SAND2015-2253J

The Approximability of Partial Vertex Covers in Trees

Vahan Mkrtchyan* Ojas Parekh† Danny Segev‡ K. Subramani§

January 13, 2015

Abstract

Motivated by applications in risk management of computational systems, we focus our attention on a special case of the partial vertex cover problem, where the underlying graph is assumed to be a tree. Here, we consider four possible versions of this setting, depending on whether vertices and edges are weighted or not. Two of these versions, where edges are assumed to be unweighted, are known to be polynomial-time solvable (Gandhi, Khuller, and Srinivasan, 2004). However, the computational complexity of this problem with weighted edges, and possibly with weighted vertices, has not been determined yet.

The main contribution of this paper is to resolve these questions, by fully characterizing which variants of partial vertex cover remain intractable in trees, and which can be efficiently solved. In particular, we propose a pseudo-polynomial DP-based algorithm for the most general case of having weights on both edges and vertices, which is proven to be NP-hard. This algorithm provides a polynomial-time solution method when weights are limited to edges, and combined with additional scaling ideas, leads to an FPTAS for the general case. A secondary contribution of this work is to propose a novel way of using centroid decompositions in trees, which could be useful in other settings as well.

*LDCSEE, West Virginia University, Morgantown, WV, USA. Email: vahan.mkrtchyan@mail.wvu.edu.

†Sandia National Laboratories, Albuquerque, NM, USA. Email: odparek@sandia.gov. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy National Nuclear Security Administration under contract DE-AC04-94AL85000.

‡Department of Statistics, University of Haifa, Haifa 31905, Israel. Email: segevd@stat.haifa.ac.il.

§LDCSEE, West Virginia University, Morgantown, WV, USA. Email: ksmani@csee.wvu.edu. The research of this author is supported by the National Science Foundation through Award CCF-1305054.

1 Introduction

The general setting. In the partial vertex cover problem we are given an undirected graph $G = (V, E)$ on n vertices, as well as a coverage requirement $P \geq 0$, standing for the number of edges to be covered. In this context, a feasible solution corresponds to a vertex set $U \subseteq V$ that covers at least P edges, where an edge (u, v) is said to be covered by U when the latter contains at least one of the vertices u and v. The objective is to compute a minimum cardinality vertex set that covers at least P edges.

The above-mentioned setting is a well-studied generalization of the classical vertex cover problem, where we wish to cover all edges. The latter is known to be APX-complete [21], and moreover, cannot be approximated within a factor smaller than 1.3606 unless P = NP [5]. In addition, assuming the unique games conjecture, the vertex cover problem cannot be approximated within factor $2-\epsilon$, for any fixed $\epsilon > 0$ [15]. These hardness results apply to partial vertex cover as well since it is a generalization of vertex cover. On the positive side, however, several polynomial-time algorithms have been devised for computing approximate partial vertex covers in general graphs. These algorithms attain a performance guarantee of 2 [2, 16], or slightly improve on this constant by lower order terms [1, 3, 7, 13].

The case of tree networks. In this paper, we focus our attention on a special case of the partial vertex cover problem where the underlying graph is assumed to be a tree. We consider four possible versions of this setting, depending on whether vertices and edges are weighted or not. Specifically, in the most general version, to which we refer as weighted partial vertex cover on trees (WPVCT), each vertex is associated with a cost, specified by an integer-valued function $c : V \to \mathbb{N}$. Similarly, each edge is associated with a coverage profit, given by $p : E \to \mathbb{N}$. With these definitions, our goal is to compute a vertex set of minimum cost, such that the collective profit of all edges covered is at least P. The three additional versions are obtained by restricting all edges to take unit profits (VPVCT), all vertices to take unit costs (EPVCT), and having both restrictions at the same time (PVCT).

In contrast to the intractability of computing partial vertex covers in general graphs, on tree networks the simplest version (i.e., PVCT) can be solved to optimality by means of dynamic programming. In fact, Gandhi, Khuller, and Srinivasan [7] showed that this problem can be solved in linear time even on graphs of bounded treewidth, with arbitrary vertex costs and with unit-profit edges, meaning in particular that VPVCT is polynomial-time solvable. However, to our knowledge, the computational complexity of WPVCT and EPVCT has not been determined yet and cannot be directly inferred from existing work in this direction.

