법은 행·열의 이름과 번호를 배열에 순차적으로 기록한 후, 순차적으로 검색하는 방법이다. 해싱(Hashing)방법은 행·열의 이름을 특정한 주소로 사상(Mapping)하는 함수를 이용하여 검색하는 방법이다.

여기서는 MPS형 자료를 읽어들이는 과정의 효율화를 위해 자료의 정렬방법에서는 전체의 비영요소를 정렬하는 방법보다 각 열들을 행번호 순서대로만 정렬하는 방식으로 설계하였으며, 행과 열의 검색과정에서는 해싱(Hashing)방법을 사용하였다. 또 각 열에서 비영요소의 희소도(Sparsity)가 상당히 높으므로, 전체 비영요소에 대하여 정렬하는 방법보다는 각 열의 자료에 대한 정렬을 여러 번 하는 것이 속도를 향상시킬 수 있다.

본 연구에서는 Bubble sort, Selection sort, Insertion sort, Quick sort 및 Recursion을 제거한 Quick sort의 5가지 정렬 방법들에 대해 전체요소를 모두 정렬하는 경우와 각 열별로 행번호 순서대로 정렬하는 방법을 [표 1]과 같이 비교하였다.

[표 1] 정렬 방법 및 검색 방법들 사이의 수행 시간 비교

실험기종 : SUN SPARC 10 단위 : 초

문제명	문제크기	전체 비영요소 정렬					열별 행번호 정렬					검색 방법	
		Selection sort	Insertion sort	Bubble sort	Quick sort	Recursion 제거한 Quick	Selection sort	Insertion sort	Bubble sort	Quick sort	Recursion 제거한 Quick	Linear	Hashing
AFIRO	27× 32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
SC105	105× 103	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BRANDY	220× 249	1.4	0.3	1.4	0.8	0.2	0.2	0.1	0.2	0.2	0.2	0.4	0.1
STANDATA	359× 1075	2.7	0.4	2.7	1.4	0.4	0.2	0.2	0.2	0.2	0.2	1.4	0.2
STAIR	356× 467	4.2	0.4	4.3	1.8	0.4	0.3	0.3	0.3	0.3	0.3	1.3	0.3
DEGEN2	444× 534	4.5	0.5	4.4	1.5	0.5	0.5	0.4	0.4	0.4	0.4	1.8	0.4
FORPLAN	161× 421	2.2	0.2	2.3	0.3	0.2	0.5	0.4	0.4	0.4	0.4	0.8	0.4
GANGES	1309× 1681	13.1	1.0	13.2	5.3	1.2	0.7	0.7	0.7	0.7	0.7	11.4	0.7
GROW22	440× 946	18.4	1.2	18.1	5.4	1.1	0.8	0.7	0.8	0.7	0.8	4.4	0.7
SIERRA	1227× 2036	14.6	1.3	14.9	3.0	1.3	0.8	0.8	0.8	0.7	0.8	16.9	0.8
SHIP12S	1151× 2763	18.1	1.3	17.9	8.6	1.3	0.9	0.8	0.9	0.9	0.8	11.5	0..8
PILOTNOV	976× 2172	45.0	1.9	44.9	12.6	1.8	1.2	1.2	1.2	1.2	1.2	14.5	1.2
80BAU3B	2262× 9799	1:57.3	4.8	1:56.5	57.1	4.8	3.2	3.2	3.4	3.1	3.1	1:59.5	3.2
DFL001	6071×12230	5:33.1	6.5	5:30.4	2:39.4	4.9	4.2	4.2	4.1	4.1	4.4	5:13.9	4.2

실험 결과를 보면 전체 비영요소를 정렬하는 방법에서는 Recursion을 제거한 Quicksort와 Insertion sort가 비슷하게 우수함을 알 수 있다. 또 각 열별로 비영요소를 정렬하는 방법에서는 각 정렬 방법간에 큰 차이가 없었는데, 이는 희소도가 높은 선형계획문제의 특성상, 각 열의 비영요소의 갯수가 적기 때문인 것으로 분석된다. 전체 비영요소를 정렬하는 방법에서 Recursion을 제거한 Quicksort를, 각 열별로 비영요소를 정렬하는 방법에서 Insertion sort를 선택하여 비교하였다. [표 1]의 실험 결과에서 보는 바와 같이, 열별 행 정렬이 평균 2.1배 정도 빠른 것으로 나타났다.

