OPTIMAL HAMILTONIAN COMPLETIONS AND PATH COVERS FOR TREES, AND A REDUCTION TO MAXIMUM FLOW

D. S. FRANZBLAU[1] and A. RAYCHAUDHURI[2]

Abstract

A *minimum Hamiltonian completion* of a graph G is a minimum-size set of edges that, when added to G, guarantee a Hamiltonian path. Finding a Hamiltonian completion has applications to frequency assignment as well as distributed computing. If the new edges are deleted from the Hamiltonian path, one is left with a *minimum path cover*, a minimum-size set of vertex-disjoint paths that cover the vertices of G. For arbitrary graphs, constructing a minimum Hamiltonian completion or path cover is clearly NP-hard, but there exists a linear-time algorithm for trees. In this paper we first give a description and proof of correctness for this linear-time algorithm that is simpler and more intuitive than those given previously. We show that the algorithm extends also to unicyclic graphs. We then give a new method for finding an optimal path cover or Hamiltonian completion for a tree that uses a reduction to a maximum flow problem. In addition, we show how to extend the reduction to construct, if possible, a covering of the vertices of a bipartite graph with vertex-disjoint cycles, that is, a 2-factor.

1. Definitions and results

A *Hamiltonian completion* of a finite graph $G = (V, E)$ is a set of edges that, when added to E, ensure that G has a Hamiltonian path.[3] The *Hamiltonian completion number $hc(G)$* is the minimum number of edges required in a Hamiltonian completion for G. A Hamiltonian completion with the minimum number of edges is a *minimum* or *optimal* Hamiltonian completion. A closely related concept is that of a *path cover* for a graph G, which is a set of vertex-disjoint simple paths that together contain all the vertices of G. The *path cover number $pc(G)$* is the minimum number of paths in

[1]Department of Mathematics, CUNY, College of Staten Island, 2800 Victory Blvd, Staten Island, NY 10314, USA; e-mail: franzblau@postbox.csi.cuny.edu.
[2]Department of Mathematics, CUNY, College of Staten Island, 2800 Victory Blvd, Staten Island, NY 10314, USA; e-mail: raychaudhuri@postbox.csi.cuny.edu.
[3]See [3] or [4] for graph terms not defined here.
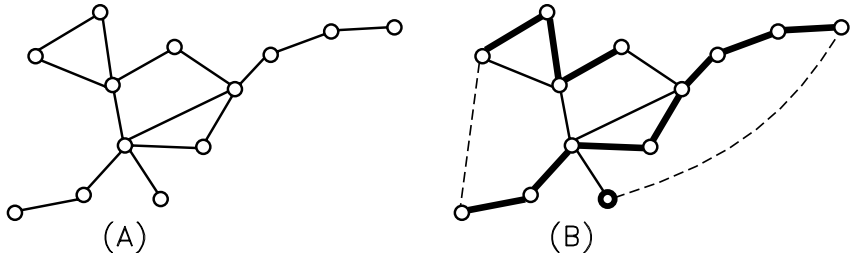
a path cover. A *minimum* or *optimal* path cover is one with the minimum number of paths. For an illustration of these terms, see Figure 1. Given a set of k paths in a path cover, one can connect them to form a Hamiltonian path by adding $k - 1$ edges, so it is easy to see that

$$pc(G) = hc(G) + 1. \tag{1}$$

Our motivation in studying minimum Hamiltonian completion arises from the problem of frequency assignment, an important issue in telecommunications. We look at a problem introduced by Hale [8] and also described in [15, p. 195]. We consider the special case of assigning frequencies in a small geographical area, with two levels of interference, as follows. There are n transmitters $x_1, x_2, \ldots, x_n$. To each, a positive integer $f(x_i)$ is to be assigned, representing a channel or frequency. Pairs of transmitters whose distance is at most a threshold d_0 are said to interfere at level 0, and must be assigned different frequencies. In our special case, all pairs of transmitters lie in a region of diameter d_0, so that all pairs interfere. Transmitters whose distance is at most d_1 $(< d_0)$ interfere at level 1 and must be assigned *nonconsecutive* frequencies, such as 3 and 5—not 3 and 4, for example. The *span* of an assignment is the maximum of $|f(x) - f(y)|$ taken over all pairs of transmitters x and y. The goal is to choose a frequency assignment to minimise the span.

