A Unified Approach to Approximation Algorithms for Bottleneck Problems

DORIT S. HOCHBAUM AND DAVID B. SHMOYS

University of California, Berkeley, Berkeley, California

Abstract. In this paper a powerful, and yet simple, technique for devising approximation algorithms for a wide variety of NP-complete problems in routing, location, and communication network design is investigated. Each of the algorithms presented here delivers an approximate solution guaranteed to be within a constant factor of the optimal solution. In addition, for several of these problems we can show that unless P = NP, there does not exist a polynomial-time algorithm that has a better performance guarantee.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*

General Terms: Theory, Verification

Additional Key Words and Phrases: Approximation algorithm, center problem, combinatorial optimization, communication network, heuristics, routing, worst-case analysis

1. *Introduction*

In this paper we investigate an extremely simple, powerful technique for devising approximation algorithms for a wide variety of NP-complete problems in routing, location, and communication network design. Each of the polynomial-time algorithms presented here delivers an approximate solution guaranteed to be within a constant factor δ of the optimal solution; such a polynomial-time algorithm is called a *δ-approximation algorithm.* In addition, for several of these problems we can show that, unless P = NP, there does not exist a $(\delta - \epsilon)$-approximation algorithm for any fixed $\epsilon > 0$; we call such approximation algorithms *best possible.* The basis for this approximation technique is the concept of the power of a graph.

The following three examples illustrate the range of problems to which the power-of-graphs technique is applicable. The k-center problem is a problem from location theory: Given n cities and the shortest path distances between all pairs of cities, the aim is to choose k cities as centers so that the city farthest from its closest center is as close as possible. A second example is the bottleneck wandering

The research of D. Hochbaum was supported in part by the National Science Foundation under grants ECS 82-04695 and ECS 85-01988. The work of D. Shmoys was supported in part by the National Science Foundation by a graduate fellowship and under grant MCS 83-11422 and in part by DARPA order 4031, monitored by Naval Electronic System Command under contract N00039-C-0235.

Authors' current addresses: D. S. Hochbaum, University of California, Berkeley, Berkeley, CA 94720; D. B. Shmoys, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139.

Communication Network Problems	δ
$(k, \mathscr{G})$-partition with diameter d	$2d$
k-clustering*	2
k-center*	2
k-switching network	3
Weighted Center Problems	δ
(singly) weighted k-center	3
weighted k-center with at most l centers	3
k-supplier*	3
m-weighted k-center	$9m - 6$
Vehicle Routing Problems	δ
k-path vehicle routing*	2
wandering salesperson*	2
repeated city TSP*	2
single depot k-vehicle routing*	2

FIGURE 1

salesperson problem: The salesperson starts in some city u and must wind his or her way to city v, visiting each city exactly once; the salesperson travels to a new city each day, and wishes to choose a route that minimizes the travel time on the longest day of traveling. For a final example, consider the k-clustering problem: We wish to partition n cities into k clusters so that the longest distance between two cities in the same cluster is minimized. For each of these problems, we wish to find a subgraph of the complete graph satisfying certain constraints such that the length of the longest edge included in the subgraph is minimized; such problems are called *bottleneck problems.* For each instance of a bottleneck problem there is a class of subgraphs of the given complete edge-weighted graph that satisfy the constraints of the problem; call this the class of *feasible* subgraphs $\mathscr{G}$. The objective of a bottleneck problem is to find a graph G contained in $\mathscr{G}$ such that the maximum weight of an edge in G is minimized. Notice that no restrictions are placed on $\mathscr{G}$, and that $\mathscr{G}$ might even depend on the instance itself. In each of the examples given above, $\mathscr{G}$ is determined by purely combinatorial restrictions, but in other examples discussed below, there are additional cost constraints that the feasible graphs must satisfy.

For all of the problems considered in this paper we make the assumption that the underlying graph is complete, so that the edge weight c_{ij} is defined for all i, j. Furthermore, we always assume that the edge weights must satisfy the triangle inequality; that is, for all i, j, k

$$c_{ij} + c_{jk} \geq c_{ik}.$$

This assumption is a natural one, since, for most of the problems that we consider, the nodes correspond to locations and the edges weights correspond to shortest path distances. The results discussed in this paper are tabulated in Figure 1. For all but one of the problems mentioned, these are the first algorithms that ensure *any* constant guarantees; for the clustering problem, Gonzalez has reported an identical bound using a different approach [7].

As indicated by the asterisks, many of the results given in Figure 1 are best possible. Best possibility results are extremely rare, and it is surprising that so general a technique can be used to prove such tight complexity bounds. To the best of the authors' knowledge, the only other such best possible results can be

```
i ← 0
repeat
  begin
  i ← i + 1
  G_i ← BOTTLENECKG(c_{e_i})  {Recall e_i is the ith shortest edge}
  P ← path with the most edges from u to v in G_i  {This is NP-hard}
  end
until P is a Hamiltonian path
output P
```

FIGURE 2

found in [7], [11], [15], and [16] and another sort of best possible approximation result can be found in [14].

2. *A Powerful Approximation Technique*

In this section we present the fundamentals of the approximation technique that we use to derive a plethora of applications. Our technique, which is a generalization of one used in [16], is based on the following crucial, and yet quite natural, observation.

Observation. For any bottleneck problem, the value of the optimal solution is always one of the edge weights in the original specification of the instance of the problem.

Throughout this paper, $G_C = (V, E_C)$ denotes the complete graph of the instance, and the edge weights are denoted c_{ij}. Furthermore, we label the edges of E_C so that $c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$ where $m = \binom{n}{2}$. The following notation will also be used extensively. If $G = (V, E)$ is an arbitrary subgraph of G_C, let

$$max(G) = \max_{(i,j)\in E} c_{ij}.$$

Furthermore, we define the *bottleneck graph* of c to be $BOTTLENECKG(c) = (V, E)$ where $E = \{(i, j) \mid c_{ij} \leq c\}$. Note that $max(BOTTLENECKG(c_{ij})) = c_{ij}$. Finally, an *edge subgraph* of $G = (V, E)$ is a graph $H = (V, F)$ such that $F \subseteq E$; note that the vertex sets are identical.

Let us return to the example of the bottleneck wandering salesperson problem. For this problem, the feasible subgraphs $\mathscr{G}$ are the Hamiltonian paths from u to v. Thus, we can restate the wandering salesperson problem in the following way: Find the minimum value c, such that $BOTTLENECKG(c)$ contains a Hamiltonian path from u to v. By the observation made above, the optimal value must be one of the c_{ij}. This leads us to the algorithm shown in Figure 2.

The procedure in Figure 2 could be mimicked for all bottleneck problems; the only change needed is that at each iteration of the **until** loop, a test is made to determine whether G_i contains an appropriate feasible edge subgraph. However, for all of the problems we consider, this decision problem is NP-complete.