1.1 Our results

The main contribution of this paper is to resolve the above-mentioned questions, by fully characterizing which variants of partial vertex cover remain intractable in trees and which can be efficiently solved. Our findings can be briefly summarized as follows:

1. We propose an exact algorithm for EPVCT, showing that this problem can be solved to optimality in polynomial time.

2. We observe that WPVCT is NP-hard and design a fully polynomial-time approximation scheme (FPTAS) for this problem.

From a technical perspective, a secondary contribution of this work is to propose a novel way of using centroid decompositions in trees. At first glance, it appears as if our approach leads to a dynamic-programming algorithm running in quasi-polynomial time[1]. However, additional insight allows us to argue that there are only polynomially-many states to be evaluated

[1] That is, having an exponent of poly-logarithmic order (in the input size).

throughout the overall computation, while all other states are actually irrelevant for our particular purposes. We believe that ideas in this spirit could be useful in additional settings.

1.2 Practical motivation

Our interest towards partial vertex covers is motivated by recent applications in risk management of computational systems or devices [17, 18, 19, 20]. Consider internet-accessible devices, such as servers, personal computers, or smartphones, routinely facing a large number of attacks on stored data, on their operating system, or on a specific software. In such scenarios, it is impossible to expect a manual response to each and every attack. Instead, such devices need to be configured in a way that enables automated response to potential attacks, which is a major technological challenge.

From a conceptual perspective, the risk to most devices of this nature depends on three factors: threats and their occurrence probabilities, existing weaknesses of the system, and the undesirable effects experienced after a successful attack. While threats cannot be controlled by system designers, the other two factors can typically be handled by decreasing functionality. Therefore, there is constant tension between the desired levels of security and functionality, and the main approach is to allow users to have maximum functionality, subject to a predefined level of risk.

Such problems were modeled by Caskurlu et al. [4] as network flows in tripartite graphs $G = (T \cup V \cup A, E)$, whose vertex partition represented threats (T), vulnerabilities (V), and the system assets (A). An edge joining two vertices represents a contribution to the overall risk, as shown in Figure 1. In this model, the objective is to reduce the system risk, measured as the total flow between T and A, below a given threshold level by either restricting user permissions, or encapsulating the system assets. In graph-theoretic terms, these strategies correspond to deleting a minimum weight subset of $V \cup A$, so that the flow between T and A reduces beyond some predefined level. As shown by Caskurlu et al. [4], the vertex set T can be merged into V by scaling each vulnerability weight appropriately. Their transformation establishes the equivalence between the above-mentioned risk management problem and the partial vertex cover problem on bipartite graphs, arguing in particular that trees form an important special case for real-life instances. We refer the reader to additional papers in this context [9, 10, 11, 12] for further discussion on risk management models in this spirit.

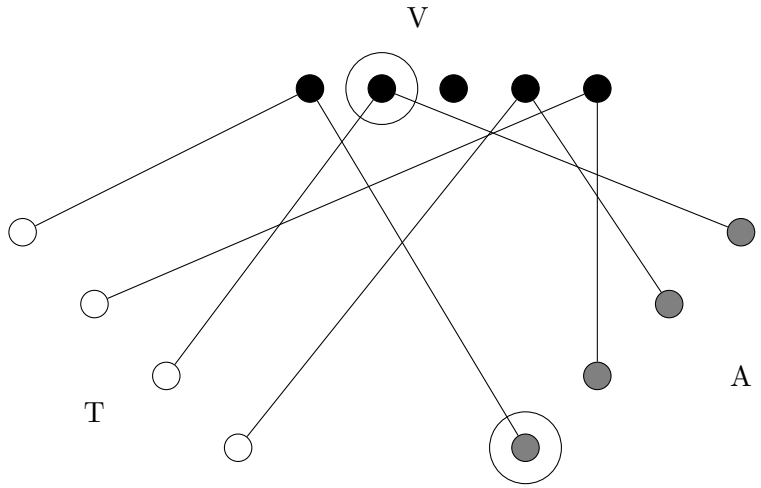

Figure 1: The tripartite graph of threats (T), vulnerabilities (V), and system assets (A). In this example, each edge has unit capacity. It is easy to notice that the initial (maximum) flow from T to A is of value 4; removing the two circled vertices decreases this value to 2.

2 Preliminaries

In what follows, we define a closely-related variant of WPVCT, which lends itself better to the divide-and-conquer approach proposed in Section 3. We explain how algorithms for this variant can be converted into ones for WPVCT and vice versa, and we prove that both problems are NP-hard.

