Discrete Applied Mathematics 13 (1986) 249–257
North-Holland

THE FAMILY CONSTRAINED NETWORK PROBLEM

John MOTE
Department of General Business, The University of Texas, Austin, TX 78712, USA

Received 18 February 1985

An extension of the classical fixed charge transportation problem is developed that allows a wide variety of practical production, distribution, and inventory planning models to be addressed. Computational results are presented for problems with up to one thousand network constraints and five thousand network variables.

1. Introduction

The Fixed Charge Transportation Problem has attracted widespread attention since it was first proposed by Hirsch and Dantzig in 1954 [8]. The closely related Facility Location Problem [5] has also been the subject of considerable research effort in recent years. Both of these problem classes involve highly structured integer and network components. Unfortunately, these extreme structural characteristics are not always present in many practical applications. In this note a related problem class is proposed that (1) completely relaxes the topological assumptions regarding the structure of the underlying network, and (2) extends the fixed charge concept to a more realistic and useful form. This proposed problem class will be called the Family Constrained Network Problem.

Virtually all formulations of the fixed charge transportation and facility location problems assume a bipartite transportation-type network component. A notable exception is the work by Rardin and Unger [10] which allows a more general network component, but maintains the simple integer structure of the fixed charge transportation problem.

While admittedly an interesting problem class, very few practical applications can be modeled with a simple bipartite network structure. Most network based models possess the more general structure associated with the capacitated transshipment problem. The family constrained network problem will be developed without any assumptions regarding the underlying structure of its network component. Since any capacitated pure transshipment model can serve as the basic framework, the family constrained network model can be used to address a wide range of production, distribution and inventory planning problems.

The integer, or binary, components of both the fixed charge transportation problem and the facility location problem are also highly structured. In the fixed charge transportation problem each binary variable controls the flow on a single network

arc, and in the facility location problem each binary variable controls the flow on the arcs associated with a single network node. The family constrained network problem dramatically extends this structure. In this new problem class, a single binary variable will be used to control a set of otherwise unrelated arcs. That is, no assumptions are made regarding the topological characteristics of the network arcs associated with a single binary variable. The set of arcs associated with a single binary variable will be referred to as a family and the binary variable will be called a family variable. A fixed charge is incurred if any member arc of a family is active. This provides a logical and powerful estension of the fixed charge concept.

The family constrained network problem has many practical application areas, particularly in the realm of multi-period and multi-commodity models. As a simple example consider the multi-period production, distribution and inventory planning problem [7]. Obtaining additional warehouse capacity, either through construction or lease, is a long-term decision. A substantial fixed charge is incurred whether the inventory capacity is used in a single or in all time periods addressed by the model. In addition, a variable cost is often based on the actual level of usage during each period. The classical fixed charge transportation and facility location models cannot adequately address this type of multi-period decision. It presents no problem, however, when formulated as a family constrained network problem.

In the following section, the family constrained network problem will be formally stated. This is followed in Section 3 by a discussion of standard penalties for a branch-and-bound implementation. The development and testing of a FORTRAN code for solving large scale family constrained network problems will be described in Section 4, and some useful model generalizations will be given in the final section.

2. Problem statement

The family constrained network problem is composed of two types of variables; the binary family variables, and the integer network arc flow variables. The family variables will be denoted by y_h and the network variables by x_k. The family constrained network problem can be formally stated as follows:

$$\text{Minimize} \quad \sum_{h \in F} f_h y_h + \sum_{k \in A} c_k x_k, \tag{1}$$

$$\text{Subject to} \quad -\sum_{k \in A(i)} x_k + \sum_{k \in B(i)} x_k = b_i \quad \text{for } i \in N, \tag{2}$$

$$M_h y_h - \sum_{k \in F(h)} x_k \geq 0 \quad \text{for } h \in F, \tag{3}$$

$$0 \leq y_h \leq 1 \text{ and integer} \quad \text{for } h \in F, \tag{4}$$

$$0 \leq x_k \leq U_k \text{ and integer} \quad \text{for } k \in A. \tag{5}$$

In this formulation of the problem, F is an index set of family variables; A is an index set of network variables, and N is an index set of network constraints or

nodes. In the statement of the network constraints, (2), two additional index sets are introduced. $A(i)$ is an index set of the arcs that originate at node i, and $B(i)$ is an index set of the arcs that terminate at node i. These two sets are often referred to as the node's forward and reverse stars, respectively [6].

The index set $F(h)$ in the statement of the family relationship constraints, (3), indicates the arcs associated with family variable h. These arcs will be referred to as members of the family. Note that a single arc may be a member of any number of families.