The key notion that we use for all of the approximation algorithms given here is that of the power of a graph. If $G = (V, E)$ is an arbitrary subgraph of G and t is a positive integer, let the tth power of G_C be $G^t = (V, E^t)$, where there is an edge (u, v) in G^t wherever there was a path from u to v with at most t edges in G; more formally, let

$$E^t = \{(i_0, i_l) \mid \exists i_1, i_2, \ldots, i_{l-1} \text{ such that } (i_{m-1}, i_m) \in E,\ m = 1, \ldots, l,\ l \leq t\}.$$

Powers of graphs are useful because of the following simple lemma, which is a consequence of the fact that the edge weights satisfy the triangle inequality.

FIGURE 3

```
procedure bottleneck(𝒢, G_C, t)
  i ← 0
  repeat
    begin
    i ← i + 1
    G_i ← BOTTLENECKG(c_{e_i})
    out.test ← test(𝒢, G_i, t)
    end
  until out.test is not a "certificate of failure" do
  output out.test.
```

LEMMA 1. $max(G^t) \leq t \cdot max(G)$.

PROOF. Consider an arbitrary edge (i, j) in G^t. By the definition of G^t, there is a path in G with at most t edges from i to j. The weight of each of these edges is at most $max(G)$. By the triangle inequality, the weight of any edge (i, j) is no more than the total weight of any path from i to j. Therefore the weight of (i, j) is at most $t \cdot max(G)$, and the claim is proved. □

For a particular instance of some bottleneck problem, let $\mathcal{G}$ denote the set of feasible subgraphs of G. Thus, we wish to minimize $max(G)$ such that $G \in \mathcal{G}$. In the algorithm presented above, at iteration i we check whether $\mathcal{G}$ contains a graph G that is a subgraph of $G_i = BOTTLENECKG(c_{e_i})$. Since this decision problem is NP-complete, we need some sort of "relaxed" decision procedure: Suppose that we run a procedure $test(\mathcal{G}, G_i, t)$ that has two termination conditions; either *test* produces a *certificate of failure* or it produces a graph $G \in \mathcal{G}$ that is an edge subgraph of $(G_i)^t$. A certificate of failure is some proof that there is no subgraph of G_i that is contained in $\mathcal{G}$. This suggests the algorithm shown in Figure 3.

The justification for the approach in Figure 3 is given by the following theorem.

THEOREM 1. *Let $\mathcal{G}$ denote the feasible subgraphs for an instance G_C of a bottleneck problem. Let G^* be an optimal subgraph in $\mathcal{G}$, and let G be the graph output by the procedure bottleneck$(\mathcal{G}, G_C, t)$. Then, the value of the approximate solution produced,*

$$max(G) \leq t \cdot max(G^*),$$

where $max(G^*)$ is the value of the optimal solution.

PROOF. Suppose that the algorithm halts after some iteration i. Since for all previous iterations, *test* produces a certificate of failure, it follows that $max(G^*) \geq c_{e_i}$. However, the graph output by the procedure is a subgraph of $(G_i)^t$, and thus, by the lemma,

$$max(G) \leq max((G_i)^t) \leq t \cdot max(G_i) = t \cdot c_{e_i}.$$

Combining these inequalities, we get the desired result. □

Thus, for any bottleneck problem, the construction of a polynomial-time procedure $test(\mathcal{G}, G, t)$ immediately yields a polynomial-time t-approximation algorithm for this problem. The remainder of this paper is devoted to exhibiting a wide variety of problems for which this is possible.

3. *Solving Communication Network Design Problems*

In this section we construct an approximation algorithm for a broad class of communication network design problems. Suppose that we wish to assign each of

n locations to one of k communication networks, where the structure of each network is set in some specific way. Furthermore, we wish to do this in such a way that the length of the longest link used for communication is minimized. This is a natural objective since the reliability of a given link usually decreases as its length increases, and we wish to ensure that all links of each network are likely to remain functional.

More formally, suppose that $\mathcal{F}$ is a family of graphs $\{F_i \mid i = 1, 2, \ldots\}$, where, for every positive integer j, there is at least one member of $\mathcal{F}$ with j vertices. These are the allowed network structures. A $(k, \mathcal{F})$-*partition* is defined to be a graph $G = (\bigcup_{i=1}^{k} V_i, \bigcup_{i=1}^{k} E_i)$ where $V_i \cap V_j = \emptyset$ for $i \neq j$, and $G_i = (V_i, E_i)$ is isomorphic to a member of $\mathcal{F}$ for each $i = 1, \ldots, k$. Thus a $(k, \mathcal{F})$-partition corresponds to the complete set of k communication networks of permissible structure. Let $part(\mathcal{F}, G)$ denote the minimum number of parts, k, into which the vertices of G can be partitioned such that a $(k, \mathcal{F})$-partition is an edge subgraph of G. We consider the following network communication design problem: Given a complete edge-weighted graph, find the $(k, \mathcal{F})$-partition, G, such that $max(G)$ is minimized. The cost of the optimal solution can be stated equivalently as the minimum value c, such that $part(\mathcal{F}, BOTTLENECKG(c)) \leq k$.

For example, if $\mathcal{F} = \{K_{1,i} \mid i = 0, 1, \ldots\}$, where $K_{m,n}$ denotes the complete bipartite graph formed between sets containing m and n vertices, then $part(\mathcal{F}, G)$ is the equivalent of the domination number $dom(G)$, the size of the smallest set of vertices such that every vertex is either in the set or adjacent to a vertex in the set. (Such a set of vertices is called a *dominating* set.) The corresponding design problem is the *k-center* problem that was described above.

Another common example is the family $\mathcal{F} = \{K_i \mid i = 1, 2, \ldots\}$ where K_i denotes the complete graph on i vertices. In this case, $part(\mathcal{F}, G)$ is the *clique partition number*, $\kappa(G)$, the minimum number of parts into which the vertex set can be partitioned such that the graph induced on each part is a clique. The corresponding optimization problem is the *clustering* problem.

As a final example, consider $\mathcal{F} = \{K_{m,n} \mid m, n = 0, 1, 2, \ldots\}$. This example differs from the previous two in that there is more than one graph for most graph sizes. Since a complete bipartite graph represents a switching network of depth 1, the corresponding design problem is called the *k-switching network* problem.

If G is an arbitrary graph, let $diam(G)$ denote the number of edges in the shortest path between the pair of vertices of G that are farthest apart (where path lengths here are just the number of edges in the path). An alternate, equivalent definition of $diam(G)$ is the minimum value t such that G^t is a complete graph. Thus it follows that $diam(K_n) = 1$ for all n and $diam(K_{m,n}) = 2$ for all $m \cdot n > 1$.