한편, 본 연구에서는 행과 열을 검색하는 방법으로 해싱(Hashing)방법을 사용하였다. 이를 위해 문자열인 행·열의 이름을 숫자로 사상하는 함수를 사용하였고, 충돌(Collision)이 발생하면 다음 위치에 기록하도록 하는 방법(Linear Probing)[12]을 사용하였다.

이를 선형검색방법과 비교한 결과를 [표 1]에 기록하였다. 두 방법 모두 정렬에는 Insertion sort를 사용하였다. 실험 결과에서 알 수 있듯이, 해싱(Hashing)방법을 사용했을 경우가 선형검색방법을 사용했을 경우에 비해 2~73배, 평균 10.4배의 입력 시간이 단축되었다. 큰 문제일수록 차이는 현저하였다.

4. 명시형과 자료구조

(1) 자료구조

명시형으로 기저역행렬을 보관하는 방법은 새로운 비영요소의 추가(Fill-in)에 대한 고려가 전혀 없는 방법이다. 따라서 새로 추가되는 비영요소의 갯수도 많고 그 위치도 예측할 수 없어서, 기저역행렬에 대한 동적인 관리가 필요하다. 여기서는 기저역행렬 전체를 보관하지 않고, 동적인 비영요소 보관 자료구조인 연결 리스트(Linked list)와 Gustavson 자료구조를 사용한다.

기저역행렬의 보관에 연결 리스트를 사용하는 경우, 기저역행렬의 각 행·열의 시작위치, 비영요소의 값과 행·열번호, 행·열에서 다음의 비영요소의 위치를 아래와 같은 배열에 보관한다.

BIN[ ] : 기저역행렬의 비영요소의 값
RCINI[2][ ] : 기저역행렬의 행·열 자료의 시작위치
ROCOL[2][ ] : 비영요소의 행·열번호
RCNEX[2][ ] : 같은 행·열에서 다음 비영요소의 위치

비영요소의 삭제·추가시에는 배열 RCINI, RCNEX를 수정하여 링크(Link)를 재조정한다.

Gustavson 자료구조를 사용하는 경우에는, 기저역행렬의 각 행·열의 시작위치, 비영요소의 값과 행·열번호, 각 행·열의 비영요소의 갯수를 아래와 같은 배열에 보관한다.

BIN[ ] : 기저역행렬의 비영요소의 값
RINI[ ] : 기저역행렬의 행자료의 시작위치
CINI[ ] : 기저역행렬의 열자료의 시작위치
ROWN[ ] : 비영요소의 행번호
COLN[ ] : 비영요소의 열번호
LENROW[ ] : 각 행의 비영요소의 갯수
LENCOL[ ] : 각 열의 비영요소의 갯수

비영요소의 삭제·추가시에는 배열 LENROW, LENCOL이 수정되고, 기억공간이 부족한 경우 RINI, CINI를 수정하여, 보다 넓은 기억공간이 있는 장소로 자료를 옮기기도 한다.

새로운 비영요소의 추가시 한 행의 자료를 모두 BIN에서 현재 사용된 위치의 다음 위치로 옮기는 작업이 있는데, 명시형 기저역행렬 보관 방법에서는 새로운 비영요소의 추가가 많으므로, 이 작업에 걸리는 시간이 늘어나게 되고, BIN의 오른쪽 끝까지 다 사용한 경우의 압축(Compression) 과정도 자주 발생하게 되어 시간이 걸리는 원인이 된다.

(2) 실험결과와 분석

명시형으로 기저역행렬을 보관하는 단체법에서 효율적인 자료구조를 알아보기 위해 연결 리스트와 Gustavson 자료구조에 대해 각각 실험을 수행하였다. 두 프로그램을 NETLIB 문제들[4]에 대해 수행해 본 결과는 [표 2]와 같다.