The BMVCT problem. An instance of budgeted maximum vertex cover on trees (BMVCT, for short) consists of an undirected graph $G = (V, E)$, where each vertex is associated with a cost given by $c : V \to \mathbb{N}$, and each edge has a coverage profit specified by $p : E \to \mathbb{N}$. Given a budget $B \geq 0$, the objective is to compute a vertex set of cost at most B such that the combined profit of all covered edges is maximized.

It is not difficult to verify that an exact algorithm $\mathcal{A}$ for BMVCT can be utilized as a subroutine, in order to obtain an exact algorithm for WPVCT. For this purpose, it is sufficient to identify the minimal budget $B(P)$ needed to meet the coverage requirement P. This task can be accomplished by making use of $\mathcal{A}$ in a binary search for $B(P)$ over the interval $[C_{\max}, nC_{\max}]$. Here, $C_{\max}$ is the maximum cost of a vertex belonging to the optimal vertex set that attains the coverage requirement. This quantity is not known in advance, but can be found by trying all vertex costs as potential values for $C_{\max}$. Overall, we perform $O(n \log n)$ subroutine calls for $\mathcal{A}$. Similarly, analogous arguments show that, in order to obtain an exact algorithm for BMVCT, $O(n \log n)$ calls to an exact algorithm for WPVCT are sufficient.

NP-hardness. We proceed by describing a polynomial-time reduction from the classical knapsack problem to BMVCT. Based on the preceding discussion, it follows that both BMVCT and WPVCT are computationally intractable.

Theorem 2.1. *WPVCT and BMVCT are NP-hard.*

Proof. In the knapsack problem, we are given n items, each of which has a value v_i and a weight w_i, both are assumed to be positive integers. Given a weight bound of W, we wish to find a maximum-value subset of items I whose total weight is at most W. In other words, the goal is to maximize $\sum_{i \in I} v_i$ subject to the packing constraint $\sum_{i \in I} w_i \leq W$. This problem is NP-hard [14, 8].

Given a knapsack instance, we construct an instance of BMVCT as follows. We initially set up a matching M that consists of n edges, $(x_1, y_1), \ldots, (x_n, y_n)$, corresponding to the items $1, \ldots, n$, respectively. For every i, the profit of (x_i, y_i) is v_i, while the vertex costs of the endpoints x_i and y_i are w_i. The matching M is now augmented into a tree by adding an auxiliary vertex z, which is joined by edges to each of the vertices $y_1, \ldots, y_n$. This vertex has a zero cost, while all edges of the form (z, y_i) have zero profits. Finally, the total budget for picking vertices is W. This construction is sketched in Figure 2.

Notice that, if I is a subset of items with total weight at most W, then the subset of vertices $U_I = \{y_i : i \in I\}$ has exactly the same weight (cost). Therefore, U_I meets the budget constraint, and covers edges with identical value (profit). Conversely, suppose that U is a subset of vertices with total cost at most W. Then, by defining the subset of items $\{i : \{x_i, y_i\} \cap U \neq \emptyset\}$, we can only decrease the combined cost (weight), while preserving the total profit (value). ∎

3 DP-Based Algorithm

In this section, we devise a divide-and-conquer method for computing an exact solution to BMVCT in pseudo-polynomial time. By converting the suggested method into a dynamic programming algorithm, we will argue that the resulting running time is $O(B^2 n^{O(1)})$. Based on

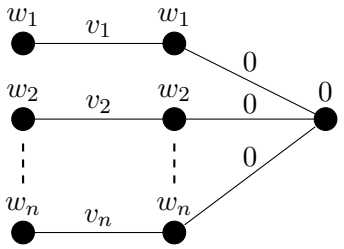

Figure 2: The tree obtained in the reduction.

the discussion in Section 2, we immediately obtain a pseudo-polynomial algorithm for WPVCT whose running time is $O(C^2 n^{O(1)})$, where $C = \sum_{v \in V} c(v)$. This claim follows by observing that each subroutine call to BMVCT is testing a candidate budget in $[C_{\max}, nC_{\max}]$, and $C_{\max} \leq C$.

3.1 Divide-and-conquer algorithm

Our method makes use of a well-known result regarding nearly-balanced decompositions of tree networks, stating that any tree can be partitioned into two edge-disjoint subtrees with roughly the same number of edges.

Definition 3.1. Let $T = (V, E)$ be a tree. A centroid decomposition of T is a partition of T into two edge-disjoint subtrees (sharing a common vertex) such that each subtree contains between $\frac{|E|}{3}$ and $\frac{2|E|}{3}$ edges.