Suppose that $\mathcal{F}$ satisfies the constraint that for all i, $diam(F_i) \leq d$. Furthermore, suppose that there is a procedure $find\mathcal{F}(i)$ that on input i produces a member of $\mathcal{F}$ with i vertices within time polynomial in i. Under these assumptions, we construct a polynomial-time $2d$-approximation algorithm to find a feasible $(k, \mathcal{F})$-partition. As discussed in the previous section, if $\mathcal{G}$ denotes the class of feasible subgraphs of G_C given by $(k, \mathcal{F})$-partitions, we need only construct the procedure $test(\mathcal{G}, G, 2d)$. Before discussing this procedure, we give a few useful definitions. An *independent set* in a graph is a set of pairwise nonadjacent vertices. A *maximal independent set* is an independent set that is not properly contained in any independent set. Unlike the problem of finding a maximum-sized independent set, it is easy to find a maximal independent set in polynomial time.

The procedure in Figure 4 makes use of a few simple facts.

FACT 1. *If G contains a $(k, \mathcal{F})$-partition, then $\kappa(G^d) \leq k$.*

```
procedure test(𝒢, G, 2d){(k, ℱ)-partition}
  S ← maximal independent set in G^d
  if |S| > k then output "certificate of failure" produced
    else do
      begin
      {Note that S is a dominating set of G^d}
      if |S| < k then add nodes arbitrarily to make |S| = k
      {Suppose that S = {v1, v2, ..., vk}}
      partition all of the nodes into V1, V2, ..., Vk where vi
          dominates all of the nodes in Vi in G^d.
      {Note that Vi induces a clique in G^2d}
      Form the (k, ℱ)-partition by replacing Vi
          by find ℱ(|Vi|)
      {Note that this (k, ℱ)-partition is a subgraph of G^2d}
      end
```

FIGURE 4

PROOF. Suppose that $H = (V, E)$ is the $(k, \mathcal{F})$-partition contained in G, where $V = \bigcup_{i=1}^{k} V_i$, $E = \bigcup_{i=1}^{k} E_i$, and $H_i = (V_i, E_i)$ is isomorphic to $F_{j_i} \in \mathcal{F}$. Since $diam(F_i) \leq d$ for all i, $(H_i)^d$ is a clique. Therefore, the partition of $V = \{V_1, V_2, \ldots, V_k\}$ induces a clique cover in H^d. As a result, the size of the smallest clique cover, $\kappa(H^d) \leq k$. □

FACT 2. *If $\kappa(G) \leq k$, then every independent set in G has size at most k.*

PROOF. Suppose that there exists an independent set S in G with more than k vertices. Consider any minimum clique cover. Since this cover has at most k cliques, two vertices in S must be in the same clique. But then S could not be an independent set. □

FACT 3. *If S is a maximal independent set of G, then it is a dominating set of G.*

PROOF. Suppose that S is not a dominating set of G. Then there exists a vertex v such that v is not adjacent to any vertex in S. However, this implies that $S \cup \{v\}$ is also an independent set, which contradicts the assumption of maximality. □

FACT 4. $(K_{1,i})^2 = K_{i+1}$.

FACT 5. *Any graph on i vertices is isomorphic to an edge subgraph of K_i.*

Using these five facts, we can prove the following theorem.

THEOREM 2. *Let $\mathcal{F}$ be a family of graphs satisfying $diam(F_i) \leq d$ for $i = 1, 2, \ldots$; suppose further that find$\mathcal{F}(i)$ constructs a member of $\mathcal{F}$ with i vertices in polynomial time. Then the procedure bottleneck yields a $2d$-approximation algorithm for the $(k, \mathcal{F})$-partition problem.*

PROOF. In order to prove this result, we show that the procedure *test* given in Figure 4 correctly produces a certificate of failure or a feasible $(k, \mathcal{F})$-partition in G^{2d}.

A certificate of failure must prove that the input G does not contain a $(k, \mathcal{F})$-partition. Suppose that the routine claimed that a certificate of failure was produced, and that there was a $(k, \mathcal{F})$-partition contained in G. By Fact 1, it follows that $\kappa(G^d) \leq k$. Applying Fact 2, we get that every independent set in G^d has size at most k. However, the set S produced by the procedure is an independent set in G^d with more than k vertices. This contradiction proves that the certificate of failure is generated correctly.

We show next that in the absence of a certificate of failure a $(k, \mathcal{F})$-partition is found in G^{2d}. By Fact 3, the set S is a dominating set, and this clearly remains true if more vertices are added to S. By the definition of a dominating set, each of the sets V_i formed must contain $K_{1,|V_i|-1}$. By Fact 4, the graph induced on V_i in $(G^d)^2 = G^{2d}$ is a clique on $|V_i|$ vertices. Using the procedure *find*$\mathcal{F}$, we can find a member F of $\mathcal{F}$ with $|V_i|$ vertices. By Fact 5, the graph induced on V_i in G^{2d} contains a graph isomorphic to F. □

COROLLARY 1. *For the clustering problem, the procedure bottleneck produces a clustering of cost at most twice the cost of the optimal clustering.*

Consider now the k-center problem; an immediate corollary of the theorem would be a 4-approximation algorithm for this problem. However, it is quite easy to improve this result.

COROLLARY 2. *For the k-center problem, the procedure bottleneck produces a center of cost at most twice the cost of the optimal.*

PROOF. In the procedure in Figure 4, the set S generated is a maximal independent set of G^d. Since the family $\mathcal{F}$ for the k-center has a maximum diameter of 2, $d = 2$, then by Fact 3, S is a dominating set in G^2. If S is the set of centers selected, every location not chosen is adjacent to some vertex of S in G^2. Therefore, we need not consider G^4. □

For both of these problems it is not hard to prove that unless P = NP, there does not exist a $(2 - \epsilon)$-approximation algorithm for any fixed $\epsilon > 0$ [8, 12]. These results, by themselves, do not exclude the possibility of tangibly better algorithms for these problems, even if P $\neq$ NP: It might be possible to have a polynomial-time approximation algorithm guaranteed to deliver a solution of cost at most αOPT + β, although no α-approximation algorithm exists unless P = NP. (Recall that an α-approximation algorithm must deliver a solution of cost at most αOPT). This behavior occurs for the problem of edge coloring multigraphs, where there is a polynomial algorithm that uses at most $\lfloor(9\chi' + 6)/8\rfloor$ colors where χ' denotes the chromatic index of a multigraph, and yet if there were a $(4/3\text{-}\epsilon)$-approximation algorithm, then P would equal NP [11]. The following theorem shows that for the k-center problem this behavior is impossible.