This can be formulated as a graph problem. Define a complete graph G whose vertices are the transmitters. Assign edge (x, y) weight $w(x, y) = 1$ if transmitters x, y interfere only at level 0, and $w(x, y) = 2$ if they also interfere at level 1: thus $w(x, y)$ is a lower bound on $|f(x) - f(y)|$. By adding edge weights along a Hamiltonian path one can construct a valid frequency assignment; moreover, a minimum-weight Hamiltonian path yields a minimum-span frequency assignment, where the span is the weight. A minimum-weight Hamiltonian path must use as few edges of weight two as possible, so finding an assignment of minimum span is

equivalent to finding the Hamiltonian completion number of the subgraph of edges of weight one.

Computation of the Hamiltonian completion number $hc(G)$ of an arbitrary graph is a difficult problem. On the other hand, $hc(G)$ is the minimum value of $hc(T)$ for all spanning trees T (see [1, 6]). Thus, finding the Hamiltonian completion number of a tree is an important problem in the area of frequency assignment.

Further applications of Hamiltonian completion include the optimization of program code [2, 11] and the assignment of processes to distributed processors [14].

The problem of finding minimum Hamiltonian completions was introduced in the 1970s by Boesch, Chen and McHugh [1] and by Goodman and Hedetniemi [6]. Path covers had been studied earlier; for example, in 1960, Gallai and Milgram proved that the path cover number for any digraph cannot exceed its independence number (see [4, p. 39]). Finding optimal Hamiltonian completions or path covers is NP-hard in general, but the authors of both [1] and [6] (independently) found essentially the same polynomial-time algorithm to construct an optimal path cover for a tree. Kundu [11] later showed that the same algorithm could be implemented in linear time. This result has been generalised in several ways since then. Goodman, Hedetniemi and Slater [7] also gave a polynomial-time algorithm to determine $hc(G)$ for a unicyclic graph. Several years later, Karejan and Mosesjan [9] gave a polynomial-time algorithm for acyclic digraphs, and Kornienko [10] reported a linear-time algorithm to solve the problem for any *cactus*, a graph such that no two cycles have a common edge. Moran and Wolfstahl [13] also gave a linear-time algorithm for cacti, generalizing earlier work of Pinter and Wolfstahl [14].

The linear-time algorithm for constructing a minimum path cover for a tree is quite elegant, however [11] gives a terse description without a proof of correctness. The proofs given in [1, 6] show that the underlying algorithm strategy is correct, but do not give efficient implementatons. In this paper, we give a more intuitive derivation of the algorithm, and a proof of correctness that is simpler than those in [1] and [6]. We show also that the algorithm can be extended to unicyclic graphs. These results are given in Section 2.

Our main result is that finding an optimal path cover for a tree can be reduced to finding a maximum flow in a certain network.[4] Moreover, we show that the same reduction can be applied to any bipartite graph to test whether it is possible to cover the vertices of the graph with vertex-disjoint simple cycles. These results are given in Section 3.

One of our main tools is a reduction of the problem of finding a minimum path cover to that of finding a *maximum-edge path cover*, a path cover that contains the maximum number of edges in any path cover for G. The *maximum-edge path cover*

---

[4]Using the classic maximum-flow algorithms of Ford and Fulkerson [5], this immediately yields a quadratic algorithm for finding an optimal path cover or Hamiltonian completion for a tree.
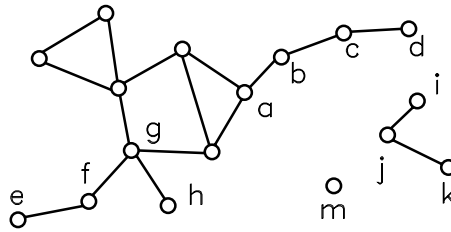
FIGURE 2. Illustration of vines and vine paths. Graph G has three components. Paths ef, dcb, h, ijk and m are vines; vertex g is the only path centre; paths $efgh$, ijk and m are vine paths.

number or $mpc(G)$ is the number of edges in such a cover. It is easy to show that the number of edges in any path cover is always the number of vertices v of G minus the number of paths. Using this observation, one can show that $mpc(G) = v - pc(G)$; combining this with (1) gives the following lemma, whose proof is straightforward.

LEMMA 1. *For any graph G with v vertices, a path cover has the maximum number of edges if and only if it has the minimum number of paths. Moreover,* $mpc(G) = v - pc(G) = v - hc(G) - 1$.

