LOCATING FACILITIES WHICH INTERACT: SOME SOLVABLE CASES

Dilip CHHAJED
Department of Business Administration, 350 Commerce West, University of Illinois at Urbana-Champaign, Champaign, IL 61820, USA

and

Timothy J. LOWE
Department of Management Sciences, University of Iowa, Iowa City, IA 52242, USA

Abstract

The network version of the m-median problem with mutual communication (MMMC) is to find the location of m new facilities on a network with n nodes such that the sum of (a) the cost of interaction between the new facilities and n existing facilities on the network, and (b) the cost of interaction between pairs of new facilities is minimized. The existing facilities are located at nodes of the network and the interaction cost between a pair of facilities is a function of the network distance between the facilities. This problem is shown to be equivalent to a graph-theoretic Node Selection Problem (NSP). We show that many other problems can be formulated as an NSP. We then provide a polynomial time algorithm to solve NSP for the case when the flow graph is Halin. Extensions to other graph families are provided.

1. Introduction

The network version of the m-median problem with mutual communication (MMMC) is to find the location of m new facilities on a *transport network* τ with n nodes such that the sum of (a) the cost of interaction between the new facilities and n existing facilities on the network, and (b) the cost of interaction between pairs of new facilities is minimized. The existing facilities are located at nodes of the network and the interaction cost between a pair of facilities is a function of the network distance between the facilities.

The new facilities can be m production plants, each producing some end products as well as several components/by-products which are used by other plants. The existing facilities may be the customer locations or the distribution centers where the customer demand for the product(s) produced by each plant is known. The transport network is the road network whose nodes include the customer points/distribution centers and other points which are candidate sites for the location of new plants.

Another application of MMMC is the location of several new machine centers in a production area. Material movements are made on a transport network (e.g. network of aisles). Each new machine center will send and/or receive material to/from one or more existing machine centers whose locations on the transport network are known. In addition, each new machine will have material flow interaction with some subset of the other new machines. We assume that the existing machines are located at nodes of the transport network. There is no loss of generality here, since as long as each existing machine is on the network, its location can be declared as a node. We consider problems where the set of possible locations on the network for each new facility is finite. We can also declare these locations as nodes of the network.

In the above examples of MMMC (as well as other examples), it is most likely the case that the cost of interaction between certain pairs of facilities will not depend upon the network distance between their locations. This would occur in the above examples if there was no material flow between a pair of facilities. In what follows, we say a pair of facilities *interacts* only if the cost of interaction is a function of the network distance between the facilities.

Most of the literature associated with MMMC deals with the case where the interaction costs are *linear* in network distances. Kolen [13] has shown that the problem is NP-hard when τ is a general network, but is polynomially solvable when τ is a tree. Picard and Ratliff [17] also give a polynomial time algorithm for the problem when τ is a tree. Dearing et al. [7] have shown that the problem is a convex optimization problem for all data choices if and only if τ is a tree. Erkut et al. [10] consider a constrained version of the problem and make use of separation conditions [11] to obtain a mathematical program. The mathematical program is equivalent to the original problem if τ is a tree; otherwise, the solution to the mathematical program provides a lower bound. A computational study of the lower bound vis-à-vis the original problem is given in Erkut et al. [9].

Xu et al. [19] consider the version of MMMC where the transport network τ is not necessarily a tree, but τ does contain two or more blocks (maximal, nonseparable subgraphs of τ). They show that by solving a related problem on a "blocking graph" (which is a tree), information can be obtained which localizes each optimal new facility to some vertex or block of τ. The problem then decomposes into a collection of independent problems, one for each localizing block of τ.

In this paper, we give a polynomial time algorithm for a class of network MMMC problems in which the transport network τ is general and where the interaction costs are general functions of network distances as long as these cost functions are such that node optimality conditions hold, i.e. there is at least one optimal solution in which each new facility is located at a node of the transport network. However, we do require a certain structure with respect to the pairs of *new* facilities that interact.

In what follows in this section, we formulate a problem known as the Node Selection Problem (NSP) and show that MMMC can be represented as an NSP. The

problem transformation, which also appears in Chhajed and Lowe [5], makes use of a graph we call a G-partite graph (to be defined shortly), which captures the essence of the underlying problem. We end the first section of the paper by citing four other problems that can be formulated as an NSP.

1.1. NODE SELECTION PROBLEM

Given a graph $G = (V(G), E(G))$ with node set $V(G)$ and arc set $E(G)$, consider the following *G-partite graph* G^0: Corresponding to each node $u \in V(G)$, we have a *node-family* σ_u in G^0 which contains n_u nodes $\{u_k: k = 1, \ldots, n_u\}$. Two nodes u_k and v_r $(v \neq u)$ are *adjacent* in G^0 if and only if arc $(v, u) \in E(G)$. Arc (u_k, v_r) in G^0 is assigned a weight ω_{kr}^{uv}. Thus, if (v, u) exists in G, nodes of node families u and v form a complete bi-partite subgraph of G^0. Figure 1 gives an example of graph G and a corresponding G-partite graph. Figure 2 shows the weights on the

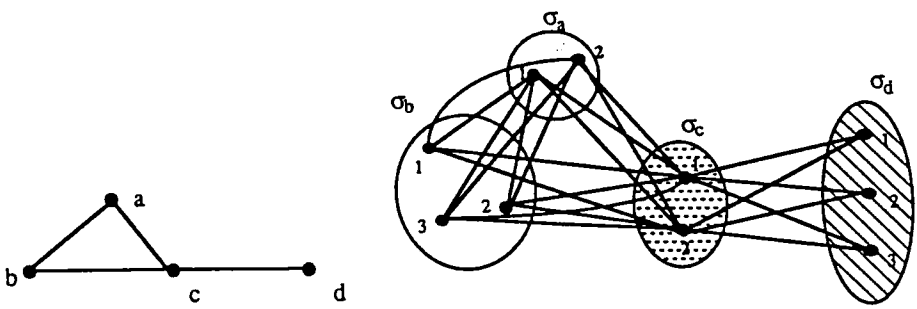


Fig. 1. Graphs G and G^0.

Weights			
c \ d	1	2	3
1	5	4	8
2	7	9	6

Weights		
c \ a	1	2
1	5	7
2	7	10

Weights			
c \ b	1	2	3
1	9	11	14
2	19	6	12

Weights			
a \ b	1	2	3
1	7	4	8
2	4	5	4

Fig. 2. Weights on graph G^0.

arcs in G^0, which are presented in the form of matrices. The entry in row 1 and column 3 of the first of the matrices in fig. 2 is the weight of the arc joining node 1 of σ_c and node 3 of σ_d. Node families σ_u and σ_v are said to be *adjacent*

if and only if every node in σ_u is adjacent with every node in σ_v. We will use the notation (σ_u, σ_v) to denote all arcs between nodes in σ_u and nodes in σ_v. The graph G is also referred to as the *flow-graph*. Given a G-partite graph G^0, let $S(G^0)$ be an induced subgraph of G^0 with one node of each node family and $Z(S(G^0))$ be the sum of the weights on the arcs in $S(G^0)$.

The following Node Selection Problem (NSP) was defined in [5]: Given a graph G and the corresponding G-partite graph G^0 with arc weights ω, find an $S(G^0)$ such that $Z(S(G^0))$ is minimum. We will denote an optimum solution of NSP by $S^*(G^0)$.

The version of the NSP in which there are also weights on nodes in G^0 and $Z(S(G^0))$ includes the node weights as well as arc weights can be easily transformed to a problem with no weights on the nodes. Let $nw(v_r)$ denote the node-weight on node v_r. For those nodes with $nw(v_r) \neq 0$, we identify a node family σ_u, adjacent to σ_v, and set the weights ω_{kr}^{uv} of arc (v_r, u_k) as $\omega_{kr}^{uv} \leftarrow \omega_{kr}^{uv} + nw(v_r)$, $\forall u_k \in \sigma_u$. Finally, we delete all node weights. If node v_r is in $S(G^0)$, $Z(S(G^0))$ will include the arc weight $nw(v_r)$. Thus, an NSP in which there are weights on nodes can be transformed to an equivalent NSP in which there are no weights on nodes.

