ELSEVIER

Discrete Applied Mathematics 74 (1997) 159–169

DISCRETE APPLIED MATHEMATICS

k-edge subgraph problems

Olivier Goldschmidt [a,1], Dorit S. Hochbaum [b,*]

[a] *Department of Mechanical Engineering, The University of Texas at Austin, TX, USA*
[b] *I.E. & O.R. Department, University of California at Berkeley, Etcheverry Hall, Berkeley, CA 94720-1777, USA*

Received 7 November 1994; revised 9 February 1996

Abstract

We study here a problem on graphs that involves finding a subgraph of maximum node weights spanning up to k edges. We interpret the concept of "spanning" to mean that at least one endpoint of the edge is in the subgraph in which we seek to maximize the total weight of the nodes. We discuss the complexity of this problem and other related problems with different concepts of "spanning" and show that most of these variants are NP-complete. For the problem defined, we demonstrate a factor 3 approximation algorithm with complexity $O(kn)$ for a graph on n nodes. For the unweighted version of the the problem in a graph on m edges we describe a factor 2 approximation algorithm of greedy type, with complexity $O(n+m)$. For trees and forests we present a polynomial time algorithm applicable to our problem and also to a problem seeking to maximize (minimize) the weight of a subtree on k nodes.

1. Introduction

In this paper, we consider a problem we call the k-edge-incident subgraph problem. In this problem one seeks to maximize the total weight on the nodes on a subgraph that spans up to k edges. We consider an edge to be spanned by a subgraph if at least one of its endpoints is in the subgraph.

Formally, let $G=(V,E)$ be a simple and undirected graph with vertex set V and edge set E. Let $\omega(v) \geqslant 0$ be the weight of node $v \in V$. If $\omega(v)=1, \forall v \in V$, then G is said to be unweighted. The *k-edge-incident* subgraph problem is to find a set of nodes $W \subset V$ of maximum weight such that the number of edges with at least one endpoint in W is at most k.

Our study of the k-edge-incident problem was motivated by an application to the loading of semi-conductor components to be assembled into products [5]. When the buffer can accommodate up to k different components, and each component is to be a

* Corresponding author.

[1] This research has been supported in part by ONR grant N00014-91-J-1241.
[2] This research has been supported in part by ONR grant N00014-91-J-1241.

PII S0166-218X(96)00030-3

part of precisely two products, the problem of choosing the components so the maximum number of products can be produced is the k-edge-incident subgraph problem. If each product has a different weight corresponding to, say – its profit contribution, the problem is to maximize the total weight of the nodes in the subgraph.

We define an algorithm to be a ρ-approximation algorithm for a maximization problem if it delivers a feasible solution the value of which is at least ρ times the optimum. Obviously, $0 < \rho < 1$. $\frac{1}{\rho}$ is frequently referred to as the *approximation factor*.

In this paper, we show that the problem of finding a set of nodes $W \subset V$ of maximum cardinality such that the number of edges with at least one endpoint in W is at most k is NP-complete in the strong sense. Further, we show that the problem remains NP-complete even if restricted to graphs with maximum degree equal to three.

We present approximation algorithms for the k-edge-incident subgraph problem: For general unweighted graphs we introduce a greedy method for finding a maximum cardinality set of nodes W such that the number of edges with at least one endpoint in W is at most k. The cardinality of a set obtained by such greedy algorithm is proved to be at least $\frac{1}{2} - o(1)$ times the size of an optimal solution. For weighted graphs, we provide a $\frac{1}{3}$-approximation algorithm of complexity $O(kn)$. Both approximations are based on an idea of relaxing the problem to a Knapsack-like problem.

We show that the k-edge-incidence subgraph problem is solvable in $O(k^2|V|)$ time for graphs that are acyclic and weighted. The unweighted case is solvable in linear time for unweighted trees or unweighted forests even with simple unweighted cycles. This k-edge problem on trees is equivalent to a k-node problem on a tree. Given a tree with weighted nodes, find a subtree (rooted at a specific node or without any rooting specification) on k nodes that maximizes the sum of node weights. This latter problem comes up in the context of database organization, and thus the algorithm stated proves the polynomiality of that problem.

