Linear Programming

Outline

- Introduction
  - A diet problem
- History of linear programming
- Applications
- The Network Flow Problems
- Network Algorithms
- Outline of Integer Programming

Introduction

- A Diet Problem

  eg: Polly wonders how much money she must spend on food in order to get all the energy (2,000 kcal), protein (50 g), and calcium (800 mg) that she needs every day. She choose six foods that seem to be cheap sources of the nutrients:

Food	Serving size	Energy (kcal)	Protein (g)	Calcium (mg)	Price per serving (c)
Oatmeal	28 g	110	4	2	3
Chicken	100 g	205	32	12	24
Eggs	2 large	160	13	54	13
Whole Milk	237 cc	160	8	285	9
Cherry pie	170 g	420	4	22	20
Pork with beans	260 g	260	14	80	19

Introduction

- Servings-per-day limits on all six foods:

Oatmeal	at most 4 servings per day
Chicken	at most 3 servings per day
Eggs	at most 2 servings per day
Milk	at most 8 servings per day
Cherry pie	at most 2 servings per day
Pork with beans	at most 2 servings per day

- Now there are so many combinations seem promising that one could go on and on, looking for the best one. Trial and error is not particularly helpful here.

Introduction

- A new way to express this—using inequalities:

minimize $3x_1 + 24x_2 + 13x_3 + 9x_4 + 20x_5 + 19x_6$

subject to

$$0 \le x_1 \le 4$$
$$0 \le x_2 \le 3$$
$$0 \le x_3 \le 2$$
$$0 \le x_4 \le 8$$
$$0 \le x_5 \le 2$$
$$0 \le x_6 \le 2$$

$$110x_1 + 205x_2 + 160x_3 + 160x_4 + 420x_5 + 260x_6 \ge 1{,}000$$
$$4x_1 + 32x_2 + 13x_3 + 8x_4 + 4x_5 + 14x_6 \ge 55$$
$$2x_1 + 12x_2 + 54x_3 + 285x_4 + 22x_5 + 80x_6 \ge 800$$

Introduction

- Problems of this kind are called “linear programming problems” or “LP problems” for short; linear programming is the branch of applied mathematics concerned with these problems.
- A *linear programming problem* is the problem of maximizing (or minimizing) a linear function subject to a finite number of linear constraints.
- Standard form:

maximize $\sum_{j=1}^{n} c_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j \le b_i \quad (i = 1, 2, \ldots, m)$

$x_j \ge 0 \quad (j = 1, 2, \ldots, n)$

History of Linear Programming

- It started in 1947 when G.B.Dantzig design the "simplex method" for solving linear programming formulations of U.S. Air Force planning problems.
- It soon became clear that a surprisingly wide range of apparently unrelated problems in production management could be stated in linear programming terms and solved by the simplex method.
- Later, it was used to solve problems of management. It's algorithm can also used to network flow problems.

History of Linear Programming

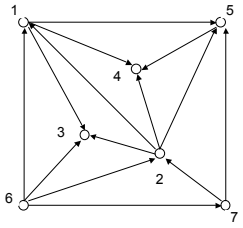
- On Oct.14th,1975, the Royal Sweden Academy of Science awarded the Nobel Prize in economic science to L.V.Kantorovich and T.C.Koopmans "for their contributions to the theory of optimum allocation of resources"
- The breakthrough in looking for a theoretically satisfactory algorithm to solve LP problems came in 1979 when L.G.Khachian published a description of such an algorithm.

Applications

- Efficient allocation of scarce resources
  - diet problem
- Scheduling production and inventory
  - multistage scheduling problems
- The cutting-stock problem
  - find a way to cut paper or textiles rolls by complicated summary of orders
- Approximating data by linear functions
  - find approximate solutions to possibly unsolvable systems of linear equations

The Network Flow Problems

- The network flow problem concerns finding the cheapest way to ship prescribed amounts of a commodity such as oranges from specified origins to specified destinations through a concrete transportation network
  - Nodes
    - Sinks
    - Sources
    - intermediate
  - arcs: an order pair (i, j) of distinct nodes i and j
  - Assumption: The total supply equals the total demand