1.2. MMMC AS NSP

To represent MMMC as an NSP, the flow graph has m nodes (one node for each new facility). For nodes u and v of G, $(u, v) \in E(G)$ if and only if new facilities u and v *interact*, i.e. the cost of interaction between the pair depends upon the distance between them. We then construct a G-partite graph G^0 with m node families, one corresponding to each new facility. The node family for node u of G consists of n_u nodes, one node corresponding to each of the n_u possible locations for new facility u on the transport graph τ. For each new facility u, we select a new facility u^0 such that there is an interaction between new facilities u and u^0 and define a function $\delta(u, u^0) = 1$ and $\delta(u, v) = 0$ for all other new facilities $v \neq u^0$. Note that $\delta(u, v)$ may not be equal to $\delta(v, u)$. The weight on the arc (u_k, v_r) in G^0 is equal to the sum of (a) $\delta(u, v)$ * (interaction costs between new facility u and *all existing* facilities if new facility u is located at node k), (b) $\delta(v, u)$ * (interaction costs between new facility v and all existing facilities if new facility v is located at node r), and (c) interaction cost between new facilities u and v if u is located at node k and v is located at node r.

$S(G^0)$ gives a feasible solution to MMMC with a cost $Z(S(G^0))$. Thus, solving NSP on a G-partite graph as defined in the preceding paragraph provides a solution to the MMMC. In [5], a version of the MMMC in which there is a fixed cost of locating a new facility u at node k is modeled as an NSP.

1.3. ADDITIONAL PROBLEMS AS NSP

We now cite four additional problems that can be posed as Node Selection Problems.

Problem 1: 0–1 quadratic programming

A 0–1 quadratic programming problem is [2]:

$$\textbf{(QP)} \qquad \min 1/2x^{\mathrm{T}}Qx + cx = \sum_{i=1\ldots n-1} \sum_{j=i+1\ldots n} q_{ij}x_ix_j + \sum_{i=1\ldots n} c_ix_i, \quad x \in \{1, 0\}.$$

To model QP as an NSP, we create a G-partite graph G^0 (and a flow graph $G(Q)$) with a node family for each x_i (a node for each x_i in $G(Q)$) which has two nodes n_{i0} and n_{i1}. Node n_{i0} (n_{i1}) corresponds to the variable x_i taking the value 0 (1). Join two node families (two nodes in $G(Q)$) if and only if $q_{ij} \neq 0$. The weight on arc (n_{i1}, n_{j1}) is initially set equal to q_{ij} and all other arcs between σ_i and σ_j are initially given weight 0. To account for the linear costs c_i, we select an index j such that $q_{ij} \neq 0$ and add c_i to the weight on arcs (n_{i1}, n_{j0}) and (n_{i1}, n_{j1}). It is easy to see that NSP on G^0 is a reformulation of **QP**.

Problem 2: Product design – marketing

Consider the following variation of a product design problem arising in marketing [8]: A product has to be designed with m attributes. Corresponding to an attribute u, there are n_u discrete levels of the attribute (e.g. color is an attribute with blue, green and red as three possible "levels"). For each level of each attribute, we have a measure of customer preference (main effect). In addition, there are two-way interaction effects between level k of attribute u and level r of attribute v. The objective is to design a product by choosing a level of each attribute such that the sum of the main effects and the two-way interaction effects is maximum. Transforming this product design problem to NSP is similar to the transformation for MMMC with the fixed locational costs.

Problem 3: Product design – engineering

Askin and Goldberg [1] have looked at a product design model focussing on the engineering attributes of a product, similar to the product design problem from the marketing perspective described above. Again, a level of each attribute is to be selected to minimize the sum of production cost and average cost of quality, which is a function of target design and the actual performance of a design. Production costs and the distribution of a quality variable can be an arbitrary function of attribute levels. A design point is defined by the setting for each attribute. Experimentation is carried out by selecting a set of design points and making multiple observations for each design point. The mean and variance of the quality of each design point are computed. One way to select a design is to minimize the cost of each attribute level (main effect) and the cost of quality defined by the square of the bias (difference between mean quality and the target quality of a design point), multiplied by a cost coefficient for quality loss. Askin and Goldberg develop a model, called the quadratic selection model (equivalent to NSP), to solve the problem.

Problem 4: Public transit schedule

Consider a public mass transit system (trains, subways, busses) [20] where we are given a set of transfer stations and a set of routes connecting these transfer stations, a cycle time (amount of time between successive departures at any station along its route) for each train, known number of passengers who want to change between routes at transfer stations, and known running times between stations and stopping times at stations. We want to determine the departure time of each train within its given cycle at the initial station of its route such that the sum of all waiting times for all passengers changing routes is minimized. To represent this problem as an NSP, we construct one node family for each route. The nodes in a node family correspond to different possible departure times for the train from its initial station and two node families are adjacent if there are passengers who want to change between those two routes. The weight on an arc is the total waiting time of all passengers who want to change between the routes (represented by node families) at the departure times represented by the end nodes of the arc.

The quadratic assignment problem (and hence the traveling salesman problem) also can be posed as an NSP. For these two problems, the flow graph will be complete and so the results of this paper cannot be applied to these two problems (if the results were applicable, we would have shown that P = NP!). However, the results of this paper can be used to obtain lower bounds to the QAP and the TSP be deleting (through Lagrangian relaxation) some of the arcs of the flow graph.

We would like to point out that NSP is a special case of nonserial dynamic programming. Lipton and Tarjan [15] have given a planar separator theorem and shown that a nonserial dynamic programming problem, when each variable can take only two values and the flow graph is planar, can be solved in $2^{O(\sqrt{m})}$, where m is the number of nodes in the flow graph. We will return to this in section 5 to compare the efficiency of our algorithm versus theirs.

Subsequent to the submission of this paper, Chhajed and Lowe [4] developed a general procedure for solving MMMC when the flow graph is a k-tree. If the algorithm in that paper was applied to the problem considered herein (flow graph is Halin), the complexity would be lower than that obtainable via the algorithm in this paper. However, the general procedure does not provide the same level of insight as that provided by the approach contained herein.

In the remainder of this paper, we give a polynomial time algorithm for a class of NSP which is characterized by the structure of the flow graph G. In section 2, we summarize the results from our earlier paper [5] on solving NSP when the flow graph is series–parallel. These results will be used in the later sections. In section 3, we define a reduction operation on the G-partite graph which is similar to the contraction operation defined for a graph. In section 4, we define a Halin graph and present a polynomial time algorithm for NSP when the flow graph is Halin. In concluding the paper in section 5, we show how the results can be extended to other flow graphs.

2. Previous results

In this section, we report some results from our previous paper [5], in which a polynomial time algorithm for NSP when the flow graph is series–parallel is given.

DEFINITION

A graph is *series–parallel* [21] if it can be reduced to an arc by repeated applications of the following operations:

(G1) *Series reduction*: Replace any degree-2 node q and the incident arcs (u, q) and (v, q), $u \neq v$, by a new arc $a'(u, v)$ incident to u and v.

(G2) *Cut reduction*: If q is a pendant node (a node of degree one) adjacent to node u, find a node $v \neq q$ adjacent to u, delete node q and add an additional arc $a'(u, v)$.

(G3) *Parallel reduction*: Replace two arcs e and f which are both incident to nodes u and v by a new arc g incident to u and v.

The new arcs that are added to the graph in the above operations are named pseudo-arcs in [21]. Richey describes an operation similar to operation (G2), calling it a Jackknife reduction, but does not add the new arc $a'(u, v)$. If we perform parallel reduction on (u, v) immediately after the cut reduction, we obtain Richey's Jackknife reduction. Thus, although there is a minor difference in the definition of one operator, which we need for our algorithm, the above definition of a series–parallel graph is identical to that of Richey.

Three graph operations (GP1, GP2, and GP3) on a G-partite graph are defined which are similar to the operations (G1), (G2), and (G3) discussed above, so that the new G-partite graph corresponds to G after the elementary operation. The outcome of two of these operations will result in parallel arcs in the graph. We emphasize here that if there are parallel arcs between two given nodes of a G-partite graph G^0, and if this pair of nodes is in a feasible solution $S(G^0)$ to NSP, then the parallel arcs are also in $S(G^0)$, so that the arc weights of both arcs contribute to $Z(S(G^0))$.

