Efficient Algorithms for Constructing Testing Sets, Covering Paths, and Minimum Flows

Alfred V. Aho

David Lee

AT&T Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

Although testing is an essential part of program and circuit design, the area is still more an art than a science. This paper considers several fundamental problems arising in program and circuit testing, and abstracts them in terms of path-covering problems on graphs. These problems are representative of important classes of graph-optimization problems, and we introduce a technique called ‘‘balancing’’ to solve these problems. This technique yields algorithms that are faster, simpler, and easier to implement than those obtained by applying existing methods. Included in the classes of problems are minimum network-flow problems and the Chinese-postman problem.

1. Testing Sets, Covering Paths, and Minimum Flows

Testing is an important part of the process of constructing reliable programs and circuits. To test a program one would like to construct a set of inputs that exercises, if possible, each edge (and/or node) in the program flow graph at least once. Furthermore, one would like to minimize the cost of the test, such as the size of the test suite or the time required to execute the test. Many good programmers have attested to the difficulty of constructing comprehensive test suites [Knuth, 1984].

To test a logic circuit, one can sensitize a set of paths such that each node is on at least one path. A node can be a logic element (combinational or sequential) or a block of such elements. To reduce the cost of testing, one would like to minimize the number of paths sensitized. One may also want to minimize the total length of the paths to reduce the complexity of the test [Seth and Agrawal, 1985; Uyar and Dahbura, 1986].

To simplify the problem, we will represent a program flow graph or logic circuit as a directed graph, $G = [V \cup \{ s, t \}, E]$, with n vertices and m edges. The distinguished vertex s is called a *source*; it has in-degree zero and corresponds to a unique entry point. The distinguished vertex t is called a *sink*; it has out-degree zero and corresponds to a unique exit point. The vertices in V are called *intermediate* vertices. Each edge e in E has a lower and an upper capacity bound $l(e)$ and $u(e)$, respectively, where $0 \le l(e) \le u(e)$, and also has an associated nonnegative cost, $cost(e)$.

A modification of the techniques presented here can also provide algorithms with the same time bounds for problems in which a cost and/or capacity lower and upper bound is assigned to each vertex. For simplicity, however, we will consider directed graphs where only edges have associated costs and capacity bounds.

A $v_0 - v_r$ path p in G is a sequence of vertices $v_0, \ldots, v_r$ such that there is a sequence of edges $e_0, \ldots, e_{r-1}$, where $e_i = (v_i, v_{i+1})$, for $i = 0, \ldots, r-1$. For such a $v_0 - v_r$ path p, we define $cost(p)$ to be $\sum_{i=0}^{r-1} cost(e_i)$. A set of *covering* paths for G is a set of $s - t$ paths such that each edge e is traversed at least once and no more than $u(e)$ times, where $u(e) \ge 1$. The basic versions of the testing problems can be couched as covering-paths problems:

Problem $CP_1^{(1)}$. *Minimum-cardinality covering paths*.

Find a set of covering paths for G such that the number of paths is minimum.

Problem $CP_2^{(1)}$. *Minimum-cost minimum-cardinality covering paths*.

Among the sets of minimum-cardinality covering paths, find one with minimum cost.

Problem $CP_3^{(1)}$. *Minimum-cost covering paths*.

Find a minimum-cost set of covering paths for G.

Throughout this paper, three forms of each problem will be considered. These forms will be distinguished by a subscript, where a value of one denotes a minimum-cardinality problem, a value of two a minimum-cost minimum-cardinality problem, and a value of three a minimum-cost problem. In addition, we will consider several versions of each problem form depending on what constraints, if any, are placed upon the edge capacity bounds. The version in which, for every edge e, $l(e) = 1$ and $u(e) \le \infty$ will be denoted by the superscript (1). For the purpose of program or logic circuit testing, we can let $l(e) = 1$ and $u(e) = \infty$. This version will be denoted by the superscript (2). A third version, denoted by the superscript (0), where $0 \le l(e) \le u(e) \le \infty$ will be considered later.

Problem $CP_1^{(2)}$ was studied by Ntafos and Hakimi [1979] in the context of program testing, but if one wants to minimize the cost of the test, then solutions to Problems $CP_2^{(2)}$ and $CP_3^{(2)}$ are also needed. Special cases of these problems have been considered by Krause, Smith, and Goodwin [1973], Gabow, Maheshwari, and Osterweil [1976], and Ntafos and Hakimi [1979].

Here we consider the more general covering-paths problems on directed graphs with arbitrary capacity upper bounds. In Section 4 we show that these covering-paths problems are equivalent to *positive-flow* problems, which are special cases of *minimum-flow* problems [Even, 1979]. One way to solve these minimum-flow problems is to reduce them to a sequence of maximum-flow problems as in Ford and Fulkerson [1962], Lawler [1976], or Even [1979]. Following this approach, we can find an algorithm for Problem $CP_1^{(1)}$ with cost $O(mn \log n)$, and algorithms for Problems $CP_2^{(1)}$ and $CP_3^{(1)}$ with cost $O(n^2(m + n \log n) \log n)$ using the methods of Tarjan [1983] and Galil and Tardos [1986].

Instead, we use *balancing* to solve these minimum-flow problems. We first reduce the minimum-flow problems to minimum-cost maximum-flow problems, but the reductions are different from those previously used [Ford and Fulkerson, 1962; Lawler, 1976; Even, 1979]. Using balancing, the reduced flow problem has an obvious bound on the flow value, and efficient algorithms can be devised, such as minimum-cost augmentation and scaling. Furthermore, our approach is intuitive and easy to implement. To our knowledge, there has been no formal presentation of this approach. We feel it is worthwhile to explore this method and to apply it to practical problems, since the ideas involved are simple.

We also present methods for minimizing both the flow value and the cost, which have not been well studied. Balancing can also be used for *circulation* problems. As special cases, we present algorithms for covering-paths and postman-tour problems. For example, we obtain an $O(n(m + n \log n) \log(m/n))$ algorithm for the *Chinese-postman* problem. For this problem, the best previously published algorithm was $O(n^5)$ [Papadimitriou, 1976], and an $O(mn \log n)$ algorithm can be obtained by reducing the problem to a minimum-cost flow problem using balancing and solving the reduced problem by scaling [Gabow and Tarjan, 1987]. Our algorithm is faster than either of these methods. Several versions of the covering-paths problems and the postman-tour problems can also be solved with the same run time.

We also study these problems on mixed graphs. For example, we show that Problem $CP_1^{(j)}$ remains polynomial, but Problems $CP_i^{(j)}$, $i = 2, 3$, and $j = 1, 2$, become NP-complete.

We begin by providing algorithms for the minimum-flow problems using balancing. Then we specialize the algorithms to *positive-flow* problems, which are equivalent to the covering-paths problems. Finally, we consider several versions of minimum-circulation and postman-tour problems.

2. Minimum-Flow Problems

A *preflow* f on a directed graph $G = [V \cup \{ s, t \}, E]$ is a function from E to nonnegative integers. We define *cost*(f) as $\sum_{e \in E} f(e)\, cost(e)$. The *balancing index* of a vertex v in G with a preflow f is

$$\beta(v, f) = \sum_{(v,w) \in E} f(v, w) - \sum_{(u,v) \in E} f(u, v)$$

We define *value*(f) to be $\beta(s, f)$.

A *flow* on G is a preflow f such that $l(e) \le f(e) \le u(e)$ for all e in E and such that the balancing index $\beta(v, f)$ is zero for all intermediate vertices v in V.

We can now consider the following minimum-flow problems:

Problem $F_1^{(0)}$. *Minimum flow*.

Find a flow for G with the minimum flow value.

Problem $F_2^{(0)}$. *Minimum-cost minimum flow*.

Among the minimum flows for G, find one with minimum cost.