Based on this lemma, for the remainder of the paper we shall use the term *optimal path cover* to mean a path cover with either the minimum number of paths or the maximum number of edges.

2. Constructing optimal path covers for trees

A top-level description of the algorithm   A basic strategy for creating an optimal path cover for a tree or forest is to greedily prune away special paths, which we now define for arbitrary graphs. A *vine* in a graph G is a maximal path such that at least one endpoint is a leaf and each edge (if any) is incident only to vertices of degree 1 or 2. Note that a vine is non-empty, and may be a single leaf, or G itself, if G is a single path. If a vertex of degree at least 3 is adjacent to the endpoints of at least two vines, it will be called a *path centre*. A *vine path* is either a path (vine) that is itself a connected component of the graph, or the path that is induced by a path centre and two of its adjacent vines. See Figure 2 for an illustration of these terms.

The importance of vine paths is described in the following lemma, which shows that one can remove such paths arbitrarily when finding a maximum-edge path cover, and is the heart of the path-cover algorithm described below. If H is a subgraph of G, we use the notation $G - H$ to mean the graph obtained from G by deleting all vertices and edges of H along with any other edges of G incident to vertices in H.
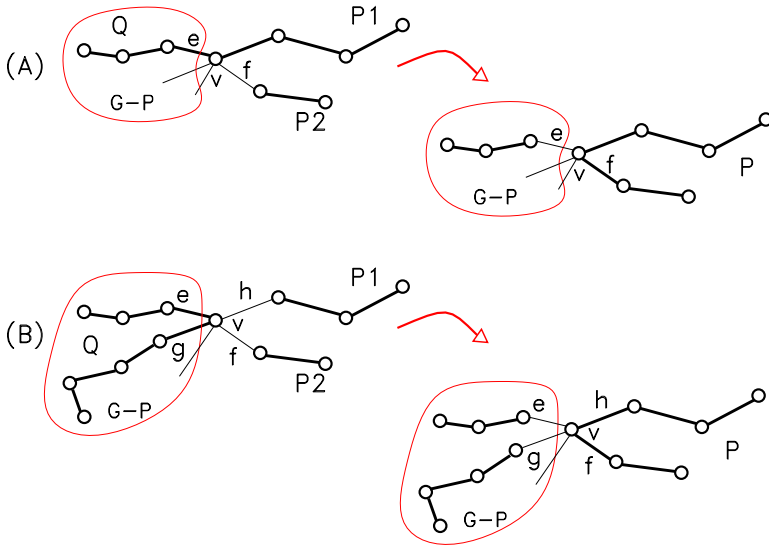
FIGURE 3. Illustration for Lemma 2. Graph G is shown with path centre v which is adjacent to two vines, P_1 and P_2, that induce a vine path P. In each case, paths in an optimal cover are shown as thickened edges. Here Q is the path containing v in the original optimal path cover for G: the figures show how to exchange pairs of edges to create a new optimal path cover containing P. (A) Case 1: If Q contains exactly one of P_1 and P_2, exchange edges e and f. (B) Case 2: If Q contains neither P_1 nor P_2, exchange edges e and g with f and h.

LEMMA 2. *Let G be any graph with a vine path P. Then we may assume that an optimal path cover for G contains P.*

PROOF. Assume that P is not an isolated component, otherwise the result is trivial. Let v be the path centre of P, and let P_1 and P_2 be the two vines contained in P. Let C be an optimal (maximum-edge) path cover for F, and let Q be the path in C that includes v. First observe that since all the vertices of P_1 lie on a unique path in G, either P_1 is in the cover C, or Q contains P_1, and similarly for P_2.

Assume that Q is not equal to P (or we are done), and suppose that Q contains exactly one of P_1 and P_2, say P_1. By our observation, the cover contains P_2. By exchanging one pair of edges incident to v one can create P, as shown in Figure 3 (A); since the exchange preserves the number of edges covered and cannot create a cycle, we have created a new maximum-edge path cover containing P. Otherwise, Q contains neither P_1 nor P_2; by exchanging two pairs of edges, as shown in Figure 3 (B), we again create a new maximum-edge path cover containing P.

Most graphs do not contain vine paths. The next lemma, however, shows that every tree does.

LEMMA 3. *Every* (*non-empty*) *tree contains a vine path.*

PROOF. If T is a single path, then it is itself a vine path. Otherwise, select an arbitrary vertex u as the root of T, and select v, a descendant of degree at least 3 that is farthest from u (possibly u itself). Then any two children of v must be endpoints of vines, so v must be a path centre, and hence lies on a vine path.