Related problems. It is useful to view the k-edge-incident problem as one in a class of problems where one is seeking a subgraph that restricts the number of spanning edges or the number of nodes. Consider the following two problems that also define a subgraph constrained by the number of edges it spans. Here however the definition of spanning is different.

a. *k-edge-in* subgraph: Find a set of nodes $W \subset V$ of maximum weight such that the subgraph induced on W has at most k edges.

b. *k-edge-cut* subgraph: Find a set of nodes $W \subset V$ of maximum (minimum) weight such that the number of edges with exactly one endpoint in W is at most k.

Like the k-edge-incident subgraph problem the k-edge-in problem is also NP-complete in the strong sense. To see this consider the recognition version of the k-edge-in subgraph problem in an unweighted graph:

Instance: A graph $G = (V, E)$ and integers k and L.

Question: Is there a set of nodes $W \subset V$, with $|W| \geqslant L$, such that the number of edges with both endpoints in W is at most k?

The k-edge-in subgraph problem is to find an induced subgraph on a maximum number of nodes and with at most k edges. For $k = 0$, this problem is equivalent to the maximum independent set problem. For $k > 0$, the problem was proved to be strongly NP-hard by Yannakakis [12]. For either $k = 0$ or $k > 0$ the problems remain NP-complete even if the input graph is cubic or planar [12].

The k-edge-cut problem is trivial: for maximization, the optimal solution is $W = V$, and for minimization – $W = \emptyset$. The versions of the k-edge-in and k-edge-incident subgraph where the number of "spanning" edges is required to be *exactly* equal to k are easily shown to be NP-complete by a reduction from the maximum cut problem.

Other related subgraph problems where the number of *nodes* in the subgraph is restricted to k, we call *k-node subgraph problems*. Such problems have been the subject of investigation recently, and it appears that they are not only NP-complete, but also for several of them it is more difficult to achieve good approximations.

One such k-node subgraph problem is the *maximum density* problem on a subgraph with k nodes at most. The density of a subgraph is the number (or weight) of edges divided by the number (or weight) of the nodes in the subgraph. Without the restriction on the number of nodes, the problem is known to be polynomial by a reduction of repeated calls to the selection problem, and applications of a minimum cut algorithm (see [7, 4]). With the restriction on the number of nodes, the problem is easily seen to be NP-hard by a reduction from the k-clique problem.

In [8], Kortsarz and Peleg presented a factor $\tilde{O}(n^{0.3885})$ approximation algorithm for the maximum density k-node subgraph problem (here $\tilde{O}$ means that polylog factors are omitted), i.e. the optimum value could be up to to $\tilde{O}(n^{0.3885})$ times the algorithm's solution value. For the special case where the weights on the edges obey the triangle inequality, a greedy approach guarantees a solution of weight at least $\frac{1}{4}$ the optimal (see [10]).

An interesting variant (pointed out by one of the anonymous referees) is when the density is measured in terms of the number of incident edges (i.e. those that have at least one endpoint in the subgraph) divided by the number of nodes in the subgraph. It is easy to see that in this case a greedy algorithm choosing a node of maximum degree at a time, will deliver a solution that is at least $\frac{1}{2}$ times the value of the optimum.

Another k-node subgraph problem is to find k nodes so that the subgraph's minimum spanning tree is of least weight compared to all subgraphs on k nodes. This problem was shown NP-complete by Ravi et al. ([11]). [11] also described a $O(\sqrt{k})$-approximation algorithm for general graphs and an $O(k^{1/4})$ approximation algorithm for Euclidean graphs. In [2], Garg and Hochbaum introduced a $O(\log k)$-approximation algorithm for the problem in the plane. For general graphs a $O(\log^2 k)$-approximation algorithm was recently established by Awerbuch et al. [1], and for Euclidean graphs Mitchell demonstrated a factor of 2 and factor of $2\sqrt{2}$ approximation algorithms for the L_1 and L_2 metrics, respectively [9].

The *sparsest* k-node subgraph problem is to find k nodes with minimum sum of edge weights connecting them. The question of whether the graph contains an independent set of size k is reducible to the sparsest subgraph problem and hence the latter is

NP-complete in the strong sense. If edge weights can be zero, then any approximation algorithm is infinitely bad, and therefore the problem is only meaningful for positive weights.

