韓國數學敎育學會誌<數學敎育>
1987. 6. Vol. XXV, No. 3

Journal of the Korea Society of Mathematical Education
Jun. 1987. Vol. XXV, No. 3

Mathematical Programming에 관하여

— 박한식 교수님의 회갑을 기념하여 —

趙 達 勳

1. 서 언

정다각형의 작도 문제와 Euclidean ring 등에 희미하게 살아 있던 algorithm의 개념은 제 2 차 세계대전 후 과학과 경제 사회의 여러 분야에서 생기는 수많은 실제적인 문제들에 대한 구체적이고 과학적인 풀이 방법이 요구되고, 이에 따라 쓸 만한 Computer가 개발되면서 동시에 풀이방법 중에서도 보다 효과적인 방법이 필요하게 되어 그 중요성이 널리 인정, 보급되었고, Mathematical Programming이라는 한 수학 분야를 형성, 발전케 하였다. Mathematical Programming은 공리론적인 추상수학에 몰두하고 있던 수학계보다는 경제 산업계에서 먼저 발전되기 시작했고, 현재도 산업계가 보다 활발한 감이 없지 않으나, 이제는 대학에도 그의 양과 질이 확산 고급화하면서 새로운 수학의 framework가 필요하다[26]고 주장하는 학자들이 있을 만큼 발전되었다.

Mathematical Progamming이란 다소 무리해서 간단하게 말하면, 어떤 변역 D에서 정의된 실함수 f가 주어졌을 때, f의 최대 또는 최소값을 구하는 방법론을 연구하는 분야라 할 수 있겠는데, 변역 D와 함수 f의 형태에 따라서 문제의 난이도와 중요성이 달라진다. 때로는 변역 D가 분명히 잘 정의되어 있어도 D가 공집합인지 아닌지를 가리기 어렵고, 그것이 곧 제일 중요한 과제일 때도 있다.

간단한 예로, 주어진 $m\times n$ 행렬 A와 m vector b에 대하여 집합 $D=\{x\in R^n : Ax\geq b\}$는 공집합인가의 문제, 주어진 graph G 안에 어떤 특수한 형태의 subgraph가 있는가의 문제 등은 이러한 유의 대표적인 문제들이다. D가 확실한 유한집합이고 f가 또한 잘 정의되어 있어도 f의 최소값을 구하는 일이 만만하지 않을 때가 많은데, 예를 들면 6 절의 assignment 문제를 들 수 있다. 이 경우에 D는 서로 다른 n개의 물건을 배열하는 순열의 전체로서 $n!$개로 구성되어 있고, D의 각 원소 x에 대하여 함수값 $f(x)=\sum\sum c_{ij}x_{ij}$을 쉽게 구할 수 있으므로 $f(x)$들을 차례로 계산해서 최선의 답을 고르면 되겠지만, n이 큰 수일 때는 좀 어렵게 된다. 가령 n이 100이라면 D의 크기는 100!이고, D의 원소 하나하나에 대하여 $f(x)$를 모두 계산해 본다고 할 때 백만분의 일초에 하나의 $f(x)$를 계산할 수 있는 고성능 computer를 지구 표면에 가득 채우고 태양계의 생성에서 종말까지 계산한다고 해도 끝내지 못한다[11]. 따라서, 가능한 모든 경우를 검토하지 않고 최선의 답을 찾아내는 좋은 방법을 연구해 내는 일이 중요하다.

이 글에서는 정리의 증명이나 구체적인 풀이 방법의 기술은 피하고, 일차형의 문제들 중에서 비교적 간단한 문제들을 통해서 기본적인 개념들을 소개하고자 한다.

2. Computational Complexity

여기 1024명의 전화번호가 적힌 자료가 있다고 할 때, 특정한 사람 K의 번호를 찾고자 한다고 하자. 즉, 주어진 1024의 자료 중에서 K씨의 번호를 찾거나 없다는 것을 밝히고자 한다. 편의상 이름들의 sequence를

$$N(1),\ N(2),\ N(3),\ \cdots,\ N(1024)$$

전화번호 sequence를

$$T(1),\ T(2),\ T(3),\ \cdots,\ T(1024)$$

즉, $N(i)$씨의 번호를 $T(i)$라 하면 위의 문제는 주어진 이름 K에 대하여 $N(i)=K$인 i를 찾아서 $T(i)$를 읽는 것이 된다.

Algorithm 1

i값의 차례로 $(i=1, 2, \cdots, 1024)$, $N(i)=K$인가를 검토하여 i를 찾도록 하고, $i=1024$일 때까지도 $N(i)\neq K$이면 K가 명단에 없다고 보고한다.

여기서 여러 가지의 이름 K에 대하여 Algorithm 1을 반복 적용한다면 각 경우에 적어도 평균 512번 이상 $N(i)$가 K인가를 검토하게 된다. 반면에 sequence $N(i)$가 가나다 순으로 잘 정리되어 있다면 다음의 binary search는 $N(i)<K$인가를 10번 검토한다.

Algorithm 2

1단계. $l:=0$, $u:=1024$, $k:=512$라 한다.

2단계. $u-l$이 1보다 큰 동안은 다음을 반복한다.

i) $N(k)<K$이면 $l:=k$라 놓고
$N(k)\geq K$이면 $u:=k$라 놓는다.

ii) $k:=\dfrac{u+l}{2}$라 한다.

3단계. $N(u)=K$이면 u번째에 K씨의 번호 $T(u)$가 있고 $N(u)\neq K$이면 K가 명단에 없다.

