네트워크 최적화 문제의 해결을 위한 LPSolve와 엑셀의 연동 방안†

(A connection method of LPSolve and Excel for network optimization problem)

김 후 곤*
(Hu-Gon Kim)

요 약 네트워크 최적화 문제는 의사결정 문제 중에서 노드와 아크로 표현되는 수 많은 문제를 포함하고 있어서, 그 응용범위가 매우 다양할 뿐만 아니라 매우 실질적인 문제를 해결하는 좋은 방법론이다. 직접적으로 관련이 없는 많은 최적화 문제들도 네트워크로 적절히 표현할 수 있는 경우가 많으며, 이를 통해 보다 심도 있는 문제에 대한 이해와 해의 도출이 가능하게 된다. 이처럼 광범위한 응용분야를 가지는 네트워크 최적화 문제는 경영과학 및 산업공학에서 기본이 중요 학문이며, 이를 체계적으로 이해하고 실제 문제를 해결하려면 최적화 이론, 계산이론, 프로그래밍 등의 종합적인 지식을 필요로 한다. 본 연구에서는 네트워크 최적화 문제를 실질적으로 해결하는 필요한 지식 전달에 중점을 두고, 선형계획법 및 정수계획법을 위한 소프트웨어인 LPSolve를 소개하고 이 LPSolve와 엑셀을 연동하는 방법을 알아본다. 또한 네트워크 자체를 엑셀에서 그리는 방법을 알아보고, 이를 통해 네트워크 최적화 문제를 보다 실질적인 다룰 수 있도록 한다.

핵심주제어 : LPsolve, 엑셀 VBA 프로그래밍, 네트워크 구현

Abstract We present a link that allows Excel to call the functions in the lp_solve system. lp_solve is free software licensed under the GPL that solves linear and mixed integer linear programs of moderate size. Our link manages the interface between Excel and lp_solve. Excel has a built-in add-in named Solver that is capable of solving mixed integer programs, but only with fewer than 200 variables. This link allows Excel users to handle substantially larger problems at no extra cost. Futhermore, we introduce that a network drawing method in Excel using arc adjacency lists of a network.

Key Words : LPSolve, Excel VBA Programming, network implementation

1. 서 론

네트워크 최적화 문제(Network Optimization Problem)는 현실에서 발생하는 매우 다양한 의사결정 문제에 사용되고 있으며, 그 응용범위가 수송, 통신, 제조, 재무, 마케팅 등의 경영 의사결정 과정 뿐만 아니라 물리, 화학, 생물, 환경 등 수많은 분야에서 이용되고 있다[1]. 그리고 직접적으로 네트워크와 관련이 없는 최적화 문제나 의사결정 문제들도 네트워크로

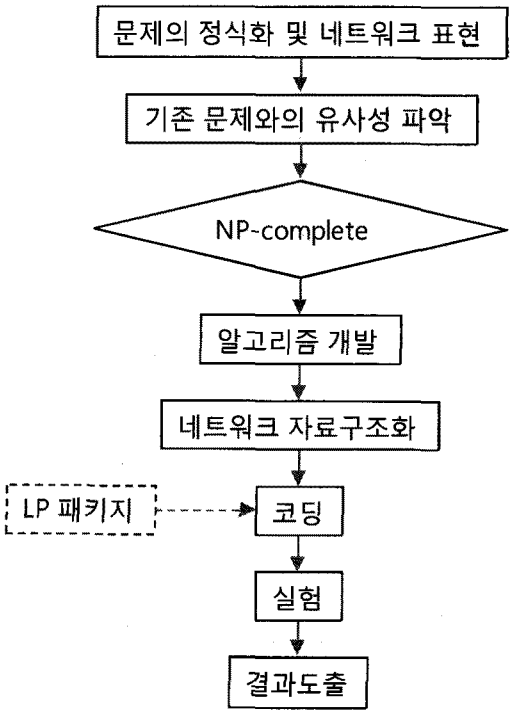
† 이 논문은 2010학년도 경성대학교 학술연구비지원에 의하여 연구되었음.
* 경성대학교 경영정보학과

적절히 표현할 수 있는 경우가 많으며, 이를 통해 보다 심도 있는 문제에 대한 이해와 해의 도출이 가능하게 된다[4]. 이처럼 매우 광범위한 응용 분야를 가지는 네트워크 최적화 문제는 경영과학 및 산업공학에서 가장 기본이 되는 학문 분야로, 이를 체계적으로 이해하고 실제 문제를 해결하려면 최적화 이론, 네트워크 이론, 프로그래밍 기술 등을 적절히 활용 할 수 있어야 한다.

2. 네트워크 최적화 문제의 해결 절차 및 표현

2.1 네트워크 최적화 문제의 전반적인 절차

네트워크 최적화 문제의 해결을 위해서는 일반적으로 <그림 1>과 같은 단계를 거치게 된다.