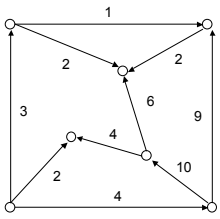


The Network Flow Problems

- It is convenient to write each positive next to the corresponding arc ij and to completely ignore arcs ij with 0

x_{ij} = x_{ij}

- Some requirements



The Network Flow Problems

Ax = b, x >= 0

with

$$A = \begin{bmatrix} -1 & -1 & -1 & 1 & & & & 1 & & & & \\ & & & -1 & -1 & -1 & -1 & & 1 & & 1 & \\ 1 & & & & 1 & & & & & 1 & & \\ & 1 & & & & 1 & 1 & & & & & \\ & & 1 & & & & 1 & -1 & & & & 1 \\ & & & & & & & & -1 & -1 & -1 & -1 \\ & & & & & & & & & & 1 & -1 & -1 \end{bmatrix} \quad x = \begin{bmatrix} x_{13} \\ x_{14} \\ x_{15} \\ x_{21} \\ x_{23} \\ x_{24} \\ x_{25} \\ x_{54} \\ x_{61} \\ x_{62} \\ x_{63} \\ x_{67} \\ x_{72} \\ x_{75} \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ 0 \\ 6 \\ 10 \\ 8 \\ -9 \\ -15 \end{bmatrix}$$

The Network Flow Problems

- Matrix A is called the incidence matrix of our network
- Each component of the demand vector b specifies the demand at node i, with supplies interpreted as negative demands b_i
- Denote the cost of shipping a unit amount along ij by c_{ij}

$$c = [c_{13} \quad c_{14} \quad c_{15} \quad c_{21} \quad c_{23} \quad c_{24} \quad c_{25} \quad c_{54} \quad c_{61} \quad c_{62} \quad c_{63} \quad c_{67} \quad c_{72} \quad c_{75}]$$
$$= [53 \quad 18 \quad 29 \quad 8 \quad 60 \quad 28 \quad 37 \quad 5 \quad 44 \quad 38 \quad 98 \quad 14 \quad 23 \quad 59]$$

- Now the total cost of a schedule x equals

$$cx = \sum c_{ij} x_{ij}$$

The Network Flow Problems

- So the network flow problems is any problem:

minimize cx subject to $Ax = b, x \geq 0$

such that A is the n ×m incidence matrix of some network and such that

this requirement stipulates assumption: The total supply equals the total demand $\sum_{i=1}^{n} b_i = 0$

Network Algorithms

- The problem of finding the minimum time necessary to complete the project is solved by finding the length of a longest path from the begin node to the end node in the graph
- Longest path algorithm
  - Two functions
    - l (i): denote the longest path length from node i to node I
    - d (i): denote the predecessor node to node i in a longest path
  - The nodes will be scanned in topological order: 1,2,…,k-1, longest paths to nodes 1,…,i+1 are known after node i has been scanned

Network Algorithms

- *Initialization of l and p*

  Put l (i)=0,i>=1

  Put p (i)=*(empty), i>=1

  denote the node to be scanned by u, begin by initializing u to 1
- *Initialization of Node to Be scanned*

  Put u = 1
- *Scanning Step*

  For each route [u, j],

  If l (j)<l (u) + len [u, j],then

  put l (j) = l (u) + len [u, j] and put p (j)=u
- *Algorithm*

  Apply the scanning step

  if u=k-1, then stop;

  otherwise, put u=u+1, and apply the scanning step

Network Algorithms

(Initial Node scanned(only changed values of l, p are recorded)

Node	value	1	2	3	4	5	6	7	8	9	10	11	12
1	0,*												
2	0,*	3,1											
3	0,*		7,2										
4	0,*		9,3										
5	0,*		10,3										
6	0,*		12,3										
7	0,*			12,4		15,6							
8	0,*				14,5								
9	0,*							20,8					
10	0,*						22,7						
11	0,*									25,9	27,10		
12	0,*			11,4		14,6							
13	0,*											27,11	

Network Algorithms

- Shortest path algorithm
  - similar to the longest path algorithm
  - differs in that the whole scanning order is not known when the algorithm begins: the node to be scanned at step n+1 is determined during step n
  - step 1, the node 1 scanned first: to find a shortest link-route beginning at node 1; this link-route is a shortest path to the other end u.
  - step 2, node u is scanned to find another node to which a shortest path is known
  - continuing step by step, shortest paths are determined one node at a step

Network Algorithms

- Two functions
  - d (i): denote the minimal path distance from node 1 to node i
  - p (i): denote the predecessor of i along the minimal path selected by the algorithm
- List S: the nodes to which a minimal path is known, initially, S = {1}
- Intermediate values of d(i) and p(i) are estimates based on shortest paths to nodes in S followed by a route-link from S to node i
- *Initialization*