In the objective function, (1), f_h is the positive fixed charge associated with family h and c_k is the per unit cost associated with arc k. In the network constraints, (2), b_i is the net exogenous demand at node i. The upper bound for arc k is indicated in (5) by U_k. Both b_i and U_k are assumed to be integer so that the network integrality conditions, (5), become natural.

In the family relationship constraints, (3), M_h is simply a large positive constant that serves to make the constraint strictly binding when the associated family variable is fixed to zero and strictly redundant if it is fixed to one. An obvious choice for this constant is the sum of the upper bounds of the individual arcs that make up the family. Since the member arcs cannot exceed their own bounds, their sum cannot exceed the sum of their bounds.

In order to simplify later notation, an additional index set will be introduced. $M(k)$ will be defined as the set of all families in which arc k is a member. So,

$$M(k) = \{h \in F \mid k \in F(k)\}.$$

This, in a sense, serves as an inverse of the $F(h)$ index set.

3. Penalty generation

One of the fundamental ingredients of a successful branch-and-bound implementation for any integer linear programming problem is the efficient use of penalties to enhance the fathoming of subproblems. Numerous penalties have been proposed and tested for the fixed charge transportation problem, the facility location problem, and other classes of related integer programming problems. In this section it will be shown how many of these penalties can be easily generalized for the family constrained network problem. In order to set the framework for the development of these penalties, a short presentation of the standard branch-and-bound algorithm will be given as well as a brief discussion of the linear programming relaxation of the family constrained network problem.

An overview of the branch-and-bound algorithm for the family constrained network problem is given below. In this presentation Z^I is used to denote the objective function value of the incumbent all-integer solution, Z^p is the objective function of the relaxed pth subproblem, and ZL^p is a lower bound on the optimal all-integer solution to the pth subproblem. ZL^p is generally based on the value of Z^p and an estimated penality to convert a non-integer solution into an all-integer solution.

Branch-and-Bound Algorithm

Step 1. [INITIALIZATION] Construct the initial branch-and-bound search tree. Set $p=1$ and $Z^I=\infty$.

Step 2. [RELAXED OPTIMIZATION] Solve subproblem p. If feasible but not all integer, go to 3. If feasible and all integer, go to 4. If infeasible, to go 5.

Step 3. [BRANCH SELECTION] Determine ZL^p. If $ZL^p \geq Z^I$, go to 5. Otherwise, select any non-integer family variable and fix it to either zero or one. Set $p=p+1$ and go to 2.

Step 4. [INCUMBENT UPDATE] If $Z^p \geq Z^I$, go to 5. Otherwise, set $Z^I=Z^p$, $y^I=y^p$, and $x^I=x^p$.

Step 5. [BACKTRACK] If all subproblems of the branch-and-bound search tree have been fathomed, then stop. Otherwise, select one, set $p=p+1$, and go to 2.

In Step 2 of the branch-and-bound algorithm, a subproblem must be solved. This subproblem is obtained by fixing some of the binary family variables to either zero or one while relaxing all other family variables to be non-integer between zero and one. This classical relaxation yields a capacitated pure transshipment problem which can be easily solved by any of the efficient algorithms designed for this broad class of network problems.

The discussion of this important subproblem is aided by introducing a three-way partitioning of the set of family variables. Let the partitioning for a given subproblem be defined by the following subsets:

$$F^0 = \{h \in F \mid y_h \text{ fixed to zero}\},$$
$$F^1 = \{h \in F \mid y_h \text{ fixed to one}\},$$
$$F^r = \{h \in F \mid y_h \text{ not fixed}\}.$$

With this partitioning of the family variables, the resulting capacitated pure transshipment problem can be stated as follows.

$$\text{Minimize} \quad \sum_{h \in F^1} f_h + \sum_{k \in A} \left(c_k + \sum_{h \in M(k)} a_n \right) x_k, \tag{6}$$

$$\text{Subject to} \quad -\sum_{k \in A(i)} x_k + \sum_{k \in B(i)} x_k = b_i \quad \text{for } i \in N, \tag{7}$$

$$0 \leq x_k \leq W_k \text{ and integer} \quad \text{for } k \in A. \tag{8}$$

In this formulation, the family variables have been implicitly removed from consideration. Due to the well-known unimodularity characteristic of this class of network problems, the integrality conditions on the network variables, (8), can also be ignored if an extreme point solution method is employed to solve the network subproblems.