Problem $F_3^{(0)}$. *Minimum-cost flow*.

Find a flow for G with minimum cost.

The superscript (0) indicates that there are no constraints on the capacity bounds on the edges; that is, for each edge e, $0 \le l(e) \le u(e) \le \infty$.

In the next section, we provide algorithms for these minimum-flow problems using balancing whenever they have a feasible solution [Lawler, 1976; Even, 1979]. The conventional approach has been to reduce a minimum-flow problem to a maximum-flow problem, then to a second maximum-flow problem, and if this has a feasible solution, optimize it by solving a third maximum-flow problem. On the other hand, with balancing we can find an optimal solution directly by solving a maximum-flow problem on a modified graph.

3. Algorithms for Minimum-Flow Problems

We first show that the minimum-flow problems are equivalent to preflow-minimization problems, which can be reduced to maximum-flow problems on a so-called *balancing graph* with only capacity upper bounds on the edges. Balancing-preflow problems are introduced to make the reduction conceptually clear. For practical purposes, one can construct the balancing graphs by a simple modification of the original graph, and then solve the final maximum-flow problems directly.

3.1. Balancing-Preflow Problems

A preflow f_0 is *basic* if for every e in E, $f_0(e) = l(e)$, where $l(e)$ is the capacity lower bound on edge e. We partition the set of intermediate vertices V into three sets: S (surplus), B (balanced), and D (deficient). An intermediate vertex v is in S, B, or D depending on whether $\beta(v, f_0)$ is less than, equal to, or greater than zero, respectively. A preflow f_b is *balancing* if $f_b(e) \le u(e) - l(e)$ for all edges e in E, and $\beta(v, f_b) = -\beta(v, f_0)$ for all intermediate vertices v in V. Note that a balancing preflow may not exist because of the capacity bounds on the edges.

Theorem 3.1. (Decomposition of a feasible flow). f is a flow on G if and only if $f = f_0 + f_b$, where f_0 is the basic preflow and f_b is a balancing preflow on G.

Proof. Let f_b be a balancing preflow. Since $l(e) \le f_0(e) + f_b(e) \le u(e)$ for all e in E, and $\beta(v, f_0 + f_b) = \beta(v, f_0) + \beta(v, f_b) = 0$ for all intermediate vertices v, $f_0 + f_b$ is a feasible flow.

On the other hand, given a flow f on G, let $f_b = f - f_0$, where f_0 is the basic preflow. Since $l(e) \le f(e) \le u(e)$ for all e in E, f_b is a preflow. For all intermediate vertices v, $\beta(v, f_b) = \beta(v, f) - \beta(v, f_0) = 0 - \beta(v, f_0)$ and therefore f_b is balancing. □

Since the basic preflow depends only on G and has fixed cost and flow value, we have:

Theorem 3.2. The minimum-flow problems $F_i^{(0)}$, for $i = 1, 2, 3$, are equivalent to the following corresponding balancing-preflow problems:

Problem BF_1. *Minimum balancing preflow*.

Find a balancing preflow for G with minimum value.

Problem BF_2. *Minimum-cost minimum balancing preflow*.

Among the minimum balancing preflows, find one with minimum cost.

Problem BF_3. *Minimum-cost balancing preflow*.

Find a balancing preflow for G with minimum cost.

3.2. Balancing Graphs

From the directed graph G, we construct a *balancing graph* $\overline{G}$, and the balancing-preflow problems on G can be reduced to corresponding maximum-flow problems on $\overline{G}$. To construct $\overline{G}$, we add a new source vertex s' and a new sink vertex t' to G. The following edges are also added to G:

i) Add the edges (s', s) and (t, t').

ii) For each v in S, add the edge (s', v) with capacity upper bound $\overline{u}(s', v) = -\beta(v, f_0)$.

iii) For each w in D, add the edge (w, t') with capacity upper bound $\overline{u}(w, t') = \beta(w, f_0)$.

The cost of each edge in G remains what it was, and the cost of each added edge is zero. The capacity lower bound of all the edges in $\overline{G}$ is zero. For each edge e in G, the capacity upper bound becomes $\overline{u}(e) = u(e) - l(e)$. The capacity upper bounds of (s', s) and (t, t') will be specified later. Obviously, $\overline{G}$ has $O(m)$ edges and $O(n)$ vertices.

Given a flow F on $\overline{G}$, the *induced* preflow f on G is defined as $f(e) = F(e)$ for all $e \in E$. We now reduce the balancing-preflow problems to maximum-flow problems on balancing graphs.

3.3. Minimum flow

To present the algorithms, we use the concept of a *residual graph* of a flow. Let f be a flow on G with capacity lower bound zero and upper bound u. The *residual capacity* (upper bound) of (v, w) in E is $\overline{u}(v, w) = u(v, w) - f(v, w)$. If (v, w) is in E but (w, v) is not, then add an edge (w, v) to G with capacity upper bound $\overline{u}(w, v) = f(v, w)$. The residual graph $R(f)$ from a flow f is the graph with vertex set V, source s, sink t, and an edge (v, w) of capacity $\overline{u}(v, w)$, where $\overline{u}(v, w) > 0$. The costs of the edges in $R(f)$ are defined as $\Delta(v, w) = cost(v, w)$ if $(v, w) \in E$, and $\Delta(v, w) = -cost(w, v)$ if $(w, v) \in E$. We need the following well-known lemma, see, for example, [Tarjan, 1983].

Lemma 3.1. Let f be any flow and f^* a maximum flow on G. If $R(f)$ is the residual graph from f, then the value of a maximum flow on $R(f)$ is $value(f^*) - value(f)$. □

To find a minimum flow, we use the following step-wise balancing on $\overline{G}$. When finding the maximum flows, we use the Sleator-Tarjan algorithm [Tarjan, 1983]. The induced preflow on G is a minimum balancing preflow.

Algorithm 1. Minimum flow.

1. Construct the balancing graph $\overline{G}$ and set $\overline{u}(s', s) = \overline{u}(t, t') = 0$.
2. Find a maximum flow F_1 on $\overline{G}$, and let R_1 be the residual graph.
3. Set $\overline{u}(s', s) = \infty$ and $\overline{u}(v, s') = 0$, $v \in S$, in R_1, and let $\overline{R}_1$ be the modified graph.
4. Find a maximum flow f_2 on $\overline{R}_1$. Let R_2 be the residual graph, and let $F_2 = F_1 + f_2$.
5. If $value(F_2) < \sum_{w \in D} \beta(w, f_0)$, then abort. There is no feasible flow on G.
6. Set $\mu = \overline{u}(s', s) = F_2(s', s)$ and $\overline{u}(t, t') = \infty$ in R_2, and let $\overline{R}_2$ be the modified graph.
7. Find a maximum flow f_3 on $\overline{R}_2$, and let $F_3 = F_2 + f_3$.

8. If $value(F_3) < \mu + \sum_{v \in S} [-\beta(v, f_0)]$, then abort. There is no feasible flow on G.

9. Let f_b be the induced preflow of F_3 on G, and let $f = f_b + f_0$, where f_0 is the basic preflow; f is a minimum flow on G. □

Intuitively, the minimum balancing preflow is constructed from three-stage balancing. Steps 1 and 2 balance vertices in S and D as much as possible without using flow from s or t. Steps 3 to 5 balance the remaining vertices in D using flow from s only, and Steps 6 to 8 balance the remaining vertices in S using flow from t only. To analyze Algorithm 1, we need the following lemma.