  Put u=1

  Put S={1}

  Put d(1)=0 and d(i)=M, i>1, where M is a big (enough) number

  Put p(i)=*(empty), i>=1: initially, no node has a predecessor

Network Algorithms

- *Scanning Step*

  Put the links (u, v) with v not in S in a list and go through the list once, applying the following "if" statement to each link in the list:

  if d(u) + len(u,v)<d(v), then

  put d(v) = d(u) + len(u, v) and

  put p(v) = u

  Put u = arg min{d(v); v is not in S}

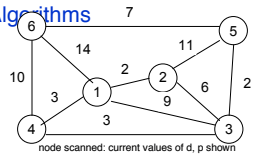
  Put S = S + {u}; us is added to the end of the list
- *Algorithm*

  Apply the scanning step. (node 1 is scanned first)

  If S contains all the nodes, then stop;

  otherwise, apply the scanning step

Network Algorithms



Node	initial values of d,p	node scanned: current values of d, p shown 1	2	4	3	5
1	0,*					
2	M,*	2,1				
3	M,*	9,1	8,2	6,4		
4	M,*	3,1	3,1			
5	M,*	M,*	13,2	13,2	8,3	
6	M,*	14,1	14,1	13,4	13,4	13,4

Network Algorithms

- After scanning node

  1, S = {1,2}

  2, S = {1,2,4}

  4, S = {1,2,4,3}

  3, S = {1,2,4,3,5}

  5, S = {1,2,4,3,5,6}
- Shortest path solution

Node	Distance to node	shortest path to node
2	2	[1,2]
4	3	[1,4]
3	6	[1,4,3]
5	8	[1,4,3,5]
6	13	[1,4,6]

Network Algorithms

- Minimum spanning tree algorithm
  - The strategy is to begin somewhere and pave short links first.
  - Start at node 1 and pave as little as you can to reach another node.
  - Then pave as little as you can to reach another node
  - Repeat this procedure until all nodes are accessible by pavement
  - Two sets
    - S: denotes the set of nodes currently accessible from node 1 by pavement
    - T: denotes the set of nodes not currently accessible by pavement

Network Algorithms

- *Initialization*

  Put S = {1}

  Put T = {2,3,…,k}
- *Iteration*

  Put d = min { len(u, v); u is in S, v is in T }

  Put y = arg min { v; u is in S, v is in T, len(u, v) = d }

  Put x = arg min { u; u is in S, len(u, y) = d }

  Pave (x, y)

  Put S = S + { y }: add y to S

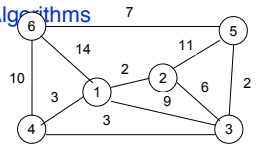
  Put T = T – { y }: delete y from T
- *Algorithm*

  If S = {1,2,…,k}, then stop

  (all nodes are accessible by pavement);

  otherwise, do an iteration

Network Algorithms



Tabular solution

Iteration	d	y	x	path paved
1	2	2	1	(1,2)
2	3	4	1	(1,4)
3	3	3	4	(4,3)
4	2	5	3	(3,5)
5	7	6	5	(5,6)

Network Algorithms

- Other network flow problems
  - Upper-bounded network flow problems
  - Maximum flows through networks
  - The primal-dual method
  - …

Outline of Integer Programming

- When formulating LP's we often found that, certain variables should have been regarded as taking integer values but, for the sake of convenience, we let them take fractional values reasoning that the variables were likely to be so large that any fractional part could be neglected. Whilst this is acceptable in some situations, in many cases it is not, and in such cases we must find a numeric solution in which the variables take integer values.
- Problems in which this is the case are called *integer programs* (*IP's*) and the subject of solving such programs is called *integer programming* (also referred to by the initials *IP*).
- IP's occur frequently because many decisions are essentially discrete (such as yes/no, go/no-go) in that one (or more) options must be chosen from a finite set of alternatives. Topics like: capital budgeting

References

- [1] Vasek Chvatal. Linear Programming. ISBN 0-7167-1195-8, 1983
- [2] Richard B.Darst. Colorado State University. Introduction to Linear Programming—Applications and Extensions. ISBN 0-8247-8383-2,1991
- [3] http://people.brunel.ac.uk/~mastjjb/jeb/or/ip.html