Another k-node subgraph problem introduced by Garg and Hochbaum [2], requires finding a maximum (or minimum) node weight subgraph, on k nodes, which is connected. The problem is NP-hard for both maximum and minimum objectives even if all node weights are 0 or 1, but is polynomial if all node weights are equal. The procedure given in Section 4 is applicable to the problem and demonstrates that the problem is polynomial on trees with either positive or negative weights. The problem has been studied recently in ([6]), but no approximation with factor better than $O(k)$ is known.

Overview. In Section 2 we prove the NP-hardness of the k-edge-incident problem. Section 3 contains the description and proofs for the approximations algorithms for the unweighted and weighted graphs. Finally, Section 4 describes the polynomial time algorithm that is applicable for acyclic graphs and those that contain in addition simple cycles.

2. Complexity of k-edge-incident subgraph problems

In this section the k-edge-incident subgraph problem is shown to be strongly NP-hard.

The k-edge-incident subgraph problem is proved NP-complete by a reduction from the maximum clique problem. We call the decision problem corresponding to the maximum clique problem – *MAXCLIQUE*.

The MAXCLIQUE problem is defined as follows:

Instance: A graph $G = (V,E)$ and an integer $C \leqslant |V|$.

Question: Does G contains a clique of size greater than or equal to C?

MAXCLIQUE remains NP-complete even if the input graph is restricted to be a r-regular graph. This is because the independent set problem is NP-hard even in cubic graph [3]. The complement of a cubic graph is a $(|V|-4)$-regular graph. An independent set in a cubic graph corresponds to a clique in its complement.

Theorem 1. *The decision problem of the k-edge-incident subgraph is* NP-*complete.*

Proof. The problem is clearly in NP: given a graph G and a set of its nodes W, $|W| \geqslant L$, one can verify in linear time whether the number of edges incident to W is smaller or equal to k. We now transform any instance of MAXCLIQUE in r-regular graph into an instance of k-edge-incident subgraph problem such that MAXCLIQUE has an answer "yes" if and only if the corresponding k-edge-incident subgraph problem has an answer "yes."

Let $\{G = (V,E), C\}$ be an instance of MAXCLIQUE in a r-regular graph. We construct the corresponding instance of k-edge-incident subgraph as follows. The input graph of k-edge-incident subgraph is the same as the input graph of MAXCLIQUE, $G = (V,E)$. $k = Cr - C(C-1)/2$ and the question is whether there exists a set of nodes W, $|W| \geq C$, with a number of edges incident to W smaller than or equal to $k = Cr - C(C-1)/2$.

We show that G has a clique of size C if and only if G has a k-edge-incident subgraph of size C with capacity $\leq k$.

For G with a clique of size C let W be the set of nodes of such clique. The number of edges incident to W is $Cr - C(C-1)/2$, hence W is a k-edge incident subgraph of size C.

Conversely, if G contains a k-edge-incident subgraph of size C, then the number of edges incident to W is equal to rC minus the number of edges which connect nodes of W. Because the capacity is $k = rC - C(C-1)/2$, the number of edges in the graph induced by W is $C(C-1)/2$. Hence, W induces a complete subgraph (or clique) in G. □

3. Approximation algorithms

In this section we present approximation algorithms for the k-edge-incident subgraph problem that are based on a greedy algorithm, or on a relaxation of the problem to a Knapsack problem.

We first present a "*greedy*" algorithm that is a $\frac{1}{2}$-approximation for the problem on unweighted graphs. The method is called "greedy" because it always selects a "best" candidate among the nodes that have not yet been chosen without backtracking. The algorithm for the unweighted case selects a node of minimum degree in the remaining graph, obtained by deleting the nodes already selected. Formally, denote by $E(W)$ the set of edges incident to the set of nodes $W \subset V$ and by $G[W]$ the subgraph induced on W.

Algorithm (H_1):
  Set $W = \emptyset$; $V' = V$; $i = 0$.
  Sort the nodes in nondecreasing order of degrees.
  repeat
    Select a node of minimum degree $v \in V'$, in $G' = G[V']$;
      If $|E(W \cup \{v\})| > k$, then **STOP**;
      Else, $W \leftarrow W \cup \{v\}$; $w_i^H = v$.
    Set $V' \leftarrow V' \setminus \{v\}$; $i \leftarrow i + 1$.
  end

Because one can bucket-sort the vertices of a graph by increasing degree in $O(|V| + |E|)$, algorithm (H_1) takes $O(|V| + |E|)$ time. The following theorem bounds the ratio of the solution obtained by (H_1) to the value of the optimal solution $|W^*| = OPT$.

