네트워크분석을 위한 계산지모형

이호창*

A Spreadsheet Modeling for Network Analysis

Hochang Lee*

ABSTRACT

In this paper we examine potentials of a spreadsheet program, one of the most widly available software system, as a mathematical optimization modeling tool. For an illustrative example, a shortest path problem is modeled on Lotus-123 for practical use and an implementational framework and a general guide to the spreadsheet modeling of network analysis is provided.

1. 서 론

종이와 연필로 수행할 수 있는 인간의 계산작업을 그대로 모방한 전자계산지(electronic spreadsheet) 프로그램이 컴퓨터 소프트웨어 시장에 처음 등장했을때 그에 대한 사용자들의 반응은 매우 경이적이었다. 그러나 초기의 사용 용도는 매우 제한적이어서 계산지의 여기저기에 흩어져 있는 많은 자료들의 수합, 동시계산 또는 변수의 변화에 따른 표의 재계산 등 방대한 계산량들이 눈앞의 화면에서 순식간에 이루어지는 일차적인 자료의 가공을 보면서 경탄을 자아내는데 그쳤다. 사실 Lotus-123, EXCEL, QUATTRO 등과 같은 계산지 응용프로그램에 관한 일반사용자들의 인식 및 응용범위는 표의 계산이나 그래프의 작성등 극히 일부분에 국한된 단순 반복계산 작업에 불과했다. 그러나 어떠한 계산지 응용프로그램을 보더라도 i) 표의 고속계산을 위한 전자계산지, ii) 데이터의 보관 및 검색을 위한 데이터베이스, iii) 데이터를 시각적으로 표현해 주는 그래프기능, iv) 매크로와 같은 범용의 프로그래밍언어 기능들을 개별 모듈로서 자체에 포함하고 있는데 이는 각 모듈을 적절하고 효과적으로 접속(interface)함으로써 과거의 일차원적인 응용개념을 바꾸어 놓을 수 있음을 시사한다. 시간이 흐

* 경희대학교 사회과학대학 경영학과

르면서 이 전자계산지에 대한 활용욕구는 점차 증대되어 각종 의사결정문제에 있어서 그 대안들을 비교하는 "what if" 분석이 전자계산지의 재계산 기능을 이용하여 널리 활용되기 시작하였다. 근래에 이보다 훨씬 어려운 "how to"방식의 분석이 의사결정 지원시스템이나 전문가시스템의 구축을 중심으로 계산지에 이식되고 있는 추세이며 목적지향적인 "what's best," 즉 최적화 문제로의 활용이 동시에 진행되고 있음은 주지의 사실이다. 그러나 이러한 최적화문제는 모형 자체가 특정한 틀을 통하여 표현되기 때문에 OR/MS에 대한 전문지식이 없는 사용자에게는 다소 부담스럽게 느껴진다. 한편 계산지 프로그램 상에서는 사용자에게 익숙한 자유형식으로 모형의 자료를 입력하고 그에 따른 결과를 표현하는 사용자 인터페이스의 설계 및 변형이 용이하다. 이에 최적화문제의 계산지모형은 종이와 연필의 사용에 익숙한 일반사용자에게 가장 자연스러운 형태로 모형을 수립하고 쉽게 이용하게 한다는 의미에서 그 활용도가 크다고 하겠다.

현재 계산지모형을 위한 소프트웨어는 일반사용자로 하여금 그들의 의사결정문제를 빠르고 쓰기 쉽고 저렴한 비용으로 해결할 수 있도록 기술적으로 충분히 개발되어 있으며 계산지모형은 새로운 모형도구로써 의사결정나무분석[6,12,14], 전문가시스템[13], 최적화기법[11,15,16], 모의실험[7,10], 통계분석 및 예측[8,9]에 이르기까지 그 활용분야를 개척하고 있으며 그 가능성을 보여주기에 이르렀다. 소프트웨어의 관점에서 모형도구로써 전자계산지의 응용유형을 구분해 보면 다음과 같다[1].

① 단일소프트웨어(Within the Spreadsheet) : 다른 소프트웨어의 도움 없이 계산지 내부의 프로그래밍언어, 매크로와 각종 유틸리티들을 사용하여 계산절차를 수행한다. 활용성이나 수행속도에 제약이 있지만 외부 시스템과의 연결절차가 불필요하므로 그 구축이 용이하고 비용도 저렴하다.

