The Bottleneck Graph Partition Problem

Dorit S. Hochbaum

Department of Industrial Engineering and Operations Research and Walter A. Haas School of Business, University of California, Berkeley, Berkeley, California 94720-1777

Anu Pathria

Department of Industrial Engineering and Operations Research, University of California, Berkeley, Berkeley, California 94720

The bottleneck graph partition problem is to partition the nodes of a graph into two equally sized sets, so that the maximum edge weight in the cut separating the two sets is minimum. Whereas the graph partition problem, where the *sum* of the edge weights in the cut is to be minimized, is NP-hard, the bottleneck version is polynomial. This paper describes an $O(n^2 \log n)$ algorithm for the bottleneck graph partition problem, where n is the number of nodes in the graph. We point out two interesting issues related to dynamic algorithms. We also generalize our polynomiality result (for fixed k) to the bottleneck k-cut problem with specified vertices and bounded components. © *1996 John Wiley & Sons, Inc.*

INTRODUCTION

The *graph partition* problem is defined on a graph, $G = (V, E)$, with a set of nodes V and a set of edges E. We will consider the weighted version where each edge $e \in E$ has a weight w_e associated with it. The problem is to find, among all partitions of V into equally sized sets V_1 and V_2, the partition that minimizes the total weight of the edges in the cut separating V_1 from V_2. The graph partition decision problem was proved NP-complete in [3]. Many heuristics have been developed for the problem, the best known being the Kernighan–Lin heuristic [6].

In this paper, we consider the *bottleneck graph partition* problem, which is similar to the graph partition problem described above, but in which the *maximum* weight edge in the cut is to be minimized. Monma and Suri [7] and Brucker [1] also tested for bipartition of a graph, in polynomial time. The difference here is that the sets have to be of a particular size; this aspect has not been addressed before. Our problem is also similar to the *p-center* problem (see [2]), except that we have to choose $p = n/2$ "suppliers" and $n/2$ "customers" so as to minimize the maximum cost associated with *any* supplier–customer pair (i.e., every customer needs supplies from *each* of the suppliers).

It is frequently the case that the bottleneck version of a min-sum NP-complete problem is also NP-complete. This is the case for the *p-center* problem, which is the bottleneck version of the *p-median* problem. Another example is the *bottleneck traveling salesperson* problem which, like the *traveling salesperson* problem, is also NP-complete. By contrast, however, we will show that the bottleneck graph partition problem (as well as a generalized version of the problem related to the K-cut problem) is polynomial, despite the fact that the regular min-sum graph partition problem is NP-complete.

The technique that we employ for this problem is of the same nature as the *Bottleneck Algorithm* used by Hochbaum and Shmoys [4], where it was applied to numerous NP-complete bottleneck problems to provide approximation algorithms for the given problems. There, the algorithm was used in conjunction with a "test" that was an NP-complete decision problem; solving the "test" problem on a related graph (a power of the original graph), however, provided sufficient information for the derivation of an *approximation* algorithm running in polynomial time. Since the "test" used for the bottleneck graph partition problem is *not* NP-complete, the adaptation of the bottleneck algorithm here is polynomial and provides an *optimal* solution.

A straightforward iterative implementation of our algorithm runs in time $O(mn^2)$. Two improvements may be possible to the iterative procedure: First, a simple improvement involves a dynamic maintenance of components in a graph to which edges are added one at a time; second, the algorithm for the graph partition may also be improved by incorporating a dynamic maintenance of solutions to the *subset-sum* problems that arise. We, however, use a binary search, rather than iterative, implementation that provides an $O(n^2 \log n)$ algorithm for the problem. Potential improvements in the dynamic algorithms mentioned above, however, may lead to an algorithm with improved running time.

1. THE BOTTLENECK ALGORITHM

Given is a weighted graph $G = (V, E)$ with n nodes and m edges. Without any loss in generality, we will assume that the graph G is connected and that the number of nodes n is even. Let the edges be sorted in nonincreasing weight:

$$w_{e_1} \geq w_{e_2} \geq \cdots \geq w_{e_m}.$$

We wish to partition the nodes of the graph into two equally sized sets, so that the maximum edge weight in the cut separating the two sets is minimum.

