Discrete Applied Mathematics 159 (2011) 746–759

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

On a labeling problem in graphs

R. Chandrasekaran [a], M. Dawande [b,*], M. Baysan [c]

[a] Department of Computer Science, University of Texas at Dallas, United States
[b] School of Management, University of Texas at Dallas, United States
[c] Department of Computer Science, University of Toronto, Canada

ARTICLE INFO

Article history:
Received 3 November 2009
Received in revised form 13 October 2010
Accepted 29 December 2010
Available online 31 January 2011

Keywords:
Graph algorithms
Software programming
Complexity

ABSTRACT

Motivated by applications in software programming, we consider the problem of covering a graph by a feasible labeling. Given an undirected graph $G = (V, E)$, two positive integers k and t, and an alphabet Σ, a *feasible labeling* is defined as an assignment of a set $L_v \subseteq \Sigma$ to each vertex $v \in V$, such that (i) $|L_v| \leq k$ for all $v \in V$ and (ii) each label $\alpha \in \Sigma$ is used no more than t times. An edge $e = \{i, j\}$ is said to be *covered* by a feasible labeling if $L_i \cap L_j \neq \emptyset$. G is said to be covered if there exists a feasible labeling that covers each edge $e \in E$.

In general, we show that the problem of deciding whether or not a tree can be covered is strongly NP-complete. For $k = 2$, $t = 3$, we characterize the trees that can be covered and provide a linear time algorithm for solving the decision problem. For fixed t, we present a strongly polynomial algorithm that solves the decision problem; if a tree can be covered, then a corresponding feasible labeling can be obtained in time polynomial in k and the size of the tree. For general graphs, we give a strongly polynomial algorithm to resolve the covering problem for $k = 2$, $t = 3$.

1. Introduction

Given an undirected graph $G = (V, E)$, two positive integers k and t, and an alphabet Σ, a *feasible labeling* is defined as an assignment of a set $L_v \subseteq \Sigma$ to each vertex $v \in V$, such that (i) $|L_v| \leq k$ for all $v \in V$ and (ii) each label $\alpha \in \Sigma$ is used no more than t times. In a feasible labeling, the *commonality index* $c(e)$ of an edge $e = \{i, j\}$ equals $|L_i \cap L_j|$. Note that $c(e) \in \{0, 1, 2, \ldots, k\}$. An edge $e = \{i, j\}$ is said to be *covered* by a feasible labeling if $c(e) \geq 1$ (i.e., if $L_i \cap L_j \neq \emptyset$). G is said to be covered if there exists a feasible labeling that covers each edge $e \in E$. In words, we need an assignment of at most k labels to each node of G such that each label is assigned to at most t nodes and there is at least one common label among the labels assigned at the two endpoints of each edge. For $k = 2$, $t = 3$, Fig. 1 shows a possible labeling that covers the graph; edge commonality indices are also shown in the diagram. There are several interesting questions concerning feasible labelings and the extent to which a graph can be covered:

1. Find a labeling that maximizes $\sum_{e \in E} c(e)$. For the special case of $k = 2$, this problem has been studied in [4]. They consider two cases. In the first case, the labels are such that for any edge $e = \{i, j\}$, either $L_i = L_j$ or $L_i \cap L_j = \emptyset$. Such a labeling is known as a *non-split labeling* [4,5]. In the second case, this condition need not hold and the corresponding labeling is referred to as a *split labeling*. It is shown that if G is a tree, then as far as this objective is concerned, no advantage is gained by splitting. A consequence of this result is a polynomial-time algorithm to solve the problem using the work in [10,11]. Further results in [4] include (i) polynomial algorithms for obtaining optimum split and non-split labelings in a general graph for $k = 2$, $t = 2$ and (ii) proofs of NP-hardness of these problems for $k = 2$, $t \geq 3$.

* Corresponding author. Fax: +1 9728835095.
E-mail addresses: chandra@utdallas.edu (R. Chandrasekaran), milind@utdallas.edu (M. Dawande), m.baysan@utoronto.ca (M. Baysan).

0166-218X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2010.12.022

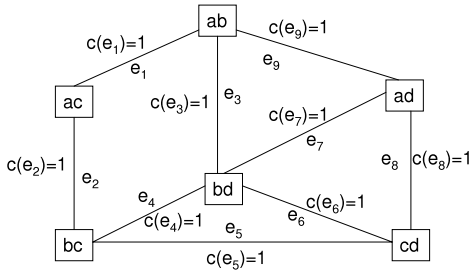

Fig. 1. $k = 2$, $t = 3$. A feasible labeling that covers the graph.

2. Can a graph be covered? If so, give a labeling of the vertices to achieve this. To our knowledge, there is no existing work on this problem in the literature.
3. If the answer in (2) is no, then what is the maximum number of edges of a given graph that can be covered and how do we label the vertices to achieve this?
4. In (3) above, if edges have weights, what is the maximum total weight of covered edges and how can this be achieved?

In answering these questions, there may be further assumptions about G, whether $|\Sigma| < \infty$ or not, etc. In this paper, we focus on Question (2) above. Note that, in general, maximizing $\sum_{e \in E} c(e)$ does not resolve the question of whether or not the graph can be covered.

1.1. Application in pair programming

An interesting application of labeling occurs in the context of assigning pairs of programmers to software development tasks. Pair Programming [1,3,12] is an agile software development approach in which two programmers sit side-by-side at a workstation and collaborate on the same piece of code. One member of the pair types the code while the other continually reviews the work. By working with different partners and on different parts of the software system, programmers develop mutual trust and learn more (as compared to traditional programming) about the software system as a whole. The benefits of pair programming include higher software quality, improved productivity, and team cohesion [3,8].

1.1.1. The underlying graph

A software module refers to some functionality that must be developed by a pair of programmers. A software system consists of several modules. The development activities of certain pairs of modules may be related, e.g., two modules may share code, one module may use the other as a subroutine, or one module may inherit methods or attributes from the other. Subsequent to their development, such pairs of modules are often integrated and tested together. Thus, a software system can be conceptualized as a graph $G(V, E)$: the vertices denote the modules and a pair of vertices is connected by an edge if the development activities of the corresponding modules are related.

1.1.2. Assigning pairs of developers to nodes

When two modules are connected by an edge, using (one or more) common developers can reduce the integration and testing effort for these modules [5]. If having common developers for the modules that are connected by an edge is the only requirement for an assignment, then a solution is trivial: we can assign the same pair of developers to each node. However, it should be noted that most projects need to be completed by a deadline. Assigning the same pair of developers to each node is typically infeasible since the time required to complete the project typically violates the deadline. On the other hand, mandating that each developer works on a pre-specified number of modules sacrifices much-needed flexibility in a real environment. A reasonable trade-off is to allow a developer to work on *at most* a pre-specified number t of modules [4].

Thus, we have two defining characteristics of a feasible assignment: (i) each node must be assigned two distinct developers and (ii) each developer can be assigned to at most t nodes. While there are several objectives of interest for such an assignment (e.g., Questions 1–4 above), the one considered in this paper is that of obtaining an assignment with the following property: for each edge (i, j) of G, there should be at least one common developer amongst those assigned at nodes i and j. It is easy to motivate generalizations of our covering problem to project management. In this context, each node in V corresponds to a project and the set of edges E between certain pairs of nodes denote an overlap in the development activities of the corresponding projects. The number of personnel to be assigned to project $v_i \in V$ is k_i. As before, each person can be assigned to at most t projects.

As mentioned earlier, there has been no prior work on the covering problem studied in this paper. However, some fundamentally different covering problems have been addressed in the literature. For example, [7] addresses a problem of covering the nodes of a tree by subtrees such that the number of nodes in each subtree is restricted and any pair of subtrees have at most one node in common. The objective is to minimize the number of subtrees used in the node cover. The problem where the subtrees used for the cover need to be chosen from an explicitly provided collection is considered in [2].