② 개별소프트웨어(Separative Software) : 독자의 소프트웨어가 모형을 정립하고 그 계산절차들을 수행하지만 사용자에게 친숙한 형태로 자료입력, 보관 및 결과를 보고하기 위하여 계산지와 독립적으로 연결된다.

③ 통합소프트웨어(Integrated Software) : 계산지의 제기능들과 필요한 개별적 소프트웨어의 기능들이 하나의 패키지로 통합된 형태이므로 가장 효율적이나 비용이 많이 든다. IBM사의 TopView와 같은 대부분의 상업용 소프트웨어가 이 범주에 든다.

단일소프트웨어의 범주에서 Kimbrough[4]는 해군함정의 성능평가를 위한 의사결정 지원시스템의 기본골격구조를 Lotus-123에 성공적으로 이식하였고 Jones[3]는 실제사용을 목적으로 의사결정나무를 모형화 하였다. 또한 Jennergren[2]은 심플렉스 알고리즘을 이용한 선형계획모형을 계산지의 내부언어를 사용하여 최적화하였다.

본 연구의 주 목적은 단일소프트웨어의 범주에서 네트워크분석을 위한 계산지모형을 수립하고 그 실용 가능성을 타진하는것이다. 예로써 선형네트워크상 최단경로문제(shortest path problem)의 계산지모형을 수립 분석하고 이를 통하여 수학적 최적화(mathematical optimization)문제의 계산지모형 수립에 관한 개념적 틀을 제시한다. 2장에서는 네트워크분석을 위한 계산지모형의 구조를 개괄적으로 설명하고 3장에서 이를 구성하는 각 모듈들을 열거한다. 4장은 실험적으로 구축된 최단경로문제를 위한 계산지모형의 실행예를 보여준다. 마지막 장에서는 단일소프트웨어의 범

주에서 계산지모형의 실용적 가능성과 그 한계점을 분석한다.

2. 계산지모형의 구조

계산지모형은 계산지상에 독립적으로 위치한 아래 5개의 기능별 부모듈들로 구성되며 각 모듈간의 자료전송 및 변형은 계산지 프로그램의 기본기능인 셀간 자료복사 기능을 이용한다.

① 문제작성 부모듈 (Problem Specification Submodule : PSS) : 최적화모형의 계수자료들을 입력하는 부분으로 개인 사용자의 편의에 따라 자유형식으로 고안된다.

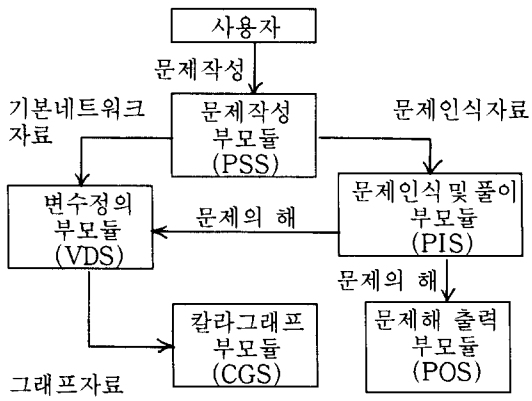
② 문제인식 및 풀이 부모듈 (Problem Identification and Solving Submodule : PIS) : 입력된 자료에 따라 최적화 문제를 인식하여 해를 구하는 부분으로 모형 개발자가 특정의 알고리즘을 택하여 계산지 모형내에 매크로로 프로그래밍한다.

③ 변수정의 부모듈 (Variable Definition Submodule : VDS) : 구해진 해를 기본 네트워크상에 표현하기 위해 자료를 재구성하는 부분이며 사용되는 그레픽모듈에 따라 자료의 재구성 방식이 달라진다.

④ 문제해 출력 부모듈 (Problem Output Submodule : POS) : 구해진 해를 사용자에게 보여주는 부분으로 개인 사용자의 편의에 따라 자유형식으로 고안된다.

⑤ 컬러그래프 부모듈 (Color Graphics Submodule : CGS) : 재구성된 기본자료와 해로 부터 네트워크를 표현하고 그 위에 해를 표시하는 출력부분이다.