[표 2] 명시형 기저역행렬을 보관하는 단체법에서 자료구조에 따른 수행 시간 비교

실험기종 : SUN SPARC 10

문제명	문제크기	연결 리스트			Gustavson		
		①	②	③	①	②	③
ADLITTLE	56× 97	81	1.4	17.2	99	2.0	20.2
BLEND	74× 83	81	2.5	30.8	83	3.4	41.0
RECIPE	91× 180	37	0.9	24.3	37	0.9	24.3
SHARE1B	117× 225	273	25.3	92.6	219	24.0	109.6
SCAGR7	129× 140	91	2.5	27.5	118	4.2	35.6
GROW7	140× 301	326	35.8	100.6	433	102.4	236.5
LOTFI	153× 308	256	11.6	45.3	238	12.8	53.8
BEACONFD	173× 262	140	8.6	61.4	141	13.3	94.3
ISRAEL	174× 142	270	19.4	71.9	264	23.1	87.5
VTP.BASE	198× 203	116	6.0	51.7	200	14.4	72.0
SC205	205× 203	121	15.0	124.0	121	22.2	183.5
CAPRI	271× 353	246	18.2	74.0	236	26.7	113.1
SCTAP1	300× 480	287	13.3	46.3	294	15.6	53.1
BANDM	305× 472	537	249.5	464.6	546	228.6	418.7
STANDATA	360×1075	104	8.2	8.8	104	11.0	105.8
SCORPION	388× 358	153	13.2	86.3	165	21.5	130.3
ETAMACRO	400× 688	649	113.8	175.3	691	139.8	202.3
SHIP04S	402×1458	240	19.9	82.9	240	30.4	126.7
AGG2	516× 302	147	20.7	140.8	148	27.5	185.8
AGG3	516× 302	157	22.5	143.3	175	32.9	188.0
SHELL	536×1775	329	54.3	165.0	329	75.0	228.0
GFRD-PNC	616×1092	550	121.4	220.7	547	154.6	282.6
SHIP08S	778×2387	520	86.5	166.3	569	152.8	268.5
GANGES	1309×1681	676	292.5	432.7	791	560.1	708.1

① : Iteration수(회) / ② : 총수행시간(초) / ③ : Iteration당 수행시간(ms)

[표 2]와 같이, 연결 리스트가 Gustavson 자료구조의 경우에 연결 리스트보다 약 50%정도 수행시간이 길어지는 것으로 나타났다.

두 프로그램에서 기저역행렬에 관련된 서브루틴들 각각의 소요 시간을 분석해 본 결과, 주로 기저 수정 과정(Updating) 및 재역산(Reinversion)과정에서 Gustavson 자료구조의 수행 시간이 길어지는 것으로 나타났는데, 이는 이들 과정에서 비영요소의 추가·삭제가 많이 일어나며, Gustavson 자료구조의 경우 비영요소의 추가·삭제 처리에 연결 리스트에 비해 많은 연산을 필요로 하기 때문인 것으로 분석된다.

따라서 실험결과, 명시형에서는 연결 리스트가 Gustavson 자료구조보다 보다 효율적인 자료구조임이 판명되었다.

5. 상하분해와 자료구조

기저행렬을 상삼각행렬 U와 하삼각행렬 L로 분해할 때 이기서는 U, L^{-1}로 보관한다. 상삼각행렬은 명시적으로 보관하는데 대형문제의 경우에 행렬이 희소하기 때문에 비영요소만을 보관하는 방법이 모든 요소를 보관하는 방법보다 기억공간을 절약할 수 있어 비영요소만을 보관하는 방법을 사용한다. 하삼각행렬의 역행렬은 기본 행 연산(Elementary row operation)을 나타내는 eta-file(η)들을 순차적으로 보

관한다.

한편, 기저행렬의 한 열이 바뀌는 경우에 U를 수정해야 하는데 필요에 따라 U의 행과 열을 바꾸어야 하고 비영요소의 삽입이나 삭제가 편리하도록 해야 한다. 따라서 여기서는 U를 다루기에 편리한 연결 리스트 구조와 Gustavson 구조로 보관하는 방법을 설계하고 비교한다.