THEOREM 3. *If there exists a polynomial-time approximation algorithm $\mathcal{A}$ for the k-center problem, such that $\mathcal{A}$ is guaranteed to produce a solution of cost at most $\alpha OPT(I) + \beta$ for every instance I, where $\alpha < 2$, $\beta > 0$, and $OPT(I)$ denotes the value of the optimal solution, then P = NP.*

PROOF. We show that such an algorithm can be used to solve the dominating set problem in polynomial time. Since the dominating set problem is NP-complete [6], this implies that P = NP. Suppose that we wish to decide whether G has a dominating set of size k. Form the matrix $C = (c_{ij})$ so that $c_{ij} = w$ if (i, j) is an edge of G, and $c_{ij} = 2w$ otherwise, where $w = 2\beta/(2 - \alpha)$. Notice that C satisfies the triangle inequality. It is not hard to see that C has a k-center of cost w if and only if G has a dominating set of size k. Therefore, if G has a dominating set of size k, the algorithm must deliver a set of cost no more than

$$\alpha w + \beta = \alpha\left(\frac{2\beta}{2 - \alpha}\right) + \beta = \frac{\alpha\beta + 2\beta}{2 - \alpha} = \frac{(a + 2)\beta}{2 - \alpha} < \frac{4\beta}{2 - \alpha} = 2w.$$

```
procedure test(𝒫, G, 3){k-switching network}
  S ← maximal independent set in G²
  if |S| > k then output "certificate of failure" produced
    else do
      begin
      {Note that S is a dominating set of G²}
      if |S| < k then add nodes arbitrarily to make |S| = k
      {Suppose that S = {v₁, v₂, ..., v_k}}
      partition all of the nodes into V₁, V₂, ..., V_k where v_i
          dominates all of the nodes in V_i in G²
      further partition each V_i into U_i and W_i = V_i − U_i
          where U_i is the set of vertices adjacent to v_i in G
      form the k-switching network by generating the complete bipartite graph
          between U_i and W_i for each i.
      end
```

FIGURE 5

Since every choice of centers has a cost of either w or $2w$, the algorithm must produce a set of cost w. If G does not have a dominating set of size k, then clearly the algorithm will yield a set of cost $2w$. Notice that this proof does not even require that α and β be constants, only that they be computable in polynomial time. Finally, we note that an analogous result can be obtained for the clustering problem. □

Consider the k-switching network problem. As a corollary to Theorem 2, we get a 4-approximation for this problem, but once again this can be improved (see Figure 5).

COROLLARY 3. *For the k-switching network problem, the procedure bottleneck produces a network of cost at most three times the cost of the optimal switching network.*

PROOF. This result follows almost directly from the proof of Theorem 2. Consider the modified procedure given in Figure 5. As was argued in the proof of Theorem 2, the certificate of failure is produced correctly, so we need only show that a feasible solution is found in G^3 when a certificate is not found. Consider the graph induced by V_i in G^3. Choose a vertex $u \in U_i$ and a vertex $w \in W_i$. Note that by definition u is adjacent to v_i in G and w is adjacent to v_i in G^2. Therefore, u and w must be adjacent in G^3, and the solution constructed is indeed a subgraph of G^3. □

For the k-switching network problem it can be shown that the existence of a $(2 - \epsilon)$-approximation algorithm would imply that P = NP, but it is not known whether any stronger statement is possible.

4. *Solving Weighted Center and Weighted Supplier Problems*

For the problems discussed in the previous sections, the set of feasible subgraphs for each bottleneck problem was constrained by purely combinatorial restrictions. In this section, in addition to a weighted version of the k-center problem, we consider a closely related problem, the k-supplier problem. For this problem there are customers and suppliers, and we wish to choose the centers only from the set of suppliers, and only the customers must be serviced efficiently. Using these two problems as examples, we show how to use the power-of-graphs technique for problems where the feasible subgraphs $\mathscr{P}$ are further constrained by additional cost functions. Furthermore, in the case of the k-supplier problem, we obtain a best

```
procedure test(𝒮, G, 3){Weighted k-center}
  S ← maximal independent set in G²
  for all j ∈ S do
    begin
    w ← min_{l∈N_G(j)} w_l
    let l(j) be a node in N_G(j) with w_{l(j)} = w
    end
  S' ← ∪_{j∈S} l(j)
  if Σ_{j∈S'} w_j > k then output "certificate of failure" produced
    else output S'
```

FIGURE 6

possible result. For the k-center problem, we also present algorithms for the case in which the feasible choice of centers is constrained by multiple weighting functions.

An instance of the (singly) weighted k-center problem is specified by a complete edge-weighted graph $G_C = (V, E_C)$ with edge weights c_{ij} that satisfy the triangle inequality and node weights w_i. A feasible center is a set S of nodes such that

$$\sum_{i \in S} w_i \leq k.$$

Thus a certificate of failure for the routine *test* must prove that the input G does not contain a dominating set of total weight at most k. Furthermore, if *test* does not produce a certificate of failure, then it must find a set of nodes of total weight at most k such that it is a dominating set for G^t for some t. The routine given in Figure 6 yields a dominating set in G^3. It will be useful to define $N_G(v)$ as the neighborhood of v in G; that is, the set of vertices adjacent to v in G as well as v itself.

THEOREM 4. *The procedure bottleneck yields a 3-approximation algorithm for the weighted k-center problem.*

PROOF. Consider the algorithm given in Figure 6. First, we show that if S' is output, then it is a feasible dominating set in G^3. By Fact 3, it follows that S is a dominating set in G^2. In other words, for every vertex in G there is path with at most two edges to some vertex in S. Then, for every vertex v of S there is some vertex in S' that is adjacent to v in G. Therefore, for every vertex in G, there is a path with at most three edges to some vertex in S'. Equivalently, S' is a dominating set in G^3. Furthermore, if S' is output, then it follows that the total vertex weight of S' is at most k. Therefore, S' is a feasible dominating set in G^3.

To complete the proof, we need only show that the certificate of failure is produced correctly. To do this, we show that the total weight of S' is no more than the weight of the minimum weight dominating set of G. Suppose that $S^* = \{v_1, v_2, \ldots, v_m\}$ is the minimum total weight dominating set in G. Partition the vertices of G into $V_1, V_2, \ldots, V_m$ so that $V_j \subseteq N_G(v_j)$. In G^2, the graph induced on each V_j is a clique. Thus, S contains at most one vertex from each V_j; let $\tilde{v}_j$ be this vertex (if it exists). Since $v_j \in N_G(\tilde{v}_j)$, the vertex $l(\tilde{v}_j)$ that is added to S' must have weight at most w_{v_j}. Thus S' has weight at most the total weight of S^*. □

Notice that this result can be extended almost immediately to the weighted k-center problem where we further constrain feasible solutions to have at most l centers. We simply add to the routine *test* a check whether $|S| = |S'| \leq l$; if not, this also constitutes a certificate of failure. Thus we have obtained the following corollary.

```
procedure test(𝒢, G, 3)  {k-supplier}
  if there exists v ∈ V_cust that is not adjacent to any supplier
    then output "certificate of failure" produced and return
  S ← maximal independent set in the graph induced in G² by V_cust
  for all j ∈ S do
    begin
    w ← min_{l∈N_g(j)∩V_sup} w_l
    let l(j) be a node in N_G(j) ∩ V_sup with W_{l(j)} = w
    end
  S' ← ∪_{j∈S} l(j)
  if Σ_{j∈S'} w_j > k then output "certificate of failure" produced
    else output S'
```

FIGURE 7

COROLLARY 4. *For the weighted k-center problem, where the feasible solutions are further constrained to have at most l centers, the procedure bottleneck yields a* 3*-approximation algorithm.*

Consider now the *k-supplier* problem. An instance of the k-supplier problem is given by an edge-weighted complete graph and a partition of the nodes into a supplier set V_{sup} and a customer set V_{cust}; furthermore, each supplier i has a weight w_i. The aim is to choose a set of supplier of total weight at most k, so that every customer is as close as possible to some supplier that was selected. This problem is very similar to the ordinary weighted k-center problem, when some subset of the nodes is given infinite weight. The essential difference is that in the k-supplier problem, the suppliers not selected need not be close to any of the suppliers selected. The algorithm is nearly identical to the one for the weighted k-center problem (see Figure 7).