사용자가 문제를 자신이 원하는 형식으로 PSS에 기록하면 PIS는 PSS로부터 입력된 데이터를 읽고 이를 자신의 형식에 맞게 재구성하여 문제를 인식하고 이에따라 네트워크의 구조를 파악한다. 동시에 기본 네트워크의 구조는 PSS로부터 VDS로 이송되어 변수별로 분산 구성되어 기록된다. 한편 PIS는 내부 프로그래밍언어로 짜여진 임의의 알고리즘으로 문제를 풀고 그 결과를 각각 문자와 그래프로 출력하기 위하여 POS와 VDS에 동시에 전달한다. VDS는 PIS로부터 이송되어진 네트워크 자료와 문제의 해를 그래프로 표현될 수 있도록 변환하여 CGS로 전달한다. CGS는 변환된 자료를 기초로하여 사용자에게 보여줄 기본 네트워크와 문제의 해를 표시하는 컬러그래프를 작성한다. 각 모듈간의 자료이송은 계산지내 셀들간의 자료이송기능을 이용하여 동시에 이루어짐으로써 시간적 지연이 무시될 정도로 그 이송시간이 짧으나 PIS에서의 알고리즘 수행과 VDS에서의 컬러그래프를 위한 자료변환과정은 내부 프로그래밍언어의 실행을 필요로 하기 때문에 상대적으로 길다. 사용자는 자신에게 친숙한(user friendly) 형식으로 PSS를 고안할 수 있으며 사용자 메뉴형식으로 부모듈간의 화면이동을 자유롭게할 뿐만 아니라 메뉴안의 항목을 고름으로써 부모듈의 실행을 지시한다. 부모듈간의 자료이송 및 상호관계는 아래의 그림과 같다.



[그림 1] 부모듈간의 자료흐름

3. Lotus－123를 이용한 최단경로 문제 풀이

3.1 메뉴방식의 절차

문제풀이의 절차는 아래의 메뉴중 임의의 항목을 선택함으로써 진행된다.

① Specification : 문제자료를 입력함으로써 최단경로문제를 작성한다.
② Execution : 입력된 자료에 의하여 문제를 인식하고 이에 따라 최적해를 구한다.
③ Numerical Solution : 최단경로를 마디번호들로 연결하여 최단길이와 함께 숫자로 표시한다.
④ Graphical Solution : 기본 네트워크와 그위의 최단경로를 색깔로 구분하여 그래프 형태로 표시한다.
⑤ Quit : 프로그램을 종료한다.

3.2 최단경로문제의 작성

계산지상의 셀안에 사용자가 원하는 형식으로 다음과 같은 최단경로문제의 자료들을 입력함으로써 PSS를 작성한다.

① 분석대상이 되는 네트워크상의 마디갯수
② 사용자가 최단경로를 알기 원하는 두 지점
③ (X,Y) 좌표로 표현된 네트워크상 마디의 위치
④ 두 지점간의 거리를 행렬형태로 표시한 거리표

3.3 최단경로문제의 인식 및 풀이

문제의 인식 및 풀이절차는 Lotus－123에 내장된 매크로로 짜여지며 이는 순차적으로 실행되는 아래의 서브루틴들로 구분된다.

① CLEAR : 새로운 문제를 위한 변수정의를 위하여 VDS를 공백상태로 만든다.
② IDEN : 입력된 문제자료로부터 기본네트워크의 구조를 인식하고 CGS로 이송될 기본 그래프자료를 작성한다.
③ DIJ : Dijkstra의 알고리즘에 의하여 최단경로를 찾아낸다.
④ SOS : DIJ의 실행결과로 얻어지는 최단경로의 좌표를 재구성하고 이를 IDEN에 덧붙임으로써 기본 네트워크상에 최단경로를 컬러그래프로 표시한다.

3.4 Dijkstra의 알고리즘

문제풀이를 위하여 비음의 아크길이를 갖는 네트워크상에서 임의의 지점과 다른 모든 지점들간의 최단경로를 찾아내는 Dijkstra의 알고리즘[5]을 이용한다. 이의 개괄적인 구조는 아래와 같다.