(1) 연결 리스트 구조

· 상삼각행렬

연결 리스트는 비영요소들을 링크(Link)로 연결해 놓은 자료구조인데 링크는 행과 열 두 가지가 있다. 연결 리스트를 구현하기 위해서는 우선 비영요소를 보관하는 배열과 그 행 지수와 열 지수를 보관하는 배열이 있고, 다음의 비영요소가 있는 주소를 가지는 배열이 있어야 한다. 또 행이나 열을 따라 가면서 비영요소를 찾기 위해서 각 행과 열의 시작을 가리키는 배열이 필요하다. U를 수정할 때 행과 열을 바꿔주는 작업을 하게 되는데 이를 편리하게 하기 위해서 행 치환(Permutation)과 열 치환을 보관하는 배열이 있다.

프로그램에 구현한 배열은 다음과 같다.

BIN[ ] : U의 비영요소 값을 보관
ROCOL[2][ ] : BIN[ ]에 보관된 비영요소의 행과 열의 지수를 보관
RCINI[2][ ] : 행과 열의 BIN[ ]에서의 시작 주소를 보관
RCNEX[2][ ] : 현재의 비영요소에서 이어진 다음 비영요소의 주소를 보관
P[ ] : 행 치환의 결과를 보관
Q[ ] : 열 치환의 결과를 보관

· 하삼각행렬

하삼각행렬은 L^{-1} 형태로 보관하는데 이는 기본 행 연산을 나타내는 eta-file(η)을 단위행렬에 차례로 곱한 것이므로 L^{-1}의 각 요소 값을 명시적으로 보관하지 않고 eta-file을 순차적으로 보관한다. 하였다. 한 번의 기본 행 연산을 기억하기 위해서는 두 행의 지수와 한 행에 곱해지는 승수(Multiplier)가 필요하다. 즉, 하나의 eta-file은 (행지수1, 행지수2, 승수)로 구성된다. 프로그램의 구현시 기억공간을 줄이기 위해서 승수는 BIN[ ]에, 행지수1과 행지수2는 ROCOL[0][ ], ROCOL[1][ ]에 저장하는데 U와 겹치지 않게 하기 위해서 뒤에서부터 기록한다.

프로그램에서는 다음과 같이 구현하였다.

BIN[ ] : 승수를 보관
ROCOL[0][ ] : 행지수1을 보관
ROCOL[1][ ] : 행지수2를 보관

(2) Gustavson 구조

· 상삼각행렬

Gustavson 구조는 U의 비영요소를 행단위로 보관하는데 비영요소를 보관하는 배열과 이 비영요소의 열 지수를 보관하는 배열이 있고 각 행의 시작 주소와 비영요소 갯수를 보관하는 배열이 있다. 또한, 각 열에서의 비영요소의 행 지수와 시작 주소, 비영요소 갯수를 보관하는 배열이 있다.

Gustavson 구조는 행단위로 그 행의 비영요소를 연속적으로 저장하기 때문에 U의 행을 따라 가면서 행하는 연산에서는 우수하나 U의 열을 따라 가면서 행하는 연산에서는 연결 리스트에 비해서 불리하다.

한편, 한 행에 새로운 비영요소가 생길 경우에 그 행에 여유 공간이 없을 경우에는 맨 뒤로 그 행 전체를 옮기는 연산이 필요하므로, 해법의 수행 도중에 자료 내부에 사용하지 않는 공간이 많이 발생한다.

따라서 더 이상 옮길 공간이 없을 경우에는 비어있는 공간을 제거하는 압축(Compression) 과정이 필요하다. Gustavson 구조에서도 연결 리스트와 같이 행과 열의 치환(Permutation)을 기록하는 배열이 있다.

프로그램에서는 다음과 같이 구현된다.