Lemma 3.2. Let $f_b{}^*$ be a minimum balancing preflow on G with value μ^*. Then

i) there exists a flow $F_1{}^*$ on $\overline{G}$ with $\overline{u}(s', s) = \overline{u}(t, t') = 0$ such that $value(F_1{}^*) = \sum_{w \in D} \beta(w, f_0) - \mu^*$;

ii) there exists a flow $F_2{}^*$ on $\overline{G}$ with $\overline{u}(s', s) = \infty$ and $\overline{u}(t, t') = 0$, such that $value(F_2{}^*) = \sum_{w \in D} \beta(w, f_0)$; and

iii) there exists a flow $F_3{}^*$ on $\overline{G}$ with $\overline{u}(s', s) = \mu^*$ and $\overline{u}(t, t') = \infty$ such that $value(F_3{}^*) = \mu^* + \sum_{v \in S} [-\beta(v, f_0)]$.

Proof. We define a flow $F_3{}^*$ on $\overline{G}$ with $\overline{u}(s', s) = \mu^*$ and $\overline{u}(t, t') = \infty$ as follows. Let $F_3{}^*(e) = f_b{}^*(e)$ for $e \in E$, and let $F_3{}^*(s', s) = \mu^*$, $F_3{}^*(s', v) = \beta(v, f_b{}^*)$, $v \in S$, $F_3{}^*(t, t') = -\beta(t, f_b{}^*)$, and $F_3{}^*(w, t') = -\beta(w, f_b{}^*)$, $w \in D$. Then $F_3{}^*$ is a flow on $\overline{G}$ with $\overline{u}(s', s) = \mu^*$ and $\overline{u}(t, t') = \infty$. We call F the *derived* flow on $\overline{G}$ from $f_b{}^*$. Since $f_b{}^*$ is balancing, $value(F_3{}^*) = F_3{}^*(s', s) + \sum_{v \in S} F_3{}^*(s', v) = \mu^* + \sum_{v \in S} \beta(v, f_b{}^*) = \mu^* + \sum_{v \in S} [-\beta(v, f_0)]$. Part (iii) has now been proved.

We successively reduce the minimum balancing preflow $f_b{}^*$ along $v-t$ paths, where $v \in S$, such that the reduced preflow $\overline{f}_b{}^*$ has $\beta(t, \overline{f}_b{}^*) = 0$. Similarly, let $F_2{}^*$ be the derived flow from $\overline{f}_b{}^*$ on $\overline{G}$ with $\overline{u}(s', s) = \infty$ and $\overline{u}(t, t') = 0$. Since $f_b{}^*$ is balancing, and only the balancing indices of v in S were changed during the reduction of $f_b{}^*$, $F_2{}^*(w, t') = -\beta(w, \overline{f}_b{}^*) = -\beta(w, f_b{}^*) = \beta(w, f_0)$, $w \in D$. Now, $value(F_2{}^*) = \sum_{w \in D} F_2{}^*(w, t') = \sum_{w \in D} \beta(w, f_0)$, and Part (ii) has been established.

We further reduce $\overline{f}_b{}^*$ along $s-w$ paths, where $w \in D$, such that the reduced preflow $\overline{\overline{f}}_b{}^*$ has $\beta(s, \overline{\overline{f}}_b{}^*) = 0$. Let $F_1{}^*$ be the derived flow from $\overline{\overline{f}}_b{}^*$ on $\overline{G}$ with $\overline{u}(s', s) = \overline{u}(t, t') = 0$. Similarly, we have $value(F_1{}^*) = \sum_{w \in D} \beta(w, f_0) - \mu^*$, and Part (i) has been proved. □

Theorem 3.3. Algorithm 1 determines the existence of a flow on G and constructs a minimum flow f in time $O(mn \log n)$.

Proof. We first show that if the algorithm aborts at Steps 5 or 8, there is no feasible flow on G. We then show that the preflow f_b constructed in Step 9 is a minimum balancing preflow and, therefore, f is a minimum flow on G.

In Step 4, we construct a maximum flow on graph $\overline{R}_1$ after setting $\overline{u}(v, s')$ to zero for each $v \in S$. Since t' is not reachable from s' in R_2, and setting $\overline{u}(v, s')$ back to $F_1(s', v)$ does not provide any $s'-t'$ paths in R_2, F_2 is a maximum flow on $\overline{G}$ with $\overline{u}(s', s) = \infty$ and $\overline{u}(t, t') = 0$.

Assume that Algorithm 1 aborts at Step 5 and that there exist balancing preflows on G. Let $f_b{}^*$ be a minimum balancing preflow with value μ^*. Then by (ii) of Lemma 3.2, there exists a flow $F_2{}^*$ on $\overline{G}$ with $\overline{u}(s', s) = \infty$ and $\overline{u}(t, t') = 0$, with $value(F_2{}^*) = \sum_{w \in D} \beta(w, f_0)$. On the other hand, the maximum flow F_2 on $\overline{G}$ with $\overline{u}(s', s) = \infty$ and $\overline{u}(t, t') = 0$ has $value(F_2) < \sum_{w \in D} \beta(w, f_0) = value(F_2{}^*)$. This is a contradiction. Therefore, if Algorithm 1 aborts at Step 5, there is no balancing preflow on G, and by Theorem 3.2, there is no feasible flow on G.

Assume that Algorithm 1 aborts at Step 8 and that there exist balancing preflows on G. Let $f_b{}^*$ be a minimum balancing preflow with value μ^*. Then by (i) of Lemma 3.2, there exists a flow $F_1{}^*$ on $\overline{G}$ with

$\bar{u}(s', s) = \bar{u}(t, t') = 0$, with $value(F_1{*}) = \sum_{w \in D} \beta(w, f_0) - \mu{*}$. Since Algorithm 1 did not abort at Step 5, $value(F_2) = \sum_{w \in D} \beta(w, f_0)$. Since F_1 is a maximum flow on $\overline{G}$ with $\bar{u}(s', s) = \bar{u}(t, t') = 0$, $value(F_1) = value(F_2) - \mu = \sum_{w \in D} \beta(w, f_0) - \mu \geq value(F_1{*}) = \sum_{w \in D} \beta(w, f_0) - \mu{*}$. Thus $\mu \leq \mu{*}$.

Since F_3 is a maximum flow on $\overline{G}$ with $\bar{u}(s', s) = \mu$ and $\bar{u}(t, t') = \infty$, a maximum flow on $\overline{G}$ with $\bar{u}(s', s) = \mu{*}$ and $\bar{u}(t, t') = \infty$ has value no more than $(\mu{*} - \mu) + value(F_3) < (\mu{*} - \mu) + \{\mu + \sum_{v \in S} [-\beta(v, f_0)]\} = \mu{*} + \sum_{v \in S} [-\beta(v, f_0)]$. That is, a maximum flow on $\overline{G}$ with $\bar{u}(s', s) = \mu{*}$ and $\bar{u}(t, t') = \infty$ has value less than $\mu{*} + \sum_{v \in S} [-\beta(v, f_0)]$. This contradicts Part (iii) of Lemma 3.2. Therefore, if Algorithm 1 aborts at Step 8, there is no balancing preflow on G, and by Theorem 3.2, there is no feasible flow on G.

From Steps 5 and 8, F_3 saturates all the edges (s', v) and (w, t'), $v \in S$ and $w \in D$. Therefore, the induced preflow f_b constructed in Step 9 is balancing. From the arguments in the previous paragraph, $value(f_b) = \mu \leq \mu{*}$, where $\mu{*}$ is the value of a minimum balancing preflow. Therefore, f_b is a minimum balancing preflow.