입　　력 : 아크길이 $c_{uv} \geq 0$ $\forall u,v \in V$, $\forall (u,v) \in A$ 를 갖는 유향그래프 $D=(V, A)$ 출발마디 $s \in V$

출　　력 : 출발마디 s에서 모든 마디 $v \in V$ 까지의 최단거리 π

알고리즘 : BEGIN
　　set W : = {s}, $\pi[s]$: = 0
　　do $\pi[y]$: = Csy $\forall y \in V - \{s\}$
　　while W = V do
　　BEGIN
　　find $min\{\pi[y] \mid y \notin W\}$, say $\pi[x]$
　　set W : = W ∪ {x}
　　do π[y] : = $min\{\pi[y], \pi[x]$

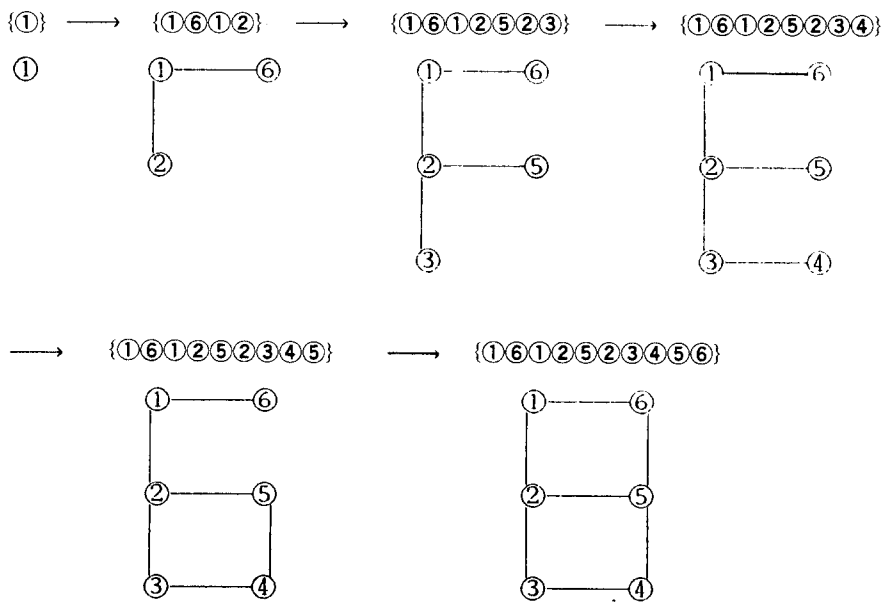
$+ c_{xy}\} \ \forall y \in V - W$

END

END

3.5 네트워크 표현을 위한 그래프자료의 재구성

Lotus-123의 그래프기능은 선, 막대, XY, 파이와 같은 기본적인 유형에 국한되어 있기 때문에 임의의 네트워크와 그위의 최단경로를 구분하여 표현하기 위해서는 그래프자료를 특별히 재구성하고 변형하는 것이 필요하다. 예를 들어 XY 그래프 기능을 이용하여 日자 형태의 네트워크를 최단시간내에 그리기 위해서는 임의의 지점에서 출발하여 손을 떼지 않은 상태에서 이미 그린 선분을 다시 그리지 않고 모든 선분을 그림으로써 日자를 완성해야 한다. 즉 XY 그레프 기능을 이용하여 日자 형태의 네트워크를 그리는 문제는 네트워크안에 포함된 6개의 마디를 연결하는 Eulerian circuit을 찾는 문제와 동일하다. 임의의 유향그래프로 표현된 네트워크상에서 Eulerian circuit이 항상 존재하는 것도 아닐 뿐만 아니라 있다고 하더라도 그것을 찾는 것은 용이한 일이 아니며 이는 분석 대상이 되는 최단경로문제보다 훨씬 어려운 부속문제가 된다. 차선의 방법으로는 마디를 순차적으로 옮겨가며 인접한 마디와 연결하는 greedy식의 휴리스틱방법을 들 수 있으며 이에 따르면 아래에 예로 든 日자 형태의 네트워크에 대해 PSS에 입력된 마디좌표 {①②③④⑤⑥}의 변형 및 재구성 과정은 다음과 같다.



[그림 2] 네트워크의 표현을 위한 마디좌표의 재구성

사용자로부터 입력된 기본 네트워크상의 마디 좌표자료 {①②③④⑤⑥}는 VDS에서 {①⑥①②⑤②③④⑤⑥}로 팽창, 재구성되며 XY 그래프형식으로 화면에 표시된다. PIS로부터 이송된 최단경로는 무순환적(acyclic)이므로 이상과 같은 마디좌표의 변형없이 기본네트워크 위에 다른 색깔로 덧씌어짐으로써 화면상 구별이 가능하다. 마디좌표의 재구성 절차에 대한 자세한 설명에 관해서는 부록에 포함된 매크로를 참조하기 바란다 (II. IDENTIFICATION OF BASE NETWORK, IV. SOLUTION AND DRAWING).