Observe that, in general, deleting a vine path from a tree creates a forest; the lemma also guarantees that in any forest, each (nonempty) component tree contains a vine path. Thus, based on Lemmas 2 and 3, an algorithm for producing a maximum-edge path cover for a forest is simply the following.

> **Path Cover Algorithm**
> **Input**: a forest, F.
> Repeat until F is empty:
>   Let P be any vine path in F; add P to the path cover.
>   Continue, replacing F with $F - P$.
> (End repeat) **End**

Extension to unicyclic graphs   It is not hard to extend the algorithm to construct optimal path covers for unicyclic graphs. Let G be a unicyclic graph. After using Lemma 2 repeatedly, we may assume that G contains no vine paths. In that case, assuming G is nonempty, we may assume that G consists of a single cycle C with at most one vine attached to each vertex: examples are shown in Figure 4 (A) or (B).

If $G = C$ or C has exactly one attached vine, it is easy to construct an optimal path cover with one path (and $|G| - 1$ edges). If C has a vine attached to each vertex, as in Figure 4 (A); we must delete at least one edge from C, which, by symmetry, may be any edge. If there is a vertex v with no incident vine, let P and Q be the two vines closest to v. Using an exchange argument similar to that in the proof of Lemma 2, one can show that there is an optimal cover containing the path induced by P, v and Q, which can be removed from G. In each case, the new graph is a tree, whose path cover can be constructed as described above.

Linear-time implementation for trees   We now describe a linear-time implementation of the Path-Cover Algorithm for a tree (or forest). Assume that tree T has its edges given as adjacency lists. The output will be a labelling of the edges as *In* or *Out* (of the cover). The paths can be easily recovered in linear time and space by finding the connected components of the graph using the *In* edges only.

Our strategy is simply to implement Lemma 3. We first perform a breadth-first search (BFS) (see, for example, [17, Section 3.2]) to number the vertices of T in order of their distance from the root, which is chosen arbitrarily. See Figure 5 (A) for a BFS numbering of a tree. The vertices are processed in reverse BFS order so that each time
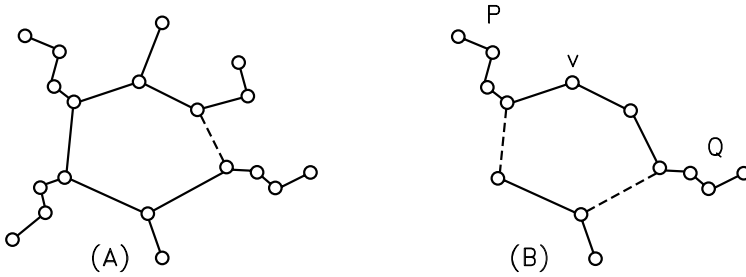
FIGURE 4. Graph G is a cycle C with vines incident to two or more vertices. (A) (Case 1) If every vertex of C is incident to a vine, delete any edge of C (as shown by the dashed line) and proceed with the resulting tree. (B) (Case 2) If some vertex v has no incident vine, let P and Q be the two vines closest to v; add the path induced by P, v and Q to the cover by deleting the dashed edge(s) as shown and proceed with the resulting tree.

we encounter a vertex of degree 3 or more, we will know that it is a path centre. The core of the algorithm, called *Path-Cover* (T), is the subroutine *Process* (v), described right after the main algorithm, which is as follows.

> **Path-Cover** (T)
> **Input**: Tree T; List of pointers to the vertices in reverse BFS order;
>   For each vertex v, a pointer Parent (v) to its parent in the rooted tree.
> **Initialise**: Label all edges *In*; Degree $(v) :=$ number of edges incident to v;
> **For** each vertex v on the list, **Process** (v); **End For**
> **End**
>
> **Process** (v)
> **If** Degree (v) is at least 3
>   Let x, y be any two children of v {with vx, vy labeled *In*}
>   **For** each $z \neq x$, y, with edge vz labeled *In*
>   Label vz *Out*; Degree $(z) :=$ Degree $(z) - 1$
>   **End For**
>   Degree $(v) := 2$
> {*Otherwise, do nothing*} **Return**

The algorithm is illustrated in Figure 5. Note that the procedure Process (v) is defined whether or not v is the root; if v is not the root and has degree at least 3, the edge between v and its parent will be labeled *Out*. The implementation clearly uses linear time and space; the correctness is established in the proof of the following theorem.