<그림 1> 네트워크 최적화 문제의 해결을 위한 단계별 수행 과정

즉 현실에서 발생한 문제를 네트워크 형태로 표현하고, 이를 의사결정변수, 제약식, 목적식으로 정식화(formulation)하여야 한다. 그리고 만들어진 네트워크 최적화 문제가 기존에 알려진 문제(a well-known problem)으로 나타나는지 또는 유사한지를 알아보게 된다[5]. 대부분의 네트워크 문제는 정형화되어 있으며 이들 문제와 관련된 많은 연구가 이루어져 있으므로, 기존 문제와 유사한 문제이면 해결 방안 또한 쉽게 찾을 수 있다. 뿐만 아니라 기존 문제의 확장된 문제이거나 기존 문제를 하위문제로 포함하는 경우에도 해의 도출 및 알고리즘 개발에 기존의 아이디어들을 이용할 수 있을 것이다.

새로운 네트워크 최적화 문제의 경우 이 문제가 P인지 NP-complete인지를 파악하는 복잡성 분석(complexity analysis)이 필요하다. 많은 기존의 NP-complete 문제를 이용하여 새로운 문제가 NP-complete임을 증명할 수 있는 경우에는 최적해를 구할 수 있는 폴리노미얼 알고리즘이 존재하지 않으므로, 다양한 휴리스틱 해법들을 모색하여야 한다. 그렇지 않고 새로운 네트워크 최적화 문제가 P임을 보인다는 것은 폴리노미얼 알고리즘을 만들 수 있다는 것이 되는데, 현실적으로 이러한 경우는 매우 드물 뿐만 아니라 실제 알고리즘을 만들어서 증명한다는 것도 매우 힘들 일이 된다. 어떠한 경우이던 문제의 복잡성을 증명하는 과정은 필수적이며, 이를 위해서는 NP-complete의 개념 정립 및 증명 기법에 대한 지식이 필요하다[5, 7].

복잡성 분석을 통해 NP-complete인 경우 폴리노미얼 알고리즘을 만들 수 없으므로, 특수한 경우의 최적해법이나 휴리스틱 해법을 개발하여야 한다. 또한 만들어진 해법의 이론적 수행시간 분석(running time analysis)이 필요하며, 이는 주로 이론적 최악시간분석(worst case analysis)을 통해 이루어진다[3,9].

제안된 알고리즘을 실제 컴퓨터를 통해 실행하기 위해서는, 먼저 네트워크를 적절한 형태로 표현하여 컴퓨터 메모리에 저장하여야 한다. 이를 네트워크 표현(network representation)이라 하며, 이용되는 네트워크의 특성을 고려하여 적절한 형태로 표현하여야 한다[1]. 이는 구현하게 될 알고리즘의 입력요소(input)가 되며, 알고리즘의 수행시간에도 영향을 미치게 된다. 네트워크 표현을 위해서는 자료구조(data structure)에 대한 지식을 바탕으로, 특정한 프로그래밍 언어를 선택하고 이를 프로그래밍으로 연결시켜야 한다. 만약 C++, Java 등과 같은 객체지향언어인 경우 STL

(Standard Template Library) 등을 이용하여, 보다 쉽게 효율적으로 네트워크 표현이 가능하다[6].

특정한 프로그래밍 언어를 이용하여 알고리즘의 구현이 이루어지면, 이 알고리즘의 이론적 성능분석은 최악시간분석을 통해 이루어지지만 실제 문제에서 얼마나 효율적인지를 측정하려면, 현실문제를 반영할 수 있는 적절한 문제군(test problem set)을 정의하고 이를 이용한 반복실험을 실행하여야 한다. 이를 위해서는 문제군의 선정과 선정된 문제의 수행시간 등을 반복 측정하고, 이 결과를 적절한 형태의 표나 그림으로 나타내어야 한다. 이러한 결과도출에는 문제별 수행시간, 하한과 상한간 차이 분석, 기존 해법의 수행시간 등을 비교한 결과를 보여주어야 한다.

이처럼 네트워크 최적화 문제는 [그림 1]과 같이 여러 단계를 거치게 되는데, 각 과정 자체가 하나 또는 여러 가지 지식을 필요로 한다. 이론적 부분보다는 문제의 해결에 중점을 두고, 이를 위한 네트워크의 표현 방법과 최적화패키지의 연동을 중심으로 설명하기로 한다.

2.2 네트워크의 표현 방안[1]

네트워크 최적화는 네트워크로 표현되는 문제들을 대상으로 한다. 통상 유방향 네트워크(a directed network)은 $G=(N,A)$로 나타내며, 이때 N은 n개의 노드(node) 집합이고 A는 m개의 유방향 아크의 집합을 나타낸다. 네트워크 $G=(N,A)$를 표현 방법으로는 노드-노드 행렬(node-to-node matrix), 노드-아크 행렬(node-arc incidence matrix), 인접리스트(adjacency list) 등으로 나눌 수 있다.

노드-노드 행렬은 행과 열에 노드를 배열하고, $(i,j) \in A$이면 i행 j열은 1의 값을 가지고 $(i,j) \notin A$이면 i행 j열은 0의 값을 가지는 행렬로 나타낸 것이다. 노드-아크 행렬은 행에는 노드를 열에는 아크를 배열하는데, $(i,j) \in A$이면 i행 j열은 1의 값을 가지고 j행 i열은 -1을 가지고 나머지들은 0의 값을 가지는 행렬이다. 인접리스트는 노드 i에서 나가는 아크가 있는 노드의 집합으로서, $A(i)=\{j \in N: (i,j) \in A\}$로 나타낸다. 인접리스트는 실제로 자료구조를 이용하여 프로그래밍 할 때 가장 많이 사용되는 형태인데, 특히 sparse network의 경우 메모리 및 알고리즘 구현에 매우 효율적인 구조이기 때문이다.

