Discrete Applied Mathematics 177 (2014) 14–18

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

On the weighted k-path vertex cover problem

B. Brešar [b,d], R. Krivoš-Belluš [a,*], G. Semanišin [a], P. Šparl [d,c]

[a] Institute of Computer Science, Faculty of Science, Pavol Jozef Šafárik University in Košice, Jesenná 5, 040 01 Košice, Slovakia
[b] Faculty of Natural Sciences and Mathematics, University of Maribor, Koroška cesta 160, SI-2000 Maribor, Slovenia
[c] Faculty of Organisational Sciences, University of Maribor, Kidričeva cesta 55a, SI-4000 Kranj, Slovenia
[d] Institute of Mathematics, Physics and Mechanics, Jadranska 19, 1000 Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 7 February 2014
Received in revised form 23 May 2014
Accepted 25 May 2014
Available online 14 June 2014

Keywords:
Weighted graph
Vertex cover
Tree
Graph algorithm

ABSTRACT

A subset S of vertices of a graph G is called a *k-path vertex cover* if every path of order k in G contains at least one vertex from S. The cardinality of a minimum k-path vertex cover is called the *k-path vertex cover number* of a graph G, denoted by $\psi_k(G)$. It is known that the minimum k-path vertex cover problem (k-PVCP) is NP-hard for all $k \geq 2$. In this paper we consider the weighted version of a k-PVCP (k-WPVCP), in which vertices are given weights, and the problem is to find a minimum weight set in G such that the graph obtained by deleting this set from G has no P_k. This problem was briefly introduced by Tu and Zhou (2011) but has not been studied to much extent. We give solutions and efficient algorithms for the k-WPVCP for some special classes of graphs. In particular, for complete graphs and cycles, and most importantly an algorithm that computes the k-path vertex cover number of a tree with time complexity $O(k \cdot |V(G)|)$ is presented.

1. Introduction

The central problem studied in this paper is related to the concept of *minimum k-path vertex cover* that was recently introduced in [3]. The same concept was studied also in [1] under the name k-observer problem. In both cases the concept is motivated by a real-life problem from computer science. The first motivation comes from the design of secure protocols for communication in wireless sensor networks (see [10]). The second motivation is related to controlling traffic in computer networks. The analogous motivation from the area of controlling traffic at street crossings can be found in [11]. The essential features of all mentioned practical problems can be expressed in the language of graph theory.

All graphs in this paper are finite without loops and multiple edges, but may have positive weights on their vertices; we use standard graph theory terminology (see e.g. [13]). In particular, by the *order of a path P* we understand the number of vertices on P while the *length of a path* is the number of edges of P. Given a graph $G = (V, E)$ and a positive integer k, a subset S of vertices of G is called a *k-path vertex cover* if S intersects all paths of order k in G; in other words, each path of order k contains a vertex from S. The minimum cardinality of a k-path vertex cover is called the *k-path vertex cover number* of a graph G, denoted by $\psi_k(G)$. Clearly for $k = 1$ the k-path vertex cover number corresponds to the order of a graph, and for $k = 2$ one obtains the well-known *vertex cover number*. For $k = 3$ the dual invariant is called *dissociation number*; cf. [4,14]. The general version of the k-path vertex cover number was studied in [1–3,7].

* Corresponding author. Tel.: +421 552342226; fax: +421 556222124.
E-mail addresses: bostjan.bresar@um.si (B. Brešar), rastislav.krivos-bellus@upjs.sk (R. Krivoš-Belluš), gabriel.semanisin@upjs.sk (G. Semanišin), petra.sparl@fov.uni-mb.si (P. Šparl).

http://dx.doi.org/10.1016/j.dam.2014.05.042
0166-218X/

It is no surprise that the problem of determining the k-path vertex cover number, i.e. the *k-Path Vertex Cover Problem* (abbreviated by k-PVCP) which generalizes the Minimum Vertex Cover Problem, is NP-hard for all $k \geq 2$, as shown in [3] and later on in [1]. Some efficient algorithms for the k-PVCP for trees, cycles, outerplanar graphs and grids can be found in [1,3]. Randomized approximation algorithms are provided in [8], while the factor 2 approximation algorithm for the k-PVCP is presented in [11] and interpreted in [12]. A polynomial-time approximation scheme (PTAS) for a modified version of the problem (it is required that k-path vertex cover must be connected) and for unit disc graphs is designed in [9].