We define $G_i = (V, E_i)$ to be an unweighted subgraph of G with the set of edges $E_i = \{e_1, \ldots, e_i\}$, for $i = 1, \ldots, m$.

Consider the following test, which is equivalent to the decision problem of the graph partition problem: "Is there a partition with cut weight zero?":

Test (G_i). *Is there a partition of the connected components of G_i into two sets each containing $n/2$ nodes?*

We will later show that **Test** (G_i) can be answered in polynomial time, but first note the following straightforward lemma and resulting corollary:

Lemma 1.1. **Test** (G_i) = *"yes"* $\Leftrightarrow$ $\exists$ *a solution of cost less than* w_{e_i}.

Proof. [$\Rightarrow$] Suppose **Test** (G_i) = "yes." Then, there exists a partition of the nodes of G into two equally sized sets V_1 and V_2 such that no edges in G_i cross between V_1 and V_2. Therefore, any edge in G that crosses between V_1 and V_2 must have weight less than w_{e_i}, i.e., we have a solution with cost less than w_{e_i}.

[$\Leftarrow$] Now assume that there exists a solution of cost w_{e_j}, where $w_{e_j} < w_{e_i}$. Then, there is a partition of the vertices into two equally sized sets V_1 and V_2 such that no edge with weight greater than w_{e_j} crosses between V_1 and V_2. In other words, **Test** (G_k) = "yes" for any k such that $w_{e_k} > w_{e_j}$; in particular, **Test** (G_i) = "yes." ■

```
1. Let $\mathcal{C}$ be the set of connected components of $G_i$.
2. Let $c_1, \ldots, c_p$ be the sizes of the components in $\mathcal{C}$.
   Call subset-sum, with $B = n/2$:
      if subset-sum returns "no" then output "no";
      else output "yes" ($V_1$ contains the components chosen for the set $S$).
```

Fig. 1. Algorithm Test (G_i)

Corollary 1.1. *Let* $j = \text{argmin}_i\ \{$**Test** $(G_i) = $ *"no"*$\}$. *The optimal solution to the bottleneck graph partition problem is* w_{e_j}.

The above corollary leads immediately to the following algorithm:

```
procedure Bottleneck
  for $i = 1 \cdots m$ do
    if Test $(G_i)$ = "no" then output $w_{e_i}$; stop else
    continue
  end
```

We show now that **Test** (G_i) can be solved in polynomial time. Each application of the test involves finding the sizes of the connected components of the graph G_i. The connected components and their sizes can be found in time $O(|E_i|) = O(m)$. Let these sizes be $c_1, \ldots, c_p$. The problem is then an instance of the subset-sum problem with $B = n/2$:

Subset Sum: *Given values $c_1, \ldots, c_p$ and B, is there a subset, $S \subseteq \{1, \ldots, p\}$, such that $\Sigma_{i \in S} c_i = B$?*

The problem of subset-sum is *weakly* NP-complete (see [5]): It is NP-complete, but there is a dynamic programming algorithm running in time polynomial in B, namely, $O(pB)$. Since in our case $B = O(n)$ and $p = O(n)$, the running time of the dynamic programming algorithm is $O(n^2)$. See Figure 1 for a description of the **Test** (G_i) algorithm; details of the dynamic programming procedure used to solve the subset-sum problem can be found in the next section, where we consider a generalization of the bottleneck graph partition problem.