실제 자료구조(data structure)를 이용하여 네트워크를 표현하려면, 상당한 정도의 프로그래밍 지식을 필요로 한다. 이처럼 복잡한 자료구조를 쉽게 이용할 수 있도록 표준화한 노력의 결실이 C++에 내장된 STL (Standard Template Library)이다[6]. STL은 C++ 뿐만 아니라 Java와 같은 대부분의 객체지향언어에서 그 기능을 제공하고 있는데, STL을 이용하면 누구나 쉽게 매우 안정적이면서 메모리 면에서 효율적인 자료구조의 구현이 가능하다. STL은 그 이름에서 알 수 있듯이 ANSI에 의해 표준화되었으며, C++ 및 Java 뿐만아니라 새롭게 등장하는 프로그래밍 언어들(예를 들어 Python, PHP, C# 등)도 STL을 이식하여 사용할 수 있게 해주고 있다.

STL은 1989년 이후부터 ANSI에서 표준화를 시작하여 1998년에 완성되었는데, 이를 이용하여 네트워크 최적화 분야의 많은 파생 라이브러리들이 만들어져 있다. 이러한 예로는 Boost Graph Library(BGL)를 들 수 있는데, 이에는 네트워크 표현 방법에 해당되는 Undirected Graph, Adjacency Graph, Vertex List Graph, Edge List Graph, Adjacency Matrix 등의 자료구조와 Minimum Spanning Tree, Shortest Path, Network Flow 등과 같은 기본적인 알고리즘들이 제공되고 있다[8]. 이처럼 STL 및 BGL을 이용하게 되면, 자료구조 구현에 대한 지식이 없어도 쉽게 알고리즘을 구현할 수 있다. 아쉽게도 STL은 객체지향언어에 대한 기본적인 지식 및 활용능력을 바탕으로 하므로, 본고에서는 이에 대한 자세한 논의는 생략하기로 한다.

3. LPSovle와 엑셀의 연동 및 엑셀을 이용한 네트워크 그리기

대부분의 네트워크 최적화 문제의 해결을 위해서는 LP 패키지를 필요로 한다. 특히 NP-complete에 속한 문제의 경우 branch-and-bound, branch-and cut 등을 통해 최적해를 구할 때 대개의 경우 LP를 하위모듈로 이용하게 된다. 대부분 LP 패키지들은 프로그래밍 언

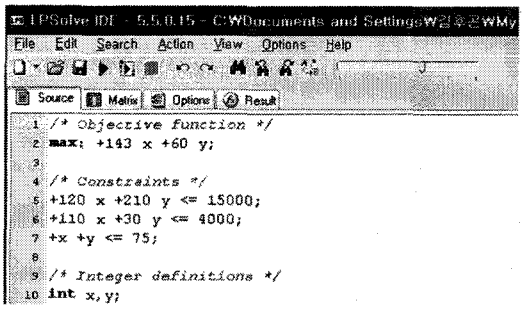
어를 통해 LP 모듈을 접근할 수 있는 API (Application Programming Interface)를 제공하고 있는데, 많은 패키지들이 C, C++, Java 등을 지원한다. LP 패키지 중에서 가장 유명한 것으로는 상용프로그램으로는 CPLEX를 들 수 있고, 누구나 사용할 수 있는 프리웨어로는 GNU의 GLPK (GNU Linear Programming Kit)와 LPSolve를 들 수 있다. CPLEX 및 GLPK는 C, C++, Jave를 위한 API를 제공하고 있으며, LPSolve의 경우 Visual BASIC을 위한 API를 지원하고 있다. 본 연구에서는 LPSolve의 Visual BASIC API를 통한 엑셀과의 연동과 엑셀의 그리기 기능(실제로는 MS 오피스의 그리기 기능)을 통해 네트워크를 직접 그리기(drawing)하는 예를 소개하기로 한다.

3.1 LPSolve와 엑셀의 연동

LPSolve는 프리웨어(엄밀히는 GNU 라이센스인 GPLK)로서 상용 패키지에 비해 성능은 못하지만, 다양한 선형계획법 모듈을 포함하고 있으면서 branch-and-bound 기반의 정수계획 문제의 해법도 제공하고 있다[2]. 또한 LPSovle IDE를 통해 다양한 LP 포맷을 읽어서, 해를 구할 수 있는 통합환경을 제공하고 있다. LPSovle IDE는 "http://sourceforge.net/projects/lpsolve/"에서 다운로드 하면 된다. LPSolve IDE에서 제공되는 LP 포맷으로는 MPS, MathProg, Cplex, LINDO, Dimacs 등이 있다. LPSolve에 대한 간단한 설명을 위해 다음과 같이 정수제약을 가지는 (Ex1)을 정의하기로 한다.

$$
\begin{aligned}
\text{(Ex1)} \quad & \max\ 143x + 60y && \text{(C0)}\\
& \text{s.t}\ \ 120x + 210y \le 15{,}000 && \text{(C1)}\\
& \qquad 110x + 30y \le 4{,}000 && \text{(C2)}\\
& \qquad x + y \le 75 && \text{(C3)}\\
& \qquad x, y \ge 0 && \text{(C4)}\\
& \qquad x, y\text{는 정수} && \text{(C5)}
\end{aligned}
$$

<그림 2>는 텍스트 파일 형식인 LPSolve 포맷으로 저장된 (Ex1)을 LPSolve IDE에서 읽어들인 화면이다.