It is also natural to consider the weighted version of a k-PVCP (abbreviated by k-WPVCP), in which vertices are given weights, arising from the practical situation where installing measuring devices in the nodes of the network requires different prices. The problem of finding a minimum weight set such that the graph has no P_k was briefly introduced in [11] but has not been studied to much extent. Obviously this problem is a generalization of the *Minimum Weight Cover Problem* that plays a central role in the computational complexity theory (see e.g. [6]). It is a special case of the *Vertex Deletion Problem* that can be stated as follows: in a given vertex weighted graph finds a minimum weight set of vertices whose deletion gives a graph satisfying a prescribed property. For instance an important special case of the Vertex Deletion Problem is the *Feedback Problem*: given a graph $G = (V, E)$ find a minimum weight set F of vertices such that the graph $G[V \setminus F]$ induced by $V \setminus F$ has no cycle. We mention that a different kind of weighted generalization of the vertex cover problem, called capacitated vertex cover problem, was studied in [5].

The k-WPVCP problem has various practical motivations. One of them is allocating specific servers to nodes of a communication network so that the set of servers has a non-empty intersection with any path of order k in the network. Such set of servers can be used to monitor all messages that travel at least k hops in the network, because it is guaranteed that they are observed by at least one server. Suppose that the cost of the installation of a server to a node is given by a function f. Then the problem can be formulated in graph theoretic terms as follows.

Definition 1 (*k-WPVCP—Weighted k-Path Vertex Cover Problem*)**.** Given a graph $G = (V, E)$, a positive weight function $f : V \to \mathbb{R}^+$, find a k-path vertex cover $S \subseteq V$ with the minimum value of $f(S) = \sum_{v \in S} f(v)$.

The instance of k-WPVCP with input $G = (V, E)$ and f will be denoted by $I = (k, G, f)$.

In Section 2 we give solutions and algorithms for the k-WPVCP for some special classes of graphs, i.e. complete graphs, paths and cycles. The main result is in Section 3 — an algorithm for determining k-WPVCP for trees. At the end of the paper we present an example which illustrates the main idea of the algorithm.

2. Complete graphs, paths and cycles

In this section we present efficient algorithms for determining optimal solutions for k-WPVCP of complete graphs, paths and cycles, respectively.

Clearly in the complete graph K_n at most $k - 1$ vertices may be unpicked in a k-path vertex covering. We immediately derive the following observation.

Proposition 1. *Let V be the vertex set of the complete graph K_n, and let f be a positive weight function on V. Then the optimal solution of $I = (k, K_n, f)$ can be computed in $O(n \cdot k)$ time.*

The optimal solution is obtained by letting the vertices with the biggest weights unpicked. Naturally, if the order of the weight function is not given, we need to compute it. The complexity can be realized by, say, using the first $k - 1$ steps of the Selection Sort Algorithm.

We continue by determining k-WPVCP for paths.

Theorem 1. *Let P_n denote a path on n vertices. The optimal solution of $I = (k, P_n, f)$ can be computed in $O(n \cdot k)$ time.*

Algorithm 1 Weighted k-Path Vertex Cover Algorithm for Paths

if $n < k$ **then**
  return 0
else
  for $i \leftarrow 1, 2, \ldots, n$ **do**

$$\psi_k(i) \leftarrow f(v_i) + \begin{cases} 0 & \text{for } i \leq k \\ \min\limits_{j=i-1,i-2,\ldots,i-k} \{\psi_k(j)\} & \text{otherwise} \end{cases}$$

  return $\min\limits_{j=n,n-1,\ldots,n-k+1} \{\psi_k(j)\}$

Proof. Algorithm 1 realizes Theorem 1. Note that $\psi_k(i)$ represents the minimum weight of a covering set of the subpath $v_1, v_2, \ldots, v_i$, when the vertex v_i is picked.

It is easy to see that the algorithm correctly computes the optimal value of $I = (k, P_n, f)$, and that it is done in time $O(n \cdot k)$.

For finding which vertices are picked in the optimal solution, one could store backpointer for every vertex (representing the vertex from which the minimum was obtained from). The construction of the optimal solution is then retrieved backwards from the vertex v_n to vertex v_1. This can also be done in time $O(n \cdot k)$. □

Note that the algorithm for paths is a simplified version of the algorithm for trees that will be explained in more detail in Section 3. We conclude this section by focusing on cycles.