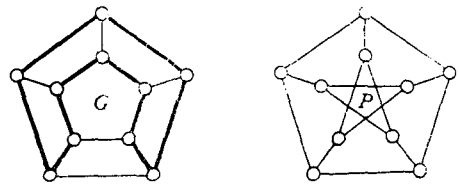
보다 구체적으로 말하면 Algorithm 2는 $N(k)<K$인가를 10번, $N(u)=K$인가를 1번 검토하게 되어 Algorithm 1보다는 평균 계산 속도가 훨씬 빠르다. Algorithm 1은 최악의 경우 $N(i)=K$인가를 1024번 조사해야 한다.

위의 문제에서 자료의 크기를 n(=사람수)이라 하면 Algorithm 1, 2는 각각 평균 $\frac{1}{2}n$, $\log_2 n$의 상수배만큼의 시간이 걸린다. Computational complexity를 논할 때는 계수와 낮은 차수의 항을 무시하고 기호 $O(\cdot)$를 써서 Algorithm 1, 2는 각각 $O(n)$, $O(\log_2 n)$만큼의 시간이 걸린다고 한다. 위의 문제에서 n명의 명단이 가나다 순으로 잘 정리되어 있지 않을 때 이것을 크기 순으로 정리하는 문제를 생각해 보자. 한가지 방법은 제일 앞에 오게 될 이름을 찾아서 $N(1)$, 그 다음에 올 이름을 찾아서 $N(2)$ 등이라 하는 것인데, 이 방법은 이름을

$$\sum_{k=1}^{n-1} k=\frac{1}{2}(n-1)(n-2)$$

번 비교하게 되어 시간이 $O(n^2)$만큼 걸린다. 반면에 여기서는 생략하지만 $O(n\log_2 n)$인 많은 좋은 방법들이 알려져 있다[1, 24].

다음은 graph theory의 한 문제를 생각해 보자. 몇 개의 점(vertex, node)들의 집합 V와 이들을 연결하는 선(edge, arc)들의 집합 E로 이루어진 체계를 graph라 하고 $G=(V, E)$로 나타낸다. 아래 그림에서 두 graph G와 P는 모두 10개의 vertex와 15개의 edge로 구성되어 있다.



주어진 graph에서 모든 vertex들을 차례로 연결하는 circuit를 Hamiltonian circuit(줄여서 HC)라 할 때 문제는 이러한 HC가 graph 안에 있느냐 하는 것이다. 위의 두 graph 중에서 G에는 HC가 있고(진한 선) Peterson graph P에는 HC가 없다. 주어진 graph 안에 HC의 존재 여부를 밝히기 위하여 vertex의 배열을 모두 검토하는 것은 좋은 방법이 못 된다. 왜냐 하면 vertex의 수를 n이라 하면 모두 $(n-1)!$가지의 순열이 있고, 이 $(n-1)!$은 n에 관한 좋은 함수라고 할 수 없기 때문이다. 여기서 좋은 함수라 하는 것은 다항식이나 다항식보다 작은 함수를 말한다.

위의 HC 문제는 NP-complete라고 하는 수많은 문제 중의 하나인데, 그 의미는 polynomial time 이내에 해답을 주는 좋은 algorithm이 알려지지도 않았고, 그러한 좋은 algorithm이 있을 수 없다는 것이 증명된 것도 아니지만 그러나 그 어려움의 정도는 비슷하다는 것이 증명된 많은문제 중의 하나라는 것이다[16]. 이 NP-

omplete에 속하는 문제는 여러 분야에서 생기는데, 문헌 [16]에는 300개 이상의 문제가 수록되어 있고, Journal of Algorithms의 NP-ompleteness column에는 새로운 문제가 계속 소개되고 있다.

주어진 정수가 솟수인가를 판정하는 문제도 중요하고 재미있는 문제이다. 예를 들면 수 $\frac{1}{9}(10^{317}-1)$은 솟수인가? 수 $2^{128}+1$은 합성수인가? 처음의 수는 317개의 1로 구성되어 있는 솟수이고 둘째수는 합성수로서

$$2^{128}+1=59,649,589,127,497,217$$
$$\times 5,704,689,200,685,129,054,721$$

이다[30]. 주어진 정수 n의 자리수는 $\log n$에 비례하므로 input의 크기는 $O(\log n)$이다. 따라서 $\log n$의 polynomial time 이내에 해답을 주는 algorithm을 찾는 것이 목적이다.

솟수 판정의 문제보다 더 잘 알려진 문제로서 주어진 자연수 n에 대하여 n보다 작은 솟수 전체를 찾는 문제가 있는데, 잘 알려진 방법으로는 Eratosthenes의 체가 있지만 그보다 더 빠르고 n보다 작은 솟수 전체를 구하는 방법이 알려져 있다[27]. 위의 솟수 판정의 문제는 NP-complete는 아니고 polynomial time의 algorithm이 있으리라고 믿고 있지만 아직 확실하지 않다.

Polynomial time algorithm의 개념은 1965년 Edmond에 의하여 소개된[12] 후 R. Karp, Cook 등의 많은 업적에 힘입어 많은 발전을 하고, 요즘은 가장 인기 있는 한 연구 분야가 되었다고 할 수 있겠고, NP-complete의 문제가 해결된다면 아마도 금세기 최대 업적 중의 하나가 될 것이다.

3. Linear Programming 문제

LP 문제

$$\max f(x) : f(x)=cx,\ Ax=b,\ x\geqq 0 \qquad (P)$$

생각해 보자. 여기서 A는 $m\times n$ 행렬이고 c, x는 이에 준하는 vector들이다. 편의상

$$\text{rank}(A)=m,\ m<n$$

라 하고,

$$X=\{x\in R^n \mid Ax=b,\ x\geqq 0\}$$

이라 하면, 문제 (1)은

$$\max f(x) : f(x)=cx,\ x\in X$$

가 된다. 여기서 f는 일차함수이고 X는 다면체이지만, 먼저 이와 같은 특수한 조건을 갖지 않는 일반적인 경우의 풀이 방법을 보자.