The first term in the objective function, (6), is a constant term associated with

the family variables that have been fixed to one. The second term reflects the relaxed per unit arc costs. For a given arc, this consists of the original arc cost, c_k, plus the allocation fixed costs of the families to which it belongs. The allocated fixed costs, a_h, are defined as follows:

$$a_h = \begin{cases} \infty & \text{if } h \in F^0, \\ 0 & \text{if } h \in F^1, \\ f_h/M_h & \text{if } h \in F^r. \end{cases} \tag{9}$$

The upper bounds, (8), of this capacitated pure transshipment problem reflect both the original upper bounds, (5), and the status of the family variables. The new upper bounds are given by:

$$W_k = \begin{cases} 0 & \text{if } h \in F^0 \text{ for some } h \in M(k), \\ U_k & \text{otherwise.} \end{cases}$$

Given a feasible solution to problem (6)–(8), various penalties can be calculated to aid the branch-and-bound search process. Foremost among these are the classic up and down penalties proposed by Driebeek [4]. They reflect a conservative one-step-ahead estimate of the degradation to the objective function when a free binary variable is fixed to either one or zero. Both penalties are based on measuring the impact of a single dual simplex iteration. The up penalty results from a dual simplex iteration to drive the slack variable associated with the family relationship constraints, (3), from the basis, while the down penalty is based on a dual simplex iteration to drive the binary variable from the basis. Computationally, these iterations can be carried out by using a method based on the poly-ω technique of Charnes and Cooper [3].

The development of the penalties for the family constrained network problem can best be presented by considering the simple down penalty. Since fixing a family variable to zero is equivalent to fixing each of its member arcs to zero, the down penalty for family variable y_h is simply equal to the largest standard down penalty associated with the family member arcs. This useful observation shows that many of the penalty methods developed for the classical fixed charge problem can be readily extended to this broader class of problems.

4. Branch-and-bound implementation

A FORTRAN implementation of the branch-and-bound algorithm was developed to solve the family constrained network problem. This implementation makes use of the commercially available optimizer ARCNET [1] to solve the numerous capacitated pure transshipment subproblems. ARCNET, a highly specialized implementation of the upper bounded primal simplex algorithm, has recently been used in

another application to solve capacitated pure transshipment problems with 10,000 constraints and 750,000 variables [9].

ARCNET, as well as the branch-and-bound routines that were developed for this study, make use of various specialized list functions to efficiently store the necessary problem description and working parameters [2]. The basic network optimizer makes use of the four arc length lists and seven node length lists. In addition to these, the branch-and-bound component for the family constrained network problem uses nine more special list functions. Seven of these are family length lists, of which four are used to store problem data, two are used to store the branch-and-bound search tree, and one is used to store the incumbent solution. The two additional lists used by the branch-and-bound component of the optimization system are used in a sparse storage representation scheme for the family relationship constraints, (3). Their length is given by the number of non-zero coefficients in these constraints. Taken as a whole, the total storage requirements (integer words of computer memory) for the branch-and-bound optimizer for the family constrained network problem is

$$4\,|A| + 7\,|N| + 7\,|F| + 2 \sum_{h \in F} |F(h)|$$

where $|X|$ denotes the cardinality of the index set X.

A set of sixteen test problems were used for benchmark purposes during the development of the optimization system. Table 1 provides a basic description of the test problems. No explicit attempts were made to specialize either the network optimizer or the general branch-and-bound process to exploit any underlying network structure of the problems.

Table 1. Test problems

Problem	Network constraints	Network variables	Family variables	Average family size
1	75	150	5	8.0
2	100	225	5	5.4
3	100	225	5	8.8
4	100	275	10	4.1
5	100	275	10	10.1
6	200	450	10	11.2
7	200	450	20	10.2
8	200	950	10	8.8
9	200	950	10	24.9
10	200	1540	10	5.5
11	500	1350	10	9.1
12	500	1350	10	24.5
13	500	2500	10	6.1
14	500	2500	20	6.3
15	1000	5000	20	6.4
16	1000	4800	20	16.5

Numerous combinations of penalties and branching rules were studied in the development of the optimization system for the family constrained network problem. Only three implementations will be presented in any detail. The first implementation serves merely as a reference point. It makes no use of penalties, so fathoming is accomplished only when an infeasible or all-integer solution is obtained. It employed a branching strategy based on a simple LIFO ordering of the family variables.

The other two strategies reported here make use of various penalties to aid the fathoming process. Not surprisingly, it was found that the overhead associated with computing the penalties was more than offset by the improvement in the search process. The same set of penalties were used in the second and third implementations for the family constrained network problem. The two implementations differ only in their choice of branching variables.

In the second implementation, the free family variable with the largest up or down penalty was selected as the branching variable. After selection, the family variable was always fixed to zero. Other strategies, such as always fixing to one or fixing in the direction of the largest or smallest penalty, were also tested to some extent, but the choice of fixing to zero tended to dominate the others.

The third implementation, unlike the first two reported, attempted to fix multiple family variables before resolving the network problem. This implementation used penalties to help fathom subproblems but did not use them in the branching variable selection process. Instead, the optimal solution values of the free family variables were used to select their branching direction. Specifically, all free variables with an activity level between zero and α were fixed to zero, and all between β and one were fixed to one. When no free variable met this selection criteria, the variable closest to either zero or one was fixed to its nearest bound. While other parameter settings were examined, the testing reported here is based on values of $\alpha = 0.25$ and $\beta = 0.75$.