Theorem 2. *Let $V = \{v_1, v_2, \ldots, v_n\}$ be the vertex set of C_n, and let f be a positive weight function on V. Then the optimal solution of $I = (k, C_n, f)$ can be computed with time complexity $O(n \cdot k^2)$.*

Algorithm 2 Weighted k-Path Vertex Cover Algorithm for Cycles

if $n < k$ **then**
  return 0
else
  for $j \leftarrow 1, 2, \ldots, k$ **do** //subgraph P^j
    for $i \leftarrow 1, 2, \ldots, n-1$ **do**

$$\psi_k^j(j+i) \leftarrow f(v_{j+i}) + \begin{cases} f(v_j) & \text{for } i \le k \\ \min\limits_{l=1,2,\ldots,k} \left\{\psi_k^j(j+i-l)\right\} & \text{otherwise} \end{cases}$$

    $\psi_k^j(j) \leftarrow \min\limits_{l=1,2,\ldots,k} \left\{\psi_k^j(j-l)\right\}$ //min for P^j
  return $\min\limits_{j=1,2,\ldots,k} \left\{\psi_k^j(j)\right\}$

Proof. Trivially, the optimal solution in the case of $n < k$ is zero. Otherwise, at least one of the vertices $\{v_1, \ldots, v_k\}$ has to be picked. Suppose that v_j is the picked vertex for some $j \in \{1, \ldots, k\}$. Then we can use modified Algorithm 1 for paths (modification is done by specifying that the first vertex is always picked, e.g. $\psi_k(i) = f(v_1) + f(v_i)$ for $i = 2, 3, \ldots, k$ in the case of picking the vertex v_1). For each possible picked vertex v_j we compute solution for $I^j = (k, P^j, f)$, where P^j is the subgraph of C_n without the edge $v_j v_{1+(j \bmod n)}$. Computation of I^j is specified in Algorithm 2, where the code is simplified in the manner that all vertex indices are calculated as a cyclic list (for vertices v_i and minimum weights $\psi_k(i)$), i.e. $v_{-n} = v_0 = v_n = v_{2*n}$, $\psi_k(i) = \psi_k(1 + (i-1) \mod n)$. Then the optimal solution of $I = (k, C_n, f)$ is the solution from $I^1, \ldots, I^k$ with the minimum value. It is not difficult to see that the time complexity is $O(n \cdot k^2)$. □

3. Trees

The main result of this paper is given in this section. We present an algorithm that determines the weight of an optimal k-WPVCP for an arbitrary tree in $O(k \cdot n)$ time and space. Hence for a fixed k the algorithm is linear with respect to the number of vertices of a tree. At the end of this section we provide an example, which gives a general idea how the algorithm works.

Let T be a tree with the vertex set V, the edge set E and weight function $f : V \to \mathbb{R}^+$. If r is an arbitrary vertex of V then we can use the Breadth-First-Search algorithm to create a topological enumeration τ of T with the source r (i.e. $\tau(r) = 1$). In such a way we obtain a rooted tree that we shall denote by $T(r)$. In addition, if v is any vertex of $T(r)$ then $T(v)$ stands for the rooted subtree of $T(r)$ with the root v.

Let us denote by $\Gamma^+(v)$ the set of successors of a vertex v (that is, the set of children of v in $T(v)$). Obviously, a vertex u is a leaf of $T(r)$ if and only if $\Gamma^+(u) = \emptyset$. It is also clear that $\tau(v)$ is less than $\tau(w)$ for all $w \neq v$ belonging to $T(v)$.

Theorem 3. *Let $T = (V, E)$ be a tree on n vertices, and let $f : V \to \mathbb{R}^+$ be a weight function. Then the optimal solution of $I = (k, T, f)$ can be computed with time and space complexity $O(k \cdot n)$.*

Proof. We start with the proof of the correctness of Algorithm 3. If $v \in V$ then let $S_k^i(v)$ denote an optimal k-WPVCP cover of $T(v)$, satisfying an additional condition that the order of the longest path of $T(v)$ that starts in v and contains no vertex of $S_k^i(v)$ is at most i, for $i = 0, \ldots, k-1$. We show that for each vertex $v \in V(T)$, Algorithm 3 computes the k-tuple $(\psi_k^0(v), \ldots, \psi_k^{k-1}(v))$, where $\psi_k^i(v)$, $i = 0, \ldots, k-1$, is the optimal weight of $S_k^i(v)$. According to the definition of $\psi_k^i(v)$ the sequence $\psi_k^0(v), \psi_k^1(v), \ldots, \psi_k^{k-1}(v)$ is always non-increasing. The output of the algorithm $\psi_k^{k-1}(r)$ is indeed the weight of an optimal k-WPVCP of T, since by definition, the longest path in the k-Path Vertex Cover S of T that starts in r and has no vertex from S is of length less than $k-1$.