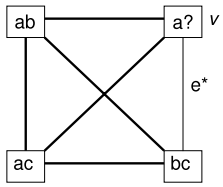

Fig. 2. $k = 2$, $t = 3$. For some edge e, $c(e) = 2$ in any feasible covering.

2. Labeling trees

We assume an infinite alphabet Σ. In Question (2) above, since we are only interested in making sure that $c(e) \geq 1$ for all edges, we do not need more than one common label at the two ends of an edge. However, in some cases, we may be forced to have more common labels. A simple example is when $G = K_4$, $k = 2$, $t = 3$: For the labeling in Fig. 2, each bold edge is covered and has a commonality index of 1. However, edge e^* remains to be covered; regardless of the label assigned to node v so as to cover e^*, one edge will have a commonality index of 2. It can easily be verified that every feasible labeling for K_4 with $k = 2$ and $t = 3$ will have this property. This, however, is never the case with trees as the result below shows.

Lemma 1. *Let G be a tree and let $|\Sigma| = \infty$. If G can be covered by a feasible labeling, then it can be covered by a feasible labeling such that for each edge e, $c(e) = 1$.*

Proof. Let $\mathcal{L}$ be a labeling that covers the graph and for which $\sum_{e\in E} c(e)$ is the minimum. Suppose that an edge $e = \{i, j\}$ has $c(e) = |L_i \cap L_j| \geq 2$ in $\mathcal{L}$. Since G is a tree, removing edge e results in two subtrees, say T_1 and T_2. Let $\mathcal{L}'$ be the labeling obtained as follows: let $\alpha \in L_i \cap L_j$ and let $\beta \in L_i \cap L_j - \{\alpha\}$. In tree T_1, replace all occurrences of the label β by a new label $\gamma \in \Sigma - \cup_{i\in V} L_i$. This is possible since $|\Sigma| = \infty$. $\mathcal{L}'$ also covers G and $\sum_{e\in E} c'(e) < \sum_{e\in E} c(e)$, contradicting the choice of $\mathcal{L}$. □

From now on, we will try to find such a labeling when it exists. For simplicity, we will refer to Question (2) above as the covering problem. Let $\deg_G(i)$ denote the degree of vertex i in G. First, we note some simple observations.

- $k = 1$, $t \geq 1$: G can be covered iff each of its connected components has no more than t vertices.
- $k \geq 1$, $t = 1$: G can be covered iff it has no edges.
- $k \geq 1$, $t = 2$: G can be covered iff $\deg_G(i) \leq k$ for all $i \in V$. If $\deg_G(i) \leq k$ for every node $i \in V$, then we cover each edge by assigning a common label to the end points of that edge. Thus, node $i \in V$ uses $\deg_G(i)$ distinct labels. Conversely, suppose there exists a covering for G. Since $t = 2$, each label can be used to cover at most one edge. Therefore, as at most k labels are allowed at each node, $\deg_G(i) \leq k$ for every node $i \in V$.

It is clear that G can be covered only if $\deg_G(i) \leq k(t - 1)$ for all $i \in V$. However, this condition is not sufficient as seen by the following examples for $k = 2$; $t = 3$.

The following result allows us to simplify the structure of the vertices with low degrees in graphs.

Lemma 2. *Let G be a graph. If $1 < \deg_G(i) \leq k$, and node i is an articulation point, covering G is equivalent to covering the "split" graph obtained as follows: we split node i into $|\deg_G(i)|$ copies such that each copy has incident on it one distinct neighbor of node i in G. Fig. 4 shows an example of the splitting for $k = 2$.*

Proof. Since node i of G is an articulation point, the splitting operation generates disconnected subgraphs of G. It is easy to see that if one of the subgraphs cannot be covered, then the original graph cannot be covered either. On the other hand, if all the subgraphs are covered (by possibly using entirely different alphabets in each copy of node i), merge the copies back to get node i and assign all the labels used by the copies to the node i. Then, node i will get no more than k labels since its original degree was no more than k. □

For a tree, since each non-leaf node is an articulation point [6], Lemma 2 can be used to split each node with degree at least 2 and at most k. In this case, the splitting operation results in a set of subtrees. By repeatedly applying Lemma 2, we can reduce the question of covering a tree G to one for a tree G' in which for each node i we have $\deg_{G'}(i) = 1$ or $\deg_{G'}(i) \geq k+1$.

2.1. Labeling trees for $k = 2$, $t = 3$

In this section, we settle the covering question for the case when G is a tree, $k = 2$, and $t = 3$. The main result follows.

Theorem 1. *For $k = 2$, $t = 3$, the problem of whether or not a tree can be covered can be solved in linear time.*

Proof. Clearly, if $\deg_G(i) \geq 5$ for any node $i \in V$, then G cannot be covered. Also, if $\deg_G(i) = 2$ for some $i \in V$, then the splitting technique of Lemma 2 can be applied. Thus, we assume that $\deg_G(i) \in \{1, 3, 4\}$ for all $i \in V$.

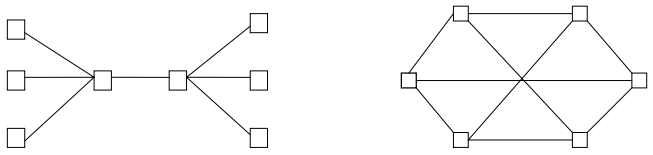

Fig. 3. Graphs with $\deg_G(i) \leq k(t-1) \forall i$ that cannot be covered for $k = 2$, $t = 3$.

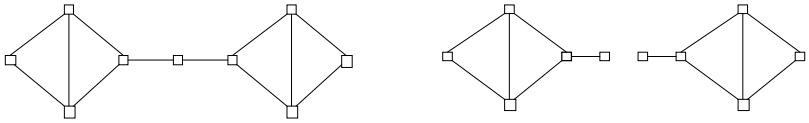

Fig. 4. Node splitting in a general graph ($k = 2$).

We first consider the case when G be a tree with $\deg_G(i) \in \{1, 3\}$ for all $i \in V$. Such a tree can *always* be covered. A feasible covering can be obtained by the following linear-time algorithm:

Algorithm I.

1. Root the tree at any arbitrary *leaf* node r. //*Now it is a binary tree.*//
2. $L \leftarrow \Sigma$; $S \leftarrow \emptyset$; $T \leftarrow \{r\}$
3. **while** $T \neq \emptyset$ select a node $x \in T$, label x and its children with any label $\alpha \in L$.
4. $S \leftarrow S \cup \{x\}$; $T \leftarrow T \cup \{j : j \text{ is a child of } x\} - \{x\}$; $L \leftarrow L - \{\alpha\}$. //*$S$ is the set of completely labeled nodes*; *T is the set of partially labeled nodes from which we select the next node to scan*; *L is the set of unused labels.*//

Since any node has at most two children, and $t = 3$, this labeling works without any problem. Next, let G be a tree with $\deg_G(i) \in \{1, 3, 4\}$ for all $i \in V$. If we now apply a similar algorithm, it may not work since the rooted tree is no longer binary. For example, the tree in Fig. 3 cannot be labeled even though $\deg_G(i) \in \{1, 4\}$ for all $i \in V$. If we root the tree at a node with degree equal to 4 (if one such exists), then if we encounter no other node with three children, we can cover the tree: We arbitrarily give the labels to children from among the two labels given to the root. Thereafter, there is no confusion since each internal node has two children. If, starting with a root that has four children, we encounter an internal node with three children, we stop and claim that the tree cannot be covered. Since there is no choice that matters to the outcome, the result follows. We, therefore, have the following algorithm. Let $\Delta = \max_i \deg_G(i)$.

Algorithm II.