THEOREM 5. *The procedure bottleneck yields a* 3*-approximation algorithm for the k-supplier problem.*

PROOF. The proof of this result is nearly identical to that of Theorem 4. If there exists a customer that is not adjacent to any supplier, it is clear that no feasible solution exists, so we are justified in producing a certificate of failure. Suppose that S' is output by the procedure. By the same reasons as above, there must be a path with no more than three edges from each customer to some supplier. Therefore, S' is a feasible choice of suppliers in G^3.

To complete the proof, we show that the total weight of S' is at most the weight of the minimum weight choice of suppliers so that every customer is adjacent to some chosen supplier. Suppose that $S^* = \{v_1, v_2, \ldots, v_m\}$ is the minimum weight choice of suppliers serving every customer. Partition V_{cust} so that $V_i \subseteq N_G(v_i)$. In G^2, the graph induced on each V_i is a clique, and therefore S contains at most one vertex from each V_i; let $\tilde{v}_i$ be this vertex (if it exists). Since v_i is adjacent to $\tilde{v}_i$, the vertex $l(\tilde{v}_i)$ has weight at most w_{v_i}. Therefore, the total weight of S' is at most the total weight of S^*. □

It should be noted that this problem could have been modeled in terms of a complete bipartite graph (thereby relaxing the requirement for distances between customers or between suppliers.) With a suitable modification of the notion of a triangle inequality, the entire framework can be carried over to this setting to provide a more general result. Surprisingly, the above algorithm can be shown to be best possible. The following result is due to Howard Karloff (private communication).

THEOREM 6. *If there exists a* $(3-\epsilon)$*-approximation algorithm for the k-supplier problem for any fixed* $\epsilon > 0$*, then* $P = NP$.

PROOF. We show that such an algorithm could be used to solve the hitting set problem in polynomial time.

Hitting Set

INSTANCE: A collection $\mathscr{C}$ of subsets of a finite set S and a positive integer k.

QUESTION: Does there exist a subset $S' \subseteq S$ where $|S'| \leq k$ such that S' contains at least one element from each subset in $\mathscr{C}$?

Given an instance of the hitting set problem, form an instance of the k-supplier problem in the following way. Let $V_{\text{sup}} = S$ and let $V_{\text{cust}} = \mathscr{C} = \{D_1, \ldots, D_m\}$. Define the distance matrix $C = (c_{ij})$ to be

$$c_{ij} \begin{cases} 2 & \text{if } \{i, j\} \subseteq V_{\text{sup}} \text{ or } \{i, j\} \subseteq V_{\text{cust}}, \\ 3 & \text{if } i \in V_{\text{sup}} \text{ and } j \in V_{\text{cust}} \text{ and } i \notin D_j, \\ 1 & \text{if } i \in V_{\text{sup}} \text{ and } j \in V_{\text{cust}} \text{ and } i \in D_j. \end{cases}$$

It is straightforward to verify that these distances satisfy the triangle inequality. It is not hard to see that the k-supplier problem has a solution of cost 1 if and only if there is a k-hitting set S'. Furthermore, if there is no k-hitting set, the cost of the optimal k-supplier must be 3. Therefore, if the algorithm guarantees a ratio of $3 - \epsilon$ and there is a k-hitting set, the algorithm delivers a set of suppliers of cost 1. If there is no k-hitting set, then clearly the set of suppliers produced by the algorithm has a cost of 3. □

In another type of generalization of the k-center problem, the feasible choice of centers is restricted by several independent resource constraints. With a substantial increase in the bound of the performance guarantee, Theorem 4 can be extended to any fixed number of independent vertex weight constraints. That is, each vertex has m nonnegative weights, $w_i^{(1)}, \ldots, w_i^{(m)}$, and a center S must satisfy $\sum_{i \in S} w_i^{(j)} \leq k$ for all $j = 1, \ldots, m$. The following lemma is essential in the construction of our algorithm.

LEMMA 2. *Let* $D = \{v_1, \ldots, v_l\}$ *be a dominating set for a connected graph* G*. If* $m \leq l$*, then there exists a partition of* D *into* $D_1, D_2, \ldots, D_m$ *such that each set* D_i *is a dominating set in* G^{3m-2}.

PROOF. Partition the vertices of G into l parts, $V_1, \ldots, V_l$, so that $v_i \in V_i$ and $V_i \subseteq N_G(v_i)$. Form the graph $H = (D, F)$, where $(v_i, v_j) \in F$ if and only if some vertex in V_i is adjacent to some vertex in V_j. Note that H is connected and let T be any spanning tree of H. It is useful to observe that any dominating set in T is a dominating set in H and is also a dominating set in G^4. More generally, any dominating set in T^p is a dominating set in G^{3p+1}. Therefore, to prove the lemma, we need only find a partition of D so that each part is a dominating set in T^{m-1}.

Suppose that any single vertex in D is a dominating set in T^{m-1}. In this case constructing the partition is easy, since any partition suffices. Otherwise, suppose that $\{v\}$ is not a dominating set in T^{m-1}. Root the tree T at v. Partition the vertices of T by the length of the (unique) path from v to each vertex; let S_i denote the set of vertices that are distance i from v. By the choice of the root, each of $S_0, S_1, \ldots, S_m$ is nonempty. Let $D_{i+1} = \{v \mid v \in S_j, j = i \bmod m\}$. It is clear that these D_i form a partition of D. Furthermore, it follows directly from the construction that D_i is a dominating set in T^{m-1} for all i. □

FIGURE 8

```
procedure test(𝒢, G, 9m − 6)   {m-weighted k-center}
  for i := 1 to |V| do
    w_i ← Σ_{j=1}^{m} w_i^(j)
  call test.wkcenter(G^{3m−2}, 3)
```

The restriction in Lemma 2 that G be connected is unnecessary, since each connected component can be treated separately. Using Lemma 2 we can prove the following theorem.

THEOREM 7. *For the m-weighted k-center problem, the procedure bottleneck yields a* $(9m - 6)$*-approximation algorithm.*

PROOF. Let *test.wkcenter* denote the procedure *test* in Figure 6 that was used to prove Theorem 4. Consider the procedure in Figure 8.