Algorithm 3

한 점 $x^1\in X$을 알고 있다면 x^1에서 f의 gradient ∇f를 구하고 내적 $\nabla f\cdot d$가 양수가 되는 방향 d를 정한다. $x^2=x^1+\lambda d$가 조건

$$x^2\in X,\ f(x^2)>f(x^1)$$

을 만족하는 범위내에서 적당한 양의 실수 λ를 정하고 $f(x^2)$가 만족할 만하면 정지하고, 그렇지 않으면 위의 과정을 반복한다.

위에서 방향 d를 정할 때 $d=\nabla f$라 하는 것이 가장 쉬운 방법이지만, 실제에서 이것이 가장 좋은 방법이 못 되는 때가 있어서 다른 여러 변형이 있고, step size λ를 정하는 것도 여러 가지 방법이 있다.

LP문제에 대해서는 X가 다면체이고 f가 일차함수이기 때문에 optimal point가 X의 꼭지점들 중에 있다. 그러므로 X의 꼭지점들에만 주의를 집중해서 꼭지점에서 꼭지점으로 이동하면서 optimal point를 찾으려하는 방법이 G. Dantzig의 Simplex Method이다.

A의 한 $m\times m$ 부분행렬 B가 조건

(P_1) B는 정칙행렬

(P_2) $B^{-1}b\geqq 0$

을 만족할 때 A의 열순서를 적당히 바꾸어서 $A=(B,N)$이라 하면, 식 $Ax=b$는

$$Bx_B+Nx_N=b$$

가 되고, B의 역행렬을 왼쪽에 곱하면

$$x_B+B^{-1}Nx_N=B^{-1}b$$

가 된다. 따라서

$$x^B=(x_B^*,\ x_N^*),\ x_B^*=B^{-1}b,\ x_N^*=0$$

이라 하면, $x^B\geqq 0$이고

$$\begin{aligned} f(x)&=C_Bx_B+C_Nx_N\\ &=C_B(B^{-1}b-B^{-1}Nx_N)+C_Nx_N\\ &=C_BB^{-1}b+(C_N-C_BB^{-1}N)x_N\\ &=f(x^B)+(C_N-C_BB^{-1}N)x_N \end{aligned}$$

이 되므로, B가 조건

$$(P_3)\quad C_N - C_B B^{-1}N \leqq 0$$

까지를 만족시킨다고 하면, 어떤 $x \geqq 0$에 대해서나

$$(C_N - C_B B^{-1}N)x_N \leqq 0$$

이 되어 $f(x^B)$가 $f(x)$의 최대값이 된다. 즉, A의 한 $m \times m$ 부분행렬 B가 조건 (P_1), (P_2), (P_3)을 만족하면 $f(x)$는 점 $x^B = (B^{-1}b, 0)$에서 최대값을 갖는다.

예를 들면, 문제

$$\begin{aligned} \max\ & x_1 - x_2 + x_3 + x_4 \\ \text{s.t. } & x_1 - x_2 + x_3 = 1 \\ & x_1 - x_2 + 2x_3 + x_4 = 2 \\ & x_1, x_2, x_3, x_4 \geqq 0 \end{aligned}$$

에서 $B_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$, $B_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ 이라 하면, B_1은 (P_1), (P_2)를 만족하지만 (P_3)을 만족하지 않고, B_2는 (P_1), (P_2), (P_3) 모두를 만족한다. 따라서 이 문제는 $x^* = (1, 0, 0, 1)$에서 최대값 2를 갖는다.

조건 (P_1), (P_2)를 만족하는 B를 feasible base라 하고, 위의 예제에서 B_1과 B_2처럼 $m-1$개의 열이 서로 같고 한 열만 서로 다를 때 neighboring bases라 하면 Simplex Method는 간단히 다음과 같이 쓸 수 있다.

Simplex Method

한 feasible base B를 알고 있다면

1단계. B가 조건 (P_3)을 만족하면 정지한다. 그렇지 않으면

2단계. 새로운 점에서의 $f(x)$ 값이 이전의 값보다 작지 않으면서 B의 neighbor에 있는 feasible base B'을 찾는다.

3단계. B'을 새로운 B로 간주하고 위의 과정을 반복한다.

이 Simplex Method에서 해결해야 할 문제는 우선 처음 시작하는 feasible base B를 어떻게 찾느냐 하는 것과, 위의 방법이 유한회의 반복으로 끝나겠는가 하는 것들이라 하겠다. 그리고 물론 반복하는 횟수를 줄이는 것도 실질적으로 중요한 문제라 하겠다.

위의 문제 (P)와 관련하여 LP 문제

$$\min g(y) : g(y) = yb,\ yA \geqq c \qquad (D)$$

를 (P)의 dual이라 한다. 이 문제를 dual이라 하는 이유는 $Y = \{y \in R^m \mid yA \geqq c\}$라 할 때 첫째, 모든 $x \in X$, $y \in Y$에 대하여 부등식

$$f(x) \leqq g(y)$$

가 성립하고(weak duality), 둘째, $f(x)$가 x^*에서 최대값을 가지면 $f(x^*) = g(y^*)$가 되는 $y^* \in Y$가 있고 $g(y^*)$가 g의 최소값이 된다(strong duality). 따라서 집합

$$\{(x, y) \mid x \in R^n,\ y \in R^m,\ x \geqq 0,\ Ax = b,\ yA \geqq c,\ cx = yb\}$$

안의 한 점을 찾으면 두 문제 (P), (D)가 동시에 해결된다.