It takes time $O(mn \log n)$ to construct a maximum flow using the Sleator-Tarjan algorithm in Steps 2, 4, and 7. The other steps take time $O(m)$. □

3.4. Minimum-cost minimum flow

The algorithm for finding a minimum-cost minimum flow for G is based on the output of Algorithm 1, which computes the value of a minimum balancing preflow μ, if it exists. Since for any balancing preflow f_b, $\beta(s, f_0) + \beta(s, f_b) = -[\beta(t, f_0) + \beta(t, f_b)]$, let $\nu = -\beta(t, f_b) = \mu + \beta(s, f_0) + \beta(t, f_0)$. From G we construct a balancing graph $\overline{G}$, in which the capacities of edges (s', s) and (t, t') are μ and ν, respectively. Let F be the derived flow on $\overline{G}$ from a minimum balancing preflow. Then F saturates all the edges incident to s' or t'. Therefore, a flow F on $\overline{G}$ is maximum if and only if all the edges incident to s' or t' are saturated. This is the case if and only if the induced preflow f_b of F is balancing and has the minimum value μ. Since $cost(F) = cost(f_b)$, we have:

Lemma 3.3. If there exists a balancing preflow on G, then a flow F on $\overline{G}$ is a minimum-cost maximum flow if and only if the induced preflow f_b on G is a minimum-cost minimum balancing preflow.

□

To construct the minimum-cost maximum flows on $\overline{G}$, different techniques can be used. Note that from the capacity constraints of the edges incident to s' in $\overline{G}$, the flow values are bounded by $L = \sum_{e \in E} l(e)$.

Algorithm 2. *Minimum-cost minimum flow.*

1. Use Algorithm 1 to determine the existence of balancing preflows on G. If there exist such flows, compute the value μ of the minimum-balancing preflows, and ν as well.
2. Construct $\overline{G}$, the balancing graph for G, setting $\bar{u}(s', s) = \mu$ and $\bar{u}(t, t') = \nu$.
3. Find a minimum-cost maximum flow F on $\overline{G}$.
4. From F construct the induced balancing preflow f_b on G.
5. Let $f = f_b + f_0$, where f_0 is the basic preflow; f is a minimum-cost minimum flow for G. □

All steps except 3 take total time $O(mn \log n)$. Different algorithms for Step 3 yield different costs for Algorithm 2. We shall use algorithms for finding the shortest paths frequently. Recall that single source shortest paths can be obtained in time $O(n^2)$, $O(m \log_{m/n+2} n)$, and $O(m + n \log n)$, using Dijkstra's algorithm [Aho, Hopcroft, and Ullman, 1974; Tarjan, 1983], implicit heaps [Johnson, 1977; Tarjan, 1983], and Fibonacci heaps [Fredman and Tarjan, 1984], respectively. The first two algorithms are relatively easy to implement, and the last one yields the best-known time bound. The time bounds reported here are from using Fibonacci heaps, and practitioners might want to choose the other two alternatives.

If we use minimum-cost augmentation for finding a minimum-cost maximum flow in Step 3, since the flow value is bounded by L and each augmentation takes time $O(m + n \log n)$, the total cost is $O((m + n \log n)L)$.

Since the flow value is bounded by L, we can set the capacity upper bounds to L for the edges with capacity upper bounds exceeding L, and then use scaling on capacities [Edmonds and Karp, 1972]. A minimum-cost maximum flow can be found in time $O(m(m + n \log n) \log L)$.

Theorem 3.4. Algorithm 2 constructs a minimum-cost minimum flow f in time $O(mn \log n + (m + n \log n)L)$ if minimum-cost augmentation is used in Step 3, and in time $O(m(m + n \log n) \log L))$ if scaling is used, where $L = \sum_{e \in E} l(e)$.

Proof. The correctness of the algorithm follows from Lemma 3.3 and Theorem 3.2. □

3.5. Minimum-cost flow

We first reduce the minimum-cost balancing preflow problem on G to a problem on an augmented graph, and then further reduce this problem to a minimum-cost maximum-flow problem on a balancing graph. Essentially, we reduce the problem to a *circulation* problem (see Section 6). Because of practical applications in programming and circuit testing, we prefer to present solutions in this order. We identify the problems in the reduction only for the purpose of making the reduction conceptually clear. For practical purposes, one can construct the balancing graphs by a simple modification of the original graph, and then solve the final maximum-flow problems directly.

To construct the augmented graph G^* from G, we add an edge (t, s) with cost zero, and capacity bounds $l(t, s) = 0$, $u(t, s) = \infty$. Given a balancing preflow f_b on G, we define a preflow $f_b{}^*$ on G^* as follows. Let $f_b{}^*(e) = f_b(e)$ for $e \in E$, and let $f_b{}^*(t, s) = \beta(s, f_0) + \beta(s, f_b)$. Let β^* be the balancing index of vertices in G^*. Obviously, the balancing index of f_0 for all vertices in G^* remains the same as it was in G, and $\beta^*(v, f_b{}^*) = \beta(v, f_b)$ for $v \in S \cup D$. On the other hand, since $\beta(s, f_b) + \beta(s, f_0) = -[\beta(t, f_b) + \beta(t, f_0)]$, we have $\beta^*(t, f_b{}^*) = \beta(t, f_b) + f_b{}^*(t, s) = \beta(t, f_b) + [\beta(s, f_0) + \beta(s, f_b)] = -\beta(t, f_0)$, and $\beta^*(s, f_b{}^*) = \beta(s, f_b) - f_b{}^*(t, s) = \beta(s, f_b) - [\beta(s, f_0) + \beta(s, f_b)] = -\beta(s, f_0)$. Therefore, if f_b is a balancing preflow on G then $f_b{}^*$ is a balancing preflow on G^*, where s and t are considered as intermediate vertices, and the restriction of $f_b{}^*$ to G is f_b. On the other hand, the restriction of a balancing preflow on G^* to G is a balancing preflow. Since $cost(f_b{}^*) = cost(f_b)$, we only have to find a minimum-cost balancing preflow on G^*.

Note that t is a surplus vertex and s is a deficient vertex in G^* now. To construct the balancing graph $\overline{G}^*$, we add a new source vertex s' and a new sink vertex t' to G^*. The following edges are also added to G^*:

i) For each v in $S \cup \{t\}$, add the edge (s', v) with $\overline{u}(s', v) = -\beta(v, f_0)$.

ii) For each w in $D \cup \{s\}$, add the edge (w, t') with $\overline{u}(w, t') = \beta(w, f_0)$.

The cost of each edge in G remains what is was, and the cost of each added edges is zero. The capacity lower bound of each edge in $\overline{G}^*$ is zero, and for each edge e in G, the capacity upper bound becomes $\overline{u}(e) = u(e) - l(e)$. Obviously, $\overline{G}^*$ has $O(m)$ edges and $O(n)$ vertices.

Given a flow F on $\overline{G}^*$, the induced preflow $f_b{}^*$ is balancing if and only if all the edges incident to s' or t' are saturated, and in this case, F is a maximum flow. Thus

Lemma 3.4. There exists a balancing preflow on G^* if and only if there exists a balancing preflow on G. In this case, F on $\overline{G}^*$ is a minimum-cost maximum flow if and only if the induced preflow $f_b{}^*$ on G^* is a minimum-cost balancing preflow, and this is the case if and only if f_b, the restriction of $f_b{}^*$ to G, is a minimum-cost balancing preflow on G.

□

Algorithm 3. *Minimum-cost flow.*

1. From G construct the augmented graph G^*.
2. From G^* construct the balancing graph $\overline{G}^*$.
3. Find a minimum-cost maximum flow F on $\overline{G}^*$. If not all the edges incident to the source s' or sink t' are saturated, then abort. There is no feasible flow on G.

4. Compute the induced balancing preflow f_b* from F on $G*$.
5. Let f_b be the restriction of f_b* to G.
6. Let $f = f_b + f_0$, where f_0 is the basic preflow; f is a minimum-cost flow for G. □

We can now prove the following analog of Theorem 3.4:

Theorem 3.5. Algorithm 3 constructs a minimum-cost flow f in time $O(mn \log n + (m + n \log n)L)$ if minimum-cost augmentation is used in Step 3, and in time $O(m(m + n \log n) \log L))$ if scaling is used, where $L = \sum_{e \in E} l(e)$. □

4. Positive Flow and Path-Covering Problems on Graphs with Capacity Upper Bounds

A flow f is *positive* if $l(e) = 1$ for all edges e. We now specialize results obtained for the minimum-flow problems to the following *positive-flow* problems:

Problem $F_1^{(1)}$. *Minimum positive flow*.