Suppose that there is a feasible dominating set D in G; that is, $\sum_{i \in D} w_i^{(j)} \leq k$ for all $j = 1, \ldots, m$. Therefore,

$$\sum_{i \in D} w_i = \sum_{i \in D} \sum_{j=1}^{m} w_i^{(j)} = \sum_{j=1}^{m} \left(\sum_{i \in D} w_i^{(j)} \right) \leq mk.$$

Consider the partition promised by Lemma 2, and choose the part D_l such that $\sum_{i \in D_l} w_i$ is smallest. By the pigeonhole principle, it follows that $\sum_{i \in D_l} w_i \leq k$. Therefore, if G contains a feasible m-weighted center, then G^{3m-2} contains a feasible singly weighted center with the modified weights. As a consequence, if a certificate of failure is produced by the subroutine call *test.wkcenter*$(G^{3m-2}, 3)$ it is a valid certificate of failure for *test*. Furthermore, if the subroutine produces a dominating set D' in $(G^{3m-2})^3 = G^{9m-6}$ such that $\sum_{i \in D'} w_i \leq k$, it follows from the nonnegativity of the $w_i^{(j)}$ that $\sum_{i \in D'} w_i^{(j)} \leq k$ for each $j = 1, \ldots, m$. Thus, D' is a feasible center for G^{9m-6}. □

The unfortunate aspects of this result is that, although the bound is constant for a fixed value of m, the performance guarantee, even for $m = 2$, is not very good. For any fixed value of m, it is possible to find a solution guaranteed to be within a factor of 3 of the optimal solution, on the condition that we do not require the algorithm to be polynomial time, but only pseudopolynomial time. Furthermore, the algorithm has a running time that is exponential in m. Thus we are trading decreased efficiency for an improved performance guarantee. Further details can be found in an earlier version of this paper [10].

5. *Solving Routing Problems*

In this section we consider several routing problems. Most of the algorithms presented here make use of the following theorem.

THEOREM 8 [4]. *If G is a biconnected graph with more than* 2*-vertices, then* G^2 *contains a Hamiltonian cycle.*

Recently, Lau [13] and Parker and Rardin [16] independently found a polynomial-time algorithm that constructs a Hamiltonian cycle in the square of a biconnected graph. This algorithm is used as a subroutine in some of the results of this section, and we denote it *findtour*(G^2). As Parker and Rardin [16] show, this immediately leads to a 2-approximation algorithm for the bottleneck traveling salesperson problem, since, if G is not biconnected, this is a certificate of failure. Furthermore, biconnectivity can be tested in polynomial time.

Recall that for the *wandering salesperson problem*, we seek a route for the salesperson, from a fixed starting point to a fixed destination, that passes through all of the remaining cities. Under the bottleneck cost criterion, we wish to find such a route where the longest intercity distance traversed is as small as possible. Another interesting generalization of the traveling salesperson problem is the *vehicle routing problem*, where there are some k vehicles that may be used to traverse all of the cities. We consider the following variant: Designate some subset of the cities as potential endpoints of routes of the vehicles; this subset may contain no more than $2k$ points. In this model the initial and final endpoints of each route must be distinct. We wish to use at most k vehicles so that all of the cities, both the potential endpoints and the remaining ones as well, are visited exactly once. We refer to this problem as the *k-path vehicle routing problem*. Note that, if $k = 1$ and the subset contains two cities, then this problem reduces to the wandering salesperson problem mentioned above. We present a 2-approximation algorithm for the k-path vehicle routing problem (and thus for the wandering salesperson problem as well). It is possible to show that, even in the special case of the wandering salesperson problem, improving this guarantee would prove that P = NP.

As was done in the previous sections, we construct a procedure *test* that either produces a certificate of failure in G or produces a feasible routing in G^2. We call a collection of (simple) paths in G a *path covering* for G if each vertex of G is contained in exactly one path. If X is the set of potential endpoints, G has a feasible routing if there exists a path covering of G with at most k paths where the endpoints are in X; such a path covering is called *feasible*. The following lemma provides us with a suitable certificate of failure.

LEMMA 3. *If there exists a vertex v such that a component of $G - v$ does not contain a vertex of X, then G cannot contain a feasible path covering.*

PROOF. Suppose that the vertices of G can be covered with $l \leq k$ paths, and yet there exists a component of $G - v$ that does not contain a vertex of X. This component is nonempty; it must contain some vertex u, which must lie on one of the l paths, say P. Since removing v disconnects u from X, all paths from u to a vertex of X must pass through v. But then P cannot be a path, since the portions of this path from each of the endpoints of P to u must contain v. □

It is easy to see that it is possible to determine whether there exists such a certificate of failure in polynomial time. Although this condition seems weak, we can find a path covering in G^2 in the event that this certificate of failure is not found. The routing is found by constructing a biconnected graph H such that any Hamiltonian cycle found in H^2 gives a feasible routing in G^2. The construction used, which we call the H-construction, is a slight modification of one given by Chartrand et al [2].

Suppose that Lemma 3 does not provide a certificate of failure for $G = (V, E)$. Let $X \subseteq V$ be the subset of possible endpoints. We have assumed that $2 \leq |X| \leq 2k$. (If $|X| < 2$, then no path covering could exist.) Let $H_i = (V_i, E_i)$, $i = 1, \ldots, 5$ be five disjoint copies of G. Let x_1 and x_2 be two additional vertices not contained in $\bigcup_i V_i$. We construct a graph H consisting of the five copies H_i and the two vertices x_1 and x_2, where both x_1 and x_2 are adjacent to all of the vertices in the copies of X in each of the H_i.

It is not hard to see that H must be biconnected; that is, the removal of any one vertex cannot disconnect H. (This follows from the fact that no certificate of failure was found.) Therefore, we may use the procedure *findtour* to find a Hamiltonian

```
procedure test(𝒮, G, 2)  {k-path vehicle routing}
  if there exists v such that a component of G − v does not contain a vertex of X then
    output "certificate of failure" produced
  else
    form the H-construction on G
    call findtour(H²)
    {let τ denote the tour output by this procedure}
    let H_i be the copy of G such that no vertex in H_i is
    adjacent in τ to either x_1 or x_2
  output the portion of τ in H_i
```

FIGURE 9

cycle in H^2. Let x_{is} and x_{ip} be the successor and predecessor of x_i in this Hamiltonian cycle. Consider the four vertices x_{1s}, x_{2s}, x_{1p}, x_{2p}; there must be at least one copy H_i, such that none of these vertices is in V_i. Consider the portions of the Hamiltonian cycle that lie within H_i. These portions must form a path covering of H_i.