<그림 2> (Ex1)의 LPSolve 포맷

LPSovle 포맷에서 목적함수는 "max:" 또는 "min:"되며, 주석은 "/* */" 사이에 표시하면 되고, 목적함수 및 제약식의 끝에는 ";"을 넣어서 구분하며, 비음제약은 디폴트로 제공되며, 정수제약은 "int x, y;"로 표시하면 된다. 『Action-Solve』 메뉴(또는 툴바의  버튼)를 실행하면 최적해를 구하게 되고, 

탭을 선택하면 최적해(목적함수=6266, x=22, y=52)를 보여준다.

이제 LPSolve를 엑셀과 연동하는 법을 알아보자. 본 연구에서는 엑셀은 2003을 기준으로 하고, LPSolve는 5.5.0.15를 가지고 설명하기로 한다. 엑셀에서 LPSolve의 모듈을 연동하려면 먼저 "http://sourceforge.net/projects/lpsolve/files/"에 가서 파일 lp_solve_5.5.0.15_COM.zip를 다운로드 하고, 이 파일 내에 있는 lpsolve55COM.dll를 "C:\Windows\System32" 폴더로 복사를 한다. 그리고 엑셀을 실행한 후 "도구-매크로-Visual Basic Editor"를 실행하면, 엑셀의 VBA (Visual Basic Application) 통합환경이 실행이 된다. VBA에서 lpsolve55COM.dll을 사용하기 위해서는 "도구-참조"를 실행하여 COM을 등록하는 창이 나타나면, 이 창에서 "lpsolve version 5.5 COM interface"를 찾아서 체크를 해 주면 된다.

이제 VBA에서 LPSolve의 명령어들을 이용하여 Visual Basic으로 코딩하여 최적해를 구하보자. <그림 3>은 (Ex1)의 최적해를 구하는 프로그램이다. 각 라인에는 적절한 주석을 통해 프로그램의 실행과정을 설명하고 있다. 이를 보다 자세히 알아보도록 하자.

1라인(Private Ex1 As lpsolve55)은 Ex1을 lpsolve55 개체로 선언하는 것이고, 4라인(Set Ex1 = New lpsolve55)은 Ex1이 lpsolve55 개체를 참조하도

```
1 : Private Ex1 As lpsolve55

2 : Sub main()
3 :   Dim lp As Long, col(0 To 1) As Long, param(0 To 1) As Double
4 :   Set Ex1 = New lpsolve55

5 :   Ex1.Init "."          'LPSolve가 실행될 경로를 설정함
6 :   lp = Ex1.make_lp(0, 2)  ' 제약식의 수 0, 변수의 수 2로 Ex1을 설정
7 :   Ex1.set_col_name lp, 1, "x" ' 1번째 변수의 이름을 'x'로 함
8 :   Ex1.set_col_name lp, 2, "y" ' 2번째 변수의 이름을 'y'로 함
9 :   Ex1.set_int lp, 1, True     ' 1번째 변수에 정수제약을 둠
10:   Ex1.set_int lp, 2, True     ' 2번째 변수에 정수제약을 둠
11:   col(0) = 1
12:   param(0) = 120
13:   col(1) = 2
14:   param(1) = 210
15:   Ex1.add_constraintex lp, 2, param(0), col(0), LE, 15000 '제약식 (C1) 추가
16:   col(0) = 1
17:   param(0) = 110
18:   col(1) = 2
19:   param(1) = 30
20:   Ex1.add_constraintex lp, 2, param(0), col(0), LE, 4000 '제약식 (C2) 추가
21:   col(0) = 1
22:   param(0) = 1
23:   col(1) = 2
24:   param(1) = 1
25:   Ex1.add_constraintex lp, 2, param(0), col(0), LE, 75  '제약식 (C3) 추가
26:   col(0) = 1
27:   param(0) = 143
28:   col(1) = 2
29:   param(1) = 60
30:   Ex1.set_obj_fnex lp, 2, param(0), col(0)
31:   Ex1.set_maxim lp                          '목적함수 (C0) 추가
32:   Ex1.solve lp                              'Ex1의 최적해를 구함
33:   Debug.Print "Obj= " & Ex1.get_objective(lp)
34:   Ex1.get_variables lp, param(0)
35:   Debug.Print Ex1.get_col_name(lp, 1) & "=" & param(0)
36:   Debug.Print Ex1.get_col_name(lp, 2) & "=" & param(1)
37:   Ex1.write_lp lp, "test.lp"
38: End Sub
```

<그림 3> (Ex1)의 최적해를 구하는 Visual Basic 코딩

록 지정하는 것이다. 5라인(Ex1.Init ".")은 LPSolve가 실행될 폴더를 지정하는 것으로, "."는 "내문서" 폴더를 가리킨다. 6라인(lp = Ex1.make_lp(0, 2))은 Ex1을 make_lp를 통해 제약식 0개와 결정변수 2인 선형계획법 문제로 정의하고, lp라는 변수가 이를 참조하도록 지정하는 것이다. 7라인(Ex1.set_col_name lp, 1, "x")은 1번째 변수의 이름을 'x'로 지정하는 것이고, 8라인은 2번째 변수의 이름을 'y'로 지정하는 것이다. 9라인(Ex1.set_int lp, 1, True)은 1번째 변수에 10라인은 2번째 변수에 정수제약을 부과하는 것이다.