Find a positive flow for G with a minimum flow value.

Problem $F_2^{(1)}$. *Minimum-cost minimum positive flow*.

Among the minimum positive flows for G, find one with minimum cost.

Problem $F_3^{(1)}$. *Minimum-cost positive flow*.

Find a positive flow for G with minimum cost.

Given a set of covering $s-t$ paths P, let $f(e)$ be the number of times e appears in P (if e appears more than once on the same path, each occurrence of e is counted). Then f is a positive flow on G, called the *derived* flow from P. Obviously, the cardinality of P is $value(f)$, and $cost(f) = cost(P)$.

Conversely, given a positive flow f on G, one can successively reduce the flow along $s-t$ paths and cycles, constructing $s-t$ paths and cycles. When $f \equiv 0$, we have constructed a set of $s-t$ paths and cycles P such that the derived flow from P is f. Since $f(e) \geq 1$ for all e in E, we can append the cycles to $s-t$ paths, such that there are only $s-t$ paths in P.

On the other hand, the flow value is bounded by $L = \sum_{e \in E} l(e) = m$, and each flow augmentation in Algorithms 1-3 is taken along a path of no more than n edges. Therefore, the value of $\sum_{e \in E} F(e)$ for the constructed flow F is $O(mn)$, and that for the corresponding balancing preflow $\sum_{e \in E} f_b(e)$ is $O(mn)$. The flows f computed by Algorithm 1-3 have the property: $\sum_{e \in E} f(e) = O(mn)$. Therefore, from such a positive flow, the corresponding covering paths can be constructed in time $O(mn)$. We summarize:

Theorem 4.1. The covering-paths problems $CP_i^{(1)}$ are equivalent to the corresponding positive-flow problems $F_i^{(1)}$, $i = 1, 2, 3$, respectively. An optimal solution of a positive-flow problem can be transformed to an optimal solution of the corresponding covering-paths problem, and vice versa, in time $O(mn)$. □

We can use Algorithms $1-3$ for the positive-flow problems $F_i^{(1)}$, $i = 1, 2, 3$, and then obtain solutions of the covering-paths problems.

Since the flow value on the balancing graph $\overline{G}$ is bounded by m, instead of using the Sleator-Tarjan algorithm, we can use Dinic's algorithm for finding maximum flows. One can easily check that it takes time $O(mn)$ [Gabow, 1985]. We summarize:

Theorem 4.2. The positive-flow problem $F_1^{(1)}$ and the covering-paths problem $CP_1^{(1)}$ can be solved in time $O(mn)$. The positive-flow problems $F_i^{(1)}$ and the covering-paths problems $CP_i^{(1)}$, $i = 2, 3$, can be solved in time $O(m(m + n \log n))$. □

5. Positive-Flow and Covering-Paths Problems on Graphs without Capacity Upper Bounds

For the purpose of constructing testing sets, there is no a priori capacity upper bound on the edges. Therefore, we can set $u(e) = \infty$ for all e in E, and denote the corresponding positive-flow and covering-paths problems by $F_i^{(2)}$ and $CP_i^{(2)}$, $i = 1, 2, 3$, respectively. As special cases, we can apply Algorithms 1-3 for these problems. However, since there is no capacity upper bound, it is easy to determine the existence of flows on G:

Lemma 5.1. For a directed graph G without capacity upper bounds, the positive-flow and covering-paths problems have a solution if and only if for every vertex v in V, there is an $s-v$ and a $v-t$ path in G. □

After checking the existence of feasible solutions, Algorithms $1-3$ can be applied to these problems:

Theorem 4.2′. The positive-flow problem $F_1^{(2)}$ and the covering-paths problem $CP_1^{(2)}$ can be solved in time $O(mn)$. The positive-flow problems $F_i^{(2)}$ and the covering-paths problems $CP_i^{(2)}$, $i = 2, 3$, can be solved in time $O(m(m + n \log n))$. □

By taking advantage of the fact that the capacity upper bounds are infinite for all the edges in the balancing graphs except those incident to the source s' or sink t', we can further reduce the time bounds for Problems $F_i^{(2)}$ and $CP_i^{(2)}$, $i = 2, 3$.

5.1. $O(mn \log n)$ Algorithms

Since the sum of capacity bounds of edges incident to the source or sink in a balancing graph is bounded by m, and the rest of the edges have infinite capacity upper bound, we can use capacity scaling [Edmonds and Karp, 1972; Gabow and Tarjan, 1987]. It provides an $O(n(m + n \log n) \log n)$ algorithm for finding a minimum-cost maximum flow on a balancing graph.

If $m \geq n \log n$, then this bound is $O(mn \log n)$. Otherwise, we can use minimum-cost augmentation for finding a minimum-cost maximum flow with cost $O(m(m + n \log n))$, which is $O(mn \log n)$ when $m < n \log n$. We summarize:

Theorem 5.1. The positive-flow problems $F_i^{(2)}$ and the covering-paths problems $CP_i^{(2)}$, $i = 2, 3$, can be solved in time $O(mn \log n)$. □

Next, we use *partial scaling* to further improve the efficiency of the algorithms.

5.2. Partial Scaling: $O(n(m + n \log n) \log(m/n))$ Algorithms

In Algorithms 2 and 3, the positive-flow problems $F_i^{(2)}$, $i = 2, 3$, are reduced to minimum-cost maximum-flow problems on balancing graphs, where the edges incident to the source or sink have bounded capacities and the rest of the edges have an infinite capacity upper bound. Instead of scaling all the edges after setting the capacity bounds of some of the edges to m, as was done in the previous subsection, we now use *partial scaling* to further improve the efficiency of the algorithms. We only scale the edges incident to the source or sink. Our approach is a generalization of scaling used by Edmonds and Karp [1972] for the transportation problem, and we adopt their terminology. The interested readers are referred to the original paper.

We construct a minimum-cost maximum flow on a balancing graph $\overline{G} = (\overline{V} \cup \{s', t'\}, \overline{E})$. A *labeling function* is a mapping from the set of vertices of $\overline{G}$ to the real numbers. Note that for $v, w \in \overline{V}$, the capacity of edge (v, w) is infinity. A flow f on $\overline{G}$ is *extreme* if and only if there is a labeling function π such that for every edge (v, w) in the residual graph $R(f)$, $\pi(v) - \pi(w) + \Delta(v, w) \geq 0$, where Δ is the cost function on the edges in the residual graph as defined in Section 3.3. A flow f is extreme on $\overline{G}$ if and only if there exists π such that the following six conditions hold:

(i) For $v, w \in \overline{V}$ and $(v, w) \in \overline{E}$, $\pi(v) - \pi(w) + cost(v, w) \geq 0$;

(ii) For $v, w \in \overline{V}$ and $(v, w) \in \overline{E}$, $\pi(v) - \pi(w) + cost(v, w) > 0$ implies $f(v, w) = 0$;

(iii) For $v \in \overline{S}$, $\pi(s') > \pi(v)$ implies $f(s', v) = 0$;

(iv) For $v \in \overline{S}$, $\pi(s') < \pi(v)$ implies $f(s', v) = \overline{u}(s', v)$;

(v) For $w \in \overline{D}$, $\pi(w) > \pi(t')$ implies $f(w, t') = 0$;

(vi) For $w \in \overline{D}$, $\pi(w) < \pi(t')$ implies $f(w, t') = \overline{u}(w, t')$.