Theorem 2. *Algorithm* (H_1) *delivers a solution* W^H *which is at least* $\lfloor OPT/2 \rfloor$.

Proof. Let $t = \lfloor OPT/2 \rfloor$. We show that Algorithm (H_1) picks at least t nodes without having the sum of degrees exceeding k. Let $\{w_1^*, w_2^*, \ldots, w_{OPT}^*\}$ be nodes of an optimal solution ordered such that for d_v the degree of node $v \in V$, $d_{w_1^*} \leqslant d_{w_2^*} \leqslant \cdots \leqslant d_{w_t^*}$. Let $W^H = \{w_1^H, w_2^H, \ldots, w_s^H\}$ be a solution obtained by algorithm (H_1). Let $\bar{W} = \{w_1^*, w_2^*, \ldots, w_t^*\} \cap \{w_1^H, w_2^H, \ldots, w_s^H\}$.

Let $d'_{w_i^H}$ be the degree of w_i^H in the remaining graph G' during the iteration when w_i^H is selected. Clearly, the number of edges incident to W^* is at least $(\sum_{w_i^* \in W^*} d_{w_i^*})/2$. Because the greedy algorithm always selects a node of minimum degree in the remaining graph, the number of edges incident to the first $\lfloor OPT/2 \rfloor$ selected nodes by the greedy is at most

$$\sum_{i=1}^{t} d'_{w_i^H} \leqslant \sum_{i \in \bar{W}} d'_{w_i^H} + \sum_{\substack{i=1 \\ i \notin \bar{W}}}^{t} d_{w_i^H} \leqslant \sum_{i=1}^{t} d_{w_i^*} \leqslant \frac{\sum_{i=1}^{OPT} d_{w_i^*}}{2} \leqslant k.$$

Hence the required result. □

This greedy algorithm could be viewed as a special case of the following "Knapsack" algorithm devised for the weighted case. This will be further detailed below. Let OPT be the value of an optimal solution to the weighted k-edge-incident subgraph problem and let $\omega(v)$ be the weight of node v. Obviously, each node in a feasible solution satisfies $d_v \leqslant k$, hence we can remove from further consideration all nodes of degree exceeding k. For the weighted case, the following knapsack problem is a relaxation of the k-edge-incident subgraph problem for $2k = M$.

$$\begin{aligned} z(M) = \max & \sum_{v \in V} \omega(v) x_v \\ s.t. & \sum_{v \in V} d_v x_v \leqslant M \quad (\text{Knap}(M)) \\ & x_v \in \{0, 1\}. \end{aligned}$$

The sum of nodes' degrees in the optimal solution is less than or equal to $2k$ and therefore OPT is a feasible solution to Knap($2k$). It follows that the value of the optimal solution to Knap($2k$), $z(2k)$, is an upper-bound on the optimal solution, $OPT \leqslant z(2k)$.

Consider the approximation algorithm is to solve optimally the problem Knap(k). The optimal solution to Knap(k), $V(k) = \{v \in V \mid x_v = 1\}$, is feasible for the k-edge-incident problem. In Theorem 3, we show that $z(k)$ is at least $OPT/3$. For the unweighted case, the greedy algorithm (H_1) solves optimally Knap(k). Hence theorem 3 is also a proof that the greedy is a $\frac{1}{3}$-approximation algorithm for the unweighted case, which is a weaker statement than the one proved in Theorem 2. As we shall see though, the proof of Theorem 3 implies in fact Theorem 2.

Knap(k) is solved by dynamic programming in $O(k|V|)$ time. Since $k \leqslant |E|$, this time is polynomial in the input size which is $O(\sum_{v \in V} \log \omega(v) + |E|)$.

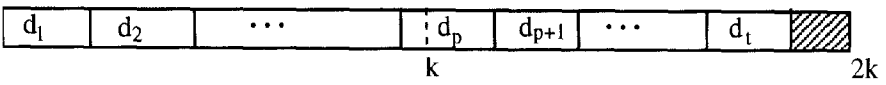

Fig. 1. Illustration of the proof of Theorem 3.