1. Root the tree at any arbitrary node r with $\deg_G(r) = \Delta$.
2. $L \leftarrow \Sigma$; $S \leftarrow \emptyset$; $T \leftarrow \{r\}$.
3. **if** $\deg_G(r) \geq 5$, stop. //*G cannot be covered* //
4. **else if** $\deg_G(r) < 4$, then use **Algorithm I**.
5. **else** label r and its children with any two labels $\alpha, \beta \in L$; $\alpha \neq \beta$, so that no label is used more than 3 times.
6. $T \leftarrow T \cup \{j : j \text{ child of } r\} - \{r\}$; $S \leftarrow S \cup \{r\}$; $L \leftarrow L - \{\alpha, \beta\}$.
7. **while** $T \neq \emptyset$ select a node $x \in T$.
8. **if** x has three children, **STOP**. // *G cannot be covered*//
9. **else** label x and its children with any label $\alpha \in L$.
   $S \leftarrow S \cup \{x\}$; $T \leftarrow T \cup \{j : j \text{ child of } x\} - \{x\}$; $L \leftarrow L - \{\alpha\}$.
   //*S is the set of completely labeled nodes*; *T is the set of partially labeled nodes from which we select the next node to scan*; *L is the set of unused labels*//
10. **end** □

If the condition in Step 8 is valid, then the tree has a structure of the type shown in Fig. 5, and Algorithm II concludes that there is no feasible labeling. Otherwise, the algorithm produces a feasible labeling. Here, the middle path can be of any length as long as each node on the path, not including the two ends, has a degree of 3. This type of graph allows only one labeling (modulo permutations of the labels) till the last edge, and this edge cannot be covered.

Theorem 2. *For $k = 2$, $t = 3$, a tree has a feasible labeling if and only if* (i) $\Delta \leq 4$ *and* (ii) *it does not have a subtree in which two nodes of degree* 4 *are connected by a path in which all intermediate nodes have degree* 3 *or higher (see Fig. 5 for an example).*

This completes our discussion of the case when G is a tree, $k = 2$, $t = 3$. When we turn to larger values of t, some additional features turn up that we have not seen so far. For example, let $k = 2$, $t = 4$, and consider the example in Fig. 6. Theorem 2 implies that this graph cannot be covered for $k = 2$, $t = 3$. However, it is easy to see that we can cover it for $k = 2$, $t = 4$. Thus, it is clear that in order to process higher values of k and t we must have an approach that is easily generalizable.

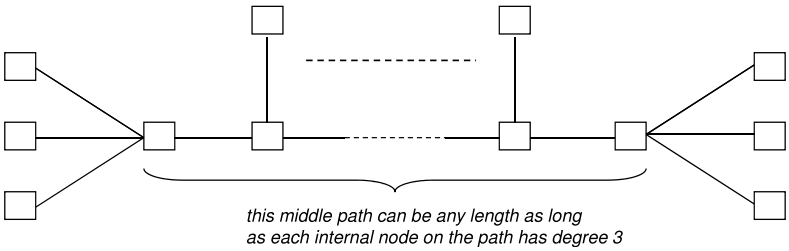

Fig. 5. Forbidden subtrees for a tree for $k = 2$, $t = 3$.

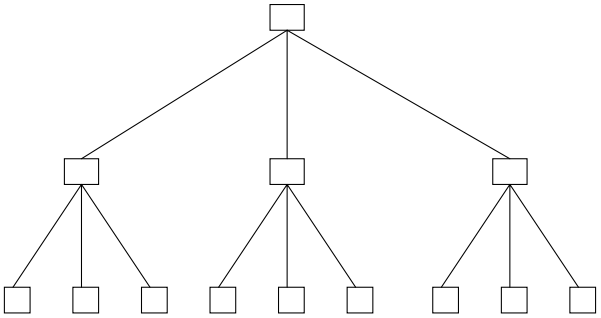

Fig. 6. Tree that can be covered for $k = 2$; $t = 4$, but not for $k = 2$; $t = 3$.

2.2. Labeling trees for higher values of k and t

We show that the problem is strongly NP-complete for trees for general values of k and t. For this purpose, we use the classical bin-packing problem.

BIN-PACKING PROBLEM:

INSTANCE: A set of positive integers $x_1, x_2, \ldots, x_n$; a positive integer bin capacity B; a positive integer k (that may depend on n).

DECISION QUESTION: Is it possible to partition the set $\{x_1, x_2, \ldots, x_n\}$ into at most k subsets $S_i, i = 1, \ldots, k$, such that $\sum_{j \in S_i} x_j \leq B$ for $i = 1, 2, \ldots, k$?

This decision problem is known to be strongly NP-complete (see, e.g., [9]). To see that an arbitrary instance of BIN-PACKING can be formulated as an instance of our tree labeling problem, consider the the following tree:

Let $t = B + 1$. We have a root node r with n children. Let the ith child of r have $n_i = (k-1)(t-1) + x_i - 1$ children and let these be leaf nodes of the tree. If BIN-PACKING has an affirmative answer, then consider a distinct label a_j corresponding to the subset $S_j, j = 1, 2, \ldots, k$. A label, say "a_1", at the root node r is used as follows: "a_1" is assigned to those children that correspond to the set S_1 and also to $x_j - 1$ children of each node $j \in S_1$. The total usage of the label "a_1" is, therefore, $1 + |S_1| + \sum_{j \in S_1} (x_j - 1) = 1 + \sum_{j \in S_1} x_j \leq B + 1 = t$. Each distinct label a_j corresponding to the set S_j is assigned similarly. The remaining labeling for the tree is now easy. Each child of the root node has $k - 1$ labels remaining to be assigned and $(k-1)(t-1)$ children remaining to be covered. This can be achieved trivially: every child of the root node uses $(k-1)$ distinct labels and uses each to cover $(t-1)$ of its children. Conversely, consider any labeling that covers such a tree. Consider the label, say "a", used to cover the edge (r, i). The $k-1$ other labels at node i can cover a maximum of $(k-1)(t-1)$ edges of the type $\{i, j\}$, where j is a child of i. Thus, label "a" *must* be used also to cover $y_i = x_i - 1$ edges of the type $\{i, j\}$ where j is a child of i. Furthermore, label "a" *can* be used to cover edges $\{r, q\}$, $q \in S \subseteq \{1, 2, \ldots, n\}$, if and only if

$$\sum_{q \in S} y_q + |S| + 1 \leq t = B + 1.$$

Using $y_q = x_q - 1$, this condition is equivalent to

$$\sum_{q \in S} x_q \leq B.$$

Hence, for the entire tree to be covered, we need a partition of the children of r into at most k subsets $S_1, S_2, \ldots, S_k$, such that the above condition holds for each subset. Thus, an affirmative answer to the covering problem provides an affirmative answer to the bin-packing problem. We, therefore, have the following result.

Theorem 3. *The labeling problem is, in general, strongly NP-complete on trees.*

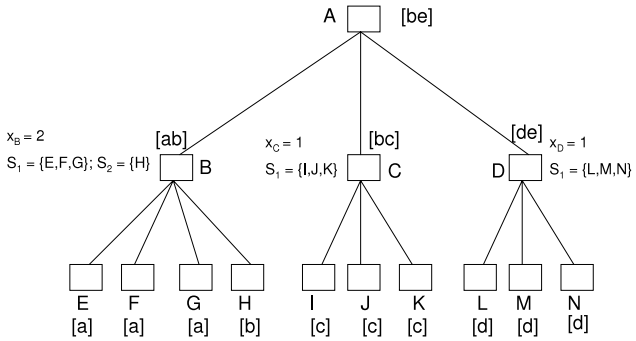

Fig. 7. Illustration of Step 6 of Algorithm III.

Note: The reduction above from an arbitrary instance of the bin-packing problem to the specific instance of the covering problem is pseudo-polynomial (in the input size of the bin-packing instance). Nevertheless, the conclusion in Theorem 3 is valid since the decision question corresponding to the bin-packing problem is NP-complete in the strong sense.

2.2.1. A polynomial algorithm for fixed $t \geq 1$

We now turn our attention to the covering problem for some fixed $t \geq 1$. For now, we assume that there is an oracle that solves the following bin-packing type problem and use it as a subroutine to solve the tree labeling problem. Later, in Section 2.2.2, we will analyze the issue of computational complexity.