3.6 매크로

계산지모형내 매크로로 짜여진 실행프로그램은 크게 세종류의 부프로그램들로 구분되며 그들의 기능은 다음과 같다.

① 모형소개 매크로 (AUTOEXEC MACRO) : 화일의 부팅과 함께 자동적으로 실행하여 사용자에게 계산지모형의 설명과 그 사용방법을 설명한다.

② 실행메뉴 매크로 (MENU) : 메뉴형태로 실행내용들을 표시함으로써 사용자로 하여금 각 실행절차들로 분기하게 한다.

③ 문제풀이 매크로 (EXECUTION) : 문제를 인식하여 해를 구하고 그래프로 결과를 출력하며 다음의 서브루틴들로 구성된다.

I. CLEAR
II. IDENTIFICATION OF BASE NETWORK (IDEN)
  A. ROW
  B. COLUMN
III. DIJKSTRA'S ALGORITHM (DIJ)
  A. MAIN
  B. INITIALIZATION
  C. LOOP
  D. LOOP1
  E. ITERATION
  F. FIND
  G. UPDATE
IV. SOLUTION AND DRAWING (SOS)
  A. SOLUTION
  B. TRIVIAL SOLUTION
  C. FIND A PATH
  D. REPORT
  E. SWITCH AND ADD
  F. REVERSE
  G. COMPARE

4. 실행예

20개의 도시들을 포함한 교통망을 가상하여 계산지모형의 실행예를 보인다. 최단거리 계산지모형 화일을 불러들임과 동시에 자동실행 매크로가 자동적으로 실행되면서 [그림 3]과 같은 초기화면이 나타난다. Alt-A 키에 의하여 계산지모형의 사용자 메뉴가 화면상단에 [그림 4]의 형태로 나타나며 문제작성을 위하여 1. Specification을 고름으로써 [그림 5]에서 보는바와 같은 최단경로문제의 자료입력모듈로 화면이 이동된다. 문제자료가 입력되면 사용자메뉴의 실행항목을 문제의 인식 및 풀이를 실행하여 자료입력모듈에서 명시한 마디간의 최단경로를 찾는다. 실행결과는 [그림 6]과[그림 7]에서와 같이 숫자와 컬러그래프의 형태로 사용자의 요구에 따라 각각 출력된다. 실행시간은 분석대상이 되는 네트워크의 형태와 실행컴퓨터의 기종에 따라 다소 차이를 보이지만 IBM-PC 386에서 20마디를 포함한 본 예의 실행시간은 약 10초 정도로 네트워크 표현을 위한 그래프자료의 팽창 및 재구성 절차를 보완하면 실용적인 측면에서도 그 가능성을 찾을 수 있다.

*** WELCOME TO THE SHORTEST PATH PROBLEM SOLVER ON LOTUS-123 ***

- How to use it
  1. Specify problem
     a. # of cities.
     b. Names of two cities between which
        you want to know the shortest path.
     c. Locations of cities in (X,Y) coordinates.
     d. Distances between cities.
  2. Solve problem
  3. Get solution
  4. Get color graphics

- How to start it
  Just press Alt-A and choose item in the menu.

[그림 3] 초기화면

1. Specification
   Specify the problem

2. Execution
   Solve the problem

3. Numerical Solution
   Get the solution

4. Graphical Solution
   Get the color graphical solution

5. Quit
   Quit the spreadsheet model

[그림 4] 사용자 메뉴

***** SPECIFICATION OF PROBLEM *****

[After specification, press Alt－A to choose next item in the menu]

Ⅰ. Number of Cities : 20

Ⅱ. From City : 1

To City 20

Ⅲ. City Map

City #	1	2	3	·	·	·	·	·	19	20
X	0.5	2	1	·	·	·	·	·	5	6
Y	9.5	8	7	·	·	·	·	·	7	9

Ⅳ. Distance Matrix

		TO CITY									
		1	2	3	·	·	·	·	·	19	20
F	1	***	15	13	·	·	·	·	·	***	***
R	2	***	***	2	·	·	·	·	·	***	***
O	3	***	2	***	·	·	·	·	·	·	***
M	·	·	·	·	·	·	·	·	·	·	·
	·	·	·	·	·	·	·	·	·	·	·
C	·	·	·	·	·	·	·	·	·	·	·
I	·	·	·	·	·	·	·	·	·	·	·
T	·	·	·	·	·	·	·	·	·	·	·
Y	19	***	***	***	·	·	·	·	·	***	***
	20	***	***	***	·	·	·	·	·	***	***