With each node and each arc of G^0, a label is associated in the form of a set of nodes. Initially, the label of each arc e is set as $L_a(e) = \{\ \}$, where $\{\ \}$ denotes the empty set, and the label of each node u_k is set as $L_n(u_k) = \{u_k\}$. We will represent the label of an arc e defined by two nodes p and q by $L_a(p, q)$ rather than $L_a((p, q))$. During the graph operations on G^0, arcs and nodes of graph G^0 are deleted, in some cases (new) arcs are added, and labels of the remaining arcs, as well as the arc weights, are modified to reflect the change. The labels are used, basically, to carry pertinent information about the deleted portion of the graph. In modifying the labels, we typically add two labels, where addition of labels is defined as the set union operation on the sets corresponding to the two labels. In the remainder of the paper, we will denote $\lambda = \max\{n_u: u \in V(G)\}$.

(GP1): *Series reduction*: In this process, a node family σ_q such that node q is adjacent to exactly two distinct nodes u and v of G, where $q \neq v \neq u$, is eliminated in G^0 and node families σ_u and σ_v are made adjacent. Thus, the reduced graph has one less node family. This reduction has time complexity $O(\lambda^3)$.

(GP2): *Cut reduction*: Given two node families σ_q and σ_u such that node q has degree one in G and $\{(u, v), (q, u) \in E(G)\}$, we delete node family σ_q and add parallel arcs (σ_u, σ_v) between nodes of σ_u and σ_v in the G-partite graph. Also, cut reduction can be done in $O(\lambda^2)$ time.

(GP3) *Parallel reduction*: Given two node families σ_u and σ_v such that there are two arcs between every node $u_k \in \sigma_v$ and $v_r \in \sigma_v$, we replace the parallel arcs by a single arc. The weights and the labels associated with the two parallel arcs are added and they form the weight and label, respectively, of the new arc. Furthermore, parallel reduction can be performed in $O(\lambda^2)$ time.

Additional details about these three reduction operations can be found in the appendix, where they are presented as procedures. In [5], it has been proven that each of these operations preserves the solution to NSP. Finally, an algorithm (algorithm SP) which repeatedly uses the above three reductions to solve NSP on a G-partite graph when the flow graph is series–parallel is presented in the appendix. If the flow graph has m nodes (m node families in G^0) and each node family has no more that λ members, then algorithm SP is $O(m\lambda^3)$.

3. Contraction reduction

In this section, we define a fourth elementary operation (G4) on G and (GP4) on G^0. In what follows, given two nodes u and v of G (node families σ_u and σ_v of G^0), when we refer to the set of nodes adjacent to $\{u, v\}$ (node families adjacent to $\{\sigma_u, \sigma_v\}$), we mean those nodes s in G (node families σ_s in G^0) adjacent to u or v or both (adjacent to σ_u or σ_v or both), where $s \neq u \neq v$.

(G4): *Contraction reduction*: *Contraction* of two nodes u and v in G is defined as the removal of u and v, the insertion of a *new* node w, and the insertion of an arc between w and each node which was adjacent to $\{u, v\}$.

(GP4): *Contract reduction of two super nodes*: In this operation, we reduce two node families σ_u and σ_v in G^0 to a single node family σ_w. If $\underline{G}$ denotes the graph G after nodes u and v are contracted, then contracting node families σ_u and σ_v will result in a G-partite graph corresponding to graph $\underline{G}$. The number of nodes in σ_w is $n_u * n_v$. The following procedure gives details of this process.

PROCEDURE CO(σ_u, σ_v)

Step 1: Create a node family σ_w with $n_u * n_v$ nodes $\{w_r: r = 1, \ldots, n_u * n_v\}$. For each $i = 1, \ldots, n_u$ and $j = 1, \ldots, n_v$, with $n_u \geq n_v$, choose the unlabeled node w_r where r satisfies $r = n_u * (i-1) + j$, and set its label $L_n(w_r)$ as $L_n(u_i) \cup L_n(v_j)$. Let the function $\kappa(r) = [i, j]$.

Step 2: Choose a node family σ_s adjacent to $\{\sigma_u, \sigma_v\}$.
For every node $s_k \in \sigma_s$ and $w_r \in \sigma_w$ where $\kappa(r) = [i, j]$:

Add a new arc $a'(s_k, w_r)$ with weight,

$\omega_{kr}^{sw} \leftarrow \omega_{ki}^{su} + \omega_{kj}^{sv}$ and label

$L_a(s_k, w_r) \leftarrow L_a(s_k, u_i) \cup L_a(s_k, v_j)$.

{Here, ω_{ki}^{su} and $L_a(s_k, u_i)$ (ω_{kj}^{sv}, and $L_a(s_k, v_j)$) are defined to be zero and the null set, respectively, if σ_s and σ_u (σ_v) are not adjacent.}

Step 3: Remove all the arcs connecting node family σ_s to σ_u and σ_v. If σ_u and σ_v become disconnected from the remainder of G^0, then go to step 4; else, go to step 2.

Step 4: If σ_u and σ_v are connected, then choose any node family σ_s connected to σ_w. Set the weight of arcs joining nodes in σ_s and σ_w as:

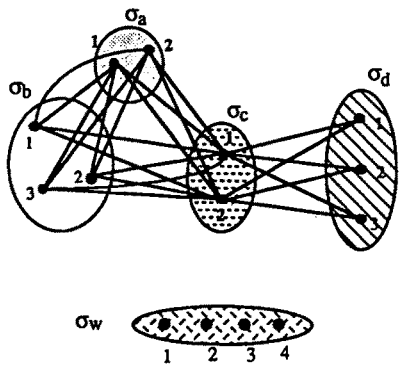
$$\omega_{kr}^{sw} \leftarrow \omega_{kr}^{sw} + \omega_{ij}^{uv} : \kappa(r) = (i, j); \quad \forall (s_k, w_r) \in E(G^0),$$

$$L_a(s_k, w_r) \leftarrow L_a(s_k, w_r) \cup L_a(u_i, v_j) : \kappa(r) = (i, j); \quad \forall (s_k, w_r) \in E(G^0).$$

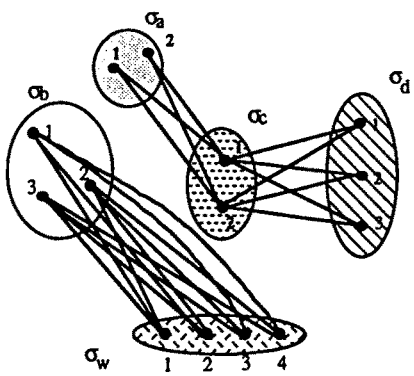
Step 5: Delete σ_u and σ_v. Return.

As an example, consider the G-partite graph G^0 shown in fig. 1 and the arc weight data shown in fig. 2. Suppose we want to contract-reduce node families σ_a and σ_c. Since $n_a = n_c = 2$, we first create a node family σ_w with four nodes as shown in fig. 3, and set the labels as $L_n(w_1) = \{a_1, c_1\}$, $L_n(w_2) = \{a_1, c_2\}$, $L_n(w_3) = \{a_2, c_1\}$, and $L_n(w_4) = \{a_2, c_2\}$. Also, let $\kappa(1) = (1, 1)$, $\kappa(2) = (1, 2)$, $\kappa(3) = (2, 1)$, and $\kappa(4) = (2, 2)$. In step 2, we choose node family σ_b, which happens to be connected to both σ_a and σ_c. We connect σ_b and σ_w (fig. 3) and compute the weights and labels. Since the labels on arcs in (σ_a, σ_b) and (σ_c, σ_b) are empty, the labels on the new arcs are also empty and are not shown in fig. 3. In step 3, we delete arcs (σ_b, σ_a) and (σ_b, σ_c) and go back to step 2, since σ_a and σ_c are still connected to G^0.