Modified Bin-packing Problem:

Instance: A set of positive integers $x_1, x_2, \ldots, x_n$; a positive integer k.

Question: Is there a partition of the set $\{x_1, x_2, \ldots, x_n\}$ into k disjoint sets S_i; $i = 1, 2, \ldots, k$, such that $\sum_{j\in S_i} x_j \leq t - 1$ for all i? If the answer is yes, find such a packing with min $\sum_{j\in S_k} x_j$.

Output: A yes or no answer to the decision question and, if yes, the value $1 + \min \sum_{j\in S_k} x_j$.

It is advantageous to use the smallest value of k possible as well for this problem in order to use as few labels as possible. Using this subroutine, we solve our tree labeling problem by the following algorithm.

Algorithm III.

1. Initialize with given values of k and t
2. $S \leftarrow \{p : p$ is a leaf node of the tree$\}$
3. **if** $p \in S$, **do** $x_p \leftarrow 1$.
4. $T \leftarrow \{q \notin S$: all children of q are in $S\}$
5. **while** $T \neq \emptyset$, select $j \in T$ and solve the Modified Bin-packing Problem with the set of values of x_i for children of j and given values of k and t. If the answer is NO, stop; the tree cannot be covered. If yes, go to Step 6. //*Nodes in the same bin will get one common label that they will share with the parent. The parent will share a common label with its parent and* $\sum_{i\in S_k} x_i$ *of its descendants.*//
6. $x_j \leftarrow 1 + \sum_{i\in S_k} x_i$; $S \leftarrow S \cup \{j\}$ and go to step 4.
7. end

We illustrate the algorithm first by an example and then state and prove its properties.

Example. Consider the graph in Fig. 7 for $k = 2, t = 4$. The algorithm starts by assigning a value of 1 to each of the leaf nodes of the tree. Thus, $x_i = 1$ for $i \in \{E, F, G, H, I, J, K, L, M, N\}$. Now consider the instance of the modified bin-packing problem at node B. The input to this instance is $x_E = x_F = x_G = x_H = 1$; $k = 2$; $t = 4$. The output is an answer of "yes" and $x_B = 2$. Suppose the actual assignment to bins is as follows: $S_1 = \{E, F, G\}$, and $S_2 = \{H\}$; see Fig. 7. The output for the instance at node C (resp., node D) is "yes" and $x_C = 1$ (resp., $x_D = 1$). Finally, the input to the instance of the modified bin-packing problem at the root node A is $x_B = 2, x_C = x_D = 1$; $k = 2$; $t = 4$. The corresponding output is $x_A = 2$. Suppose the actual assignment is as follows: $S_1 = \{B, C\}$ and $S_2 = \{D\}$.

Obtaining a feasible labeling is now straightforward. The assignment obtained from solving each instance of the bin-packing problem is used as follows: nodes in the same set (i.e., either S_1 or S_2) will get one common label that they share with the parent. The parent will share a common label with its parent and $\sum_{i\in S_k} x_i$ of its descendants. Let us use these two rules to complete the labeling. For the instance at node B, we have $S_1 = \{E, F, G\}$ and $S_2 = \{H\}$. So, nodes E, F, and G share a common label, say "a", with their parent (i.e., node B). Node B shares a common label, say "b", with its parent (i.e., node A) and node H. Similarly, nodes I, J, and K share a common label, say "c" with their parent C. Node C shares a common label "b" with its parent A. Finally, nodes L, M, and N share a common label, say "d", with their parent D. Node D shares a common label "e" with its parent A. The complete assignment is shown in Fig. 8.

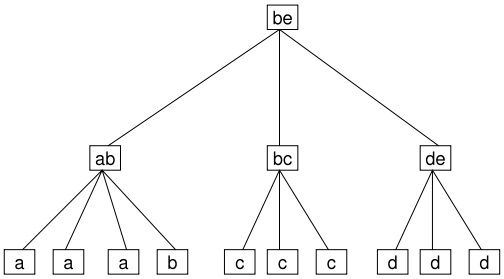

Fig. 8. A complete labeling for $k = 2$ and $t = 4$.

Table 1
"Yes" instances for $k = 2$, $t = 4$, in Step 5 of Algorithm III.

n	2	2	2	2	2	2
input x output x_j	[3, 3] 4*	[3, 2] 3	[3, 1] 2	[2, 2] 3	[2, 1] 1	[1, 1] 1
n	3		3	3		3
input x output x_j	[3,2,1] 4*		[2, 2, 1] 3	[2, 1, 1] 2		[1, 1, 1] 1
n	4		4	4		4
input x output x_j	[3, 1, 1, 1] 4*		[2, 2, 1, 1] 4*	[2, 1, 1, 1] 3		[1, 1, 1, 1] 2
n	5		5	6		
input x output x_j	[2, 1, 1, 1, 1] 4*		[1, 1, 1, 1, 1] 3	[1, 1, 1, 1, 1, 1] 4*		

Recall that n is the number of positive integers in the input instance of the MODIFIED BIN-PACKING PROBLEM. In the instances that we will encounter for $k = 2, t = 3$, we have $n \leq k(t-1) = 4$, with equality possible only at the root of the tree. Since $k = 2, t = 3$, the input values of x_i, $i = 1, 2, \ldots, n$, that we will encounter are at most 2 and $\sum_i x_i \leq 4$. The answer is "no" for the following vectors (and vectors with each entry at least as large as the corresponding entry in these vectors): [2, 1, 1, 1] and [2, 2, 1]. For the remaining vectors, the following table gives the resulting value for x_j in Step 6 of the above algorithm. Here, $*$ indicates that the process stops here since this must be the root; if not, then the graph cannot be covered.

n	2	2	2	3	3
input x	[2, 2]	[2, 1]	[1, 1]	[2, 1, 1]	[1, 1, 1]
output x_j	3*	2	1	3*	2

For $k = 2, t = 4$, we have $n \leq 6$, with equality possibly only at the root. The input values x_i, $i = 1, \ldots, n$, satisfy $x_i \leq 3$ and $\sum x_i \leq 6$. Table 1 lists the output values for those instances that have a "yes" answer:

We are now ready to prove that Algorithm III finds a feasible labeling of G if one exists, and otherwise demonstrates a subgraph of G for which no feasible labeling exists. To prove this, we formally state a simple observation about the MODIFIED BIN-PACKING PROBLEM.

Lemma 3. *For given values of k, t, and n, consider two instances of the* MODIFIED BIN-PACKING PROBLEM: *one with input vector $x = (x_1, x_2, \ldots, x_n)$, and the other with input vector $x' = (x'_1, x'_2, \ldots, x'_n)$. If* (a) $x'_i \geq x_i$, $i = 1, 2, \ldots, n$, *and* (b) *the answer to the instance with input vector x is "no", then the answer to the instance with input vector x' is "no" as well.*

Proof. If x_i, $i = 1, 2, \ldots, n$, cannot be packed into k bins each of size at most $t - 1$, then x'_i, $i = 1, 2, \ldots, n$, cannot be be packed as well since $x'_i \geq x_i$, $i = 1, 2, \ldots, n$. □

Theorem 4. *Algorithm* III *finds a feasible labeling if one exists. Otherwise, it provides a subgraph of G for which no feasible labeling exists.*

Proof. First, consider the case when the algorithm terminates with $T = \emptyset$. Thus, in this case, the answer to all the instances the MODIFIED BIN-PACKING PROBLEM encountered in Step 5 is in the affirmative. The solution to the instance of the bin-packing problem that is encountered at, say, $j \in T$, precisely specifies a labeling to the children of j: the children in the same bin will get one common label that they will share with node j. Furthermore, node j shares a common label with its parent and $\sum_{i \in S_k} x_i$ of its children in the kth bin. At this point, the output of the bin-packing problem is $x_j = 1 + \sum_{i \in S_k} x_i$; this means

that the total number of nodes, including node j, in the subtree rooted at node j at which the label corresponding to the kth bin has been used is x_j. Since $\sum_{i\in S_k} x_i \leq t-1$ (with equality possible only for the root), we have $x_j \leq t$. Assigning labels in this manner, a complete and feasible assignment can be obtained. This is guaranteed by the affirmative answer to each instance of the bin-packing problem.