A straightforward implementation of **Bottleneck** requires m applications of the test, leading to an overall running time of $O(mn^2)$ for **Bottleneck.**

We now propose several possible improvements to speed up the running time of **Bottleneck.**

The **Bottleneck** algorithm searches for the optimal w_{e_i}, as described in Corollary 1.1, in an iterative manner. The best running time that we have been able to achieve is realized by employing a binary, rather than iterative, search for the optimal w_{e_i}. We can reduce the number of calls to **Test** (G_i) from $O(m)$ to $O(\log m)$ and thereby find the optimal w_{e_i} in time $O(n^2 \log n)$.*

Another approach is to improve the running time of an iterative implementation of **Bottleneck.** For instance, the application of the algorithm can be preceded by a preprocessing step, producing for each graph G_i the list of sizes of its connected components. As this is equivalent to the *disjoint sets union-find* problem, it can be accomplished in time $O(m\alpha(m, n))$, where α is Tarjan's inverse of Ackermann's function (see [8]). Since for the graph $G_0 = (V, E_0)$ the number of components is n (all isolated vertices), there are, in the process of scanning the lists for G_i, $i = 1, \ldots, m$, at most n steps when an edge is added and the sizes of components of the graph are changed. Hence, we need consider only the indices $i_1, \ldots, i_p$ that correspond to the highest index graph prior to a change in the number of connected components. Consequently, the procedure **Bottleneck** only goes through at most n steps rather than m steps, yielding a running time of $O(m\alpha(m, n) + n^3)$, which is $O(n^3)$.

Notice that the $O(n^3)$ term dominates the running time of the above procedure. Hence, a second *potential* improvement to an iterative **Bottleneck** may be realized by improving the efficiency of our dynamic programming procedure that solves the subset-sum problem arising in **Test** (G_i). Observe that two consecutive calls to the subset-sum differ only in that, in the latter call, some of the subset sizes have been combined. Thus, it may be possible to take advantage of earlier calls to reduce the $O(n^2)$ running time of the dynamic programming procedure and thereby provide an implementation of **Bottleneck** that has a better running time than has the $O(n^2 \log n)$ algorithm. We have not, however, been able to find such an improvement.

2. GENERALIZATIONS

Using the same general strategy as above, we can easily handle several generalizations to the bottleneck graph partition problem. It is convenient to first consider the *k-subset-sum* problem that will arise when solving what we will refer to as the *generalized bottleneck partition problem.*

* Note that we are assuming that the edge weights are presorted; otherwise, we must add $O(m \log n)$ to the running times. This does not affect the asymptotic running time of our algorithm, but could be the bottleneck in an "improved" algorithm.

k-Subset Sum: *Given values $c_1, \ldots, c_p$, and $A_1, \ldots, A_k$, $B_1, \ldots, B_k$, is there a partition of the set $\{1, \ldots, p\}$ into k subsets, $S_1, \ldots, S_k$, such that $A_j \le \sum_{i \in S_j} c_i \le B_j$, for $j = 1, \ldots, k$?*

Observe that the subset-sum problem is a special case of the k-subset-sum problem, with $k = 2$ and $A_j = B_j$. As is the case for the subset-sum problem, the k-subset-sum problem is weakly NP-complete. We can solve the problem in $O(pk \prod_{j=1}^{k-1} B_j)$ time using a dynamic programming procedure, as follows:

Let

$$T_i(n_1, \ldots, n_{k-1}) = \begin{cases} \textit{true}, & \text{if the numbers } c_1, \ldots, c_i \text{ can be partitioned} \\ & \text{into } k \text{ sets with the } j\text{'th set having weight } n_j \\ & \text{(Note: the weight of } n_k \text{ is implicit from the other parameters)} \\ \textit{false}, & \text{otherwise.} \end{cases}$$

The function, T, satisfies the following recurrence and boundary condition:

Boundary Condition:

$$T_0(0, \ldots, 0) = \textit{true}$$

Recurrence:

$$T_i(n_1, \ldots, n_{k-1}) = T_{i-1}(n_1 - c_i, \ldots, n_{k-1}) \text{ or } \cdots \text{ or } T_{i-1}(n_1, \ldots, n_{k-1} - c_i) \text{ or } T_{i-1}(n_1, \ldots, n_{k-1})$$

The answer to the k-subset-sum problem is "yes" if $T_p(N_1, \ldots, N_{k-1}) = \textit{true}$ for some $\{N_j\}$ satisfying $A_j \le N_j \le B_j$, for $1 \le j \le k$ (again, note that N_k is implicitly given).