We now choose node σ_d which is connected to σ_c, connect σ_w and σ_d, and compute the weight and label of these arcs. We then delete the arcs connecting nodes in σ_d and σ_c. This disconnects σ_a and σ_c from the rest of the graph (fig. 4), so we go to step 4. Since σ_c and σ_a are connected, we select node σ_b which is connected to σ_w and modify the weights and labels of arcs joining σ_b and σ_w. Node families σ_c and σ_a are now deleted. The final result of applying procedure CO is shown in fig. 5.



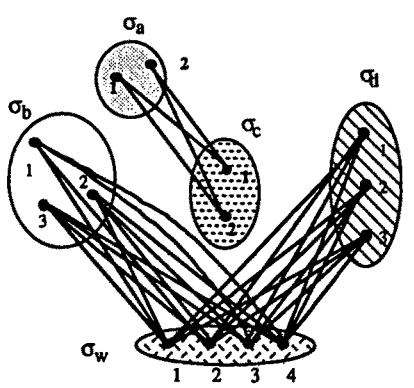
Step 1



Weights				
b \ w	1	2	3	4
1	16	26	13	23
2	15	10	16	9
3	22	20	18	16

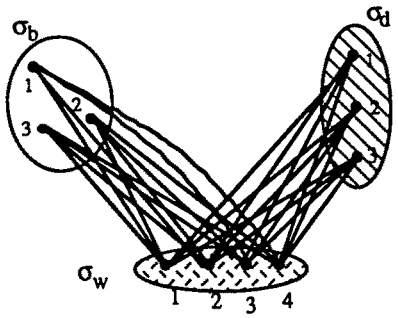
Step 3

Fig. 3. Contraction-reduction: example.



Weights				
d \ w	1	2	3	4
1	5	7	5	7
2	4	9	4	9
3	8	6	8	6

Fig. 4. Contraction-reduction example: partial solution.



Weights				
w / d	1	2	3	4
1	5	7	5	7
2	4	9	4	9
3	8	6	8	6

Weights				
w / b	1	2	3	4
1	21	33	20	33
2	20	13	23	19
3	27	23	25	26

Label on Each Arc: { }

Node Labels

$L_n(w_1)=\{a_1,c_1\}$, $L_n(w_2)=\{a_1,c_2\}$, $L_n(w_3)=\{a_2,c_1\}$, $L_n(w_4)=\{a_2,c_2\}$

$L_n(b_1)=\{b_1\}$, $L_n(b_2)=\{b_2\}$, $L_n(b_3)=\{b_3\}$

$L_n(d_1)=\{d_1\}$, $L_n(d_2)=\{d_2\}$, $L_n(d_3)=\{d_3\}$

Fig. 5. Contraction-reduction: final solution.

The complexity of step 1 is $n_u * n_v$. Step 2 can be carried out in $n_s * n_u * n_v$ time, which is repeated for all σ_s connected to σ_u or σ_u, giving it a complexity of $O((\sum_{s \text{ is adjacent to } \{u, v\}} n_s) * n_u * n_v)$. The complexity of steps 4 and 5 is no larger than that of step 2. Thus, the complexity of CO is $O((\sum_{s \text{ is adjacent to } \{u, v\}} n_s) * n_u * n_v)$. We now show that contraction-reduction preserves an NSP solution on G^0.

Let ψ be an arbitrary subset of nodes of a G-partite graph G^0 such that there is at most one node from each node family in ψ. Let $S^*(G^0, \psi)$ be an optimal solution of a *constrained* version of NSP on G^0 with the set of nodes ψ fixed, and let $Z(S^*(G^0, \psi))$ be the value of this solution. Thus, $S^*(G^0, \{\ \})$ is a solution to NSP.

LEMMA 1

For a G-partite graph G^0 with node families σ_u and σ_v, let $\underline{G}^0$ and $\underline{G}$ be the results of contracting node families σ_u and σ_v in G^0 and nodes u and v in G. Given an optimal solution $S^*(\underline{G}^0)$ to NSP($\underline{G}^0$), an optimal solution to NSP(G^0) can be constructed using the nodes and arc labels of $S^*(\underline{G}^0)$.

Proof

Let $S^*(\underline{G}^0)$ be an optimal solution to NSP($\underline{G}^0$) and let $w_{r^*} \in \sigma_w$ be in this optimal solution, where σ_w is the node family introduced in G^0 as a result of contraction of σ_u and σ_v. Let $u_{i^*} \in \sigma_u$ and $v_{j^*} \in \sigma_v$ be such that $\kappa(r^*) = (i^*, j^*)$. Let $\psi^* = \{q_{p^*}: (w_{r^*}, q_{p^*}) \in S^*(\underline{G}^0)\}$, i.e. ψ^* is the set of nodes adjacent to w_{r^*} in $S^*(\underline{G}^0)$.

All nodes in ψ^* are also in graph G^0 and belong to node families adjacent to σ_u and/or σ_v. Let $\underline{\Delta}^*$ denote the set of arcs between w_{r^*} and the nodes in ψ^*.

In order to obtain a solution to NSP($\underline{G}^0 \backslash \sigma_w, \psi^*$), we can delete the arcs in $\underline{\Delta}^*$ from $S^*(\underline{G}^0)$. That is, $S^*(\underline{G}^0 \backslash \sigma_w, \psi^*) = S^*(\underline{G}^0) \backslash \underline{\Delta}^*$. $S^*(\underline{G}^0 \backslash \sigma_w, \psi^*)$ is also an optimal solution to NSP($G^0 \backslash \{\sigma_u, \sigma_v\}, \psi^*$) and has the same objective function value on graphs $\underline{G}^0 \backslash \sigma_w$ and $G^0 \backslash \{\sigma_u, \sigma_v\}$ because these two graphs are the same. If Δ^* is the set of arcs between $\{u_{i^*}, v_{j^*}\}$ and nodes in ψ^*, including the arc (u_{i^*}, v_{j^*}) if it exists, then the sum of the weights and labels on the arcs in Δ^* are the same as the sum of the weights and labels on arcs in $\underline{\Delta}^*$. Thus, $S^*(\underline{G}^0 \backslash \sigma_w, \psi^*) \cup \Delta^*$ is a *feasible* solution to NSP(G^0) with the objective function value $Z(S^*(\underline{G}^0))$.

What remains to be shown is that $S^*(\underline{G}^0 \backslash \sigma_w, \psi^*) \cup \Delta^*$ is also an optimal solution to NSP(G^0). To do so, we first assume that a solution $S^{*\prime}(G^0)$ with an objective function value better (lower) than $Z(S^*(\underline{G}^0))$ exists and then arrive at a contradiction. Let $u_{i'} \in \sigma_u$ and $v_{j'} \in \sigma_v$ be such that they are in this optimal solution. Let $w_{r'} \in \sigma_w$ be such that $\kappa(r') = (i', j')$. Let $\psi' = \{q_{p'} : q_{p'} \in S^{*\prime}(G^0), q$ is in the set of nodes adjacent to $\{u, v\}\}$, Δ' is the set of arcs between $\{u_{i'}, v_{j'}\}$ and nodes in ψ', including the arc $(u_{i'}, v_{j'})$ if it exists, and $\underline{\Delta}'$ is the set of arcs between $w_{r'}$ and the nodes in ψ'. Now, $S^{*\prime}(G^0) \backslash \Delta'$ is an optimal solution to NSP($G^0 \backslash \{\sigma_u, \sigma_v\}, \psi'$) and NSP($\underline{G}^0 \backslash \sigma_w, \psi'$). Also $\{S^{*\prime}(G^0 \backslash \Delta'\} \cup \underline{\Delta}'$ is a feasible solution to NSP($\underline{G}^0$). However, the sum weights and labels on Δ' and $\underline{\Delta}'$ are the same. Thus, we have obtained a feasible solution to NSP($\underline{G}^0$) with a value smaller than $Z(S^*(\underline{G}^0))$, a contradiction. □

In the next section, we define a Halin graph and give an algorithm to solve NSP when the flow graph is Halin. This algorithm makes use of GP1, . . . , GP4 defined in the previous two sections.

4. Halin graphs

In this section, we give an algorithm to solve (NSP) on a G-partite graph corresponding to a Halin flow graph. A *Halin graph* is constructed as follows: Take a tree T having no nodes of degree 2, with a planar embedding, and add a cycle $\mathcal{C}$ formed by all the leaf nodes of T such that $G = T \cup \mathcal{C}$ remains planar (fig. 6). A procedure to recognize a Halin graph in polynomial time is given in [22].