Now consider the case when the bin-packing problem has a negative answer for some node $l \in T$, and let this be the first instance of a negative answer. Suppose the bin-packing instance that the algorithm encountered at node l has an input vector $(y_1, \ldots, y_n)$. We claim that no feasible labeling exists for G. Let G_l denote the subtree rooted at node l. To obtain a proof by contradiction, suppose a feasible labeling exists for G. In particular, consider a feasible labeling for the subtree G_l. This feasible labeling defines instances of the bin-packing problem at exactly those nodes of G_l for which the algorithm formulated such a problem. Furthermore, the feasible labeling also provides a solution to these bin-packing problems. Consider a node $j \in G_l$ such that all the children of j are leaf nodes. Thus, each child i of node j has $x_i = 1$. The algorithm formulated an instance of the bin-packing problem at node j. If the answer to this instance was "yes", then $x_j = 1 + \sum_{i\in S_k} x_i$. Recall that x_j is the number of times the label corresponding to the kth bin is used in the subtree rooted at node j. The assumed feasible labeling also provides a solution to this bin-packing instance and also provides a value, say x'_j, of the number of times the kth label is used in the subtree rooted at node j. However, since our modified bin-packing problem minimizes the contents of the kth bin, we have $x'_j \geq x_j$.

Arguing in this manner for each non-leaf node of the subtree G_l, we finally arrive at node l. Recall that the bin-packing instance at node l has no feasible solution for the input vector $(y_1, \ldots, y_n)$. However, the assumed feasible labeling for G implies a solution for another instance at node l with input vector, say, $(y'_1, \ldots, y'_n)$. Moreover, we have $y'_i \geq y_i$, $i = 1, \ldots, n$. Now, Lemma 3 implies that if the instance with input vector $(y_1, \ldots, y_n)$ had a negative answer, then the instance with $(y'_1, \ldots, y'_n)$ must also have a negative answer. We, therefore, arrive at a contradiction. The result follows. □

2.2.2. Complexity

We now show that for a fixed value of t, Algorithm III is a strongly polynomial algorithm for solving the decision question of whether or not a tree can be covered.

Definition. Let $u = (u_1, u_2, \ldots, u_n)$ and $w = (w_1, w_2, \ldots, w_n)$ be vectors in $\Re^n$. Then, vectors u and w are lexicographically ordered $u <_{lex} w$ if and only if there exists $k \in \{1, 2, \ldots, n-1\}$ such that $u_i = w_i$, $i = 1, 2, \ldots, k-1$, and $u_k < v_k$.

To illustrate, in $\Re^3$, we have $(1, 2, 1) <_{lex} (2, 3, 7)$ and $(2, 5, 3) <_{lex} (2, 6, 2)$.

To resolve the decision question, we formulate the modified bin-packing problem as an integer linear program. Let t be a given positive integer. In this case, the values of x_j that we encounter in our problem will of the form $1 \leq x_j \leq t-1$. The set of possible configurations of bins can be described as the set of solutions to the system:

$$\sum_{l=1}^{t-1} l z_l \leq t-1$$
$$z_l \geq 0, \quad \text{integer}.$$

Let $A = (a_{ij})$ be a $(t-1) \times m$ matrix containing all vectors z that are solutions to the above system. Each column of A corresponds to a distinct configuration; the entry a_{ij} denotes the number of items of size i in configuration j. For a reason that will become clear soon, the columns are arranged in ascending order of the total space consumed by the corresponding configuration. For example, when $t = 4$, the corresponding A is shown below:

$$A = \begin{pmatrix} 1 & 2 & 0 & 3 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The configuration corresponding to the first column consumes a total space of 1. The configurations corresponding to the second and third column both consume a total space of 2; we order these two columns in lexicographic decreasing order. The last three columns each consume a total space of 3; again, these are ordered in lexicographically decreasing order.

Now, consider the following Integer Program (IP):

$$\min e \cdot y$$
$$Ay = b$$
$$y \geq 0; \quad \text{integer}$$

where (i) b is the vector in which the component b_l is the number of input variables x_j whose value is l and (ii) e is an m-dimensional vector in which each entry is 1. Note that this IP has a size determined only by the value of t. Several observations immediately follow from the optimum solution value of IP:

1. If the optimum objective function value is greater than k, then the answer to the modified bin-packing problem is "no".
2. If the optimum value is less than or equal to k, then the answer is "yes". Furthermore, if the value is less or equal to $k-1$, then the output value in Step 6 of Algorithm III is 1.

3. If the optimum value is equal to k, then we find the output value by finding the lexico-maximal optimal solution to the IP (i.e., the solution vector that is lexicographically maximum among all optimum solutions). This minimizes the content of the kth bin. Note that 1 plus the content of the kth bin is the output value in Step 6 of Algorithm III.

For example, if $k = 2$, $t = 4$, and the input vector is [2, 1, 1, 1] our IP has the above matrix A as the constraint matrix and the vector $b = [3, 1, 0]$ and the solution of the IP is [0, 1, 0, 0, 1, 0] and the answer to the modified bin-packing problem is "yes". This solution is also a lexico-maximal optimal solution. Thus, the output value (in Step 6) is 3 in this case.

The only issue that remains to be addressed is that of obtaining a lexico-maximal optimal solution required in Observation 3 above. There are several ways in which this can be done. We outline a straightforward iterative procedure below.

Suppose the optimum solution to the IP is k. The total content in the kth bin can be one of the following $t - 1$ quantities: $1, 2, \ldots, t - 1$. We would first like to find a solution in which the content of the kth bin is 1. We, therefore, impose an additional constraint $y_1 \geq 1$. If there exists an optimum solution to the modified IP of value k, we are done. In this case, the output value in Step 6 of Algorithm III is 2. If the modified IP has a solution strictly greater than k, then all solutions with objective value k have $y_1 = 0$. So, variable y_1 can be deleted. In general, if γ_r is the index set of configurations with total content r, $r = 1, \ldots, t - 1$, then we impose an additional constraint $\sum_{j \in \gamma_r} y_j \geq 1$. If the modified IP has an optimum solution of value k, we stop and output $r + 1$ in Step 6. Otherwise, we delete all the columns corresponding to γ_r and proceed with the next largest content value.

The iterative procedure above requires us to solve a modified IP at most $t - 1$ times. To summarize, for fixed t, obtaining a lexico-maximal optimal solution for the modified bin-packing problem requires constant time (i.e., time $O(1)$).

Observe that a call to the modified bin-packing problem is made as many times as the number of non-leaf nodes in the tree. Thus, Algorithm III is a strongly polynomial algorithm for the decision question of whether or not a tree can be covered. However, it is important to note that if a feasible labeling exists, then the complexity of producing a feasible labeling is *not* strongly polynomial. This is because producing a feasible labeling at each node in Algorithm III itself requires time linear in k. Thus, if the answer to the decision question is in the affirmative, then producing a corresponding feasible labeling requires time polynomial in k and the size of the tree.

3. Labeling general graphs for $k = 2$, $t = 3$

We now turn to the problem of covering general graphs for $k = 2$, $t = 3$. As we have already discussed (Section 2), a necessary condition for a graph to be covered is that $\Delta \leq k(t - 1) = 4$. However, as seen from the examples in Fig. 3, this condition is not sufficient. Let us first consider graphs for which $\Delta \leq 3$.

3.1. Graphs with $\Delta \leq 3$

We start with the following observation.