Theorem 3. *The knapsack algorithm is a $\frac{1}{3}$-approximation algorithm for the k-edge-incident problem.*

Proof. As stated above, $OPT \leqslant z(2k)$.

To complete the proof, it suffices to show that $z(2k) \leqslant 3z(k)$. Let $W^* = \{w_1^*, w_2^*, \ldots, w_t^*\}$ be an optimal solution to $z(2k)$ where the vertices of W are arbitrarily ordered. Vertex $w_j^* \in W^*$ occupies d_j slots in the knapsack, from slot $\sum_{i=1}^{j-1} d_i + 1$ to slot $\sum_{i=1}^{j-1} d_i + d_j$. Let w_p^* be the vertex which occupies slot $k+1$ in the knapsack (see Fig. 1). Note that w_p^* may also occupy slots to the left and/or to the right of slot $k+1$ but cannot occupy more than k slots because we may assume that $d_i \leqslant k$, $\forall i$. $\{w_1^*, \ldots, w_{p-1}^*\}$ and $\{w_{p+1}^*, \ldots, w_t^*\}$ are both feasible solutions for Knap(k) because $\sum_{i=1}^{p-1} d_i \leqslant k$ and $\sum_{i=p+1}^{t} d_i \leqslant k$. Therefore, $\sum_{i=1}^{p-1} \omega(w_i) \leqslant z(k)$ and $\sum_{i=p+1}^{t} \omega(w_i) \leqslant z(k)$. For the same reason $\omega(w_p) \leqslant z(k)$. Hence $z(2k) \leqslant 3z(k)$, which completes the proof of the theorem.

For unweighted graphs, the vertex w_p^* contributes only a single unit of weight to the objective function, so the total solution value in only at most twice the optimum plus this single unit. This implies the result of Theorem 2. □

4. Polynomially solvable instances

In this section we present polynomial time algorithms for the k-edge-incident problem on acyclic graphs. We first present the easy unweighted case for a tree. Because of the "packing" nature of the k-edge problems, solving on a tree does not immediately imply a solution on a forest, as the number of edges packed in each component needs to be considered. We then extend the algorithm for forests with isolated cycles.

We then present a dynamic programming algorithm for weighted trees (extensions to forest and forests with isolated cycles work analogously). The dynamic programming overcomes the potentially exponential hurdle of allocating the appropriate number of nodes to each child.

The weighted problem comes up as a k-nodes problem in applications related to organization of databases. For acyclic graphs and trees the node problem is equivalent to the edge problem: assign the weight of the node to the edge connecting it to its parent. This is therefore the first known polynomial algorithm for the problem of finding a subtree on k nodes of maximum (or minimum) weight. The closely related problem of finding a maximum weight closed set of k-nodes in a DAG is NP-complete (using a reduction from CLIQUE via the selection problem).

4.1. Unweighted polynomially solvable instances

On an unweighted graph tree $T=(V,E)$, the problem is solvable in linear time. Recall that the number of edges of a tree is exactly $(|V|-1)$. If $|V| \leqslant k+1$, then the optimal solution is the entire graph. Otherwise we use the following linear time algorithm:

Algorithm (H_2):
  $W = \emptyset$; $T' = T$
  Repeat k times:
    Select a leaf $v \in T'$;
    $W \leftarrow W \cup \{v\}$;
    $T' \leftarrow T' \setminus \{v\}$.

The correctness of algorithm (H_2) follows since any proper subtree $T' \subset T$ has at least $|T'|$ edges incident to it. The algorithm delivers a set W with exactly $|W|$ edges incident to it, which meets the lower bound and is hence optimal. The linear time complexity follows from the easy search for a leaf node. A new leaf joins the list of leaf nodes only when the last of its children is removed. It is therefore sufficient to check, upon deletion of a node, whether its parent has become a leaf.

For a forest as the unweighted input graph G, the following $O(|V|)$ algorithm delivers an optimal solution:

Algorithm (H_3):
  Sort the trees of the forest by increasing size;
  Let $T_1, T_2, \ldots$ be the list of sorted trees;
  $W = \emptyset$; $j = 0$;
  Do while $|E(W)| \leqslant k$:
    $j \leftarrow j+1$, $W \leftarrow W \cup T_j$
  Apply algorithm (H_2) to T_{j+1}
    with capacity equal to $k - |E(W)|$.