Our solution procedure to solve NSP on a Halin flow graph proceeds by first identifying a set of pairs of nodes of G. Each pair of nodes is contracted, after which the original flow graph reduces to an *outerplanar* graph, which is known to be a series–parallel graph [18]. Corresponding operations are also performed on G^0. Subsequently, we use algorithm SP to solve NSP on this resulting outerplanar graph and recover the solution to the original problem on the Halin graph. We begin by introducing some additional notation.

Given a Halin graph G, let C be the set of cycle nodes (i.e. $V(\mathcal{C})$) and let I be the non-cycle nodes (non-tip nodes of T). We assume that G has no nodes of

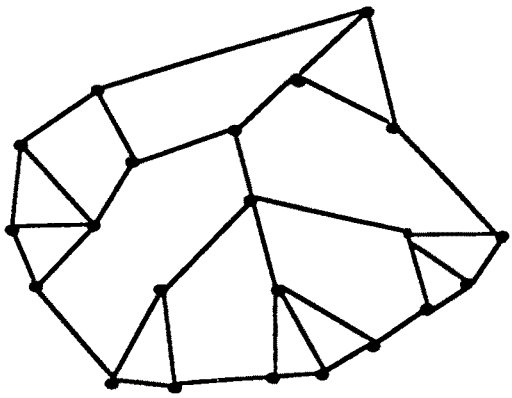


Fig. 6. A Halin graph.

degree 2 (see section 5 for relaxation of this assumption). Let $|C| = \theta$ and $|I| = \eta$, so that $\theta + \eta = m$. Select a node $r \in I$ which is adjacent to no more than one non-cycle node. Letting v be the number of cycle nodes adjacent to r, we now number all of the cycle nodes of G consecutively in a counter-clockwise fashion, in such a way that the cycle nodes adjacent to r are numbered $1, 2, \ldots, v$.

Now direct all of the arcs of T away from node r. If there is a directed path from node $p \in I$ to node $q \in V(G)$, we say that q is a *descendant* of p, and if p is adjacent to q, then p (q) is the *parent* (*child*) of q (p). Given any node $i \in I$, we denote the set of cycle nodes (C-nodes) which are descendants of i by $C(i)$. We also define $C(i) = \{i\}$ for node $i \in C$. With this construction, we note that for any $i \in I$, the member(s) of $C(i)$ are numbered consecutively. In fact, $C(r) = \{1, \ldots, \theta\}$ and $\{1, \theta\}$ is a subset of $C(i)$ if and only if $i = r$. Also, if $q \in I$ is a descendant of p, then $C(p) \supset C(q)$. We also note that if q is not a descendant of p, then $C(q) \cap C(p) = \emptyset$.

For any node $i \in I$, we call the *youngest* child of i that *child*, denoted by y_i, which has the lowest numbered C-node as a descendant. If one or more children of i are themselves C-nodes, the indices of the C-node children are used to define the youngest child of i.

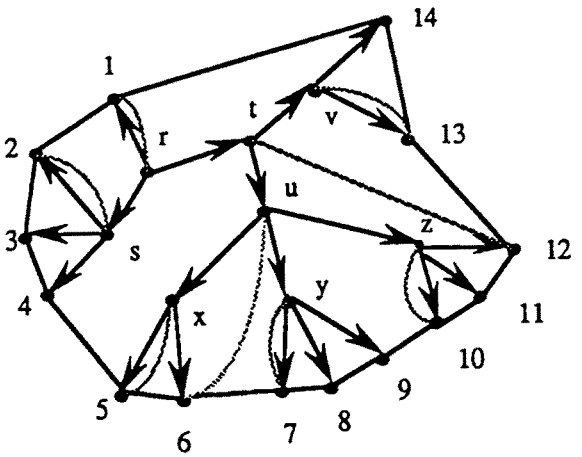
As an overview, we will contract every non-cycle node with an appropriately selected cycle node, i.e. η pairs will be contracted and each pair to be contracted will be made up of a non-cycle node and a matched cycle node. We now specify for each $i \in I$ a cycle node (denoted by $c(i)$) to contract with i. As will be shown shortly, we choose the nodes $c(i)$ so that:

(i) $c(i) \neq c(j)$ for $i \neq j$ so that each node family of $\hat{G}^0$ (graph G^0 after the contractions) contains no more than λ^2 nodes;

(ii) the graph $\hat{G}$ resulting after the η contractions is planar; and

(iii) each node of $\hat{G}$ is in the exterior boundary of $\hat{G}$ so that $\hat{G}$ is outerplanar.

The nodes $c(i)$ are chosen as follows: For each node $i \in I$, if $y_i \in C$, then $c(i) = y_i$. Otherwise, if $y_i \in I$, then $c(i)$ is the largest indexed member (oldest member) of $C(y_i)$. We note in particular that $c(r)$ is that cycle node which is numbered 1 since node r is adjacent to the cycle node numbered 1. An example of the above construction and definitions is shown in fig. 7. The dashed lines in the figure are



C(r) = {1,...,14}, C(s) = {2,3,4}, C(u)=C(x)∪C(y)∪C(z) = {5,6,7,8,9,10,11,12}

The youngest child of node t is node u.

c(r) = 1, c(s) = 2, c(t) = 12, c(u) = 6, c(y) = 7, c(z) = 10, c(v) = 13, c(x) = 5

Fig. 7. An example to show $c(\cdot)$.

between nodes i and $c(i)$ to be contracted. The graph $\hat{G}$ which results from the contraction is shown in fig. 8. We note that $\hat{G}$ contains parallel arcs. Figure 9 shows the result of performing all parallel reductions on the graph in fig. 8 (after some rearrangement of the embedding).

We first show part (i) for the above choice of $c(i)$'s.

LEMMA 2

$c(i) \neq c(j)$ for any $i, j \in I$, $i \neq j$.

Proof

Letting y_i (y_j) be the youngest child of i (j), we have two cases.

Case I: y_i is not on a directed path from i to j.

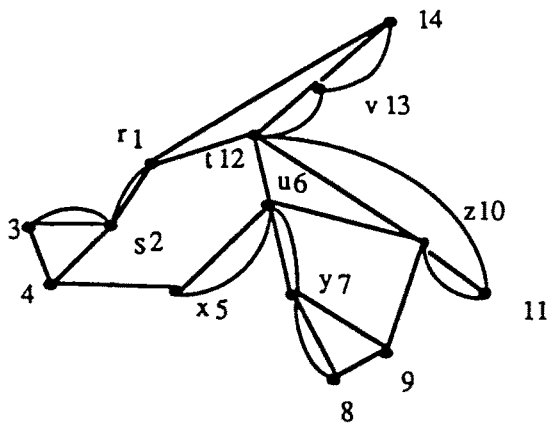


Fig. 8. Graph after contraction of pairs $(*, c(*))$.

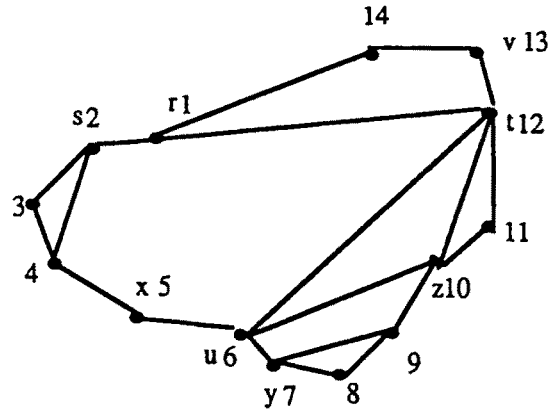


Fig. 9. Simplification of fig. 8 to show an outerplanar graph.

In this case, $C(y_i) \cap C(y_j) = \varnothing$ so that $c(i) \neq c(j)$. (A similar argument holds for y_j not on a directed path from j to i.)

Case II: y_i is on a directed path from i to j.