Lemma 4. *Let $G = (V, E)$ be a graph with $\deg_G(i) \leq 3$ for all $i \in V$. Let $k = 2, t = 3$, and let K_3 be a subgraph of G on vertices $\{v, w, x\} \subseteq V$. Let H be the graph obtained from G by deleting edges $\{v, w\}, \{w, x\}, \{x, v\}$; note that $\deg_H(i) \leq 1$ for $i \in \{v, w, x\}$. Then, G can be covered iff H can be covered.*

Proof. Clearly if H cannot be covered then G cannot be covered either. The subgraph K_3 on the vertex set $\{v, w, x\}$ can be covered with one label on these vertices. In any covering of H, these vertices will not get more than one label each since $\deg_H(i) \leq 1$ for $i \in \{v, w, x\}$. The result follows. □

It follows from Lemma 4 that covering the three edges of K_3 by assigning the same label, say "a", at nodes v, w, and x, does not affect the outcome of the covering problem. Thus, without loss of generality, we will assume further that G has no subgraph isomorphic to K_3. Therefore, a single label can cover at most two edges and (if this is the case) these two edges must be incident at some vertex. Moreover, at each vertex i for which $\deg_G(i) = 3$, two of the edges incident at i must be covered by the same label if the graph G can be covered. Further, two such pairs cannot share an edge in common. Let p denote the number of vertices of G whose degree is 3. So, in order to cover G, we must have p such pairs of edges no two of which have a common edge. As discussed below, we can check this condition in polynomial time and find such a set of pairs of edges as well.

Let the graph L be obtained from G as follows: The vertices of L correspond to the edges of G incident at vertices with degree in G equal to 3. Two nodes in L are connected by an edge in L if the corresponding edges in G are both incident at the same degree 3 node. Next, solve the maximum cardinality matching problem on L. As will be clear from the discussion below, the cardinality of the maximum matching cannot be higher than p. Therefore, we check if the cardinality of the maximum matching is p. If not, G cannot be covered. Otherwise, it is straightforward to obtain a feasible labeling for G. The two edges in G corresponding to the endpoints of an edge in the matching (in L) are covered by the same label; see Figs. 9 and 10 for two examples. For the example in Fig. 9 (resp., Fig. 10), the required matching exists (resp., does not exist). After having done so, each node i in G with $\deg_G(i) = 3$ has either (a) one label that remains to be assigned and one edge that remains to be covered or (b) no labels that remain to be assigned and no edge that remains to be covered. To see this, we first observe that for each node i in G with $\deg_G(i) = 2$, we have one of the following three possibilities:

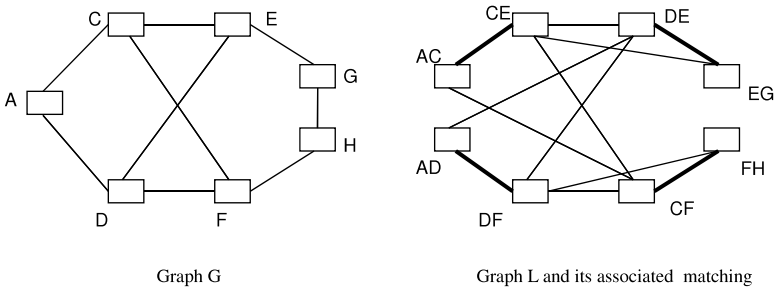

Fig. 9. G and its associated graph L: an example where the required matching exists.

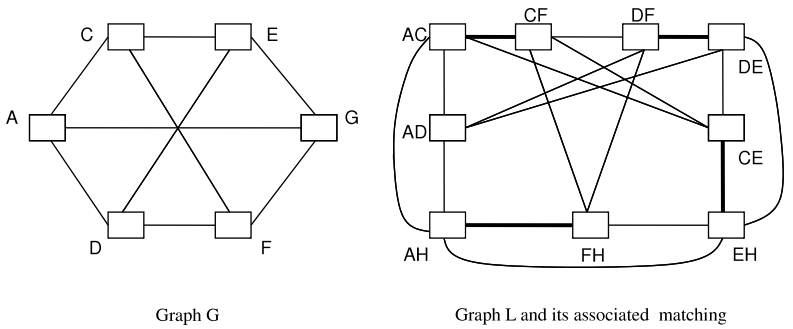

Fig. 10. G and its associated graph L: an example where the required matching does not exist.

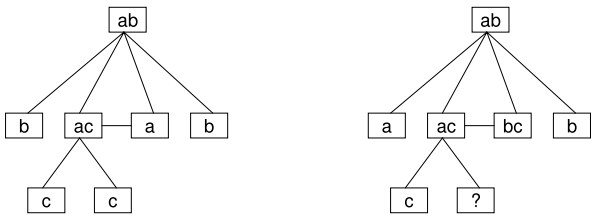

Fig. 11. A graph with $\Delta = 4$. Assigning labels arbitrarily to the neighbors of a node with degree 4 can lead to an erroneous conclusion.

(i) both labels remain to be assigned and two edges remain to the covered, or
(ii) one label remains to be assigned and one edge remains to be covered, or
(iii) no remaining label and no edge that remains to be covered. Note that the two nodes in L, corresponding to the two edges incident at i, are not connected. Thus, this case occurs if and only if the nodes corresponding to the two edges incident at node i are part of two different matching edges in L. An example is illustrated in Fig. 9 (node "A").

Now consider a node i in G with $\deg_G(i) = 3$ and let e_1^i, e_2^i, and e_3^i denote the three edges incident on i. Then, of the triplet of edges $\{e_1^i, e_2^i\}$, $\{e_1^i, e_3^i\}$, and $\{e_2^i, e_3^i\}$ in L, at most one can be in a matching. If none of these three edges is in a matching, then the cardinality of the matching is strictly less than p (since every edge in L is part of such a triplet of edges). Thus, if a matching of cardinality p exists in L, then exactly one of these three edges belongs to the matching. That is, exactly two of the three nodes e_1^i, e_2^i, and e_3^i in L are endpoints of a single matched edge; the third node may or may not be an endpoint of a matched edge.

It is now trivial to assign the labels to cover all the remaining edges. This completes our discussion for graphs with $\Delta \leq 3$.

3.2. Graphs with $\Delta = 4$

For any node i with $\deg_G(i) = 4$, we must use two labels and these two labels are fully used to cover the edges incident at node i. The only question is on the pairing of the neighbors of node i that receive the same label. Arbitrarily labeling the nodes incident on node i so that no label is used more than 3 times can land us into trouble even though a feasible covering exists; for example, see Fig. 11.

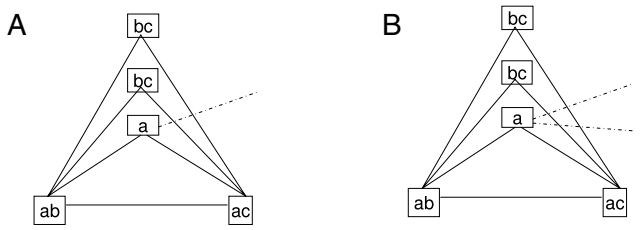

Fig. 12. Three distinct triangles sharing an edge.

As our analysis will soon reveal, the primary difficulty comes from nodes of degree 4 that are part of one or more triangles.[1] We start by considering the case when G has at least one triangle.

3.2.1. G has one or more triangles: covering a triangle by the same label

To resolve the covering problem, our basic idea is to identify a triangle, say $\{i, j, k\}$, such that the same label can be assigned at i, j, and k, without affecting the outcome of the covering problem. We show that either there is no feasible covering or such a triangle exists. We start by making the following observations.