THEOREM 1. *For any tree (forest) with n vertices, the algorithm Path-Cover (T) produces an optimal path cover using $O(n)$ time and space.*
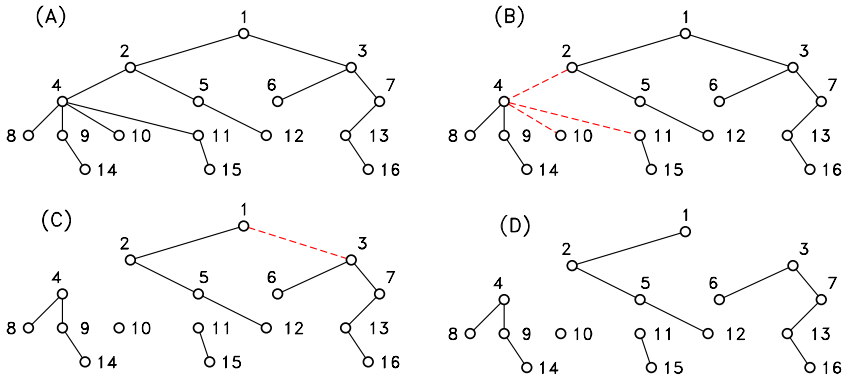
FIGURE 5. Illustration of the algorithm Path-Cover (T). (A) A tree with a breadth-first search (BFS) numbering of the vertices. (B) When Vertex 4 is processed, three edges are deleted (labeled *Out*); these are shown as dashed lines. (C) When Vertex 3 is processed, only its parent edge is deleted. (D) The path cover found by the algorithm.

PROOF. We only need to show that after algorithm Path-Cover (T) has been performed, the vertices of T along with the edges labeled *In* form an optimal path cover. First, it is easy to show that the following invariant holds, which is the key to the proof.

Invariant. If the subroutine Process (v) is called and vertex x was processed before v, then Degree (x) is 0, 1 or 2.

In the next part of the proof, $T(v)$ represents the (modified) tree just after Process (v) is called, that is, the tree induced by the root and all edges labeled *In*. Also $C(v)$ represents the remaining connected components induced by the edges labeled *In*. In order to show that Process (v) works correctly, we prove the following claim by induction.

Claim. Throughout the algorithm Path-Cover (T), every component in $C(v)$ is a path in an optimal cover.

To prove the claim, we first note that if v is the first vertex processed it is a leaf and must have degree 0 or 1; thus $C(v)$ is empty and $T(v) = T$. Now suppose that we have just finished running Process (x), and are about to call Process (v). Assume that the claim holds for $C(x)$. Observe that every vertex that has not yet been processed is still a member of $T(x)$; this is true because the BFS numbering guarantees that a vertex can only be deleted from the tree when it is processed or when one of its ancestors is processed later. In particular, v is in $T(x)$.

If Degree $(v) \leq 2$ then $C(v) = C(x)$, so we may assume that Degree $(v) \geq 3$. Then the invariant guarantees that all descendants of v in $T(x)$ must have degree 1 or 2, and
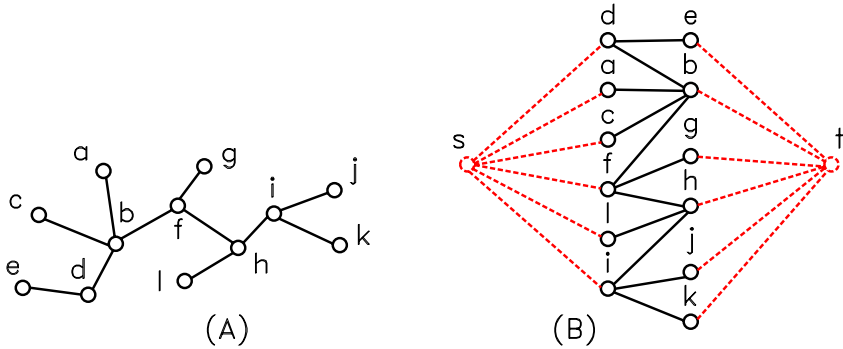
FIGURE 6. (A) Tree T; (B) the network $N(T)$ is formed by creating a bipartition (X, Y) of the vertices of T, then adding new vertices s and t along with new arcs to all vertices in X or Y, respectively (shown with dashed lines). All arcs from original edges have capacity 1; the new arcs have capacity 2.

hence lie on vines; therefore v is a path centre for some path P in $T(x)$.