The function values of T can be calculated in increasing values of $n_1, \ldots, n_k$, i, for $0 \le n_j \le B_j$ and $0 \le i \le p$. Each value of T can be evaluated in $O(k)$ time, leading to the $O(kp \prod_{j=1}^{k-1} B_j)$ running time. Clearly, it is advantageous to let $c_k = \max_i \{c_i\}$.

```
1. Let $\mathcal{C}$ be the set of connected components of $G_i$.
2. Initialize $V_j \leftarrow \emptyset, j = 1, \ldots, k$.
3. for $C \in \mathcal{C}$ do:
   a) if $\exists j_1 \neq j_2$ such that $C \cap W_{j1} \neq \emptyset$ and $C \cap W_{j2} \neq \emptyset$ then:
        output "no";
        STOP.
   b) if $C \cap W_j \neq \emptyset$ then:
        $\mathcal{C} \leftarrow \mathcal{C} - \{C\}$;
        $f_j(n) \leftarrow f_j(n) - |C|$;
        $g_j(n) \leftarrow g_j(n) - |C|$;
        $V_j \leftarrow V_j \cup \{C\}$.
4. Let $c_1, \ldots, c_p$ be the sizes of the components that remain in $\mathcal{C}$.
   Call $k$-subset-sum, with $A_j = f_j(n)$ and $B_j = g_j(n)$, for $j = 1, \ldots, k$:
     if $k$-subset-sum returns "no" then output "no";
     else output "yes" ($V_j \leftarrow V_j \cup S_j$, where $S_j$ contains the components chosen for set $j$).
```

Fig. 2. Algorithm Test (G_i) for generalized problem.

We now define the generalized bottleneck graph partition problem:

Instance: A weighted graph $G = (V, E)$ and integer k, specified subsets of the vertices, W_j, and integers $f_j(n)$, $g_j(n)$, for $j = 1, \ldots, k$. A *legal* partition of the vertices, V, into k subsets, $V_1, \ldots, V_k$, satisfies, $\forall j$: $g_j(n) \leq |V_j| \leq f_j(n)$, and $W_j \subseteq V_j$.

Optimization Problem: Find a legal partition of the vertices such that the maximum weight edge crossing between any two of the sets, V_{j1} and V_{j2}, is minimum.

To handle this generalized problem using **Bottleneck,** we proceed exactly as in the previous section, but modify the "test":

Test (G_i): *Is there a partition of the connected components of G_i that results in a legal partition?*

Consider the connected components of G_i. If there exists a component, C, such that $C \cap W_j \neq \emptyset$, then C must be in V_j. It follows that if there is a component C such that $C \cap W_{j1} \neq \emptyset$ and $C \cap W_{j2} \neq \emptyset$, for j_1 and j_2 different, then the answer to **Test** (G_i) = "no." Assuming that this is not the case, we can immediately assign some of the connected components to particular sets, V_j. Of the remaining components, we wish to assign them to the V_j's to create, if possible, a legal partition; this can be accomplished via the k-subset-sum procedure. See Figure 2 for the details of the algorithm.

Theorem 2.1. *The generalized bottleneck graph partition problem can be solved in $O(\max\{m, knF\}\log n)$ time, where $F = \prod_{j=1}^{k-1} f_j(n)$.*

Proof. The correctness of the algorithm for the generalized graph partition problem follows from the correctness of **Test** (G_i).

Each call to test **Test** (G_i) requires finding the connected components of G_i and an application of the k-subset-sum problem. The connected components can be found in time $O(m)$, and the k-subset-sum problem can be solved in time $O(knF)$, since $p = O(n)$ and $B_j = O(f_j(n))$. Using a binary search procedure, we make $O(\log n)$ calls to the "test," which yields the overall running time. ■

We have provided an algorithm for the generalized graph partition problem which has polynomial running time for k fixed. If, however, k is considered to be part of the input, then the generalized graph partition problem is NP-hard.