위의 duality와 Simplex Method의 발견[10]과 실용화된 Computer에 의한 여러 분야에서 생긴 대형 LP 문제의 성공적인 풀이에 의하여, LP가 널리 유행 보급되어 있긴 하지만 이론적으로는 해결되지 않은 문제가 남아 있다. 즉, Simplex Method가 1, 2, 3 단계를 무한히 반복할 cycling의 가능성[5, 19]은 Bland rule[6] 등을 써서 예방을 한다고 하더라도 목적하는 함수값 $f(x)$에 별 변화없이 오랫동안 계산을 지속하는 stalling의 가능성은 아직 남아 있다[9]. Simplex Method 이외에 언급을 꼭 해야 할 두 방법이 있는데, 그 중 하나는 문제 (P)를 풀기 위해서 (P)와 동치인 어떤 부등식의 해의 존재 문제를 생각하고, 해가 있다면 그것을 포함하게 될 ellipsoid를 생각해서 그 ellipsoid를 차츰 작게 만들어서, ellipsoid가 충분히 작아진 다음에 그의 중심이 본래의 연립부등식을 만족하는지의 여부를 검토하여 구하는 해를 찾는 ellipsoid method이다[14, 23]. 이 방법은 LP 문제를 polynomial time에 푸는 것이 증명되지만, 그의 실용성은 의문시되고 있다[7]. 다른 한 방법은 Karmarkar에 의하여 가장 최근에 얻어진 것으로서, 주어진 문제를 변역 X가 simplex이면 원래 문제와 동치인 다른 LP로 바꾼 다음 simplex의 내부의 한 점에서 출발하여 사영변환을 반복 이용함으로써 원하는 꼭지점에 접근하는 방법이다[2, 8, 21]. 이 새로운 방법은 이론적으로만이 아니라 실제적으로도 매우 만족할 만한

는 Bell 연구소의 보고가 있었다[22]. 위의 LP 문제에 대한 해 $x=(x_1, x_2, \cdots, x_n)$에서 일부 또는 전부의 x_i가 정수이어야 한다는 조건이 첨가되면 문제는 지극히 어려워지지만, 많은 경우에 특히 실제 문제에서 이러한 조건이 요구되므로 정해를 못 구하면 근사해라도 구할 수 있는 방법이 필요하고, 따라서 특수한 경우의 특수한 방법들도 연구되고 있다[17].

4. 일차연립방정식과 Graph

2절에서 언급한 graph는 여러 분야에서 여러 형태로 이용되겠지만 여기서 특수한 형태의 일차연립방정식의 풀이 graph가 어떻게 이용될 수 있는가를 알아본다. 구체적으로 일차연립방정식을 가감법으로 풀 때 시행해야 하는 기본적인 계산의 횟수를 줄이는 일과 graph의 한 관계를 알아보고자 한다. 우선 다음 문제

$$\begin{cases} x_1+x_2+x_3+x_4=0 \\ x_1+2x_2 \qquad\quad =-1 \\ x_1 \quad +x_3 \quad\ =2 \\ x_1 \qquad\ -x_4=2 \end{cases}$$

에 대한 두 가지의 풀이 방법을 검토해 보자. 아래 계산 과정에서 점들은 영을 나타낸다.

풀이 1 :

$$\begin{bmatrix} \textcircled{1} & 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 & -1 \\ 1 & 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & -1 & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ \cdot & \textcircled{1} & -1 & -1 & -1 \\ \cdot & -1 & 0 & -1 & 2 \\ \cdot & -1 & -1 & -2 & 2 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ \cdot & 1 & -1 & -1 & -1 \\ \cdot & \cdot & \textcircled{-1} & -2 & 1 \\ \cdot & \cdot & -2 & -3 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ \cdot & 1 & -1 & -1 & -1 \\ \cdot & \cdot & 1 & 2 & -1 \\ \cdot & \cdot & \cdot & \textcircled{1} & -1 \end{bmatrix} \Rightarrow$$

$$\cdots \Rightarrow \begin{bmatrix} 1 & \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & \cdot & \cdot & -1 \\ \cdot & \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & \cdot & 1 & -1 \end{bmatrix} \qquad \therefore \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

풀이 2 :

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 & -1 \\ 1 & 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & \textcircled{-1} & 2 \end{bmatrix} \Rightarrow \begin{bmatrix} 2 & 1 & 1 & \cdot & 2 \\ 1 & 2 & 0 & 0 & -1 \\ 1 & 0 & \textcircled{1} & 0 & 2 \\ -1 & 0 & 0 & 1 & -2 \end{bmatrix}$$

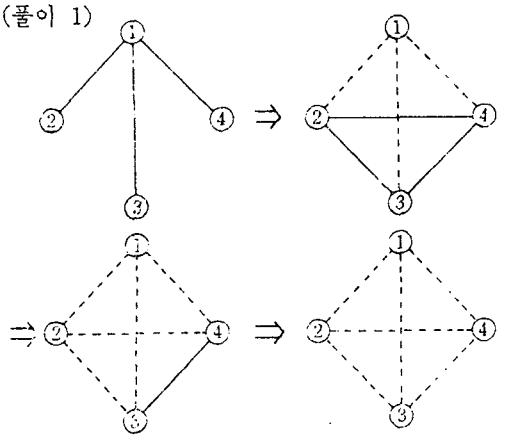
$$\Rightarrow \begin{bmatrix} 1 & 1 & \cdot & \cdot & 0 \\ 1 & \textcircled{2} & 0 & 0 & -1 \\ 1 & 0 & 1 & 0 & 2 \\ -1 & 0 & 0 & 1 & -2 \end{bmatrix} \Rightarrow \begin{bmatrix} \textcircled{1/2} & \cdot & \cdot & \cdot & 1/2 \\ 1/2 & 1 & 0 & 0 & -1/2 \\ 1 & 0 & 1 & 0 & 2 \\ -1 & 0 & 0 & 1 & -2 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 1 & \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & \cdot & \cdot & -1 \\ \cdot & \cdot & 1 & \cdot & 1 \\ \cdot & \cdot & \cdot & 1 & -1 \end{bmatrix} \qquad \therefore \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