A brief computational comparison of the three implementations of the branch-and-bound optimization system for the family constrained network problem is given in Table 2. This table shows the actual number of network subproblems examined as well as the total solution time for each of the sixteen benchmark problems. This testing was performed on a PRIME 750 minicomputer and the reported timing statistics include all computational effort except input and output.

The value of using the penalties to improve the branch-and-bound search process is clear. Not only did strategies two and three result in fewer network subproblems than the no penalty approach, but the overall solution times were considerably reduced even though the penalty generation process was tedious.

The branching rule of the third implementation was selected as an attempt to fix multiple variables at a time and therefore reduce the total number of network subproblems that had to be considered. It appears from this limited testing that the strategy was successful since it addressed the fewest number of subproblems in twelve of the sixteen cases. In terms of total cpu time, it was the fastest in ten cases. Strategy two was fastest in five cases.

Table 2. Computational results

Problem	Strategy 1		Strategy 2		Strategy 3	
	Subproblems	CPU seconds	Subproblems	CPU seconds	Subproblems	CPU seconds
1	13	1.29	11	1.77	13	2.21
2	46	5.26	8	1.14	7	1.34
3	29	4.77	12	2.36	7	1.87
4	127	13.17	9	1.67	12	2.41
5	250[a]	32.25	21	3.34	14	2.83
6	45	10.27	21	5.73	11	4.01
7	250[a]	35.72	39	6.29	22	5.42
8	83	46.71	25	15.87	12	9.17
9	58	31.69	27	16.24	14	10.99
10	91	38.70	13	9.78	17	13.13
11	27	46.58	20	37.02	31	64.26
12	70	210.82	32	41.74	19	32.40
13	29	50.56	12	32.93	12	36.19
14	143	148.51	51	74.64	23	53.27
15	250[a]	1695.05	250[a]	1805.10	41	465.31
16	250[a]	2910.81	41	482.98	28	467.34

[a] Reached limit before verifying optimality.

5. Final considerations

The FORTRAN optimization system developed for the family constrained network problem is apparently capable of solving fairly large and complex problems in a reasonable period of time. This can be largely attributed to the use of an efficient network optimizer for solving the numerous pure transshipment subproblems [1]. This allowed full exploitation of the specialized structure of the network subproblems.

The concept of family variables is very useful in many network modeling situations. Since it allows a charge to be incurred if any member of a set is active. An extension of the family relationship constraint, (3), has certain advantages. As originally formulated, M_h is a large constant that serves to make (3) redundant if the corresponding family variable is fixed to one. If, however, M_h is given as a true 'family capacity', then the constraint would limit the total activity level of the family member arcs. In some instances, this is a powerful capability. Unfortunately, the resulting subproblems considered during the branch-and-bound search process are no longer simple pure network problems. The optimization system for this extended problem class would need to make use of an efficient linear programming or embedded network/LP optimizer for the subproblems. Many other extensions of the family variable concept are also possible if a full linear programming optimizer is used by the branch-and-bound process.

References

[1] Analysis, Research and Computation, ARCNET User's Guide (1983).
[2] R. Barr, F. Glover and D. Klingman, Enhancements of spanning tree labeling procedures for network optimization, INFOR 17 (1979) 16–34.
[3] A. Charnes and W.W. Cooper, Management Models and Industrial Applications of Linear Programming (Wiley, New York, 1961).
[4] N.J. Driebeek, An algorithm for the solution of mixed integer programming problems, Management Sci. 12 (1966) 576–587.
[5] M.A. Efroymson and T.L. Ray, A branch-bound algorithm for plant location, Oper. Res. 14 (1966) 361–368.
[6] J. Gilsinn and C. Witzgall, A performance comparison of labeling algorithms for calculating shortest path trees, NBS Technical Note 772 (U.S. Department of Commerce, 1973).
[7] F. Glover, G. Jones, D. Karney, D. Klingman and J. Mote, An integrated production, distribution, and inventory planning system, Interfaces 9 (1979) 21–35.
[8] W.M. Hirsch and G.B. Dantzig, The fixed charge problem, Naval Res. Logist. Quart. 15 (1968) 413–424.
[9] D. Klingman and N.V. Phillips, Topological and computational aspects of preemptive multicriteria military personnel assignment problems, Management Sci. 30 (1984) 1362–1375.
[10] R.L. Rardin and V.E. Unger, Solving fixed charge network problems with group theory based penalties, Naval Res. Logist. Quart. 23 (1976) 67–84.