When Process (v) is complete, if v is not the root, $C(v)$ will now include P, as well as all of the vine paths (vines) in $T(x) - P$ whose endpoints are the children of v not on P. By Lemma 2, these are all paths in an optimal cover, which establishes the Claim. If v is the root, then the invariant guarantees that $T(v) = P$. By Lemma 2, $T(v) \cup C(v)$ is an optimal path cover for T.

Notice that the only place the BFS numbering is used is to guarantee that all descendants of a vertex v are processed before v. Thus any numbering with this property, such as a depth-first search numbering (see, for example, [17]) will also work in the algorithm.

The following corollary follows immediately from Theorem 1 and Lemma 1.

COROLLARY 1. *A minimum Hamiltonian completion or path cover of a tree (forest) can be constructed in linear time and space.*

3. Path covers via maximum flow

In this section, we show that finding an optimal path cover for a tree T (and thus an optimal Hamiltonian completion for T) can be reduced to finding the maximum flow in a special directed network $N(T)$ with edge capacities, which we now define. We assume throughout that T has at least one edge. The construction also works for a forest.

Since T is bipartite, we can create a bipartition of the vertices of T into nonempty sets X and Y such that all edges join vertices of X to vertices of Y. Orient all of these

edges from X to Y to create arcs, and give them capacity 1. Next, add a vertex s, with an arc oriented from s to each vertex of X; add a corresponding vertex t with an arc oriented from each vertex in y to t. Assign all of these new arcs capacity 2. An example is shown in Figure 6. In the following theorem, we show that a maximum flow in $N(T)$ can be used to compute the path cover number or Hamiltonian completion number of T. The proof gives a construction of an optimal path cover from an integral maximum flow.

THEOREM 2. *Let $f^\star$ be the value of a maximum s-t flow in the network $N(T)$ where T is a tree with at least two vertices. Then $f^\star = mpc(T)$, the maximum-edge path cover number of T.*

PROOF. To prove the theorem, we will give a correspondence between flows and path covers, which is illustrated in Figure 7.

First, let $f^\star$ be any maximum flow such that the flow in each arc is an integer.[5] Let C be the subgraph of T consisting of all the vertices of T and the XY edges with flow 1 in $f^\star$. Each vertex in X or Y can have degree 0, 1 or 2 in C; and, since T is acyclic, C must be a set of vertex-disjoint paths that cover the vertices of T and which contain $f^\star$ edges. Hence $mpc(T) \geq f^\star$.

Now let $\mathrm{PC}(T)$ be a maximum-edge path cover for T. If edge xy is in $\mathrm{PC}(T)$ then assign flow 1 to the corresponding arc in $N(T)$. In $N(T)$, let the flow for each arc sx with $x \in X$ be the degree of x in $\mathrm{PC}(T)$ (which can be 0, 1 or 2), and assign flow similarly for each arc yt with $y \in Y$. It is easy to check that this is a legal s-t flow. Since the flow value is equal to the number of edges in $\mathrm{PC}(T)$, which is $mpc(T)$, $mpc(T) \leq f^\star$.

Lemma 1 yields the following corollary.

COROLLARY 2 (to Theorem 2). *If $f^\star$ is the value of a maximum s-t flow in $N(T)$, then the path cover number of T is $v - f^\star$ and the Hamiltonian completion number of T is $v - 1 - f^\star$.*

The corollary yields an $O(v^2)$ algorithm for computing the Hamiltonian completion number or path cover number, using the maximum-flow implementation of Ford and Fulkerson [5], as described in Roberts [16]. The algorithm in fact constructs an optimal path cover, and hence yields an optimal Hamiltonian completion.

The construction of $N(T)$ can be extended easily to any bipartite graph G. However, the construction of the set C given in the proof of Theorem 2 from a maximum flow in

[5]Since all the capacities are integers, such a flow exists (see, for example, Lawler [12, Theorem 2.2, p. 113]); in fact, the Ford and Fulkerson labelling and scanning algorithm [5] produces an integral maximum flow.
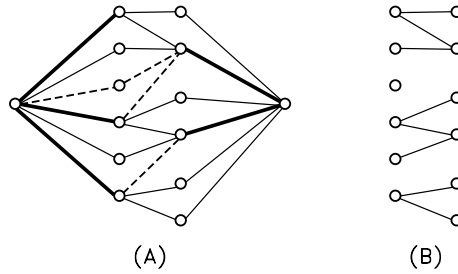
FIGURE 7. (A) The network $N(T)$ with an integral maximum flow of value 8: arcs with flow 2 are shown as thick lines, those with flow 1 as thin lines, and those with flow 0 as dashed lines. (B) The corresponding path cover in the tree T.