A flow is *pseudo-extreme* if there exists a labeling function satisfying conditions (i) and (ii). As in Edmonds and Karp [1972], it can be shown that a pseudo-extreme maximum flow on $\overline{G}$ is extreme, i.e., a minimum-cost maximum flow. The problem is now reduced to finding a pseudo-extreme maximum flow on $\overline{G}$.

Choose a positive integer l such that the capacity upper bounds of the edges incident to the source s' or sink t' have at most l digits in their binary expansions. Obviously, $l \le \lceil \log\ m \rceil$, where $\lceil \cdot \rceil$ is the ceiling function. For Problem p, where $0 \le p \le l$, edge $(s',\ v)$ has capacity $\lceil \overline{u}(s',\ v)/2^p \rceil$ and edge $(w,\ t')$ has capacity $\lceil \overline{u}(w,\ t')/2^p \rceil$, where $v \in \overline{S}$ and $w \in \overline{D}$. The capacity of the other edges remains infinite.

Then the scaling method computes maximum pseudo-extreme flows successively for Problems $l,\ l-1, \ldots,\ 0$. It can be easily shown that if f is a maximum pseudo-extreme flow computed in Problem p and π is the associated labeling function, then $2f$ can be taken as its initial pseudo-extreme flow in Problem $p-1$ with π as its associated labeling function.

The detailed implementation is similar to that of Edmonds and Karp [1972], and we omit it. However, to guarantee the correctness of the algorithm, we have to prove that each minimum-cost augmentation over a pseudo-extreme flow still gives a pseudo-extreme flow.

Since f^k is a pseudo-extreme flow and π^k is the associated labeling function, for $(v,\ w) \in \overline{E}$, $v,\ w \in \overline{V}$, $\Delta^k(v,\ w) = \pi^k(v) - \pi^k(w) + \Delta(v,\ w) = \pi^k(v) - \pi^k(w) + cost(v,\ w) \ge 0$. On the other hand, for $v,\ w \in \overline{V}$, $(v,\ w) \in \overline{E}$, and $(w,\ v) \in R(f^k)$, $\Delta^k(w,\ v) = \pi^k(w) - \pi^k(v) + \Delta(w,\ v) = -[\pi^k(v) - \pi^k(w) + cost(v,\ w)]$. If $\Delta^k(w,\ v) < 0$, then $\pi^k(v) - \pi^k(w) + cost(v,\ w) > 0$; that is, $f^k(v,\ w) = 0$, since f^k is a pseudo-extreme flow. Thus, $(w,\ v)$ cannot be in $R(f^k)$, a contradiction. Therefore, $\Delta^k(v,\ w) \ge 0$ for all $(v,\ w) \in R(f^k)$, where $v,\ w \in \overline{V}$.

Starting with $\pi^0 \equiv 0$ and $f^0 \equiv 0$, we do the following minimum-cost augmentations. Given a pseudo-extreme flow f^k with its associated labeling function π^k, augment along a minimum-cost path from s' to t' in the residual graph $R(f^k)$, with respect to the costs $\Delta^k(v,\ w) = \pi^k(v) - \pi^k(w) + \Delta(v,\ w)$ for all edges $(v,\ w)$ in $R(f^k)$. If $\sigma^k(v)$ denotes the cost of a shortest path from s' to v with respect to the costs Δ^k, set $\pi^{k+1}(v) = \pi^k(v) + \sigma^k(v)$. Obviously, $\Delta^k(v,\ w) \ge 0$ for all $(v,\ w) \in R(f^k)$. Since $\pi^0(s') = 0$ and Δ^k is nonnegative, $\pi^k(s') = 0$ for all k. We now have:

Lemma 5.2. For each v in $\overline{V}$, $\pi^{k+1}(v)$ gives the cost of a shortest path from s' to v in the residual graph $R(f^k)$ with respect to the cost function Δ. If f^k is a pseudo-extreme flow, then after a minimum-cost augmentation in $R(f^k)$ with respect to the cost Δ^k, f^{k+1} is still a pseudo-extreme flow with π^{k+1} as the associated labeling function.

Proof. Let $p(v)$ be a path from s' to v in $R(f^k)$. Then $\sum_{p(v)} \Delta^k(\cdot,\ \cdot) = \pi^k(s') - \pi^k(v) + \sum_{p(v)} \Delta(\cdot,\ \cdot)$, where Δ is the cost function in $R(f^k)$. Therefore, an $s' - v$ path in $R(f^k)$ is of minimum cost with respect to Δ if and only if it is of minimum cost with respect to Δ^k.

Let $p^*(v)$ be a shortest path from s' to v with respect to the cost function Δ^k (and Δ). Then $\pi^{k+1}(v) = \pi^k(v) + \sigma^k(v) = \pi^k(v) + [\pi^k(s') - \pi^k(v) + \sum_{p^*(v)} \Delta(\cdot,\ \cdot)] = \pi^k(s') + \sum_{p^*(v)} \Delta(\cdot,\ \cdot) = \sum_{p^*(v)} \Delta(\cdot,\ \cdot)$. Therefore, $\pi^{k+1}(v)$ gives the cost of a shortest path from s' to v in the residual graph $R(f^k)$ with respect to the cost function Δ. The first part of the lemma is proved.

To show that f^{k+1} is pseudo-extreme, we have to show that conditions (i) and (ii) hold for $(v,\ w) \in \overline{E}$, where $v,\ w \in \overline{V}$.

We have $\pi^{k+1}(v) - \pi^{k+1}(w) + cost(v,\ w) = \sum_{p^*(v)} \Delta(\cdot,\ \cdot) - \sum_{p^*(w)} \Delta(\cdot,\ \cdot) + \Delta(v,\ w) \ge 0$, since $\sum_{p^*(v)} \Delta(\cdot,\ \cdot) + \Delta(v,\ w)$ is the cost of an $s' - w$ path in $R(f^k)$ and $\sum_{p^*(w)} \Delta(\cdot,\ \cdot)$ is the cost of a shortest $s' - w$ path in $R(f^k)$. Thus (i) is proved.

If $\pi^{k+1}(v) - \pi^{k+1}(w) + cost(v,\ w) > 0$, then in $R(f^k)$, $\sum_{p^*(v)} \Delta(\cdot,\ \cdot) - \sum_{p^*(w)} \Delta(\cdot,\ \cdot) + \Delta(v,\ w) > 0$, i.e., $\sum_{p^*(v)} \Delta(\cdot,\ \cdot) + \Delta(v,\ w) > \sum_{p^*(w)} \Delta(\cdot,\ \cdot)$. This implies that $(v,\ w)$ is not on a minimum-cost augmenting path in $R(f^k)$, and, therefore, $f^{k+1}(v,\ w) = f^k(v,\ w)$.