[그림 5] 문제의 작성

***** SOLUTION *****

From City 1

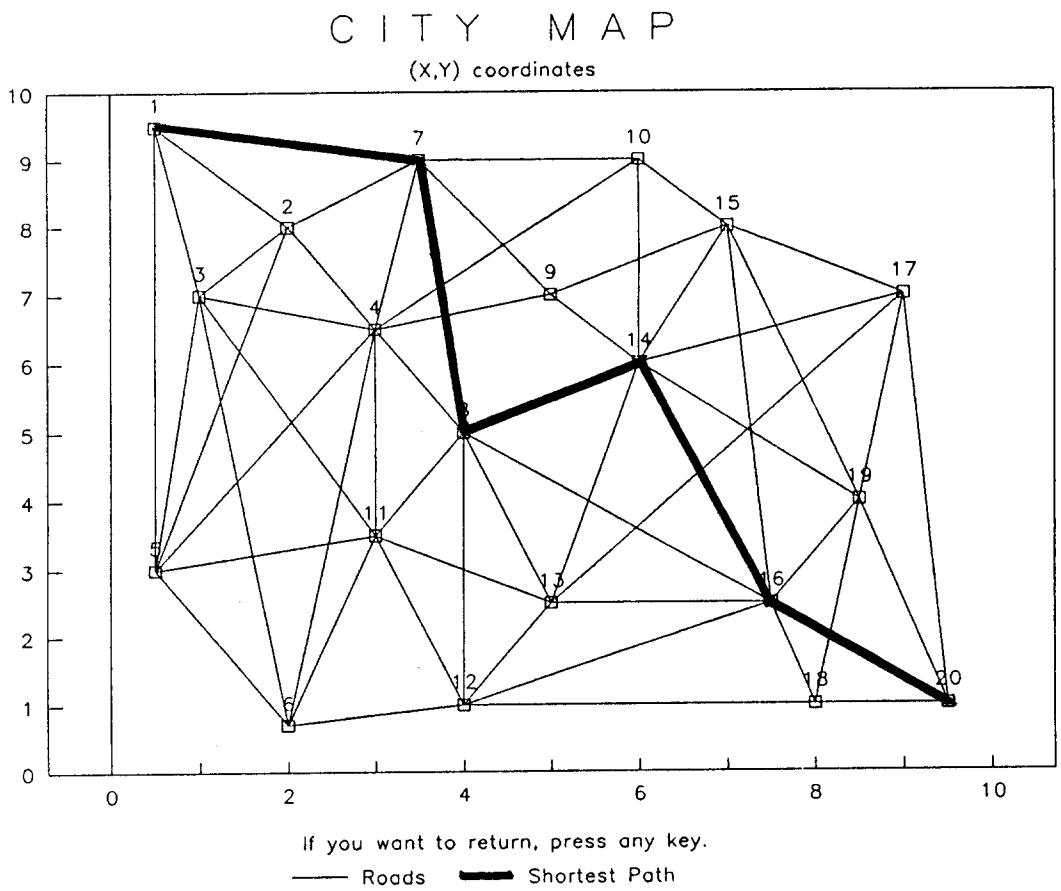
To City 20

Total Distance 19

Shortest Path 1 7 8 14 16 20

[Choose next item in the menu.]

[그림 6] 최단경로



[그림 7] 그래프로 표현된 최단경로

5. 결 론

최단경로문제를 예로 들어 네트워크분석을 위한 시험적(prototype) 계산지모형을 수립하였다. 또한 이를 통해 수학적 최적화문제의 계산지모형 구축에 관한 일반적인 지침을 제시하였는바 본장에서는 이러한 계산지모형의 실용적 가능성 및 그 한계점들을 고찰한다. 첫째, 계산지모형은 모형자료의 자유로운 입출력 형태(format free input/output)와 종이와 연필을 사용하는 듯한 자연스러운 사용자 인터페이스 고안을 통하여 최적화 문제에 익숙치 않은 일반사용자로 하여금 쉽게 모형을 이해하고 사용할 수 있게 해주는 친숙한 창구(user-friendly channel)를 제공해 준다. 뿐만아니라 단일소프트웨어로 응용될 경우에는 매우 저렴한 비용으로 실용적 모형구축이 가능하다. 둘째, 계산지 프로그램내의 각종 내부함수, 행렬계산기능, 그래프표현기능, 데이터베이스(DB)등과 같은 소프트웨어 내부기능들을 적절히 이용한다면 더욱 효과적이고 효율적인 계산지모