LPSolve는 하한이나 상한을 지정하지 않으면, 기본적으로 모든 변수가 비음제약이 있는 것으로 처리한다. 하한을 지정하려면 set_lowbo를 상한을 지정하려면 set_upbo를 이용하면 된다.

11~15라인까지는 제약식 (C1) $120x + 210y \leq 15{,}000$를 추가하는 과정인데, 배열 col에는 결정변수의 인덱스가 저장되고 배열 param에는 결정변수의 계수가 저장된다. 15라인(Ex1.add_constraintex lp, 2, param(0), col(0), LE, 4000)에서 두 번째 인수의 값 2는 col 및 param의 크기를 지정하는 것이다. 16~20라인은 제약식 (C2) $110x + 30y \leq 4{,}000$를 추가하는 과정이고, 21~25라인은 제약식 (C3) $x + y \leq 75$를 추가하는 과정이다. 26~31라인은 목적함수식 (C0) max $143x + 60y$를 추가하는 과정인데, 31 라인(Ex1.set_maxim lp)은 목적함수를 최대화로 설정하는 것이다. 만약 최소화 문제이면 set_minim을 이용하여 지정하면 된다. 32라인(Ex1.solve lp)는 (Ex1)의 최적해를 구하는 것이며, 33~36라인은 Debug.Print를 통해 목적함수 및 결정변수의 최적해를 인쇄하는 과정이다. 37라인(Ex1.write_lp lp, "test.lp")은 생성된 (Ex1)을 test.lp로 "내문서" 폴더에 저장하는 것으로, 만약 "C:\temp"에 저장하고 싶다면 "test.lp"를 "C:\\test.lp"로 바꾸어 지정하면 된다.

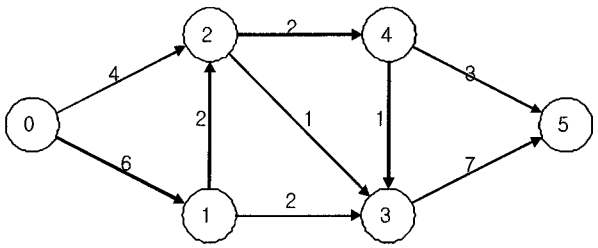
LPSolve는 하한이나 상한을 지정하지 않으면; 기본적으로 모든 변수가 비음제약이 있는 것으로 처리한다. 하한을 지정하려면 set_lowbo를 상한을 지정하려면 set_upbo를 이용하면 된다.

11~15라인까지는 제약식 (C1) $120x + 210y \leq 15{,}000$를 추가하는 과정인데, 배열 col에는 결정변수의 인덱스가 저장되고 배열 param에는 결정변수의 계수가 저장된다. 15라인(Ex1.add_constraintex lp, 2, param(0), col(0), LE, 4000)에서 두 번째 인수의 값 2는 col 및 param의 크기를 지정하는 것이다. 16~20라인은 제약식 (C2) $110x + 30y \leq 4{,}000$를 추가하는 과정이고, 21~25라인은 제약식 (C3) $x + y \leq 75$를 추가하는 과정이다. 26~31라인은 목적함수식 (C0) max $143x + 60y$를 추가하는 과정인데, 31 라인(Ex1.set_maxim lp)은 목적함수를 최대화로 설정하는 것이다. 만약 최소화 문제이면 set_minim을 이용하여 지정하면 된다. 32라인(Ex1.solve lp)는 (Ex1)의 최적해를 구하는 것이며, 33~36라인은 Debug.Print를 통해 목적함수 및 결정변수의 최적해를 인쇄하는 과정이다. 37라인(Ex1.write_lp lp, "test.lp")은 생성된 (Ex1)을 test.lp로 "내문서" 폴더에 저장하는 것으로, 만약 "C:\temp"에 저장하고 싶다면 "test.lp"를 "C:\\test.lp"로 바꾸어 지정하면 된다.

3.2 엑셀의 그리기 기능을 이용한 네트워크 표현

네트워크는 자료구조를 이용하여 적절히 표현할 수 있음을 앞에서 언급하였다. 그러나 네트워크 문제의 이해 및 해의 특성을 아는 쉬운 방법은 네트워크 자체를 그림으로 나타내어 보는 것이다. 윈도 상에서 네트워크 그리기는 많은 인내를 요구하는데, MS 오피스의 도형 그리기 기능을 이용하면 엑셀에서도 쉽게 네트워크를 그릴 수 있다.

<그림 4>는 MS 오피스의 도형 그리기 기능을 이용하여 네트워크를 그린 결과물이다. 이 네트워크를 그리기 위해서는 6개 노드의 레이블과 위치, 노드 i에서 노드 j로 가는 유방향 아크에 대한 리스트 및 거리 정보 등을 필요로 한다.