Theorem 3.2. ([6]) *Let $T = (V, E)$ be a tree with $|E| \geq 2$. A centroid decomposition of T exists and can be found in linear time.*

Now suppose we are given an instance of BMVCT, consisting of a tree $T = (V, E)$ with integer vertex costs and edge profits, as well as an integer budget $B \geq 0$. For purposes of analysis, we focus attention on a fixed optimal solution U^*, i.e., a vertex set that satisfies the budget constraint $\sum_{v \in U^*} c(v) \leq B$ and maximizes the combined profit of all covered edges. Our algorithm proceeds as follows.

Trivial scenario: $|E| = 1$. In this case, we simply compute an optimal solution by trying all 4 possible subsets of vertices as candidate solutions.

General scenario: $|E| \geq 2$ By employing Theorem 3.2, we compute a centroid decomposition (T_1, T_2) of T, and designate by r the single intersection vertex of these subtrees. We proceed by guessing two attributes of the optimal vertex set U^*, as it relates to the decomposition (T_1, T_2), in order to break the current instance into two mutually independent instances, $\mathcal{I}_1$ and $\mathcal{I}_2$, one for each subtree:

- We first guess whether the intersection vertex r belongs to the optimal solution U^* or not.
- We then guess, by means of exhaustive enumeration, how much of the budget B is spent on picking vertices (for U^*) in the subtree T_1, other than r. In other words, by trying all $B + 1$ possible options, we assume that $B_1^* = \sum_{v \in U^* \setminus T_2} c(v)$ is known.

Note that the overall number of guesses is $O(B)$. Having this information at hand, we create two independent instances as follows:

- Case 1: When $r \notin U^*$, in the instance $\mathcal{I}_1$ we would have T_1 as the underlying tree, with budget B_1^*. Similarly, in $\mathcal{I}_2$ the tree is T_2, with budget $B - B_1^*$. Furthermore, from this

point on, we will keep an additional piece of information for each of these instances, stating that the vertex r has not been picked. This means, in particular, that the edges adjacent to r have not been covered yet.

- Case 2: When $r \in U^*$, in the instance $\mathcal{I}_1$ we would have T_1 as the underlying tree, with budget B_1^*, similar to the previous case. For the instance $\mathcal{I}_2$ we need a small modification: While the tree is still T_2, since r has to be paid for, the budget is now $B - B_1^* - c(r)$. The additional piece of information for each instance would be that the vertex r has been picked. So, in particular, the edges adjacent to r have already been covered.

As an immediate consequence of the above discussion, it follows that in order to compute an optimal solution, it remains to recursively solve the instances $\mathcal{I}_1$ and $\mathcal{I}_2$. From a running-time perspective, we make $O(B)$ guesses in each step, and recurse (for every possible guess) into two subproblems, each with an underlying tree consisting of at most $\frac{2|E|}{3}$ edges. This corresponds to a recursion tree with node degree $O(B)$ and depth $O(\log n)$, implying that our algorithm runs in $O((nB)^{O(\log n)})$ time.

3.2 Dynamic programming implementation

We now explain how to efficiently implement the divide-and-conquer algorithm suggested above by means of dynamic programming. To this end, we will first solve all instances corresponding to single-edge subtrees, which constitute the leaves of the recursion tree. These will be utilized in turn to solve instances located one level up the recursion tree, consisting of two edges, which will then be recombined into subtrees with three or four edges, so forth and so on. Formally, each of these instances is a state of the dynamic program, characterized by the following properties[2]:

1. Subtree $S \subseteq T$, created during our recursive application of centroid decompositions.
2. Remaining budget $B_S \geq 0$, for picking vertices in S.
3. Two subsets of vertices, U^+ and U^-, summarizing the additional information accumulated in higher levels of the recursion. That is, U^+ are the vertices in S that have already been picked (Case 2), and similarly, U^- are the vertices that should not be picked (Case 1).

To bound the relevant number of states, note that the number of subtrees that result from recursively breaking the original tree T via centroid decompositions is $O(n)$. Furthermore, the remaining budget B_S is clearly restricted to take integer values in $[0, B]$. Now, as far as the vertex sets U^+ and U^-, one can naively argue that $|U^+| + |U^-| = O(\log n)$, since in each step of the recursion a single vertex is added, either to U^+ or to U^-. Therefore, without additional insight, the number of possible subsets could be as large as $O(n^{O(\log n)})$.