형의 수립이 가능하다. 즉 단일 계산지 프로그램 내에서 최적화모형과 DB와의 통합을 생각할수 있는데 예를 들어 전국의 도로망과 도시 및 그 특성들을 포함한 DB와 최단경로문제가 하나의 계산지모형으로 통합된다면 사용자의 여러가지 요구조건 - 한 예로 인구 40만 미만의 도시를 대상으로 충청북도를 거치지 않는 국도를 통해서 서울과 부산간의 최단경로 - 에 맞는 문제자료들이 DB로부터 선별되어 PSS로 이송됨으로써 다양한 제약조건들을 갖는 문제작성이 사용자에 의한 사전 자료처리단계를 거치지 않고 신속하게 이루어지게 된다. 또한 실행시간의 단축이나 프로그래밍의 용이성을 목적으로 매크로 대신 각종 행렬계산기능이나 내부함수를 이용함으로써 모형화과정을 단순화 할 수도 있다. 셋째, 정보의 교환이 기본적으로 계산지상의 셀주소를 인용함으로써 이루어지므로 기존의 프로그래밍기법에 익숙치 않은 사용자의 입장에서 볼때 각 부 프로그램들간의 접속은 물론이고 서로 독립적인 모형들의 통합이 용이하여 계산지모형의 확장성이 거의 무한하다. 넷째, 매크로에는 범용의 프로그래밍언어가 기본적으로 갖추어야할 구조적 개념(structural concept)이 결여되어 있기 때문에 모형전체의 구조파악이 어렵고 때에 따라서는 프로그래밍하기 어려운 경우도 발생한다. 다섯째, 교육목적이나 소규모의 최적화문제를 위해서는 단일 프로그램내의 계산지모형이 필요 충분할 것으로 보이나 빠른 계산 속도를 요구하는 상업적 이용을 위해서는 LINDO, MPSX와 같은 전문적 소프트웨어와의 접속이 불가피할것으로 사료된다. 실제로 VINO, LP83/MIP83나 TopView와 같이 자료 입출력등의 사용자접속(user interface)은 계산지에서 처리하고 문제의 풀이과정과 같이 계산부하가 큰 부분을 전문소프트웨어가 나누어 담당하게하는 통합소프트웨어의 개발이 활발히 진행되고 있다. 여섯째, 계산지프로그램에 내장되어 있는 그래프기능은 일반 네트워크를 표현하는데 매우 제한적이므로 일부 기능들의 수정, 보완이나 독립적인 그레픽전용 소프트웨어와의 접속이 필요하다. 사용예에서는 네트워크의 표현을 위한 그래프자료의 변형 및 재구성에 전체 실행시간의 60% 정도가 소요되며 재구성된 자료의 기록을 위하여 전체 계산지모형내 상당부분의 셀들을 점유하고 있는 실정이다.

참 고 문 헌

1. Bodily, E. Samuel, "Spreadsheet Modeling as a Stepping Stone," *Interfaces*, Vol. 16 (1986), pp. 34-52.
2. Jennergran, L. Peter, "Pivoting in Symphony : A Teaching Aid for Linear Programming," Working Papaer, Odense University, 1984.
3. Jones, J. Morgan, "Decision Analysis using spreadsheets," *European Journal of Operations Research*, Vol. 26(1986).
4. Kimbrough, O. Steven, "A Decision Support System for Evaluation of Advance Marine Vehicles," Working Papaer, University of Pennsylvania, 1988.
5. Papadimitriou, H. Christos and Kenneth Steiglitz, *Combinatorial Optimization - Algorithms and Complexity*, Prentice 1982.

응용소프트웨어

6. Arborist, Texas Instrument.
7. ENCORE, Ferox Microsystems.
8. 1-2-3 Forecast, Bruce L. Gates, Salem,

Oregon.