Since G is simple, two distinct triangles cannot share more than one edge in common. If an edge is shared by three distinct triangles, then exactly one of three triangles *must* by covered by the same label in *any* feasible labeling of G. This claim follows since there is only one feasible labeling of the vertices of the three triangles (modulo permutations of the labels). This is illustrated in Fig. 12. Furthermore, it is also trivial to identify the triangle that is covered by the same label. To see this, we first note that only the apexes of the three triangles can have additional edges. If none of the three apexes have an additional edge(s), then the three triangles form a disjoint component and can, hence, be labeled separately from the rest of the graph. If any two or all three of the three apexes of the triangles have additional edges, then a feasible labeling does not exist. Thus, only one apex can have either one or two additional edges incident on it (Fig. 12(A) and (B)). The triangle corresponding to this apex *must* be the triangle covered by the same label in any feasible labeling. Thus, for the purpose of proving Theorem 5 below, we assume without loss of generality that an edge is shared by at most two distinct triangles. The next two lemmas identify two other cases in which a triangle can be covered by a common label.

Lemma 5. *Suppose G has a feasible labeling. If there exists a triangle, say $\{i, j, k\}$, such that the degrees (in G) of nodes i, j, and k, are each at most 3, then there exists a feasible labeling in which a common label is assigned at i, j, and k.*

Proof. The result follows since (i) assigning a common label, say "a", at i, j, and k, covers the three edges $\{i, j\}$, $\{j, k\}$ and $\{k, i\}$, and (ii) each of the three nodes then has at most one edge incident on it that needs to be covered, for which the one remaining label at that node suffices. □

Next, we introduce the following definition.

Definition. Two distinct triangles $\{i, j, k\}$ and $\{u, v, w\}$ are said to be *adjacent* if they share a common edge.

Lemma 6. *Suppose G has a feasible labeling. Consider a triangle, say $\{i, j, k\}$. If this triangle is not adjacent to any other triangle, then there exists a feasible labeling in which a common label is assigned at i, j, and k.*

Proof. Suppose not. Consider any feasible labeling of G. The only other labeling (modulo permutations of the labels) that covers the three edges of the triangle is as follows: node i (resp., node j; node k) is assigned labels "a" and "b" (resp., "b" and "c"; "c" and "a"). Note that at least one of nodes i, j, or k, has a degree of 4, for otherwise, the result follows from Lemma 5. Furthermore, since each of the three labels "a", "b", and "c" is used twice for labeling the triangle, there is no feasible labeling if any two or all three of i, j, and k, have a degree of 4 in G. Thus, exactly one of nodes i, j, and k, say k, has a degree of 4. Furthermore, if both the remaining nodes i and j have a degree of 3 in G, then a feasible labeling does not exist. Thus, at most one of nodes i and j, say i, has a degree of 3 in G. We, therefore, have the following two possibilities:

In either case, there is an alternate feasible labeling (see Fig. 13) in which (i) the vertices of the triangle $\{i, j, k\}$ have a common label, and (ii) the total usage of the labels "a", "b", and "c" is the same as that in the assumed feasible labeling. □

Using Lemmas 5 and 6, we now prove a general result.

Theorem 5. *If G has one or more triangles and there exists a feasible labeling of G, then there exists a feasible labeling in which a triangle is covered by the same label.*

[1] By a triangle, we mean a subgraph isomorphic to K_3.

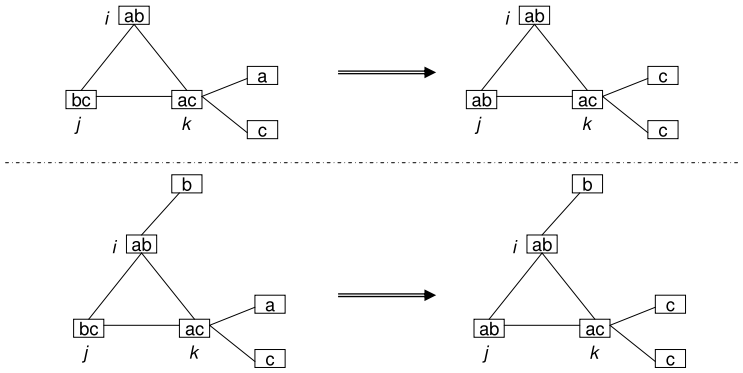

Fig. 13. The two possibilities in the proof of Lemma 6.

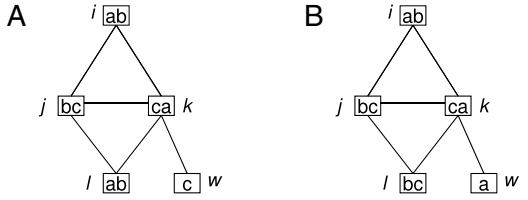

Fig. 14. The structure in the proof of Theorem 5.

Proof. If there exists a triangle such that the degree of each of its three vertices is at most 3, then the result follows from Lemma 5. Otherwise, if there exists a triangle that is not adjacent to any other triangle, then the result follows from Lemma 6. Consider a triangle, say $\{i, j, k\}$, and consider any feasible labeling of G. Without loss of generality, assume that edge $\{j, k\}$ is part of another triangle, say $\{j, k, l\}$. Thus, triangles $\{i, j, k\}$ and $\{j, k, l\}$ are adjacent; see Fig. 14. We consider the following two cases:

Case 1: If either triangle $\{i, j, k\}$ or triangle $\{j, k, l\}$ is covered by the same label, then the statement of the theorem holds.

Case 2: If neither of the two triangles is covered by the same label, then the only labeling (modulo permutations of the labels) that covers the edges of the two triangles is as follows: node i (resp., node j; node k; node l) is assigned labels "a" and "b" (resp., "b" and "c"; "c" and "a"; "a" and "b"). Then, node i cannot have a degree of 4, for otherwise this labeling is not feasible. Similarly, node l cannot have a degree of 4. Also, nodes j and k cannot both have a degree of 4, for otherwise this labeling is not feasible. Thus, exactly one of j and k, say k, has a degree of 4. As shown in Fig. 14, let node w be the node, different from i, j, and l, that is adjacent to node k. Since labels "a" and "b" have both been used 3 times, the only label that can be assigned to node w (to cover the edge $\{k, w\}$) is "c" (Fig. 14(A)). Note that each of the three labels "a", "b", and "c", has now been used $t = 3$ times and are, therefore, not available for further assignments. Therefore, any alternate feasible assignment of the labels "a", "b" and "c" that covers all the edges covered by these three labels can be used without impacting the covering of the other edges of G. Fig. 14(B) shows such an alternate feasible labeling that covers triangle $\{j, k, l\}$ with the same label.

The result follows. □

Remark 1. The proof of Theorem 5, in fact, implies that if two triangles are adjacent and if the graph admits a feasible labeling, then there exists a feasible labeling in which one of the two triangles is covered by the same label.

Although Theorem 5 guarantees the existence of a triangle that is covered by the same label in a feasible labeling (if one exists), finding a triangle with this property is an issue that remains to be addressed. We now describe an iterative procedure that propagates the impact of covering a specified triangle by the same label. The output of the procedure is a binary signal: an output of 1 implies that the vertices of the specified triangle can be assigned a common label without impacting the outcome of the covering problem. An output of 0 indicates otherwise. During the propagation, nodes are classified into three types: *marked nodes*, *double-marked nodes*, and *unmarked nodes*. We now define these terms.

Definition. A *marked node* is a node for which only one label remains to be assigned. The other label has been previously assigned during the propagation and has been completely used (i.e., has been assigned at t nodes).

Definition. A *double-marked node* is a node for which both its labels have been previously assigned during the propagation. Furthermore, both the labels have been completely used (i.e., each label has been assigned at t nodes).

An *unmarked node* is a node for which both its labels remain to be assigned. During the propagation, several nodes and edges may be removed. Therefore, at any point of time in the procedure, we refer to the graph as the *residual graph*. In the procedure below (i.e., Propagate-Labels-I), we assume for simplicity that G does not have an edge that is shared by three triangles. For otherwise, as discussed prior to stating Lemma 5, we can easily identify a triangle that is covered by the same label without affecting the outcome of the labeling question. Then, with this identified triangle as input, the propagation of the labeling in G is the same as in the procedure below.

Procedure **Propagate-Labels-I**

Input: A triangle $\{i, j, k\}$ in G.