Assume on the contrary that $f^k(v, w) > 0$. Then $(w, v) \in R(f^k)$. From $\pi^{k+1}(v) - \pi^{k+1}(w) + cost(v, w) > 0$, we have $\pi^{k+1}(w) - cost(v, w) = \pi^{k+1}(w) + \Delta(w, v) < \pi^{k+1}(v)$, where $\Delta(w, v)$ is the value of edge (w, v) of the cost function in the residual graph $R(f^k)$. Therefore, $\pi^{k+1}(v)$ is not the cost of a shortest path from s' to v in the residual graph $R(f^k)$ with respect to the cost function Δ of $R(f^k)$. This is a contradiction to the first part of the lemma. Therefore, $f^{k+1}(v, w) = f^k(v, w) = 0$, and this proves (ii). □

The total number of flow augmentations is

$$max\ (|\bar{S}|,\ |\bar{D}|)\ \left\{ 2 + \left\lceil \log \frac{\sum_{v \in \bar{S}} \bar{u}(s', v)}{max\ (|\bar{S}|,\ |\bar{D}|)} \right\rceil \right\}.$$

Since $\sum_{v \in \bar{S}} \bar{u}(s', v) = O(m)$ and $max\ (|\bar{S}|,\ |\bar{D}|) = O(n)$,

$$max\ (|\bar{S}|,\ |\bar{D}|)\ \left\{ 2 + \left\lceil \log \frac{\sum_{v \in \bar{S}} \bar{u}(s', v)}{max\ (|\bar{S}|,\ |\bar{D}|)} \right\rceil \right\} = O(n \log \frac{m}{n}).$$

Each augmentation takes time $O(m + n \log n)$ using Fibonacci heaps, and the total cost for solving the min-cost max-flow problem is $O(n(m + n \log n) \log(m/n))$. Therefore,

Theorem 5.2. The positive-flow and the covering-paths problems $F_i^{(2)}$ and $P_i^{(2)}$, $i = 2, 3$, can be solved in time $O(n(m + n \log n) \log(m/n))$. □

The following lemma shows that this bound is even tighter than $O(mn \log n)$, and we omit the routine proof.

Lemma 5.3. Let m be of order $n^{\lambda_0}(log\ n)^{\lambda_1}(log\ \log n)^{\lambda_1}\cdots$, where $1 \le \lambda_0 \le 2$. Then (i) $n(m + n \log n) \log(m/n)$ is of order $O(mn \log n)$; (ii) for $\lambda_0 > 1$, $n(m + n \log n) \log(m/n)$ is of order $\Omega(mn \log n)$; and (iii) for $\lambda_0 = 1$,

$$\lim_{n \to \infty} \frac{n(m + n \log n) \log \frac{m}{n}}{mn \log n} = 0.$$

□

6. Minimum-Circulation Problems

We now study flow and covering-paths problems on a directed graph G_O without a source s or sink t. Otherwise, the capacity bounds and costs are the same as in Section 1. A circulation on G is a preflow f such that $l(e) \le f(e) \le u(e)$ for all e in E and such that the balancing index $\beta(v, f) = 0$ for all v in V. A distinguished vertex O is called the *origin*. The flow value out of O (or equivalently, into O), i.e., $\sum_{(O, v) \in E} f(O, v)$, is called the *repetition* of the circulation. We study the following circulation problems:

Problem $C_1^{(0)}$. *Minimum circulation .*

Find a circulation for G with minimum repetition.

Problem $C_2^{(0)}$. *Minimum-cost minimum circulation.*

Among the minimum circulations of G, find one with minimum cost.

Problem $C_3^{(0)}$. *Minimum-cost circulation.*

Find a circulation for G with minimum cost.

Strongly polynomial algorithms exist for Problem $C_3^{(0)}$; see Tardos [1985]. Balancing gives an algorithm with a run time depending on the capacity lower bounds. However, it yields better algorithms for the postman-tour problems.

We can replace the origin O by two vertices s and t, and replace edges (O, v) by (s, v) and edges (w, O) by (w, t) with the same capacity bounds and costs. In this way, the minimum-circulation problems $C_i^{(0)}$ can be reduced to the minimum-flow problems $F_i^{(0)}$, $i = 1, 2, 3$, and Algorithms $1-3$ can be applied. Thus,

Theorem 6.1. The minimum-circulation problem $C_1^{(0)}$ can be solved in time $O(mn \log n)$. The minimum-cost minimum-circulation problem $C_2^{(0)}$ and the minimum-cost circulation problem $C_3^{(0)}$ can be solved in time $O(mn \log n + (m + n \log n)L)$ if minimum-cost augmentation is used, and in time $O(m(m + n \log n) \log L))$ if scaling is used, where $L = \sum_{e \in E} l(e)$. □

7. The Positive-Circulation and Postman-Tour Problems on Graphs with or without Capacity Upper Bounds

Similarly, as special cases of the minimum-circulation problems, we can define positive-circulation problems, where the capacity lower bound of every edge e in E is $l(e) = 1$. We denote the corresponding positive-circulation problems by $C_i^{(1)}$, $i = 1, 2, 3$.

Let G_O be a directed graph with origin O. A *closed* path in G_O is one in which the beginning and the ending vertex is the same origin O. A *postman tour* is a closed path that uses every edge at least once in the direction of the edges [Kwan, 1960; Edmonds and Johnson, 1973]. Similar to the covering-paths problems $CP_i^{(1)}$, $i = 1, 2, 3$, we can define the following postman-tour problems on G_O:

Problem $PT_1^{(1)}$. *Least-repetitious postman tour*.

Find a postman tour with minimum repetition.

Problem $PT_2^{(1)}$. *Minimum-cost least-repetitious postman tour*.

Among the least-repetitious postman tours, find one with minimum cost.

Problem $PT_3^{(1)}$. *Minimum-cost postman tour*.

Find a postman tour with minimum cost.

As special cases of the positive-circulation problems, we can consider the positive-circulation problems on graphs without capacity upper bound (or $u(e) = \infty$ for all e in E). We denote the three corresponding problems by $C_i^{(2)}$, $i = 1, 2, 3$. We can also define postman-tour problems on such graphs, and denote the corresponding problems by $PT_i^{(2)}$, $i = 2, 3$.

Similarly, it can be shown that the positive-circulation problems are equivalent to the corresponding postman-tour problems, and the transformation of solutions between corresponding positive-circulation and postman-tour problems can be done in time $O(mn)$.

Since the approach is similar to that for the positive-flow and the covering-paths problems, we omit the details. We summarize:

Theorem 7.1. The positive-circulation and the postman-tour problems $C_1^{(j)}$ and $PT_1^{(j)}$ can be solved in time $O(mn)$, $j = 1, 2$. The positive-circulation and the postman-tour problems $C_i^{(j)}$ and $PT_i^{(j)}$, $i = 2, 3$, can be solved in time $O(m(m + n \log n))$ if $j = 1$, and in time $O(n(m + n \log n) \log(m/n))$ if $j = 2$.

□

Problem $PT_3^{(2)}$ is the classical Chinese-postman problem [Kwan, 1960]. Balancing was used to reduce the problem to a minimum-cost maximum-flow problem on a balancing graph [Gibbons, 1985], and the previously best-known algorithm ran in time $O(mn \log n)$ [Gabow and Tarjan, 1987]. From Lemma 5.3 and Theorem 7.1, our algorithm runs in time $O(n(m + n\log n) log(m/n)$, which is asymptotically faster.

8. General Path-Covering Problems

We have not yet discussed the most general covering-paths and postman-tour problems on directed graphs with arbitrary capacity upper and lower bounds. In general, we have

Theorem 8.1. Problems $CP_i^{(0)}$ and $PT_i^{(0)}$, for $i = 2, 3$, are NP-complete.

Proof. We first show that Problem $PT_3^{(0)}$ is NP-complete. A special case of this problem is one in which all edges have an infinite capacity upper bound, some edges have a zero capacity lower bound, and some edges have a unit capacity lower bound. This special case is the well-known NP-complete *rural-postman* problem [Lenstra and Kan, 1976].

Given Problem $PT_3^{(0)}$ on a graph G with origin O, we augment the graph by adding the edges (s, O) and (O, t), giving them a cost of zero, and a capacity upper bound of infinity and a capacity lower bound of zero. Then a minimum-cost (minimum-cardinality) set of covering paths on the augmented graph provides a solution for Problem $PT_3^{(0)}$. Thus Problem $PT_3^{(0)}$ can be reduced to Problem $CP_2^{(0)}$ or Problem $CP_3^{(0)}$; therefore, they too are NP-complete.