Letting q denote the index of the node with the largest number in $C(j)$, we note that $c(i) \geq q$. Since j has at least two descendants, q is strictly greater than any member of $C(y_j)$ so that $q > c(j)$. Thus, $c(i) > c(j)$. (A similar argument holds for y_j on a directed path from j to i.) □

Parts (ii) and (iii) are established in the following:

THEOREM 1

Let $\hat{G}$ be the graph resulting from contracting the pairs $\{(i, c(i)), i = 1, \ldots, \eta\}$. Then,

(a) $\hat{G}$ is planar, and

(b) each node of $\hat{G}$ is on the exterior boundary of $\hat{G}$, so that $\hat{G}$ is outerplanar.

Proof

Each pair of consecutive cycle nodes of G is in a unique face of G. From lemma 2, $c(i) \neq c(j)$ for all $i, j \in I$, $i \neq j$, so the face of G containing $c(i)$ and $c(i) + 1$, denoted by F_i, is distinct from the face F_j containing $c(j)$ and $c(j) + 1$, for every $i, j \in I$. For every $i \in I$, construct an artificial edge in F_i connecting i and $c(i)$. Note that the resulting graph, denoted by $\overline{G}$, is planar.

From the definition of contraction of nodes, the contraction of i and $c(i)$ in G is equivalent to "shrinking" (as defined in [14]) the (artificial) edge $(i, c(i))$ in $\overline{G}$. In lemma 1 of [14], it is shown that shrinking an edge of a planar graph preserves planarity. It now follows that contraction of the pairs $\{(i, c(i))\}$ of G results in $\hat{G}$ planar.

That $\hat{G}$ is outerplanar again follows from the definition of contraction and the fact that every node $i \in I$ is contracted with a cycle node of G. □

The above results provide a justification for the following algorithm.

ALGORITHM HALIN

Given: A G-partite graph G^0 corresponding to a Halin flow graph $G = T \cup \mathcal{C}$.

Output: Solution to NSP on G^0.

Step 0: Let $C = V(\mathcal{C})$; $I = V(G) \backslash V(\mathcal{C})$; $r \in I$: r is adjacent to two or more nodes in C. Number these nodes of C adjacent to r in a counter-clockwise fashion, starting with 1 and continue numbering the remainder of the nodes of C.

Step 1: Root T at r. Let $C(i) = \{j: j \text{ is a descendant of } i;\ j \in C\}$, $\forall i \in I$, and let $C(i) = \{i\}$, $\forall i \in C$.
For all $i \in I$, let y_i be that child of node i : $\min\{j: j \in C(i)\} \in C(y_i)$.
Define $c(i) = y_i$ if $y_i \in C$; otherwise, $c(i) = \max\{j: j \in C(y_i)\}$, $\forall i \in I$.

Step 3: Contract pairs $(i, c(i))$, $\forall i \in I$ in G and the corresponding node families in G^0, using (G4) and procedure CO, respectively, deriving graphs $\hat{G}$ and $\hat{G}^0$.

Step 4: Call algorithm SP to solve NSP on $\hat{G}$ and $\hat{G}^0$.

LEMMA 3

The complexity of solving NSP for a Halin flow graph using algorithm HALIN is $O(m\lambda^6)$.

Proof

As we will show, the total effort is dominated by the work in step 4, i.e. solving the NSP on $\hat{G}^0$ when $\hat{G}$ is series–parallel.

We have argued earlier that to contract σ_u and σ_v takes $O(n_u * n_v * \sum n_s: s$ adjacent to $\{u, v\})$ effort. Thus, the total effort to create graphs $\hat{G}$ and $\hat{G}^0$ is $O(\sum_{i \in I}(|\sigma_i| * |\sigma_{c(i)}| * \sum|\sigma_s|: s$ adjacent to $\{i, c(i)\})$. Again, letting λ be an upper bound on the number of nodes in any family of the original G-partite graph G^0, some σ_s in the above expression *may* be the result of a previous contraction operation, in which case it may have up to λ^2 members. However, in any case $|\sigma_s| \leq \lambda^2$. It is *always* the case that $|\sigma_i| \leq \lambda$ and $|\sigma_{c(i)}| \leq \lambda$.

Letting k_i be the number of nodes adjacent to $i \in I$ and noting that the number of nodes adjacent to $c(i)$ is 3, we have $\sum_{i \in I}(k_i + 3) = O(m)$. Thus, the total effort for creating graphs $\hat{G}$ and $\hat{G}^0$ is $O(m\lambda^4)$. In step 4, we call algorithm SP with input graphs $\hat{G}$ and $\hat{G}^0$, where $\hat{G}$ is a series–parallel graph with $O(m)$ nodes and each node family of $\hat{G}^0$ has no more than λ^2 members. Thus, using algorithm SP on $\hat{G}$ and $\hat{G}^0$ takes $O(m\lambda^6)$ effort, which is the complexity of algorithm HALIN. □

COROLLARY

Problem MMMC on a transport graph τ with n nodes, a Halin flow graph, and each new facility can be at any one of the n nodes of τ, can be solved, via algorithm HALIN, in $O(mn^6)$.

When the flow graph is a partial k-tree (see, for example, [6]), Chhajed and Lowe [4] have shown that MMMC can be solved in $O(mn^{k+1})$. Since a Halin graph is a partial 3-tree, we note that MMMC can be solved in $O(mn^4)$. However, as we noted earlier, considerable insight is lost in using the more general approach.

5. Extensions

In this section, we give examples of other graphs for which NSP can be solved by using GP1, . . . , GP4 in polynomial time. Specifically, we define a generalized Halin graph and π-Halin graphs. Finally, we show that using our results, we obtain a linear time algorithm for a 0–1 quadratic programming problem when G^0 representing the 0–1 quadratic program corresponds to a Halin flow graph.

5.1. GENERALIZED HALIN GRAPHS

We can generalize the definition of a Halin graph to a graph constructed as $G' = T' \cup \mathcal{C}'$, where T' is a tree (which could have nodes of degree 2) and $\mathcal{C}'$ is a cycle connecting a *subset* of the leaf nodes of T'. Thus, there may be tip nodes of T' not in the cycle C' (G' could have nodes of degree 1), and G' could have nodes of degree 2 on the cycle and on the tree. After maximal application of series

and cut-reductions to G', the resulting graph will be Halin; therefore, recognition of these graphs and solving NSP on them can be done in the same time complexity as before.

5.2. π-DECOMPOSABLE GRAPHS

We now give a further generalization of the class of graphs which are solvable by the repeated application of GP1, . . . , GP4 in polynomial time. In defining this class, there are two important factors to be kept in mind: (i) a graph in the class should be recognizable in time complexity no larger than the time it will take to solve NSP, and (ii) there should be limited application of the contraction–reduction. In particular, if w is the result of contracting two nodes, then w should not be used in further contractions (in order to bound the number of nodes in any node family of the G-partite graph).

We first introduce some definitions. The *connectivity* of a graph is the minimum number of nodes whose removal results in a disconnected graph. A graph is said to be n-connected if its connectivity is at least as large as n. The *t-decomposition* of a graph, which is unique [3, 12], is a tree T, the nodes of which are graphs. Two nodes in T are adjacent if and only if they have a common arc, called the *marker arc*. In the t-decomposition, (i) every member of $V(T)$ has at least three arcs and is a polygon (cycle), bond (graph on two nodes with parallel arcs), or a prime (a graph which is 3-connected after deleting all loops and all but one arc in each parallel class), (ii) only bond members of $V(T)$ have arcs parallel to their parent marker (defined below), and (iii) no two polygons or bonds have a marker arc in common. The reverse of decomposition is a *merge* operation; while merging two adjacent nodes of T, the common marker arc is identified and then erased. If we perform all the merge operations (in any order), we obtain the graph G. An $O(V(G) + E(G))$ algorithm to compute the t-decomposition is given in [12].

If we direct all arcs of a tree T away from a node $D \in V(T)$, T is *rooted* at D. Given a rooted T, and two nodes $H, K \in V(T)$ such that arc $(a, b) \in E(H) \cap E(K)$ (i.e. (a, b) is a marker arc of H and K), H is a *parent* of K if there is a directed arc from H to K in the rooted tree T. Arc (a, b) is called the *child marker* of K. Note that every node of rooted tree T, except the root node, has one and only one child marker. For a rooting at node R, let $cmn(K, R)$ denote the nodes $\{a, b\}$ corresponding to the end nodes of the child marker of K for all $K \in V(T) \backslash R$.