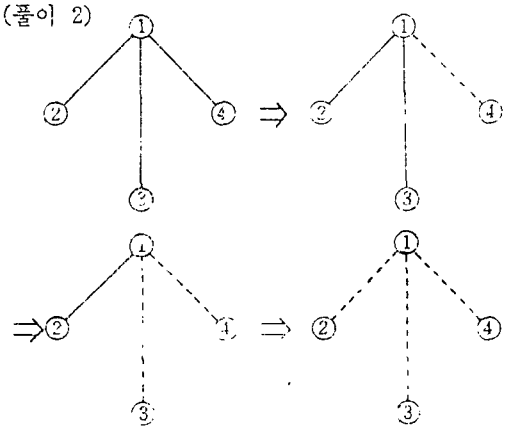
위의 두 가지 풀이 과정을 검토해 보면, 풀이 2는 풀이 1보다 기본적인 계산(가감승제)을 더 적게 하고 답을 얻었음을 알 수 있다. 보다 엄밀히 따져보면, 풀이 2에서는 원래 0이었던 자리(a_{23}, a_{24}, a_{32}, $\cdots$)에서 가감승제를 전혀하지 않았기 때문에 기본적인 계산의 횟수가 풀이 1의 경우보다 적다. 다시 말하면, 대칭행렬 A에 대하여 일차연립방정식 $Ax=b$를 가감법으로 풀 때 원래 0이었던 A의 각자리 a_{ij} 중에서 가능하면 시종 변하지 않는 자리가 많은 가감법 순서를 따르는 것이 좋겠다.

$n\times n$ 대칭행렬 A에 대하여 graph $G=(V, E)$를 $V=\{1, 2, \cdots, n\}$, $E=\{ij \mid a_{ij}\neq 0\}$와 같이 정의한다. 그리고 위의 두 풀이 과정에 따른 graph의 변화를 보면 다음과 같다.

(풀이 1)



(풀이 2)



(풀이 1)에서는 vertex 들을 1, 2, 3, 4의 순으로, (풀이 2)에서는 4, 3, 2, 1의 순으로 제거하면서 한 vertex를 제거할 때마다 그 근처에 있는 vertex 들을 edge로 연결해 준다. 이러한 과정에서 새로 첨가되는 edge들(앞 그림에서 edge 23, 24, 34)은 가감법 과정에서 생기는 바람직하지 않은 일, 즉 A의 원소들 중에서 원래 0이었던 것이 0 아닌 수로 바뀌는 것에 대응한다.

따라서, 가감법을 시행하기 전에 A의 graph G에서 vertex 순서를 잘 정해서 새로 첨가되는 edge의 수를 적게 한 후 그 순서에 따라 가감법을 시행하는 것이 바람직하다. 물론 행렬 A의 규모(n^2)가 작을 때는 아무 순서를 택하거나 관계없지만, n이 크고 A가 sparse인 경우에는 위의 방법이 매우 효과적일 수 있다. 보다 구체적으로는 boundary value가 주어진 편미분방정식에 대하여 보다 정밀한 근사해를 구하려 할 때 n이 크고 각 행과 열에 0 아닌 자리가 아주 적은 경우가 생긴다. 위에서 생긴 순수한 graph 문제, 즉 주어진 graph에 대하여 가능한 한 새로운 edge가 적게 첨가되도록 하는 vertex elimination order를 찾는 문제는 적어도 directed graph에 대하여 NP-complete이기 때문에[28] 일반적으로 최선을 찾는 algorithm은 아직 없지만, 몇 가지 특수한 형태의 graph에 대해서는 가능하다[18].

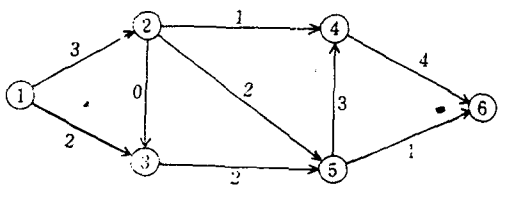
5. Network Flows

Directed graph $G=(V, E)$와 함수 $x: E\longrightarrow R$가 주어졌을 때, 각 vertex $v\in V$에 대하여 b_v를

$$b_v=\sum_{j\in V} x(v,j)-\sum_{i\in V} x(i,v) \qquad (1)$$

와 같이 정하고 $b_v>0$이면 v를 source, $b_v<0$이면 v를 sink라 한다. 아래의 network에서 각 edge 옆에 쓰인 숫자를 $x(e)$라 하면

$$b_1=5,\ b_6=-5,\ b_2=b_3=b_4=b_5=0$$



이 예에서처럼 source가 하나 ($s=1$)이고 sink도 하나($t=6$)인 경우에 x는 $s-t$ flow라고 보는 것이 자연스럽다. Directed graph의 각 edge에 용량 제한이 있고 source와 sink가 각각 하나씩 있는 경우, 용량제한

$$l(e)\leqq x(e)\leqq u(e),\ \forall e\in E$$

을 만족하는 s-t flow 중에서 b_s를 최대로 하는 x를 maximum flow라 한다. 따라서, 문제는 주어진 network에 대하여 어떻게 maximum s-t flow를 구하고, 구한 flow x가 최선의 답인가를 어떻게 확인할 수 있는가 등이다. Vertex 집합 V의 부분집합 S가 $s\in S$, $t\notin S$일 때