9. Forecast-G, Gary Roodman, SUNY Binghamton.
10. IFPS, Execucom Systems Corporation.
11. LP83/MIP83, Sunset Software.
12. Rick-Calc, Rick-Calc Associates.
13. ROME, Kosey and Wise.
14. Supertree, Strategic Decision Group.
15. TopView, IBM.
16. VINO, LINDO, Linus Schrage.

부 록(매크로)

AUTOEXEC MACRO

{goto} al~ {beep}

MENU

{menubranch topmenu}
Specification Specify the problem {goto}speci~
Execution Solve the problem {goto}wait~ {\e} {goto}fin~ {\a}
Numerical-solution Get a solution {goto}slt~ {\a}
Graphical-solution Get a color graphics {graph} {\a}
Quit Quit the program /q

EXECUTION

```
{clear}
{iden}
{dij}
{sos}
{esc}{beep}~~
{return}
```

```
I. CLEAR
{blank solution}
{blank graph}
{blank td}
{blank sp}
{blank status}
{blank rho}
{blank path}
{blank qqq}
{return}
```

```
Ⅱ. IDENTIFICATION OF BASE NETWORK (IDEN)
{let k,0}
{let qqq,0}
{for i,1,m,1,row}
{return}

A. ROW
{for j,1,n,1,column}
{return}

B. COLUMN
{if @index(distmat,j,i)>1000000}{return}
{let qqq,qqq+1}
{let k,k+1}
{put graph,k,0,i} {put graph,k,1,@index(map,i,1)}
{put graph,k,2,@index(map,i,2)}
{let k,k+1}
{put graph,k,0,j}
{put graph,k,1,@index(map,j,1)}
{put graph,k,2,@index(map,j,2)}
{let k,k+1}
{return}
```

```
Ⅲ. DIJKSTRA'S ALGORITHM (DIJ)

A. MAIN
{initialization}
{for i,1,n-1,1,iteration}
{return}

B. INITIALIZATION
{for i,1,n,1,loop}
{put status,fc,1,1}
{for i,1,n,1,loop1}
{put rho,fc,1,0}
{return}

C. LOOP
{put status,i,1,0}
{put path,i,1,fc}
{return}

D. LOOP1
{put rho,i,1,@index(distmat,i,fc)}
{return}

E. ITERATION
{let k,1000}
{let mm,100000000}
{for j,1,n,1,find}
{if k=1000}{forbreak}
{put status,k,1,1}
{for j,1,n,1,update}
{return}

F. FIND
{if @index(status,j,1)=1}{return}
{if @index(rho,j,1)>=mm}{return}
{let mm,@index(rho,j,1)}
{let k,j}
{return}

G. UPDATE
{let nn,@index(rho,k,1)+@index(distmat,j,k)}
{if @index(rho,j,1)<=nn}{return}
{put rho,j,1,nn}
{put path,j,1,k}
{return}
```

```
Ⅳ. SOLUTION AND DRAWING (SOS)
A. SOLUTION
{let td,@index(rho,tc,1)}
{let ii,0}
{if @index(path,tc,1)=tc}{branch trivial}
{if @index(status,tc,1)=0}{branch trivial}
{put solution,1,0,tc}
{let jj,tc}
{for ii,2,n,1,ssol}
{branch report1}
{return}
B. TRIVIAL SOLUTION
{put sp,0,0,"No path exists"}
{put solution,1,0,"No path exists"}
{return}
C. FIND A PATH
{let jj,@index(path,jj,1)}
{put solution,ii,0,jj}
{if jj<>fc}{return}
{forbreak}
{return}
D. REPORT
{for kk,1,ii,1,switch}
{return}
E. SWITCH AND ADD
{let i,kk-1}
{put sp,i,0,@index(solution,ii-kk+1,0)}
{if kk=ii}{return}
{let l,@index(solution,kk,0)}
{let ll,@index(solution,kk+1,0)}
{reverse}
{for oo,1,qqq,1,compare}
{return}
F. REVERSE
{let o,l}
{let l,ll}
{let ll,o}
{return}
G. COMPARE
{let p,3*(oo-1)+1}
{if @index(graph,p,0)<>l}{return}
{let p,p+1}
{if @index(graph,p,0)<>ll}{return}
{put graph,p,3,@index(graph,p,2)}
{let p,p-1}
{put graph,p,3,@index(graph,p,2)}
{forbreak}{return}
```