The algorithm (H_3) is of linear complexity as trees can be sorted in $O(|V|)$ by using a bucket-sort technique.

To see that this algorithm actually delivers an optimal solution, notice that the number of nodes in the algorithm's solution, W, is equal to k plus the number of whole trees induced by W. Any solution with k edges has at most that many nodes. So an optimal solution, W^* is one that maximizes the number of whole trees induced by W^*, which is delivered by algorithm (H_3).

The problem on unweighted graphs consisting of trees and isolated cycles is also solvable in polynomial time using an algorithm similar to (H_2). Isolated cycles are sorted by increasing size and placed in the list of sorted components after the trees.

4.2. Weighted polynomially solvable instances

For weighted acyclic graphs, we describe a dynamic programming approach. First consider the graph to be a tree rooted at an arbitrarily selected node $v_r \in V$. The

algorithm recursively computes an optimal solution for a subtree rooted at $v, \forall v \in V$ and using *exactly* p edges, for $p = 1, \ldots, k$. Let $F_v(p)$ be the value of such optimal solution. We distinguish between $F_v^{\text{in}}(p)$, the optimal value of a solution that includes node v (but the edge that connects node v to its parent is not counted) and $F_v^{\text{out}}(p)$, the optimal value of a solution that excludes node v. Then, $F_v(p) = \max\{F_v^{\text{in}}(p), F_v^{\text{out}}(p)\}$. Our optimal solution is $\max_{v \in V} F_v(k)$.

For a node $v \in V$ that is not a leaf, let $w_1, w_2, \ldots, w_{c(v)}$ be its children ordered arbitrarily. Define $G_v^{\text{in}}(q,t)$ to be the value of an optimal solution using q edges that includes v in the subtree rooted at v but of which the subtrees rooted at children $t+1, \ldots, c(v)$ have been pruned; i.e. permits only the inclusion of subtrees rooted at $w_1, \ldots, w_t$. $G_v^{\text{out}}(q,t)$ is defined in a similar way for the case that v is excluded. We are now ready to write the recurrence relations for this dynamic programming algorithm.

Algorithm (H_4):

$$F_v^{\text{in}}(p) = G_v^{\text{in}}(p, c(v)), \tag{1}$$

$$F_v^{\text{out}}(p) = G_v^{\text{out}}(p, c(v)), \tag{2}$$

$$G_v^{\text{in}}(q,t) = \max \begin{cases} \max\limits_{s=1,\ldots,q-t+1} [F_{w_t}(s) + G_v^{\text{in}}(q-s-1, t-1)], \\ G_v^{\text{in}}(q-1, t-1), \end{cases} \tag{3}$$

$$G_v^{\text{out}}(q,t) = \max \begin{cases} \max\limits_{s=1,\ldots,q} [F_{w_t}^{\text{out}}(s) + G_v^{\text{out}}(q-s, t-1)], \\ \max\limits_{s=1,\ldots,q-1} [F_{w_t}^{\text{in}}(s) + G_v^{\text{out}}(q-s-1, t-1)], \\ G_v^{\text{out}}(q, t-1). \end{cases} \tag{4}$$

The correctness of Eqs. (1) and (2) follows from the definitions of $G_v^{\text{in}}(q,t)$ and $G_v^{\text{out}}(q,t)$.

In recurrence relation (3), the number of edges allocated to the subtree rooted at the tth child of node v, w_t, cannot be more than $q - t + 1$ because $t - 1$ edges must be "reserved" for the connections of node v to its first $t - 1$ children. Actually, because we assume that node v is in the solution, every edge incident to it must be counted whether or not its children are included in the solution. The second line in recurrence relation (3) is for the case when the subtree rooted at w_t is null.

In the first line of recurrence relation (4), we assume that s edges are allocated to the subtree rooted at w_t but that node w_t itself is not part of the solution. Because we assume that node v does not belong to the solution, one can allocate all q edges to the subtree rooted at w_t because none of the edges that connect node v to its children has to be in the solution, only the edges that connect v to its children that are part of the solution. The second line of (4) assume that node w_t is taken in the solution. Therefore, s cannot be greater than $q - 1$ because one edge has to be reserved for the connection of w_t to its parent v. The third line of (4) accounts for the case the subtree rooted at w_t is null.