The k-tuple for all leaves is correctly calculated in Step 5 of the algorithm.

We proceed by induction, assuming that the values $\psi_k^i(s)$ are correctly computed for all successors s of v. Assume that an optimal solution for k-WPVCP of $T(v)$ contains v. Then in Step 7 of the algorithm $\psi_k^0(v)$ adds to the weight $f(v)$ the sum of weights of the optimal values $\psi_k^{k-1}(s)$ for each of the successors s, which are by induction hypothesis obtained as an optimal solution in each subtree $T(s)$.

Algorithm 3 Weighted k-Path Vertex Cover Algorithm for Trees

Require: a weighted tree $T = (V, E)$ of order n with vertex weight function f and a positive integer k
Ensure: the weight of an optimal k-WPVCP of T

1: **choose** an arbitrary vertex r from V and **use** Breadth-First-Search algorithm to create a topological enumeration τ of T with root r
2: **for** $x \leftarrow n, n-1, \ldots, 1$ **do**
3: **take** $v \in V$ such that $\tau(v) = x$
4: **if** $\Gamma^+(v) = \emptyset$ **then**
5: $\psi_k^i(v) \leftarrow \begin{cases} f(v) & \text{for } i = 0 \\ 0 & \text{for } i = 1, 2, \ldots, k-1 \end{cases}$
6: **else**
7: $\psi_k^0(v) \leftarrow f(v) + \sum_{s \in \Gamma^+(v)} \psi_k^{k-1}(s)$
8: **for** $i \leftarrow 1, 2, \ldots, k-1$ **do**
9: $M(v) \leftarrow \begin{cases} \sum_{s \in \Gamma^+(v)} \psi_k^{k-i-1}(s) - \max_{s \in \Gamma^+(v)} \left[\psi_k^{k-i-1}(s) - \psi_k^{i-1}(s)\right] & \text{for } i > \frac{k}{2} \\ \sum_{s \in \Gamma^+(v)} \psi_k^{i-1}(s) & \text{otherwise} \end{cases}$
10: $\psi_k^i(v) \leftarrow \min\{\psi_k^{i-1}(v), M(v)\}$
11: **return** $\psi_k^{k-1}(r)$.

A bit more involved is the case when an optimal solution for k-WPVCP of $T(v)$ does not need to contain v. In Step 9 of the algorithm we first compute the value $M(v)$. Since v need not be picked, we can construct an optimal k-WPVCP for $T(v)$ by combining optimal solutions for its children. But we have to avoid an existence of an uncovered path of order k or greater, containing v and vertices from two different branches of $T(v)$. This is guaranteed by the computation of $M(v)$, where we again remind that the values of $\psi_k^i(v)$ are non-increasing with respect to i.

If we require that an uncovered path from vertex v in $T(v)$ is of order at most i, then the order of the uncovered (sub)path from the child of v must be at most $i-1$. In the case $i \leq \frac{k}{2}$, the order of the longest uncovered path from any child of v is at most $i-1$, and thus the longest uncovered path going through v and using two branches in $T(v)$ has order at most $(i-1)+1+(i-1) = 2i-1 \leq k-1$. Hence the k-Path Vertex Cover property is preserved.

In the case $i > \frac{k}{2}$, in calculating $M(v)$ we consider the case when one path (i.e. going from the vertex v through the child v_s) is of order i. (Note that the case when all the paths from v in $T(v)$ are shorter is already computed in $\psi_k^{i-1}(v)$, which is then compared with $M(v)$ in Step 10.) Then any other uncovered path from v going through a (different) child $s \in \Gamma^+(v) \setminus \{v_s\}$ can be of order at most $k-i$, yielding that the order of its subpath from the child is at most $k-i-1$. Thus in the spirit of dynamic programming the value $M(v)$ is computed from all the paths of order $k-i-1$ from all the children s of v in $T(s)$. Then for each child, the improvement is computed, by noting that one of the uncovered paths from v may be of order i. The maximum of these improvements will give us the value $M(v)$. This is correctly computed in the part $\max_{s \in \Gamma^+(v)} \left[\psi_k^{k-i-1}(s) - \psi_k^{i-1}(s)\right]$.