<그림 4> 네트워크 그리기의 예

이러한 정보는 엑셀에 저장시키는데, <그림 5>는 엑셀에 네트워크를 위해 입력한 내용을 보여주고 있다. <그림 5>의 『Sheet1』에서 셀 B2, E2, B6, B11은 『메모』를 가지고 있는 셀로서 NoOfNodes, NodeRadius, NodeLocation, NoOfArcs라는 『메모』가 있다. 이들 셀에는 메모와 동일한 『이름』(range name이라고 함)이 부여되어 있다. 즉 B2, E2, B6, B11의 『이름』은 NoOfNodes, NodeRadius, NodeLocation, NoOfArcs이다. 나중에 엑셀의 VBA에서는 『이름』이 지정된 셀의 값을 변수로 변환시키는데, 예를 들면

```
Dim NoOfNodes As Integer
NN =WorkSheet("Sheet1")_
        .Range("NoOfNodes").Value
```

와 같은 과정을 거치게 된다. 다음은 4가지 『이름』 의 역할을 정리한 것이다.

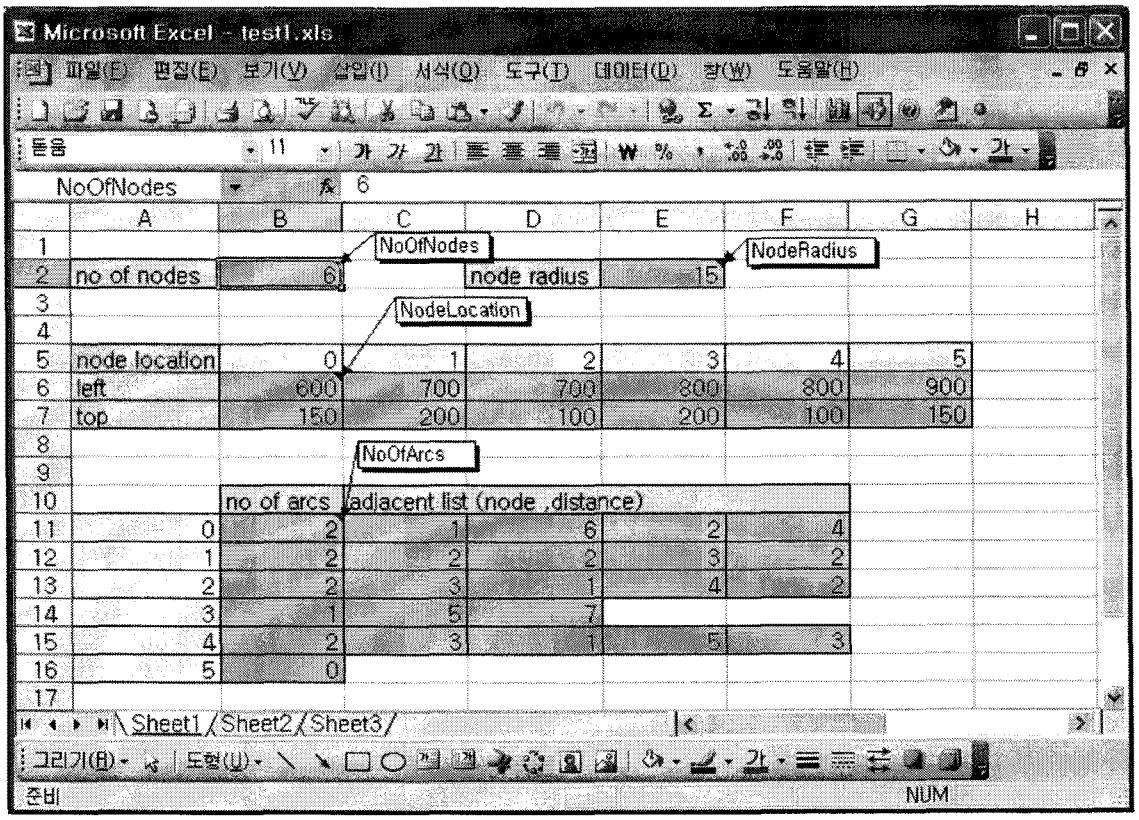
셀	『이름』	역할
B2	NoOfNodes	노드의 개수
E2	NodeRadius	그리게 될 노드들의 반지름 길이
B6	NodeLocation	노드의 중심 위치 좌표를 지정하는 셀이 시작되는 곳
B 11	NoOfArcs	인접리스트를 나타내는 셀이 시작되는 곳

<그림 5>의 Sheet1에서 범위 B11:G7까지는 노드의 위치를 나타내고 있다. 윈도 및 엑셀에서 마우스 또는 도형의 위치는 수학의 좌표와 달리 왼쪽 상단을 (0,0)으로 하고, (x,y) 좌표의 경우 x는 왼쪽에서 이동한 거리(left로 표시함)이고 y는 상단에서 이동한 거리(top으로 표시함)를 나타낸다. 『이름』 이 NodeLocation인 셀 B6는 0번째 노드의 left 값, B7은 0번째 노드의 top 값이 저장되어 있다. VBA에서는 『이름』 NodeLocation을 이용하여 다른 노드의 left 및 top 값도 쉽게 접근 할 수 있는데, 예를 들어 노드 3의 경우에

```
Dim left As Integer, top As Integer
left = WorkSheet("Sheet1")._
Range("NodeLocation").Offset(0, 3).Value
top= WorkSheet("Sheet1")._
 Range("NodeLocation").Offset(1, 3).Value
```

를 이용하여, 노드 3의 left와 top값을 읽어 들일 수 있다. 여기에서 Offset(0,3)에서 Range ("NodeLocation")를 기준으로 셀의 상대 위치를 나타내는데, 값 "0"은 동일한 행을, 값 "3"은 오른쪽으로 3열 이동한 위치로 이는 E6 셀을 된다.