Output: 1 (SUCCESS) or 0 (FAILURE). An output of 1 denotes that nodes i, j, and k, can be assigned a common label without affecting the outcome of the covering problem. An output of 0 indicates otherwise.

Step 1: Denote nodes i, j, and k, as marked nodes and delete edges $\{i, j\}$, $\{j, k\}$, and $\{k, i\}$. This corresponds to the situation where nodes i, j, and k share a common label. Thus, only one label remains to be assigned at these three nodes. Since the common label is used at three nodes, it cannot be used further.

Step 2: Consider a marked node ℓ:

(a) If the degree of node ℓ in the residual graph is 2, then (i) denote node ℓ as a double-marked node, (ii) denote each marked neighbor of node ℓ as double-marked, (iii) denote each unmarked neighbor of node ℓ as marked, and (iv) delete the two edges incident on node ℓ.
(b) If the degree of node ℓ in the residual graph is 3 or more, then return 0 (FAILURE).

Step 3: Consider a double-marked node ℓ:

(a) If the degree of node ℓ is 1 or more, then return 0 (FAILURE).
(b) If the degree of node ℓ is 0, delete node ℓ.

Step 4: If there exists a marked node with degree at least 2, go to Step 2. Otherwise, return 1 (SUCCESS); this corresponds to the situation when each node in the residual graph is either an unmarked node or a marked node with degree 1.

Remark 2. If two triangles are adjacent, say $\{i, j, k\}$ and $\{j, k, l\}$, then we can invoke Procedure Propagate-Labels-I, first for triangle $\{i, j, k\}$ and then immediately for triangle $\{j, k, l\}$. It follows from Remark 1 that the procedure *must* return a 1 (i.e., SUCCESS) on at least one of these two invocations, for otherwise no feasible labeling exists.

3.2.2. G has no triangles

If G has no triangles and $\Delta = 4$, then obtaining a feasible labeling, if one exists, is straightforward. The primary reason for the simplification is that a node with degree 4 can be labeled without any look-ahead since G has no triangles. We now discuss the procedure in detail. Consider a node $i \in V$ with $\deg_G(i) = 4$. To cover the four edges incident on i, we need two labels at node i, say "a" and "b". Then, two neighbors of node i can be assigned the label "a" and the remaining two neighbors can be assigned the label "b". Thus, both labels "a" and "b" have each been used $t = 3$ times and are not available for any further assignments. Note that the choice of the two neighbors of node i that are assigned the label "a" (resp. "b") can be done arbitrarily, without any future implications, since any choice completely utilizes the two labels "a" and "b" for labeling the four neighbors of node i.

We now describe a procedure that starts by labeling a node i with $\deg_G(i) = 4$ and propagates this labeling in G. The procedure either establishes that G cannot be covered, or reduces the covering problem to one that has already been resolved in our earlier discussion.

Procedure **Propagate-Labels-II**

Input: A graph G with no triangles and $\Delta = 4$.

Output: Either (i) 0 (FAILURE): A subgraph of G containing the forbidden structure of Fig. 5, thereby establishing that G cannot be covered, or (ii) 1 (POTENTIAL SUCCESS): A subgraph S of G in which (a) $\max_i \deg_G(i) \leq 3$ and (b) each node is either an unmarked node or a marked node with degree 1. The labeling for such a structure has already been resolved in Section 3.1.

Step 1: To begin with, all nodes of G are unmarked. Consider the nodes $i \in V$ with $\deg_G(i) = 4$ in any arbitrary sequence and do the following for each such node: If any of the four neighbors of node i is already double-marked, then return 0 (FAILURE). Otherwise, (i) denote each marked neighbor of node i as double-marked, (ii) denote each unmarked neighbor of node i as marked, and (iii) delete node i and all its incident edges.

Step 2: Consider a marked node ℓ:

(a) If the degree of node ℓ in the residual graph is 2: Return 0 (FAILURE) if either of the two neighbors of node ℓ is already double-marked. Otherwise, (i) denote each marked neighbor of node ℓ as double-marked, (ii) denote each unmarked neighbor of node ℓ as marked, and (iii) delete node ℓ and all its incident edges.
(b) If the degree of node ℓ in the residual graph is 3 or more, then return 0 (FAILURE).

Step 3: Consider a double-marked node ℓ:

(a) If the degree of node ℓ is 1 or more, then return 0 (FAILURE).
(b) If the degree of node ℓ is 0, delete node ℓ.

Step 4: If there exists a marked node with degree at least 2, go to Step 2. Otherwise, return 1 (SUCCESS); this corresponds to the situation when each node in the residual graph is either an unmarked node or a marked node with degree 1.

If Procedure Propagate-Labels-II returns 1, then (i) the degree of each node in the residual graph is at most 3, and (ii) each node in the residual graph is either an unmarked node or a marked node with degree 1. Since the degree of each marked node in the residual graph is 1, any feasible labeling of the residual graph requires one label for such a node. Therefore, without loss of generality, these nodes can be considered as unmarked nodes. Thus, the residual graph is a graph (with no marked nodes) with maximum degree at most 3 and no triangles. Then, we can follow our discussion immediately following Lemma 4 (Section 3.1) to resolve the covering problem for the residual graph.

4. Future research directions

A direct extension of our work is to consider the covering problem on a general graph G for $k = 2, t = 4$. For this problem, the complexity of testing the existence of a cover is not known. Clearly, for a cover to exist for these parameters, the maximum degree of G should be at most 6. However, the analysis for this case is fundamentally different from that for $k = 2, t = 3$. For instance, our characterization for trees in Theorem 2 does not hold for $k = 2, t = 4$; to illustrate, the graph in Fig. 6 can be covered for $k = 2, t = 4$ but not for $k = 2, t = 3$. In the case of general graphs, the complete graph K_5 is an example of a graph that can be covered for $k = 2, t = 4$ but not for $k = 2, t = 3$. Our analysis for $k = 2, t = 3$ revealed the central role of subgraphs isomorphic to K_3 in G. Similarly, the analysis for higher values of k and t is likely to depend on carefully chosen labelings for specific subgraphs. More generally, if a cover does not exist, then it might be of interest to obtain a feasible assignment that maximizes the number of edges covered. This problem has not been addressed thus far. The weighted versions of these problems are also open.

Acknowledgements

We thank Abhishek Arora and Dhruv Sethi of the Indian Institute of Technology, Roorkee, for useful discussions and for the preliminary investigation of covering problems during their summer internship. Section 2.1 includes the results of our discussions with them.

References

[1] D. Astels, G. Miller, M. Novak, A Practical Guide to Extreme Programming, Prentice Hall, 2002.
[2] I. Bárány, J. Edmonds, L.A. Wolsey, Packing and covering a tree by subtrees, Combinatorica 6 (1984) 221–233.
[3] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, 2000.
[4] M. Dawande, S. Kumar, V. Mookerjee, C. Sriskandarajah, Maximum commonality problems: applications and analysis, Management Science 54 (2008) 194–207.
[5] M. Dawande, M. Johar, S. Kumar, V. Mookerjee, A comparison of pair versus solo programming under different objectives, Information Systems Research 19 (2008) 71–92.
[6] R. Diestel, Graph Theory, 3rd ed., Springer, 2005.
[7] S. Dye, Heuristics for a tree covering problem, in: Proceedings of the 40th Annual Conference of Operational Research Society of New Zealand, 2005, 206–215.
[8] H. Erdogmus, L. Williams, The economics of software development by pair programmers, The Engineering Economist 48 (2003) 283–319.
[9] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, San Francisco, 1979.
[10] F. Hadlock, Minimum spanning forests of bounded trees, in: Southeastern Conf. on Combinatorics, Graph Theory and Computing, 1974.
[11] S. Kundu, J. Misra, A tree partitioning algorithm, SIAM Journal on Computing 6 (1977) 151–154.
[12] A. Wood, W. Kleb, Extreme programming in a research environment, Technical Report, NASA Langley Research Center, Hampton, VA, 2002.