BIN[ ] : 비영요소의 값을 보관.
COLN[ ] : BIN[ ]에 보관된 비영요소의 열 지수
RINI[ ] : U의 각 행의 BIN[ ]에서의 시작 주소
LENROW[ ] : 각 행의 비영요소 갯수
ROWN[ ] : 비영요소의 행 지수
CINI[ ] : ROWN[ ]에서 각 열의 시작 주소
LENCOL[ ] : 각 열의 비영요소의 갯수
P[ ] : 행 치환의 결과를 보관
Q[ ] : 열 치환의 결과를 보관

· 하삼각행렬

연결 리스트의 경우와 같이 구현하였으며 프로그램에서는 다음과 같이 구현하였다.

BIN[ ] : 승수를 보관
ROWN[ ] : 행지수1을 보관
COLN[ ] : 행지수2를 보관

(3) 실험결과 및 분석

상·하삼각행렬을 수정하는 방법인 Bartels-Golub 방법과 Reid 방법에 대해 효율적인 자료구조를 실험적으로 고찰해 보고자 [표 3]과 같이 NETLIB에 있는 문제에 대해 실험을 수행하였다.

[표 3] 기저역행렬 개선방법에 대한 자료구조의 효율성 비교

실험기종:SUN SPARC 10

문제명	문제크기	B-G/연결 리스트			B-G/Gustavson			Reid/연결 리스트			Reid/Gustavson		
		①	②	③	①	②	③	①	②	③	①	②	③
ADLITTLE	56× 97	99	1.5	15.2	99	1.3	13.1	84	1.0	11.9	96	1.2	12.5
BLEND	74× 83	59	1.0	16.9	59	1.2	20.3	60	0.8	13.3	59	1.0	16.9
RECIPE	91× 180	27	0.6	22.2	28	0.6	21.4	27	0.4	14.8	28	0.4	14.3
SHARE1B	117× 225	207	7.5	36.2	207	7.0	33.8	232	7.4	31.9	207	5.9	28.5
SCAGR7	129× 140	106	1.7	16.0	106	1.8	17.0	106	1.4	13.2	106	1.4	13.2
GROW7	140× 301	395	47.8	121.0	479	40.0	83.5	325	18.7	57.5	459	27.1	59.0
LOTFI	153× 308	269	10.9	40.5	255	9.9	38.8	247	5.9	23.9	259	6.1	23.6
BEACONFD	173× 262	35	3.0	85.7	35	2.9	82.9	35	2.9	82.9	35	2.7	77.1
ISRAEL	174× 142	264	14.2	53.8	264	12.7	48.1	274	10.7	39.1	271	9.8	36.2
VTPBASE	198× 203	265	6.2	23.4	241	5.8	24.1	230	4.6	20.0	211	4.5	21.3
SC205	205× 203	121	4.0	33.1	121	3.9	32.2	121	2.9	24.0	121	2.9	24.0
CAPRI	271× 353	235	10.9	46.4	244	10.7	43.9	251	9.1	36.3	251	8.8	35.1
SCTAP1	300× 480	258	8.0	31.0	255	8.0	31.4	266	8.1	30.4	273	8.3	30.4
BANDM	305× 472	592	58.5	98.8	614	54.0	87.9	592	54.1	91.4	614	44.3	72.1
STANDATA	359×1075	79	7.8	98.7	84	8.2	97.6	79	7.4	93.7	84	7.8	92.9
SCORPION	388× 358	158	8.5	53.8	157	8.4	53.5	157	7.7	49.0	157	7.7	49.0
ETAMACRO	400× 688	705	74.4	105.5	792	81.5	102.9	722	40.3	55.8	745	43.0	57.7
SHIP04S	402×1458	247	21.2	85.8	255	21.3	83.5	247	20.2	81.8	255	20.1	78.8
AGG2	516× 302	189	13.4	70.9	202	13.5	66.8	195	13.6	69.7	190	12.9	67.9
AGG3	516× 302	266	19.4	72.9	237	17.5	73.8	250	18.0	72.0	251	19.0	75.7
SHELL	536×1775	362	101.4	280.1	329	77.7	236.2	365	32.6	89.3	329	29.4	89.4
GFRD-PNC	616×1092	547	47.9	87.6	547	48.7	89.0	527	33.2	63.0	551	36.1	65.5
SHIP08S	778×2387	531	75.0	141.2	533	74.4	139.6	488	67.0	137.3	533	69.5	130.4
GANGES	1309×1681	636	126.5	198.9	644	122.6	190.4	703	135.3	192.5	715	126.2	176.5

① : Iteration수(회) / ② : 총수행시간(초) / ③ : Iteration당 수행시간(ms)

[표 3]과 같이, 상하분해 방법에서는 연결 리스트와 Gustavson 자료구조 사이에 거의 속도의 차이를 발견할 수 없었다. 이것은 명시형 기저역행렬 방법에서 Gustavson자료구조가 50%정도 수행 시간이 더 걸리는 결과를 보였던 결과와 대조된다.