Finally, in Step 10 we compare the value $M(v)$ (which is obtained by assuming that one of the uncovered paths from v in $T(v)$ can be of order i) with $\psi_k^{i-1}(v)$ (where the order of the longest uncovered path from v in $T(v)$ is at most $i-1$), which is obtained in the previous step. Hence we derive that for each $v \in V$ the values $\psi_k^i(v)$, $i = 0, \ldots, k-1$ determine the weight of $S_k^i(v)$'s, which completes the proof of correctness.

Now, let us estimate the time complexity of the algorithm. Step 1 can be done in time $O(n+m)$, where m is the number of edges. The loop in Steps 2–10 is performed for each vertex $v \in V$. For each leaf it is sufficient to perform $O(k)$ operations. If v is not a leaf, then the computation of $\psi_k^0(v)$ requires $O(k)$ operations and the number of other operations is proportional to $k \cdot |\Gamma^+(v)|$. But one can easily see that each edge is considered just once when it connects actual root with some of its children. It means that altogether Steps 2–10 require $O(k \cdot (m+n))$ time. The last step can be performed in constant time. Since for trees we have $m = n-1$, the time complexity of the algorithm is $O(k \cdot n)$.

For finding the optimal solution (not only its weight) one can use backpointers like for Algorithm 1. The construction of the optimal solution is done backwards from the root vertex to the leaves. This can be done easily using the computation with time complexity $O(k \cdot n)$. □

Fig. 1 represents an example of a tree and the corresponding values. The left-hand side of the figure represents a vertex weighted tree T on 13 vertices, while the right-hand side shows the computed values for $k = 4$, obtained by Algorithm 3 for every vertex. Each vertex is represented by the 4-tuple (the legend is given at the bottom of the figure), the value of the optimal solution is displayed in a square. Edges represent how the value for the particular vertex (used in the optimal solution) was computed using the values of its successors (backpointers). Vertices which are filled by gray color are picked in the optimal solution.

Let us explain the obtained values for vertex v_3 with weight $f(v_3) = 5$. The upper edge indicates that the solution $S_4^1(v_3)$ of the subtree $T(v_3)$ was used in the optimal solution for the original tree T. This solution has the weight $\psi_4^1(v_3) = \min$

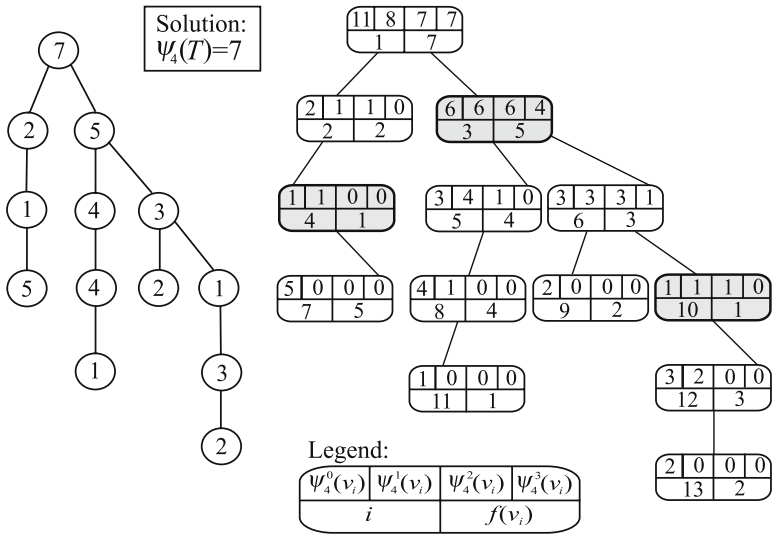

Fig. 1. An example of a tree T and the corresponding values computed with Algorithm 3.