Theorem 2.2. *No polynomial algorithm exists for solving the generalized graph partition optimization problem, unless $P = NP$.*

Proof. We proceed by showing that a polynomial algorithm for the generalized graph partition problem implies the existence of a *pseudopolynomial* algorithm for the 3-*partition* problem. Since 3-partition is known to be *strongly* NP-complete (see [2]), such an algorithm would imply that $P = NP$.

Consider an instance of 3-partition: a set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$, such that each $s(a)$ satisfies $(B/4) < s(a) < (B/2)$ and such that $\Sigma_{a \in A}\, s(a) = mB$. The problem is to find a partition of A into m disjoint sets $S_1, S_2, \ldots, S_m$ such that $\Sigma_{a \in S_i}\, s(a) = B$, for $i = 1, \ldots, m$.

Now, for such an instance of 3-partition, construct a complete weighted graph $G = (V, E)$ as follows: $V = \cup_{i=1}^{3m} X_i$, where $|X_i| = s(a_i)$; if $e \in E$ is an edge between two vertices in the same W_i, then assign it weight 1, and otherwise, assign it weight 0. It is easy to see that the optimal solution to the generalized graph partition problem on G (with $k = m$ and, for $j = 1, \ldots, k, f_j(n) = g_j(n) = B$ and $W_j = \emptyset$) is 0 if and only if a solution exists to the 3-partition instance. Since $k = m$ and $|V| = n = mB$, and noting that the reduction described can be carried out in pseudopolynomial time, it follows that a polynomial algorithm for a generalized graph partition implies a pseudopolynomial algorithm for 3-partition. ■

3. SUMMARY

We have shown that the bottleneck graph partition problem can be solved in polynomial time, despite the fact that the min-sum version of the problem is NP-hard. Our algorithm uses the *Bottleneck Algorithm* described in [4] and takes advantage of the pseudo-polynomiality of the subset-sum problem. Our algorithm runs in time $O(n^2 \log n)$, where n is the number of vertices in the graph; for dense graphs, this running time is asymptotically optimal for any algorithm that requires, but does not presume, the edge weights to be sorted, since we have a lower bound of $O(m \log m)$ for sorting (of course, we have not established that having sorted edge weights is necessary for solving the bottleneck graph partition problem).

Having established the polynomiality of the bottleneck graph partition problem, it may be interesting to discover other polynomial algorithms using different approaches. For instance, perhaps some type of iterative local neighborhood search algorithm is possible. We observe, however, that a simple neighborhood based on swapping vertices between sets is not exact, i.e., a local improvement algorithm using a neighborhood based on swapping vertices between sets can get trapped in minima that are locally, but not globally, optimal.

This research was supported in part by ONR Contract N00014-91-J-1241 (D.S.H.). A.P. was supported in part by an NSERC '67 scholarship provided by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] P. Brucker, On the complexity of clustering problems. *Lecture Notes in Operations Research and Mathematical Systems*, Vol. 25 (R. Henn, Ed.). Springer-Verlag, Berlin (1970).

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco (1979).

[3] M. R. Garey, D. S. Johnson, and L. Stockmeyer, Some simplified NP-complete graph problems. *Theor. Comput. Sci.* **1** (1976) 237–267.

[4] D. S. Hochbaum and D. Shmoys, A unified approach to approximation algorithms for bottleneck problems. *J. Assoc. Comput. Mach.* **33**(3) (1986) 533–550.

[5] R. M. Karp, *Reducibility Among Combinatorial Problems.* Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, Eds.). Plenum Press, New York (1972) 85–103.

[6] B. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning. *Bell Syst. Tech.* **49** (1976) 291–307.

[7] C. Monma and S. Suri, Partitioning points and graphs to minimize the maximum or the sum of diameters. *Proceedings of the Sixth International Conference on the Theory and Applications of Graphs.* Wiley, New York (1988).

[8] R. E. Tarjan, *Data Structures and Network Algorithms.* Regional Conference Series in Applied Mathematics, No. 44. SIAM, Philadelphia, PA (1983).

Received December 1, 1993
Accepted July 2, 1996