We now define [5] a family of graphs π. Graph B is a member of π if and only if there are two terminal (nodes) u and v of B such that for arbitrary fixed nodes $u_{k^0} \in \sigma_u$ and $v_{r^0} \in \sigma_v$, $S^*(G^0(B), \{u_{k^0}, v_{r^0}\})$ can be computed in polynomial time, where $G^0(B)$ is the G-partite graph corresponding to B. In addition, graph B should be recognizable in polynomial time. Note that π may contain non-planar graphs, e.g. K_5 with any pair of nodes as terminals is in π. Members of π include Halin graphs, series–parallel graphs, as well as graphs obtained by taking a series–parallel graph or a Halin graph, G, two additional nodes (which will be terminals),

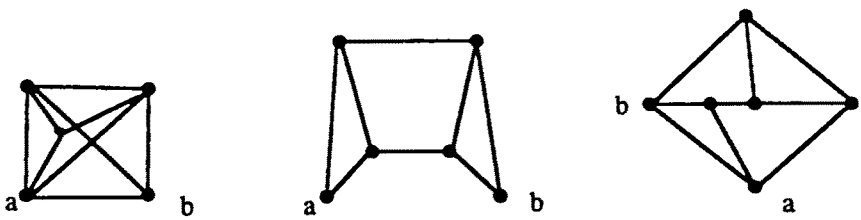


Fig. 10. Examples of graphs which are in π.

$\{u, v\}$, and connecting node u (v) to an arbitrary subset of nodes of G. The graphs in fig. 10 are examples of members of π.

G is said to be a *π-decomposable* graph if there exists a node $\tilde{R}$ such that when the t-decomposition tree of G is rooted at $\tilde{R}$, each component $H \in V(T)\backslash\tilde{R}$ is in π with terminals $cmn(H, \tilde{R})$ and $\tilde{R}$ is in π for some pair of nodes as terminals. Note that if $\tilde{R}$ is series–parallel or Halin, then $\tilde{R}$ is in π. Figure 11 gives an example of a π-decomposable graph together with its t-decomposition.

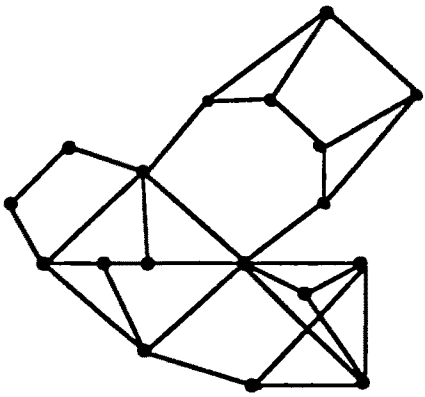
To recognize a π-decomposable graph G, we first find its t-decomposition. Then we select a node R of this tree, root T at R, and test whether R is in π for some pair of nodes of R. This can be done in polynomial time since there are at most $O(V(R)^2)$ pairs of nodes to be tested. If R is in π, we then test for each member of $V(T)\backslash R$ whether it is in π with terminals $cmn(H, R)$. If the answer is affirmative for each component in $V(T)\backslash R$, we are finished; otherwise, we select another unselected node of T and continue. Thus, determining whether a graph is π-decomposable can be done in polynomial time.

In order to solve NSP on a π-decomposable flow graph, we first find the node $\tilde{R}$ and root the tree at $\tilde{R}$. The following recursive step is carried out until node $\tilde{R}$ is obtained.

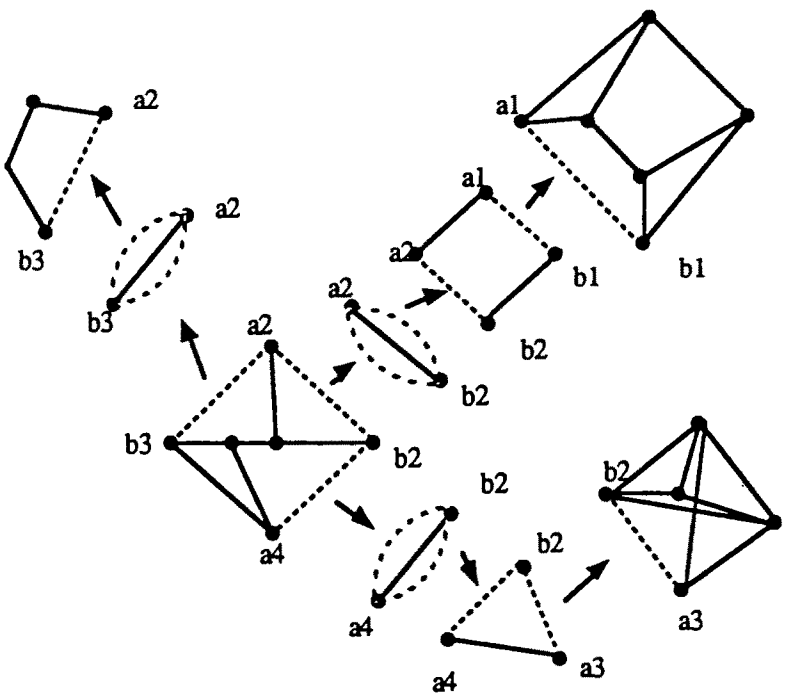
Consider a component H ($\neq \tilde{R}$) which is a leaf node in the t-decomposition tree.

Case I: If H is a polygon, we perform series-reduction on both H and $G^0(H)$ until only the marker arc remains. Merge H with the component adjacent to it and perform parallel reduction, if necessary.

Case II: If H is prime, connect (arcs initially have zero costs) the node families corresponding to the $cmn(H, \tilde{R}) = \{u, v\}$ in $G^0(H)$. Select two nodes $u_k \in \sigma_u$ and $v_r \in \sigma_v$ and compute $S^*(G^0(H), \{u_k, v_r\})$. This can be done in polynomial time since H is in π with terminals $cmn(H, \tilde{R})$. Set the weight on the arc (u_k, v_r) as $Z(S^*(G^0(H), \{u_k, v_r\}))$ and label as nodes in $S^*(G^0(H), \{u_k, v_r\})$. Perform this operation for every choice of a node in σ_u and a node in σ_v. Delete all arcs and nodes of H except the marker arc. Merge H with the component adjacent to it and perform parallel reduction, if necessary.



Graph G



The t-decomposition of graph G

Fig. 11. A π-decomposable graph.

Case III: If H is a bond, merge H with the component adjacent to it and perform parallel reduction, if necessary.

In either case, the resulting tree T will have one less node. At the end of the process, we obtain graph $\tilde{R}$ for which NSP can be solved in polynomial time.

5.3. QUADRATIC ZERO-ONE PROGRAMMING

Pardalos and Jha [16] have provided an algorithm which uses the planar separator theorem [14, 15] and runs in $O(m\log(m)2^{c\sqrt{m}\log m})$ with $c > 0$, for quadratic zero–one programming (see problem 1, section 1) for planar graphs. Halin graphs have a 3-separator which will make the algorithm in [6] of $O(m\log(m)2^{3\log m})$. The initial m is an upper bound on the complexity of finding a separator and updating weights. It may be that due to the special structure of a Halin graph and the resulting components, this can be done in constant time. In addition, the exponent of 3 comes from the fact that a Halin graph has a 3-separator, but the resulting components and subcomponents will have a 2-separator. Thus, it may be possible to reduce the overall upper bound to $O(\log(m)2^{2\log m}) = O(m^2\log m)$. If we represent the quadratic 0–1 programming problem as an NSP with flow graph $G(Q)$ (see section 1), then there will be two nodes in every node family, i.e. $\lambda = 2$. Thus, applying algorithm Halin to solve the quadratic 0–1 programming problem when $G(Q)$ is a Halin graph will result in a time complexity of $O(m2^6)$, which is linear. These results are also applicable when $G(Q)$ is π-decomposable. We note that Barahona [2] has provided a linear time algorithm for quadratic 0–1 programming when $G(Q)$ is series–parallel which can also be achieved by formulating the problem as an NSP and applying algorithm SP.