$T=VS$라 하면 edge들의 집합

$$[S, T]=\{(i,j)\mid i\in S,\ j\in T \text{ 이거나 } i\in T,\ j\in S\}$$

이 s와 t를 분리하는 cut가 된다. 이러한 cut에 대하여 그의 용량 $c(S, T)$을

$$c(S, T)=\sum_{i\in S, j\in T} u(i,j)-\sum_{i\in T, j\in S} l(i,j)$$

와 같이 정의하면 s-t flow의 최대값은 $c(S, T)$의 최소값과 같다(Max Flow Min Cut정리). Maximum flow를 찾는 algorithm으로는 현재까지는 $O(|v|^2)$인 것이 가장 빠르다[13, 29]. 유한집합 S의 크기를 $|S|$로 나타낸다. 한편 G의 각 edge에 단위 비용 C_e가 주어져 있을 때 s-t flow 양이 일정한 것 중에서 총비용 $\sum C_e x(e)$이 최소인 x를 구하는 Max Flow, Min Cost Flow 문제에 대하여는 그렇게 좋은 algorithm이 알려져 있지 않다[9].

Directed graph $G=(V, E)$에 대하여 $|V|\times|E|$ 행렬 $A=(a_{ie})$를

$$a_{ie}=\begin{cases} 1 & \text{edge } e \text{가 } i \text{에서 시작할 때} \\ -1 & \text{edge } e \text{가 } i \text{로 향하고 있을 때} \\ 0 & \text{기타} \end{cases}$$

로 정하면 위 예제에서의 식 (1)은 $Ax=b$이고 A, b는

$$A=\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

$$b=\begin{bmatrix}5\\0\\0\\0\\0\\-5\end{bmatrix}$$

가 된다. G가 connected이면 A의 rank는 $|V|-1$이 되고, A의 행들 중에서 아무것이나 하나를 지우면 남은 A의 행 vector들은 일차독립이 된다. 편의상 A의 행 하나를 지운 행렬도 A라고 하면, A는 $(|V|-1)\times|E|$ 행렬이고 A의 $(|V|-1)\times(|V|-1)$ 부분행렬 B를 생각할 때 B가 정칙이 되기 위한 필요충분조건은 B의 열들에 해당하는 edge들이 circuit을 포함하지 않을 것, 즉 G의 spanning tree가 되는 것이다. 따라서, B가 정칙이면 B는 triangular형이 되고, 역행렬 B^{-1}이 쉽게 구해질 뿐만이 아니라 B^{-1}의 각 자리수가 0 또는 ±1이 된다. 그러므로 Min Cost Flow 문제를 LP 문제로 보고 Simplex Method를 적용하면 중간 계산 과정이 빠르고 빠를 뿐더러, 일반적인 LP에 대한 Simplex Method의 많은 본질적인 요소들을 잃지 않기 때문에 좋은 점도 많다.

6. 짝짓기문제

주어진 graph $G=(V,E)$에서 matching M은 E의 한 부분집합으로서 어느 두 edge도 한 vertex를 공유하지 않은 것을 말한다. 아래 그림 1에서 $M_1=\{14,23,67\}$은 한 matching이다.

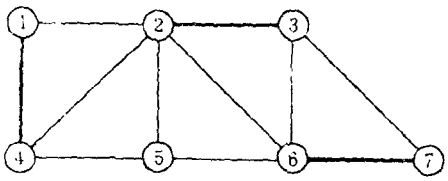


그림 1

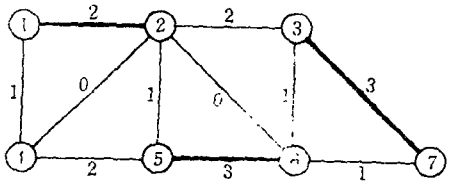
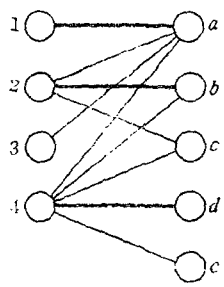


그림 2

Matching 중에서 짝의 수가 가장 많은 것을 최대 matching이라 하면, 첫번째 문제는 주어진 graph에 대하여 최대 matching을 어떻게 찾을 수 있으며, 찾은 것이 최선이라는 것을 어떻게 쉽게 확인할 수 있느냐이다. 둘째로는 각 edge에 무게가 주어졌을 때 최대 matching 중에서 무게의 합이 최대가 되는 것은 어떻게 찾을 수 있는가이다. 위의 그림 2에서 $M_2=\{12,37,56\}$은 무게의 합이 최대인 최대 matching이다. 위의 matching 문제는 Edm의 blossom algorithm에 의하여 $O(|V|^4)$ time에 풀리는데 [12] 여기서는 그 algorithm의 설명은 피하고 특수한 몇 가지의 예를 설명하고자 한다.