All that remains to be shown is that the endpoints of each of the paths must be contained in X. Suppose that this is not true. Suppose that one of these paths has an endpoint w not contained in X. Since it is an endpoint of one of these segments, it must either be preceded or succeeded on the Hamiltonian cycle by a vertex not in V_i. However, consider the graph H^2. If w is not equal to any copy of a vertex in X, then x_1 and x_2 are the only vertices not in V_i that are adjacent to w. Thus w must be either succeeded or preceded by either x_1 or x_2. This is a contradiction, since H_i was chosen so that this would not be true. Therefore, the path cover produced must have the desired property that each path has its endpoints in X. Since X contains at most $2k$ points, there cannot be more than k such paths, and we have obtained the required routing. Therefore, we have verified the correctness of the procedure in Figure 9 and thus have the following result.

THEOREM 9. *For the bottleneck k-path vehicle routing problem, the procedure bottleneck yields a 2-approximation algorithm.*

It is convenient to refer to the procedure used to find a feasible k path covering in the absence of a certificate of failure as *findpaths*(G, X).

Consider the variant of the wandering salesperson problem where we do not fix the endpoints of the route. We can run the procedure given above for all $\binom{n}{2}$ choices of endpoints to get a path that is again guaranteed to be within a factor of 2 of the optimum of all such paths. However, there is a more direct approach to finding an approximate solution in this case, and we outline it. If $G = (V, E)$ is a connected graph with $m > 1$ biconnected components, define the graph $B(G) = (V', E')$ in the following way. Let $V' = \{u_1, u_2, \ldots, u_m\} \cup \{v \mid v \text{ is a vertex in at least two biconnected components}\}$ where each u_i corresponds to a biconnected component and $u_i \notin V$. Let $E' = \{(u_i, v) \mid v \text{ is a vertex in the biconnected component corresponding to } u_i\}$. It is easy to see that $B(G)$ must be a tree (Figure 10). The reader is referred to [1] for a detailed exposition on the biconnected components of a graph.

THEOREM 10. *The procedure bottleneck yields a 2-approximation algorithm for the wandering salesperson problem without fixed endpoints.*

PROOF. First we show that the certificate of failure is produced correctly; that is, if a certificate is produced, then G does not contain a Hamiltonian path. If G is disconnected, then it clearly does not contain a Hamiltonian path. Consider any path in G, and consider the subgraph of this path that is induced by the vertices of

```
procedure test(𝒢, G, 2)  {wandering salesperson without fixed endpoints}
  if G is disconnected then output "certificate of failure" produced and return
  if G is biconnected then call findtour(G²)
  else
    if B(G) is not a path
      then output "certificate of failure" produced
      else
        for each biconnected component of G, Gᵢ with two vertices
        uᵢ and vᵢ that are contained in other components do
          call paths(Gᵢ, {uᵢ, vᵢ})
        for each of the two "end" biconnected components Gᵢ do
          call findtour(Gᵢ²)
          "paste" the paths together to obtain a Hamiltonian path
```

FIGURE 10

any biconnected component of G. Since a path cannot repeat vertices, this subgraph must also be a connected path. As a result, if G has a Hamiltonian path and is connected, but not biconnected, then the graph $B(G)$ is a path.

Now suppose that no certificate of failure is generated. If G is biconnected, *findtour* produces a Hamiltonian cycle, so any edge can be deleted from the cycle to form a Hamiltonian path. If G is not biconnected and no certificate of failure is produced, then $B(G)$ is a path. Consider a biconnected component G_i that contains two articulation points u_i and v_i. If there is a Hamiltonian path in G, there must be a Hamiltonian path for G_i from u_i to v_i. Fortunately, there is such a path in G_i^2 and it can be constructed by the same method used in the fixed endpoint case. Note that since G_i is biconnected, if $X = \{u_i, v_i\}$, then $G - v$ is connected, and therefore contains a vertex of X for every choice of v. As a result, the procedure *findpaths* is applicable, and will provide us with the desired Hamiltonian path from u_i to v_i. For each of the two biconnected components that share only one vertex with some other biconnected component, we may simply use *findtour* to construct a Hamiltonian cycle, and then convert this cycle into path by deleting one of the edges of the cycle incident to the vertex that is contained in another biconnected component. These pieces, together, form a Hamiltonian path in G. □

In a well-known relaxation of the bottleneck traveling salesperson problem, cities may be visited more than once, but edges may be used at most once. Thus, for this problem we wish to find the smallest value of c such that $G = BOTTLENECKG(c)$ contains a Hamiltonian walk. (A *walk* is a connected collection of cycles; that is, vertices are allowed to be repeated, but edges are not. A *Hamiltonian walk* is a walk that contains all of the vertices of a given graph.) Once again it is possible to prove the following result.

THEOREM 11. *For the bottleneck traveling salesperson problem with multiple visits allowed, the procedure bottleneck gives a* 2*-approximation algorithm.*

PROOF. For this algorithm we need to find a Hamiltonian walk in G^2, or else prove that G does not contain a Hamiltonian walk. If G is not connected, then it clearly does not contain a Hamiltonian walk; this will serve as our certificate of failure. If G is connected, then it contains a spanning tree T. We give a procedure for constructing a Hamiltonian walk in the square of a tree T rooted at a vertex v. To do this, we give a recursive procedure for finding a path in T^2 that visits every vertex, possibly repeating vertices but not edges. The *path* found starts at some child of v in T and ends at v. If T has more than 2 vertices, the edge between v and this child is not used in the path, and thus the path can be completed to a Hamiltonian walk by adding this edge. (Note that, despite its name, *walk* only

```
procedure walk(T, v)
  let u_1, u_2, ..., u_l be the children of v in T
  if v is adjacent to all other vertices of T
    then return (u_1, u_2, ..., u_l, v)
    else
      for all i = 1, ..., l do
        begin
        let T_i be the subtree of T rooted at u_i
        W_i ← walk(T_i, u_i)
        end
      W_1 ← W_1 traversed in the reverse direction
      return (W_1, v, W_2, ..., v, W_l, v)
```

FIGURE 11

produces this path, and not the entire walk.) It is not hard to verify that the procedure *walk* (Figure 11) does produce such a path. □

In the *single-depot k-vehicle routing* problem we wish to choose a routing where all (nondepot) cities are serviced exactly once by at most k vehicles, such that the longest intercity distance traveled is minimized. An instance of this problem consists of a complete edge-weighted graph G_C, an integer k, and a special vertex v; this is the depot. We construct a 2-approximation algorithm for this problem by reducing it to the ordinary traveling salesperson problem. In order to do this, we show how to transform an instance of the single-depot problem G_C into an instance of the traveling salesperson problem $G_{C'}$, such that G_C has a feasible vehicle routing solution of cost c if and only if $G_{C'}$ has a feasible traveling salesperson solution of cost c. We construct the instance $G_{C'}$ as follows. We add $k-1$ vertices $\{u_1, u_2, \ldots, u_{k-1}\} = U$ so that the new vertex set is $V \cup U$ where $V \cap U = \varnothing$. Define the edge costs in the following way:

$$c'_{ij} = \begin{cases} c_{ij}, & i, j \in V, \\ 0, & i, j \in U \cup \{v\}, \\ c_{iv}, & i \in V - \{v\}, \quad j \in U. \end{cases}$$

It is straightforward to verify that these new edge costs also satisfy the triangle inequality.