각 서브루틴별로 수행 시간을 비교해 본 결과에서도 두 자료구조의 수행 시간에는 거의 차이를 발견할 수 없었다. 다만, 단체승수 π의 계산이나, 진입열 수정 등의 연산 과정에서 Gustavson 자료구조가 연산 시간이 근소한 차이로 적은 반면, 비영요소의 추가·삭제가 비교적 많은 상하분해(Factorization)과정에서는 연결 리스트의 수행 시간이 약간 적은 것으로 나타났다.

실험결과, 상하분해 방법에서는 연결리스트와 Gustavson 자료구조가 모두 경쟁력이 있는 것으로 판명되었다.

6. 결론

본 연구에서는 기저역행렬의 계산방법에 따른 효율적인 자료구조를 실험적으로 검토하고, 입력자료방식과 효율화 방법을 제안하여 구현하였다.

기저역행렬의 계산방법에 따른 자료구조는 명시형에서는 연결 리스트 방식이 유리하였으며, 상하분해형에서는 연결 리스트와 Gustavson 방식이 비슷한 효율을 나타내었다. 또한, Gustavson 자료구조는 비영요소의 추가·삭제가 많은 경우 효율적이지 못한 것으로 나타났다. 그리고 입력자료의 방식과 효율화 방안에서는 각 열별로 행 정렬을 실시하고 해싱(Hashing)함수를 도입하여 효율화를 기하였다.

추후연구로는 상하분해법에서 Gustavson 방식의 효율화를 위해 압축 전략에 대한 연구와 비영요소의 추가·삭제시의 효율적인 처리방법에 대한 연구가 필요하다.

참고문헌

[1] 박순달, 「선형계획법(3정판)」, 민영사, 1992

[2] 김우제, 강완모, 김민정, 박순달, "일반한계 선형계획법에서 비영요소만 보관하는 자료구조와 평가전략의 효율성에 관한 연구", 전산활용연구, 제6권 제1호, pp 55-66, 1993

[3] Bartels. R. H., G. H. Golub., "The Simplex method of linear programming using LU decomposition." Communication of ACM, 12, pp 266-268, 1969

[4] Bixby, R. E., "Implementing the Simplex Method : The Initial Basis", *ORSA J. on Computing*, Vol. 4, No. 2, pp 267-284, 1992

[5] Duff. I. S., A. M. Erisman, J. K. Reid, *Direct Methods for Sparse Matrices,* Clarendon Press, Oxford 1986

[6] Forrest, J. J. H., J.A. Tomlin, "Implementing the Simplex Method for the Optimization Subroutine Library", *IBM Systems Journal,* Vol. 31, No. 1, pp 11-25, 1992

[7] Forrest. J. J. H., J. A. Tomlin, "Updating triangular factors of the basis to maintain sparsity in the product-form simplex method." *Math. Prog.* 2, pp 263-278, 1972

[8] Murtagh, B. A., *Advanced Linear Programming : Computation and Practice*, McGraw-Hill, 1981

[9] Murty, K. G., *Linear Programming*, Wiley, 1983

[10] Reid. J. K., "Fortran Subroutines for Handling Sparse Linear Programming Bases", *Computer Science and Systems Devision*, A.E.R.E., Harwell R.8269 pp 1-23, 1976

[11] Reid J. K., "A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases", *Computer Science and Systems Devision*, A.E.R.E., Harwell, CSS20 pp 1-23, 1975

[12] Sedgewick, R., *Algorithms*, 2nd ed., Addison Wesley, New york, 1988