Bipartite matching. 그림과 같이 vertex 집합이 두 부분 $V=A\cup B$로 나누어진 graph에서의 짝짓기 문제를 보자. 이 경우는 결혼 중매 문제로 볼 수도 있겠다. 즉, 좌우 점들이 남여를 나타내고 서로 가능한 남과 여를 선으로 연결해 놓은 것으로 보면 중매인의 문제는 보다 많은 짝을 맞추는 것이 된다. 그림과 같은 특수한 경우에는 굵은 선으로 표시한 $M=\{1a,2b,4d\}$이 한 최대 matching이다. 다시 말하면 이 경우에 네 쌍을 맞출 수는 없다. 왜냐하면 크기가 3인 vertex 집합 $C=\{a,2,4\}$가 edge들을 모두 cover(어떤 edge C의 한 vertex에 연해 있음) 하고 있고 증명은 생략하지만 König-Egervary 정리에 의하면 matching의 최대값이 이러한 cover의 최소값과 같기 때문이다[25].



Bipartite graph에 대하여 왼쪽에 있는 vertex들을 행, 오른쪽에 있는 vertex들을 열로 하는 (0,1) 행렬 $A=(a_{ij})$를 ij가 edge이면 $a_{ij}=1$, 아니면 $a_{ij}=0$과 같이 정의하면 위의 graph에 대한 A는

$$A=\begin{bmatrix}1&0&0&0&0\\1&1&1&0&0\\1&0&0&0&0\\1&1&1&1&1\end{bmatrix}$$

이 된다. 그리고 König-Egervary 정리는, 모든 $(0,1)$ 행렬에 대하여, 어느 행이나 열에도 두 개 이상이 있지 않는 1들의 최대 갯수는 1들을 모두 포함하는 행과 열들의 최소 갯수와 같다는 것이다. 이 정리를 쓰면 다음 Birkhoff-von Neumann 정리를 쉽게 증명할 수 있다[25]. 즉 $n\times n$ 행렬 A가 doubly stochastic:

$$a_{ij}\geqq 0,\ \sum_{i=1}^{n} a_{ij}=1\ \forall j,\ \sum_{j=1}^{n} a_{ij}\ \forall i$$

이면 A는 permutation 행렬들의 convex combination으로 나타낼 수 있다. 예를 들면

$$\begin{bmatrix} 0 & 2/3 & 1/3 \\ 5/6 & 0 & 1/6 \\ 1/6 & 1/3 & 1/2 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \frac{1}{3}\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} + \frac{1}{6}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Assignment 문제

Bipartite matching 문제 중에서 다음과 같은 특수한 경우를 보자. Bipartite graph $G=(V,E)$, $V=A\cup B$에서 $|A|=|B|=n$이고 A의 모든 vertex와 B의 모든 vertex를 연결하는 edge들이 있어서 $|E|=n^2$일 때 이 graph는 complete라 하고 $K_{n,n}$으로 나타내면 $K_{n,n}$에는 크기가 n인 matching이 $n!$가지 있다. 이는 모든 edge ij에 무게(또는 비용, 이익) c_{ij}가 주어져 있다고 하고 $n!$가지의 matching 중에서 무게의 합이 최소인 것을 찾는 문제를 assignment 문제라 한다. c_{ij}들과 matching을 행렬 $C=(c_{ij})$, $X=(x_{ij})$로 나타내면 문제는 주어진 $n\times n$ 행렬 C에 대하여 $\sum\sum c_{ij}x_{ij}$가 최소인 permutation 행렬 X를 구하는 것이다. 다음과 같은 경우에

$$C=\begin{bmatrix} 2 & 0 & 0 & 1 \\ 5 & 2 & 4 & 4 \\ 3 & 2 & 5 & 3 \\ 3 & 0 & 1 & 0 \end{bmatrix} \quad X=\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\bar{C}=\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 3 & 1 \\ 2 & 1 & 1 & 0 \end{bmatrix}$$

$\sum\sum c_{ij}x_{ij}=5$이고 X가 최선의 답이다. 그 이유는 다음과 같이 설명할 수 있다. 임의로 주어진 vector $u=(u_i)$, $v=(v_j)\in R^n$에 대하여

$$\bar{c}_{ij}=c_{ij}-u_i-v_j$$

라 하면

$$\sum c_{ij}x_{ij}=\sum\sum \bar{c}_{ij}x_{ij}+\sum u_i+\sum v_j$$

가 되어 $\sum\sum c_{ij}x_{ij}$를 최소로 하는 X와 $\sum\sum \bar{c}_{ij}x_{ij}$를 최소로 하는 X는 같은 것이 다. 위에서 $u=(-1,2,1,-1)$, $v=(2,0,1,1)$라 하면 $\bar{C}=(\bar{c}_{ij})$가 위 오른쪽 행렬과 같이 되고 X는 $\bar{C}$에 대하여 최선이므로 C에 대해서도 최선이다. 이 assignment 문제는 LP문제

$$\min \sum_i \sum_j c_{ij}x_{ij} : \sum_j x_{ij}=1,\ \sum_i x_{ij}=1,\ x_{ij}\geqq 0$$

와 "동치"가 되고 이 LP의 dual 변수들이 u, v에 해당한다. 따라서 최선의 u, v는 LP문제에 관한 algorithm이나 여기서 설명을 피했지만 network flow의 방법 등을 써서 polynomial time 이내에 효과적으로 구할 수 있다[3, 13, 20, 25].

이 assignment 문제에 관련하여 다음과 같은 재미있는 기하학적 성질들이 알려져 있다. 집합 P를

$$P=\{(x_{ij})\in R^{n\times n} \mid \forall i,j;\ \sum_j x_{ij}=1,\ \sum_i x_{ij}=1,\ x_{ij}\geqq 0\}$$

이라 하면 P는 $(n-1)^2$ 차원 다면체이고 $n!$개의 꼭지점이 있다. 여기서 중요한 점은 P를 정의하는 식의 갯수에 비하여 꼭지점의 수가 엄청나게 많을 수 있다는 사실이다. 다면체 P의 꼭지점과 모서리만을 생각할 때 얻어지는 graph $G(P)$라 하면 $G(P)$는 Hamiltonian circuit를 포함하고 어떤 두 꼭지점에 대해서나 그 두 점을 연결한 모서리가 둘 또는 하나인 길이 있어 $G(P)$의 diameter가 2 밖에 안된다[4].