LEMMA 4. *$G_{C'}$ has a traveling salesperson tour of bottleneck cost c if and only if G_C has a vehicle routing using at most k vehicles of bottleneck cost c.*

PROOF. Suppose that $G_{C'}$ has a traveling salesperson tour of bottleneck cost c. Let $e_1, \ldots, e_m$ be the edges of this tour, in order. If $e_i = (v_p, v_q)$, simply delete it from the sequence. If $e_i = (v_p, x)$ for some $x \in V - \{v_1\}$, then replace e_i with (v_1, x). It is easy to see that the resulting sequence of edges is a feasible routing for G_C of cost c. Suppose that G_C has a vehicle routing of bottleneck cost c that uses l vehicles, where $l \le k$. Suppose that vehicle i's route consists of $v_1, v_{i1}, v_{i2}, \ldots, v_1$. Consider the cycle in $G_{C'}$,

$$v_1, v_{11}, v_{12}, \ldots, v_2, v_{21}, v_{22}, \ldots, v_3, \ldots, v_l, v_{l1}, \ldots, v_{l+1}, \ldots, v_k, v_1.$$

Once again it is easy to verify that this is a traveling salesperson tour of cost c. □

Note that Lemma 4 holds for both the ordinary cost and the bottleneck cost objectives. Therefore, we may apply Christofides' algorithm to get a 3/2-approximation algorithm for the total-cost single-depot problem [3]. A similar algorithm was obtained recently by Frieze [5]. For the bottleneck cost criterion we have obtained the following result.

THEOREM 12. *The procedure bottleneck yields a 2-approximation algorithm for the single-depot k-vehicle routing problem.*

It is significant to note that *all* of the problems in this section have the property that if there existed approximation algorithms for them with a better performance guarantee, then there would exist polynomial-time algorithms to solve them to optimality. As an example, consider the following result for the k-path vehicle routing problem.

THEOREM 13. *For every fixed $\epsilon > 0$ and every fixed integer $k > 0$, if there exists a $2 - \epsilon$ approximation algorithm to solve the k-path vehicle routing problem with the triangle inequality, then P = NP.*

PROOF. We show that given such an algorithm, we could solve the Hamiltonian path problem in polynomial time. Since this problem is known to be NP-complete, this implies that P = NP. Suppose that the graph G is the instance of the Hamiltonian path problem. Make k copies of this graph. For each copy add two vertices that are adjacent to all of the vertices in the copy. Call this graph H, and let X consist of all these added vertices. It is easy to verify that G has a Hamiltonian path if and only if there are some k paths covering the vertices of H such that all of these paths have distinct endpoints in X. Now we create the weighted instance of the k-path vehicle routing problem by assigning edges of H a weight of 1, and nonedges a weight of 2. It is easy to see that the weights for the completed graph satisfy the triangle inequality. Therefore we may use the assumed algorithm. If there is good covering of H, then the bottleneck cost is 1. The algorithm must then produce a routing of cost no more than $2 - \epsilon$. But this must then be a routing of cost 1, and from this routing it is easy to extract the Hamiltonian path of G. Therefore, if there is a Hamiltonian path in G, the algorithm will find it. If the approximation algorithm produces a routing of cost 2, we know that G does not contain a Hamiltonian path. □

6. *Conclusions*

In this paper we have presented a very general technique for creating approximation algorithms for problems in location theory, routing, and communication network design. We have exhibited a wide range of problems for which this technique is applicable. In every case, the algorithm given produces an approximate solution that is guaranteed to be within a constant factor of the optimal solution. One of the most attractive features of this technique is the great flexibility that it has in dealing with additional constraints on feasible solutions, within precisely the same framework. Furthermore, for many of the problems considered, the algorithm produced by this technique is best possible, in the very strong sense that the existence of an algorithm with a better performance guarantee would imply that P = NP.

A salient feature of this approach is that for *each instance* a lower bound on the optimal value is produced, in addition to an approximate solution. As a result, these heuristics are especially well suited for use within a branch-and-bound routine. Empirical results for the k-center problem indicate that such an approach may indeed be a very fruitful one for obtaining optimal solutions to bottleneck problems.

REFERENCES

(Note: Reference [9] is not cited in text.)

1. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Pa., 1974, pp. 179–187.

2. CHARTRAND, G., HOBBS, A. M., JUNG, H. A., KAPOOR, S. F., AND NASH-WILLIAMS, C.ST. J. A. The square of a block is Hamiltonian connected. *J. Comb. Theory, Ser. B 16* (1974) 290–292.
3. CHRISTOFIDES, N. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon Univ., Pittsburgh, Pa., 1976.
4. FLEISCHNER, H. The square of every two-connected graph is Hamiltonian. *J. Comb. Theory, Ser. B 16* (1974), 29–34.
5. FRIEZE, A. M. An extension of Christofides heuristic to the k-person travelling salesman problem. *Discr. Appl. Math. 6* (1983), 79–83.
6. GAREY, M. R. AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.
7. GONZALEZ, T. F. Clustering to minimize the maximum inter-city distance. Tech. Rep. 117, Computer Science Dept., Univ. of Texas, Dallas, 1982.
8. HOCHBAUM, D. S. When are NP-hard location problems easy? *Ann. Oper. Res. 1* (1984), 201–214.
9. HOCHBAUM, D. S. Easy Solutions for the k-Center Problem or the Dominating Set Problem on Random Graphs. *Ann. Disc. Math. 25* (1985), 189–210.
10. HOCHBAUM, D. S., AND SHMOYS, D. B. Powers of graphs: A powerful approximation algorithm technique for bottleneck problems. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing* (Washington, DC, Apr. 30–May 2). ACM, New York, 1984, pp. 324–333.
11. HOCHBAUM, D. S., NISHIZEKI, T., AND SHMOYS, D. B. A better than "best possible" algorithm to edge color multigraphs. *J. Algorithms*, to appear.
12. HSU, W. L., AND NEMHAUSER, G. L. Easy and hard bottleneck location problems. *Disc. Appl. Math. 1* (1979), 209–216.
13. LAU, H. T. Finding a Hamiltonian cycle in the square of a block. Ph.D. dissertation, McGill Univ., Montreal, Que., Canada, 1980.
14. LIEBERHERR, K. J., AND SPECKER, E. Complexity of partial satisfaction. *J. ACM 28*, 2 (Apr. 1981), 411–421.
15. NISHIZEKI, T., AND SATO, M. An approximation algorithm for edge-coloring multigraphs. Preprint.
16. PARKER, R. G., AND RARDIN, R. L. Guaranteed performance heuristic for the bottleneck travelling salesman problem. *Oper. Res. Lett. 6* (1982), 269–272.

RECEIVED JULY 1984; REVISED AUGUST 1985; ACCEPTED SEPTEMBER 1985