Appendix

PROCEDURE SR(σ_q)

Step 1: Let u and v be the two nodes adjacent to node q in G, where $q \neq u \neq v$.

Step 2: For each pair of nodes $u_k \in \sigma_u$, $v_r \in \sigma_v$, find q_{p^0} giving

$$\omega^{uq}_{kp^0} + \omega^{vq}_{rp^0} = \min_{q_p \in \sigma_q} \left\{\omega^{uq}_{kp} + \omega^{vq}_{rp}\right\} \text{ (ties can be arbitrarily broken).}$$

Add an arc $a'(u_k, v_r)$ with weights equal to $\omega^{uq}_{kp^0} + \omega^{vq}_{rp^0}$ and let the label of this new arc be $L_{a'}(v_r, u_k) \leftarrow L_a(v_r, q_{p^0}) \cup L_a(u_k, q_{p^0}) \cup L_n(q_{p^0})$.

Step 3: Delete node family σ_q. Return (to the calling algorithm).

PROCEDURE PR(σ_u, σ_v)

Step 1: Let nodes $u_k \in \sigma_u$ and $v_r \in \sigma_v$ be such that there are two arcs between them. Delete one of these arcs and add its weight to the weight of the other arc. Also add the label of this deleted arc to the label of the second arc.

Step 2: Continue step 1 until no parallel arcs between nodes of σ_u and σ_v remain.

Step 3: Return.

PROCEDURE CR(σ_q, σ_u)

Step 1: With q a pendant node of G adjacent to u, select an arc (u, v) of G, $v \neq q$. (Such an arc exists because we have assumed that (u, q) is not the only arc of G, and thoughout we assume that G is connected.)

Step 2: For each node u_k of σ_u,
Find a node q_{p^0} of node family σ_q such that

$$\omega^{uq}_{kp^0} = \min_{q_p \in \sigma_q} \left\{\omega^{uq}_{kp}\right\} \text{ (ties can be arbitrarily broken).}$$

In G^0, add new arcs $a'(u_k, v_r)$ for all $v_r \in \sigma_v$ with weights $\omega^{uq}_{kp^0}$ and set the labels of these new arcs $L_{a'}(u_k, v_r) \leftarrow L_n(q_{p^0}) \cup L_a(q_{p^0}, u_k)$.

Step 3: Delete all the nodes of node family σ_v in G^0, i.e. delete σ_v.

Step 4: Return.

ALGORITHM SP

Step 0: Set $k \leftarrow 1$, $G^0_k \leftarrow G^0$, $G_k \leftarrow G$.
Let $2D$ denote the list of nodes in G_k with degree 2. PA is the list of node pairs having parallel arcs in G_k.

Step 1: If G_k is a single arc, then go to step 6; else find a node $q \in V(G_k)$ with degree 1 and go to step 2. If there exists no such node, then go to step 3.

Step 2: Let $(q, u) \in E(G_k)$ be the arc connecting q to another node u.
Cut-reduce node families σ_u and σ_q in G^0_k by calling procedure CR(σ_q, σ_u).
Cut-reduce nodes u and q in G_k.
Let node v be such that it is adjacent to u in G_k and is used in CR($\cdot$). Add (u, v) to PA.
Set $G_{k+1} \leftarrow G_k$ (after cut-reduction)
$G^0_{k+1} \leftarrow G^0_k$ (after cut-reduction)
$k \leftarrow k + 1$.
Go to step 5.

Step 3: If $|2D| = 0$, then go to step 5; else choose a node $q \in 2D$. Let nodes u and v be adjacent to q in G_k.

Step 4: If $(u, v) \in E(G_k)$, then add (u, v) to PA.
Series-reduce node family σ_q by calling procedure SR(σ_q).
Series-reduce node q.
Set $G_{k+1} \leftarrow G_k$ (after series-reduction)
$G^0_{k+1} \leftarrow G^0_k$ (after series-reduction)
$k \leftarrow k + 1$
Delete q from $2D$ and go to step 5.

Step 5: If $|PA| = 0$, then go to step 1; else let $(u, v) \in PA$.
Parallel-reduce arcs between node families σ_u and σ_v by calling procedure $PR(\sigma_u, \sigma_v)$.
Parallel-reduce arcs between nodes u and v.
Set $G_{k+1} \leftarrow G_k$ (after parallel-reduction)
$G^0_{k+1} \leftarrow G^0_k$ (after parallel-reduction)
$k \leftarrow k + 1$
If u (or v) has degree 2 in G_k, add it to $2D$.
Go to step 1.

Step 6: At this stage, G is a single arc (u, v). Find

$$\omega^{uv}_{k^0 r^0} = \min_{k \in \sigma_u, r \in \sigma_v} \left\{ \omega^{uv}_{kr} \right\}.$$

This is the value of the optimal solution and the solution can be constructed by $L_a(u_{k^0}, v_{r^0}) \cup L_n(u_{k^0}) \cup L_n(v_{r^0})$. Stop.

References

[1] R.G. Askin and J.B. Goldberg, Economic optimization in product design, Eng. Opt. 14(1988) 139–152.
[2] F. Barahona, A solvable case of quadratic 0–1 programming, Discr. Appl. Math. 13(1986)23–26.
[3] R.E. Bixby and D.K. Wagner, An almost linear-time algorithm for graph realization, Math. Oper. Res. 13(1988)99–123.
[4] D. Chhajed and T.J. Lowe, Solving structured multifacility location problems efficiently, Transp. Sci. (1993), to appear.
[5] D. Chhajed and T.J. Lowe, *M*-median and *M*-center problems with mutual communication: Solvable special cases, Oper. Res. 40(1992)S56–S66.
[6] D.G. Corneil and J.M. Keil, A dynamic programming approach to the dominating set problem on *K*-trees, SIAM J. Alg. Discr. Meth. 8(1987)535–543.
[7] P.M. Dearing, R.L. Francis and T.J. Lowe, Convex location problems on tree networks, Oper. Res. 24(1976)628–642.
[8] S.E. Eriksen and P.D. Berger, A quadratic programming model for product configuration optimization, Zeits. Oper. Res. 31(1987)B143–B159.
[9] E. Erkut, R.L. Francis and T.J. Lowe, A multimedian problem with interdistance constraints, Environ. Planning B: Planning and Design 15(1988)181–190.
[10] E. Erkut, R.L. Francis, T.J. Lowe and A. Tamir, Equivalent mathematical programming formulations of monotonic tree network location problems, Oper. Res. 37(1989)447–461.
[11] R.L. Francis, T.J. Lowe and H.D. Ratliff, Distance constraints for tree network multifacility location problems, Oper. Res. 26(1978)570–596.
[12] J.E. Hopcroft and R.E. Tarjan, Dividing a graph into triconnected components, SIAM J. Comput. 2(1973)135–158.
[13] A. Kolen, Location problems on trees and in the rectilinear plane, Stichting Mathematisch Centrum, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands (1982).
[14] R.L. Lipton and R.E. Tarjan, A separator theorem for planar graphs, SIAM J. Appl. Math. 36(1979) 177–189.
[15] R.J. Lipton and R.E. Tarjan, Applications of a planar separator theorem, SIAM J. Comput. 9(1980) 615–627.

[16] P.M. Pardalos and S. Jha, Graph separation techniques for quadratic zero–one programming (1990), Computer Science Department, The Pennsylvania State University, to appear in Comput. Math. Appl.
[17] J.-C. Picard and H.D. Ratliff, A cut approach to the rectilinear distance facility location problem, Oper. Res. 28(1978)422–433.
[18] T.V. Wimer, Linear algorithms on k-terminal graphs, Report No. URI-030, Clemson University (1987).
[19] Y. Xu, R.L. Francis and T.J. Lowe, The multimedian problem on a network: Exploiting block structure, Working Paper, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL (1988).
[20] W. Domschke, Schedule synchronization for public transit networks, OR Spektrum 11(1989)17–24.
[21] M.B. Richey, Optimal location of a path or tree on a network with cycles, Working Paper, George Mason University, Fairfax, VA (1989).
[22] G. Cornuejols, D. Naddef and W. Pulleyblank, The traveling salesman problem in graphs with 3-edge cutsets, J. ACM 32(1985)383–410.