Boundary conditions are given below.

If v is a leaf, then

$$F_v^{\text{in}}(p) = \omega(v) \quad p = 0,\ldots,k;$$

$$F_v^{\text{out}}(p) = 0 \quad p = 0,\ldots,k.$$

Also, $\forall v, t$,

$$F_v^{\text{out}}(0) = G_v^{\text{out}}(0,t) = 0;$$

$$F_v^{\text{out}}(p) = -\infty \quad p < 0;$$

$$G_v^{\text{out}}(q,t) = -\infty \quad q < 0;$$

$$F_v^{\text{in}}(p) = -\infty \quad p \leqslant 0;$$

$$G_v^{\text{in}}(q,t) = -\infty \quad q \leqslant 0.$$

Notice that the dynamic programming imposes an order on the children of a node and processes them in the prescribed order while maintaining the accumulated sum of nodes allocated to the children processes.

The computation of the value of $G_v^{\text{in}}(q,t)$ or $G_v^{\text{out}}(q,t)$ takes $O(k)$ steps. Because these functions need to be computed for all vertices $v \in V$ and all possible values of $q = 1,\ldots,k$, it follows that the complexity of the above described dynamic programming algorithm has complexity $O(|V|k^2)$.

It is worth noticing that the dynamic programming described above would also deliver an optimal solution if the edges of the tree were assigned non-negative weights. Moreover the running time of the algorithm would have identical complexity, namely $O(|V|k^2)$.

The problem on a weighted forest is solved in $O(|V|k^2)$ time using the following adjustment. Tree T_i of the forest is rooted at node v_i, arbitrarily chosen among the nodes of T_i. Node v_0 of weight zero is added and connected to all roots v_i of the trees in the forest with edges of weight zero. The dynamic programming algorithm is then executed on the connected tree rooted at v_0.

References

[1] B. Awerbuch, Y. Azar, A. Blum and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen, in: Proc. 27th Ann. ACM Symp. Theory Comput. (STOC 95) (1995).

[2] N. Garg and D.S. Hochbaum, An $O(\log k)$ approximation algorithm for the k-minimum spanning tree problem in the plane, Proc. 26th Ann. ACM Symp. on the Theory of Computing (1994) 432–438.

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

[4] A. V. Goldberg, Finding a maximum density subgraph, Tech. Report No. UCB CSD 84/171, Computer Science Division (EECS), UC Berkeley, CA (1984).

[5] O. Goldschmidt, D.S. Hochbaum and G. Yu, A Simulated Annealing Heuristic for the Semiconductor Components Assembly, Tech. report, I.E.& O.R. Department, UC Berkeley, CA (1994).

[6] D.S. Hochbaum and A. Pathria, Node-optimal connected k-subgraphs manuscript, UC Berkeley, May (1994).
[7] D.S. Hochbaum and D.B. Shmoys, A best possible parallel approximation algorithm to a graph theoretic problem, Oper. Res. 933–938, (1987).
[8] G. Kortsarz and D. Peleg, On choosing a dense subgraph, in: Proc. 34th Ann. Symp. on Foundations of Computer Science (1993).
[9] J. Mitchell, Guillotine subdivisions approximate polynomial subdivisions: a simple new method for the geometric k-MST problem, Manuscript, 1995.
[10] S.S. Ravi, D.J. Rosenkrantz and G.K. Tayi, Facility dispersion problems: heuristics and special cases, Proc. 2nd Workshop on Algorithms and Data Structures, Ottawa, Canada, Lecture Notes in Computer Science, Vol. 519, 355–366, (Springer, Berlin, 1991).
[11] R. Ravi, R. Sundaram, M.V. Marathe, D.J. Rosenkrantz and S.S. Ravi. Spanning trees short and small, in: Proc. 5th Ann. ACM-SIAM Symp. on Discrete Algorithms (1994).
[12] M. Yannakakis, Node- and edge-deletion NP-complete problems, Proc. 10th Ann. ACM Symp. on the Theory of Computing (1978) 253–264.