$N(G)$ may not yield a path cover: instead, the construction gives a *path-cycle cover*, a set of vertex-disjoint paths and cycles that cover the vertices of G. The same argument in Theorem 2 yields the following result for bipartite graphs.

LEMMA 4. *Let G be a bipartite graph with at least one edge and let $f^\star$ be the value of a maximum s-t flow in the network $N(G)$. Then $f^\star$ is also the maximum number of edges in any path-cycle cover of G.*

Using an argument similar to that used for Lemma 1, it is not hard to show the following result.

LEMMA 5. *The maximum number of edges in a path-cycle cover for any graph G is $v - k$, where k is the minimum number of paths in any path-cycle cover.*

Combining these two lemmas yields the following interesting result, which gives a method for testing whether a bipartite graph has a covering of the vertices by disjoint simple cycles (also called a 2-*factor*).

THEOREM 3. *Let $f^\star$ be the value of a maximum s-t flow in the network $N(G)$ where G is a bipartite graph with at least two vertices. Then G has a covering of the vertices by disjoint simple cycles if and only if $f^\star = v$.*

References

[1] F. T. Boesch, S. Chen and N. A. M. McHugh, "On covering the points of a graph with point disjoint paths", in *Graphs and combinatorics* (eds. A. Dold and B. Eckman), Lecture Notes in Math. 406, (Springer, Berlin, 1974) 201–212.

[2] F. T. Boesch and J. F. Gimpel, "Covering the points of a digraph with point-disjoint paths and its application to code optimization", *JACM* **24** (1977) 192–198.

[3] J. A. Bondy and U. S. R. Murty, *Graph theory with applications* (North Holland, NY, 1976).

[4] R. Diestel, *Graph theory* (Springer, NY, 1997).

[5] L. R. Ford and D. R Fulkerson, "A simple algorithm for finding maximal network flows and an application to the Hitchcock problem", *Canadian J. Math.* **9** (1957) 210–218.

[6] S. E. Goodman and S. T. Hedetniemi, "On the Hamiltonian completion problem", in *Graphs and combinatorics* (eds. A. Dold and B. Eckman), Lecture Notes in Math. 406, (Springer, Berlin, 1974) 262–272.

[7] S. E. Goodman, S. T. Hedetniemi and P. J. Slater, "Advances on the Hamiltonian completion problem", *JACM* **22** (1975) 352–360.

[8] W. K. Hale, "Frequency assignment: Theory and applications", *Proc. IEEE* **68** (1980) 1497–1514.

[9] Z. A. Karejan and K. M. Mosesjan, "The Hamiltonian completion number of a digraph", *Akad. Nauk Armyan. SSR Dokl.* **70** (1980) 129–132 (in Russian).

[10] N. M. Kornienko, "The Hamiltonian completion of some classes of graphs", *Vestsī Akad. Navuk BSSR, Ser. Fīz.-Mat. Navuk* **124** (1982) 18–22 (in Russian).

[11] S. Kundu, "A linear algorithm for the Hamiltonian completion number of a tree", *Info. Proc. Letters* **5** (1976) 55–57.

[12] E. L. Lawler, *Combinatorial optimization: Networks and matroids* (Holt, Rinehart, and Winston, NY, 1976).

[13] S. Moran and Y. Wolfstahl, "Optimal covering of cacti by vertex-disjoint paths", *Theoret. Comp. Sci.* **84** (1991) 179–197.

[14] S. S. Pinter and Y. Wolfstahl, "On mapping processes to processors in distributed systems", *Internat. J. Parallel Prog.* **16** (1987) 1–15.

[15] A. Raychaudhuri, "Intersection assignments, *T*-colorings, and powers of graphs", Ph. D. Thesis, Rutgers Univ., 1985.

[16] F. S. Roberts, *Applied combinatorics* (Prentice-Hall, Englewood Cliffs, NJ, 1984).

[17] A. Tucker, *Applied combinatorics* (Wiley, NY, 1980).