『이름』 이 NoOfAcrcs인 셀 B11부터는 노드에서



<그림 5> 네트워크의 그리기를 위한 입력 정보

나가는 노드(emanating nodes) 및 거리의 쌍으로 된 인접리스트(adjacent list)를 나타내고 있다. 노드 0의 경우 인접리스트에 속한 노드의 수가 2개(셀 B11)이며, 리스트에서 첫번째는 노드 1(셀 C11)이고 노드 0에서 노드 1사이의 거리는 6(셀 D11)이며, 리스트에서 두번째는 노드 3(셀 E11)이고 노드 0에서 노드 3사이

```
1 : Sub Macro1()
2 :   'Integer %, Long &, String$, single!, double#
3 :    Dim i%, j%, NN%, NA%, NR!, NL!, left!, top!, dstId%, label$
4 :    Dim src As Object, dst As Object, sh As Worksheet
5 :    Set sh = Worksheets("Sheet1")
6 :    NN = sh.Range("NoOfNodes").Value
7 :    NR = sh.Range("NodeRadius").Value
8 :    NL = sh.Range("NodeLocation").Value
9 :    sh.DrawingObjects.Delete
10:    For i = 0 To NN - 1
11:        left = sh.Range("NodeLocation").Offset(0, i).Value
12:        top = sh.Range("NodeLocation").Offset(1, i).Value
13:        sh.Shapes.AddShape(msoShapeOval,left,top,NR*2, NR * 2).Select
14:        Selection.Name = "node" & CStr(i)
15:        Selection.Text = i
16:        Selection.HorizontalAlignment = xlCenter
17:        Selection.VerticalAlignment = xlCenter
18:    Next
19:    For i = 0 To NN - 1
20:       NA = sh.Range("NoOfArcs").Offset(i, 0).Value
21:       Set src = sh.Shapes.Item("node" & CStr(i))
22:       For j = 0 To NA - 1
23:          dstId = sh.Range("NoOfArcs").Offset(i, j * 2 + 1).Value
24:          Set dst = sh.Shapes.Item("node" & CStr(dstId))
25:          label = sh.Range("NoOfArcs").Offset(i, j * 2 + 2).Value
26:         Call draw_arc(sh, src, dst, label, NodeRadius)
27:       Next
28:    Next
29: End Sub

30: Public Sub draw_arc(sh As Worksheet, s As Object, t As Object, label$, r!)
31:   Dim dist!, slope!, x1!, x2!, y1!, y2!, xx1!, xx2!, yy1!, yy2!
32:   Dim directed_line As Object, arc_label As Object
33:   x1 = s.left + r
34:   y1 = s.top + r
35:   x2 = t.left + r
36:   y2 = t.top + r
37:   dist = Sqr((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2))
38:   xx1 = (r * (x2 - x1) + dist * x1) / dist
39:   yy1 = (r * (y2 - y1) + dist * y1) / dist
40:   xx2 = (-r * (x2 - x1) + dist * x2) / dist
41:   yy2 = (-r * (y2 - y1) + dist * y2) / dist
42:   sh.Shapes.AddLine(xx1, yy1, xx2, yy2).Line.EndArrowheadStyle _
          = msoArrowheadTriangle
43:   sh.Shapes.AddLabel(msoTextOrientationHorizontal, (x1 + x2) / 2 - r, _
          (y1 + y2) / 2 - r, 10, 10).TextFrame.Characters.Text = label
44: End Sub
```

<그림 6> [그림 6] VBA에서 네트워크 그리기

의 거리는 4(셀 F11)임을 보이고 있다.

VBA에서는 『이름』 NoOfArcs를 이용하여 Range ("NodeLocation")의 Offset을 이용하여 인접리스트의 값을 쉽게 읽어 들일 수 있다.

<그림 6>은 <그림 5>의 입력정보를 이용하여, [그림 4]와 동일한 네트워크를 엑셀에 직접 그리도록 VBA 상에서 프로그램을 입력한 내용을 보여주고 있다. 19~28라인의 Sub Macro1은 네트워크 그리기의 메인 프로시저가 되며, 29~43라인의 Sub draw_arc는 아크를 그리는 프로시저이다. 2라인은 Visual Basic에서 자료형에 대한 단축형으로, 예를 들어 정수 자료형의 경우 변수명 뒤에 %를 덧붙이면 된다. 5라인은 작업할 워크쉬트를 변수 sh에 할당하는 것이고, 6~8라인은 『이름』으로 지정된 NoOfNodes, NodeRadius, NodeLocation에 있는 셀의 값을 변수인 NN, NR, NL에 할당하는 것이다. 9라인은 워크쉬트 sh에 이전에 DrawingObject를 모두 삭제하는 명령이다. 10~17라인은 노드를 그리는 과정으로 for문을 이용하여 NN개의 노드를 그리는데, 노드 i에 대한 left 및 top 위치 지정은 11~12라인이고, 13라인은 AddShape명령으로 msoShapeOval이라는 가로 및 세로의 길이가 NR*2로 동일한 Oval(결국 이는 원이 됨)을 만드는 명령을 실행 한후, 이 만들어진 원을 선택(Selection)하도록 하는 명령이다.