Bipartite matching의 여러 변형을 생각할 수 있겠는데 그 중 하나는 다음과 같은 것이다. 집합 A와 B를 각각 남자와 여자들, 사람들과 책 등의 집합이라 하고 각자가 자기의 우선순위를 기록하여 제출했을 때 합리적인 matching을 찾는 것이다[15].

7. 결 언

이상에서 몇 가지 예제들을 통하여 mathematical programming에서 다루는 내용의 일부를 소개하였고 필자가 강조하고 싶었던 것은 algorithm의 중요성이다. 구체적 algorithm을 별로 기술하지는 않았는데 일반적으로 algorithm의 기술과 해석은 적어도 처음에는 다소 지루하고 난해한 편이다. 서언에서 언급한 것처럼 mathematical programming은 전통 수학 밖에서 더욱 활발한 편이고 수학적 모델 안에서 아름다운 질서와 구조를 찾는 철학적 미는 부족한 편이지만 여러 가지 실제적인 문제에 구체적인 풀이 방법을 제공하려 노력하고 또한 전통 수학 자체에도 그 내용을 더욱 넓고 깊게 하며 어떤 의미에서는 잊혀졌던 한 부분을 다시 찾아 그의 발전을 가속화하고 있다고 볼 수 있다.

참 고 문 헌

1. Aho, A.V., JF Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).
2. Anstreicher, K.M., *Analysis of a Modified Karmarkar Algorithm for Linear Programming*, 12th International Symposium on Mathematical Programming, Boston (1985).
3. Balinski, M.L., *Signature Methods for the Assignment Problem*, Research Report, Ecole Polytechnique, (1982).
4. Balinski, M.L. and A. Russakoff, *On the Assignment Polytope*, SIAM Review, 16-4 (1974), 516~525.
5. Beale, E.M.L., *Cycing in the Dual Simplex Algorithm*, Naval Res. Log. Q. 2-4 (1955), 269~276.
6. Bland, R.G., *New Finite Pivoting Rules for the Simplex Method*, Math. O. R., 2-2 (1977), 103~107.
7. Bland, R.G., D. Goldfarb and M.J. Todd, *The Ellipsoid Method—A Survey*, O.R., 29-6 (1981), 1039-1091.
8. Burrell, B.P. and M.J. Todd, *An Extension of Karmarkar's Algorithm to Linear Programming Using Dual Variables*, Tech. Report no. 648, School of OR/IE, Cornell University (1985).
9. Cunningham, W.H., *Theoretical Properties of the Network Simplex Method*, Math. O.R., 4-2 (1979), 196~208.
10. Dantzig, G.B., *Programming in a Linear Structure*, Comptroller, USAF, Washington, D.C. (1948).
11. Dantzig, G.B., *Reminiscences about the Origin of Linear Programming*, O.R. Letters, 1-2 (1982), 43~48.
12. Edmonds, J., *Paths, Trees and Flowers*, Canad. J. Math., 17 (1965), 449~467.
13. Ford, L.R. and D.R. Fulkerson, *Flows and Networks*, Princeton (1962).
14. Gacs, P. and L. Lovasz, *Khachian's Algorithm for Linear Programming*. Math. Prog. Study, 14 (1981), 61~68.
15. Gále, D. and M. Sotomayor, *Ms. Machiavelli and the Stable Matching Problem*, AMS Monthly 92-4 (1985), 261~268.
16. Garey, M.R. and D.S. Johnson, *Computers and Inracterbility* Freeman and Company (1979).
17. Garfinkel, R.S. and G.L. Nemhauser, *Integer Programming*, Wiley & Sons (1972).
18. Golumbic, M.C., *Algorithmic Graph Theory*, Academic Press (1980).
19. Hoffman, A.J., *Cycling in the Simplex Algorithm*, Nat. Bureau of Standard, Report no. 2974 (1953).
20. Hung, M.S., *A Polynomial Simplex Method for the Assignment Problem*, O.R. 31-3 (1983), 595~600.
21. Karmarkar, M.K., *A New Polynomial Time Algorithm for Linear Programm-*

ing, Combinatoria. 4-4 (1984), 373∼395.

22. Karmarkar, M.K., ***Further Developments in the New Polynomial Time Algorithm for Linear Programming***, 12th international Symposium on Mathematical Programming, Boston (1985).
23. Khachian, L.G., ***A Polynomial Algorithm for Linear Programming***, Soviet Math. Dokl. 244 (1979), 1093∼1096.
24. Knuth, D.E., ***The Art of Computer Programming***, vol 3, Addison-Wesley (1982).
25. Lawler, E.L., ***Combinatorial Optimization: Networks and Matroids***, Holt, Rinehart and Winston (1976).
26. Lovász, L., ***Algorithms: A New Framework for Mathematics***, Lecture at Cornel University (1983).
27. Pritchard, P., ***Explaining the Whee Sieve***, Acta Informatica, 17 (1982), 477∼485.
28. Rose, D.J. and R.E. Tarzan, ***Algorithm Aspects of Vertex Elimination on D rected Graphs***, SIAM Appl. Math., (1978), 176∼197.
29. Tarzan, R.E., ***A Simple Version of Ka zanov's Block Flow Algorithm***, Bell L Research Report (1982).
30. Williams, H.C., ***Primality Testing on Computer***, ARS Combinatoria, 5 (1978) 127∼185.