Given Problem $CP_2^{(0)}$ on G, we add an edge (t, s) with a cost of zero, a capacity lower bound of zero, and a capacity upper bound of infinity. The covering-paths problem now becomes a minimum-cost least-repetitious postman-tour problem $PT_2^{(0)}$ on the augmented graph with s as origin. Therefore, Problem $PT_2^{(0)}$ is also NP-complete. □

In practice, however, additional constraints may allow polynomial-time algorithms to be devised for these problems. For example, if the subgraph consisting of edges with positive capacity lower bound is connected, then the problem is equivalent to a minimum-flow problem, and the reduction can be done in time $O(mn)$.

9. Problems on Mixed Graphs

We now discuss briefly the corresponding problems on mixed graphs. A *mixed* graph has both directed and undirected edges. The optimization problems considered in this paper can also be formulated for mixed graphs, and we have:

Theorem 9.1. For mixed graphs, Problems $X_i^{(j)}$, are NP-complete, where $X = C, F, CP, PT$, $i = 2, 3$, and $j = 0, 1, 2$.

Proof. We first note that Problem $PT_3^{(2)}$ can be reduced to Problems $CP_3^{(2)}$ and $CP_2^{(2)}$ and that Problem $CP_2^{(2)}$ can be reduced to Problem $PT_2^{(2)}$. Since Problem $PT_3^{(2)}$ is *NP*-complete [Papadimitriou, 1976], all of them are. The reduction is the same as that in the proof of Theorem 8.1, and we omit the details.

Since Problems $F_i^{(2)}$ ($C_i^{(2)}$) and $CP_i^{(2)}$ ($PT_i^{(2)}$) are equivalent, they are special cases of Problems $X_i^{(1)}$, which in turn are special cases of Problems $X_i^{(0)}$, respectively, for $X = F, C, P, p$, $i = 2, 3$. The theorem follows. □

10. Conclusions

We started by considering basic questions arising in program and circuit testing. We showed that these questions are instances of several natural classes of network-optimization problems, and we derived a general technique for solving these problems. Our work provides a unifying framework for these optimization problems. A few of the problems have been studied in isolation, such as the testing and Chinese-postman problems, but our methods yield the fastest-known solutions even for these problems.

The results are summarized in the following three tables. The symbols $C_i^{(j)}$, $F_i^{(j)}$, $CP_i^{(j)}$, and $PT_i^{(j)}$ refer to the four main classes of problems: circulation, flow, covering paths, and postman tours. The subscript, $i = 1, 2, 3$, distinguishes the three forms of each problem: minimizing the cardinality, minimizing the cost and cardinality, and minimizing only the cost. The superscript, $j = 0, 1, 2$, classifies the capacity bounds on the edges: 0 for arbitrary lower and upper bounds, 1 for a unit lower bound and an arbitrary upper bound, and 2 for a unit lower bound and an infinite upper bound. In the tables, m is the number of edges, n the number of nodes, and $L = \sum_{e \in E} l(e)$ where $l(e)$ is the capacity lower bound on edge e.

PROBLEM	COST	PROBLEM	COST
$C_1^{(0)}$, $F_1^{(0)}$	$mn \log n$	$CP_1^{(0)}$, $PT_1^{(0)}$	?
$C_1^{(1)}$, $F_1^{(1)}$	mn	$CP_1^{(1)}$, $PT_1^{(1)}$	mn
$C_1^{(2)}$, $F_1^{(2)}$	mn	$CP_1^{(2)}$, $PT_1^{(2)}$	mn

Table 1. Minimum-cardinality problems.

PROBLEM	COST	PROBLEM	COST
$C_2^{(0)}$, $F_2^{(0)}$	$\min \{(m+n\log n)L,\ m(m+n\log n)\log L\}$	$CP_2^{(0)}$, $PT_2^{(0)}$	NP-complete
$C_2^{(1)}$, $F_2^{(1)}$	$m(m+n\log n)$	$CP_2^{(1)}$, $PT_2^{(1)}$	$m(m+n\log n)$
$C_2^{(2)}$, $F_2^{(2)}$	$n(m + n \log n) \log (m/n)$	$CP_2^{(2)}$, $PT_2^{(2)}$	$n(m + n \log n) \log (m/n)$

Table 2. Minimum-cost minimum-cardinality problems.

PROBLEM	COST	PROBLEM	COST
$C_3^{(0)}$, $F_3^{(0)}$	$\min \{(m+n\log n)L,\ m(m+n\log n)\log L\}$	$CP_3^{(0)}$, $PT_3^{(0)}$	NP-complete
$C_3^{(1)}$, $F_3^{(1)}$	$m(m+n\log n)$	$CP_3^{(1)}$, $PT_3^{(1)}$	$m(m+n\log n)$
$C_3^{(2)}$, $F_3^{(2)}$	$n(m + n \log n) \log (m/n)$	$CP_3^{(2)}$, $PT_3^{(2)}$	$n(m + n \log n) \log (m/n)$

Table 3. Minimum-cost problems.

Acknowledgment

We are indebted to Mihalis Yannakakis for many valuable comments and stimulating discussions.

References

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. [1974]. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

Edmonds, J., and Johnson, E. L. [1973]. Matching, Euler tours and the Chinese Postman, *Mathematical Programming* **5**, 88-124.

Edmonds, J., and Karp, R. M. [1972]. Theoretical improvements in algorithmic efficiency for network flow problems, *J. ACM* **19**, 2, 248-264.

Even, S. [1979]. *Graph Algorithms*, Computer Science Press.

Ford, L. R. Jr and Fulkerson, D. R. [1962]. *Flows in Networks*, Princeton Univ. Press. Princeton, NJ.

Fredman, M. L, and Tarjan, R. E. [1984]. Fibonacci heaps and their uses in improved network optimization algorithms, *Proc. 25th Annual Symp. on Found. of Comp. Sci.*, pp. 338-346.

Gabow, H. N., Maheshwari, S. N., and Osterweil, L. J. [1976]. On two problems in the generation of program test paths, *IEEE Trans. Software Engineering* **SE-2**, 227-231.

Gabow, H. N. and Tarjan, R. E. [1987]. *Faster scaling algorithms for network problems*, Unpublished manuscript.

Galil, Z. and Tardos, E. [1986]. An $O(n^2 \log n\ (m + n \log n))$ min-cost flow algorithm, *Proc. 27th IEEE Symp. of Foundations of Computer Science*, 1-9.

Gibbons, A. [1985]. *Algorithmic Graph Theory*, Cambridge University Press, Cambridge.

Knuth, D. E. [1984]. *A torture test for TEX*. Stanford University, CS Technical Report.

Krause, K. A., Smith, R. W., and Goodwin, M. A. [1973]. Optimal software test planning through automated network analysis, *Proc. 1973 IEEE Symposium on Computer Software Reliability*, pp. 18-22.

Kwan, M. (Guan, Meigu) [1960]. Graphic programming using odd or even points, *Chinese Math.* **1**, 273-277.

Lawler, E. [1976]. *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston.

Lenstra, J. K. and Rinnooy Kan, A. H. G. [1976]. On general routing problems, *Networks* **6**, 273-280.

Ntafos, S. C. and Hakimi, S. L. [1979]. On path cover problems in digraphs and applications to programming testing, *IEEE Trans. on Software Engineering* **SE-5**:5, 520-529.

Papadimitriou, C. H. [1976]. On the complexity of edge traversing, *J. ACM* **23**:3, 544-554.

Seth, S. C., and Agrawal, V. D. [1985]. Cutting chip testing costs, *IEEE Spectrum* **22**, 38-45.

Tardos, E. [1985]. A strongly polynomial minimum cost circulation algorithm, *Combinatorica* **5**:3, 247-255.

Tarjan, R. E. [1983]. *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Uyar, M. U, and Dahbura, A. T. [1986]. Optimal test sequence generation for protocols: the Chinese postman algorithm applied to Q.931, *Proc. IEEE Global Telecommunications Conference*, 1986.