$\left\{\psi_4^0(v_3), \psi_4^0(v_5) + \psi_4^0(v_6)\right\} = \min\{6, 7\} = 6$. The computation of $\psi_4^1(v_3)$ shows that it is better to pick vertex v_3 instead of its successors v_5 and v_6. The weight of $\psi_4^1(v_3) = 6 > 5 = f(v_3)$ shows that v_3 is not the only picked vertex in the subtree $T(v_3)$. The lower two edges indicate that for the solution $S_4^1(v_3)$ the solutions $S_4^3(v_5)$ and $S_4^3(v_6)$ were used. The computation of $\psi_4^2(v_3)$ is as follows: $\psi_4^2(v_3) = \min\left\{\psi_4^1(v_3), \psi_4^1(v_5) + \psi_4^1(v_6)\right\} = \min\{6, 7\} = 6$. The computation of $\psi_4^3(v_3)$ falls into the case where in Step 9 of Algorithm 3 the maximum of the improvements among all the successors of v_3 has to be computed. In this case we get $M(v_3) = \psi_4^0(v_5) + \psi_4^0(v_6) - \max\left\{\psi_4^0(v_5) - \psi_4^2(v_5), \psi_4^0(v_6) - \psi_4^2(v_6)\right\} = 4 + 3 - \max\{3, 2\} = 4$ and consequently $\psi_4^3(v_3) = \min\left\{\psi_4^2(v_3), M(v_3)\right\} = \min\{6, 4\} = 4$.

4. Conclusion

In this paper we give solutions and linear time algorithms for k-WPVCP for complete graphs, cycles and trees. Extensibility of the used technique to other classical families of graph (say, graphs with bounded treewidth, in particular cacti that are often used in motivations) is an intriguing open problem.

Acknowledgments

The first and fourth authors were supported in part by Slovenian Research Agency under the contract BI-SK/11-12-004 (bilateral Slovenian–Slovakian project entitled "Contemporary graph invariants and applications") and by the Ministry of Education, Science, Culture and Sport of Slovenia under the grants J1-2043 and P1-0285, respectively.

The research of the second and third authors was also supported under the grant numbers APVV-0035-10, APVV SK-SI-0014-10, contract VEGA 1/0479/12, project ITMS 26220120007 and FP7 EU project CELIM 316310.

References

[1] H.B. Acharya, T. Choi, R.A. Bazzi, M.G. Gouda, The K-observer problem in computer networks, in: Stabilization, Safety, and Security of Distributed Systems, in: Lecture Notes in Computer Science, vol. 6976, 2011, pp. 5–18.
[2] B. Brešar, M. Jakovac, J. Katrenič, G. Semanišin, A. Taranenko, On the vertex k-path cover, Discrete Appl. Math. 161 (2013) 1943–1949.
[3] B. Brešar, F. Kardoš, J. Katrenič, G. Semanišin, Minimum k-path vertex cover, Discrete Appl. Math. 159 (2011) 1189–1195.
[4] F. Göring, J. Harant, D. Rautenbach, I. Schiermeyer, On F-independence in graphs, Discuss. Math. Graph Theory 29 (2009) 377–383.
[5] F. Grandoni, J. Könemann, A. Panconesi, M. Sozio, A primal–dual bicriteria distributed algorithm for capacitated vertex cover, SIAM J. Comput. 38 (2008) 825–840.
[6] E. Halperin, Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs, SIAM J. Comput. 31 (2002) 1608–1623.
[7] M. Jakovac, A. Taranenko, On the k-path vertex cover of some graph products, Discrete Math. 313 (2013) 94–100.
[8] F. Kardoš, J. Katrenič, I. Schiermeyer, On computing the minimum 3-path vertex cover and dissociation number of graphs, Theoret. Comput. Sci. 412 (2011) 7009–7017.
[9] X. Liu, H. Lu, W. Wang, W. Wu, PTAS for the minimum k-path connected vertex cover problem in unit disk graphs, J. Global Optim. 56 (2013) 449–458.
[10] M. Novotný, Design and analysis of a generalized canvas protocol, in: Proceedings of WISTP 2010, in: LNCS, vol. 6033, Springer-Verlag, 2010, pp. 106–121.
[11] J. Tu, W. Zhou, A factor 2 approximation algorithm for the vertex cover P_3 problem, Inform. Process. Lett. 111 (2011) 683–686.
[12] J. Tu, W. Zhou, A primal–dual approximation algorithm for the vertex cover P_3 problem, Theoret. Comput. Sci. 412 (2011) 7044–7048.
[13] D.B. West, Introduction to Graph Theory, third ed., Prentice Hall, 2007.
[14] M. Yannakakis, Node-deletion problems on bipartite graphs, SIAM J. Comput. 10 (1981) 310–327.