AddShape로 만들어지는 Shape 개체들은 sh.Shapes.Items 배열에 저장되는데, 나중에 sh.Shapes.Items에 저장된 Shape를 호출하려면 이 Shape가 저장된 배열의 위치를 알거나 Shape 부여된 이름(Name)을 통하면 된다. 이때 Shape의 이름은 서로 중복되면 안된다. 14라인은 만들어진 노드 Shape에 이름(Name)을 부여하는 명령으로, 노드 0의 경우 "node0"라는 이름을 부여하게 된다. 15라인은 그려진 노드의 레이블로 노드 번호를 삽입하는 명령이고, 15~16라인은 이 노드 번호가 원의 중심에 위치하도록 하는 것이다.

19~28라인은 for 문을 이용하여 NN개의 노드 각각에 대한 인접리스트를 이용하여 아크를 그리고, 그려진 아크의 중간에 거리를 레이블로 표시하는 과정이다. 20라인은 노드 i의 아크리스트 수를 변수 NA에 저장하고, 21라인은 sh.Shapes.Items에 저장된 노드 i를 src 개체에 할당하는 것이다. 23라인은 아크리스트에 속한 노드 번호의 값을 변수 dstId에 저장하고, 24라인은 sh.Shapes.Items에 저장된 노드 dstId를 dst 개체에 할당하는 것이다. 25라인은 아크리스트에 속한 노드의 거리 값을 읽어서 변수 label에 할당하는 것이다. 26라인은 draw_arc를 통해 아크 및 아크의 레이블(거리)을 그리는 프로시져이다. draw_arc는 AddLine과 AddLabel을 이용하여 두 개의 원 사이에 화살표 및 이 화살표 위에 레이블을 추가하는 과정으로 자세한 설명은 생략하기로 한다.

4. 결론 및 향후의 연구방향

대부분의 네트워크 최적화 문제의 해결을 위해서는 LP 패키지를 필요로 한다. 이러한 LP패키지들은 상용의 경우 많은 비용을 들여서 구입하여야 하는데, 실제 적당한 크기의 문제들은 프리 소프트웨어인 LPSovle를 이용하면 쉽게 최적해를 구할 수 있다. 이러한 LPSolve는 VBA를 통해 엑셀에서도 쉽게 호출할 수 있으며, 본 연구에서는 예제를 통해 이를 설명하고 있다. 또한 현실에서 네트워크 최적화 문제를 실제적으로 해결할 수 있도록 하기 위해, 최적화 문제의 정의, 네트워크의 자료 표현 방법, 최적화 패키지인 LPSolve의 이해 및 엑셀을 통한 최적화 패키지와의 연동 방안을 소개하고 있다.

또한 실제 문제에서 네트워크 최적화 문제는 네트워크를 직접 그려서 보면서 작업하면 최적해의 특성이 해법의 적용타당성 등을 검증하는데 매우 효과적이다. 본 연구에서는 엑셀에서의 네트워크의 그리기 방법(drawing method)을 네트워크 자료구조와 연계하는 방안을 실제 예를 중심으로 소개하고 있다.

본 연구는 새로운 이론적 결과보다는 실제적으로 연구자들이 네트워크 최적화 문제를 다룰 수 있는 능력을 높이기 위하여, 엑셀 및 LPSolve의 연동과 네트워크 그리기에 중점을 두었다. 향후의 연구에서는 이를 누구나 쉽게 사용할 수 있도록 모듈한하고, 이를 엑셀의 Add-in 형태로 발전시켜 나가야 할 것이다. 또한 다양한 문제를 대상으로 엑셀 및 LPSolve의 연동과 네트워크 그리기가 가능하고, 이를 누구나 쉽게

할 수 있는 형태로 정리할 필요가 있다.

참 고 문 헌

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Network Flows : theory, algorithms, and aplications, Prentice-Hall, 1993.

[2] S. E. Buttrey, "Calling the lp_solve Linear Program Software from R, S-PLUS and Excel," Journal of Statistical Software, Vol. 14, Issue 4, 2005.

[3] W. J. Cook, W. H. Cunninghan, W. R. Pulleyblank, and A. Schrijver, Combinatorial Optimization, Wiley Pub., 1998.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, The MIT Press, Boston, MA, 1990, pp. 175-177.

[5] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-complete", Freeman and Company, 1979

[6] N. M. Josuttis, C++ Standard Library, Addison Wesley, 2003.

[7] R. G. Parker and R. L. Rardin, Discrete Optimization, Academic Press, 1988.

[8] J. G. Siek, L. Q. Lee, and A. Lumsdanie, The Boost Graph Library, Addison Wesley, 2002.

[9] V. Vazirani, Approximation Algorithm, Springer, 2003.

[10] J. Walkenbach, Excel VBA Programming For Dummies, Wiley Pub., 2004.

김 후 곤 (Hu-Gon Kim)

• 경성대학교 경영학과, 경영학사
• KAIST 경영과학과, 공학석사
• KAIST 산업경영학과, 공학박사
• 경성대학교 경영정보학과 교수
• 관심분야 : 최적화, 네트워크 이론, 환경정책평가