However, the important observation is that the collection of vertices for which a decision has been made in the divide-and-conquer algorithm has a special structure. Letting T_1 be the subtree considered one level above S in the recursion, we know that the intersection vertex r_1 in the centroid decomposition of T_1 (into S and some other subtree) is the one added to either U^+ or U^-. Similarly, letting T_2 be the subtree considered one level above T_1, we know that the intersection vertex r_2 in the centroid decomposition of T_2 (into T_1 and some other subtree) is added to either U^+ or U^-, so forth and so on. For this reason, focusing on the subtree S, our recursive decomposition determines a unique sequence of intersection vertices $r_1, r_2, \ldots$ which comprises U^+ and U^-. Consequently, we can equivalently represent these two subsets as a binary sequence of length $O(\log n)$, corresponding to whether each of the vertices $r_1, r_2, \ldots$ is picked or not. The number of such sequences is only $O(2^{O(\log n)}) = O(n^{O(1)})$.

In summary, it follows that the overall number of states is $Bn^{O(1)}$, and since each one can be evaluated in $O(nB)$ time, we obtain an improved running time of $O(B^2 n^{O(1)})$. Based on the

[2]In addition to vertex costs and edge profits, which are common to all states.

relation between BMVCT and WPVCT, discussed at the beginning of this section, we have just proven the following.

Theorem 3.3. *The WPVCT problem can be solved to optimality in $O(C^2 n^{O(1)})$ time, where $C = \sum_{v \in V} c(v)$.*

As an immediate corollary, note that when all vertices are associated with unit costs, we obtain the EPVCT problem, where $C = n$. Therefore, Theorem 3.3 allows us to resolve the open question regarding the tractability of this variant (see Section 1).

Theorem 3.4. *The EPVCT problem can be solved to optimality in polynomial time.*

4 Fully Polynomial-Time Approximation Scheme

In what follows, we show that standard scaling arguments can be used to ensure that the sum of vertex costs C becomes polynomial in the number of vertices n, with only negligible loss in optimality. Specifically, given an error parameter $\epsilon > 0$, we define a rounded cost function $\tilde{c} : V \to \mathbb{N}$ satisfying the next two properties:

1. $\tilde{C} = \sum_{v \in V} \tilde{c}(v) = O(\frac{n^3}{\epsilon})$.
2. The optimal vertex set U w.r.t. $\tilde{c}$ has a near-optimal cost w.r.t. c, that is,

$$\sum_{v \in U} c(v) \leq (1 + \epsilon) \cdot \mathrm{OPT}_c \ .$$

Combined with the pseudo-polynomial algorithm given in Theorem 3.3, we complement the NP-hardness proof of WPVCT (see Theorem 2.1) with a fully polynomial-time approximation scheme.

Theorem 4.1. *The WPVCT problem admits an FPTAS, with a running time of $O(\frac{1}{\epsilon^2} \cdot n^{O(1)})$.*

Defining $\tilde{c}$. To construct the rounded cost function $\tilde{c}$, as before, let $C_{\max}$ be the maximum cost of a vertex belonging to the optimal vertex set that attains the coverage requirement. This parameter can be assumed to be known in advance, by trying all vertex costs as potential values for $C_{\max}$. Now, for each vertex v, its rounded cost is determined according to two cases:

- If $c(v) > C_{\max}$, then $\tilde{c}(v) = \lceil \frac{n^2}{\epsilon} \rceil + 1$.
- Otherwise, $\tilde{c}(v) = \lceil \frac{n}{\epsilon} \cdot \frac{c(v)}{C_{\max}} \rceil$.

Note that this definition indeed satisfies property 1, since

$$\tilde{C} = \sum_{v \in V} \tilde{c}(v) \leq n \cdot \left(\left\lceil \frac{n^2}{\epsilon} \right\rceil + 1 \right) = O\left(\frac{n^3}{\epsilon} \right) \ .$$

In addition, to establish property 2, we can upper bound the cost of the optimal vertex set U w.r.t. $\tilde{c}$ by relating it to that of the optimal set U^* w.r.t. c as follows:

$$\begin{aligned} \sum_{v \in U} c(v) &\leq \frac{\epsilon C_{\max}}{n} \cdot \sum_{v \in U} \tilde{c}(v) \\ &\leq \frac{\epsilon C_{\max}}{n} \cdot \sum_{v \in U^*} \tilde{c}(v) \\ &\leq \sum_{v \in U^*} \left(c(v) + \frac{\epsilon C_{\max}}{n} \right) \end{aligned}$$

$$= \sum_{v \in U^*} c(v) + \frac{|U^*|}{n} \cdot \epsilon C_{\max}$$
$$\leq (1+\epsilon) \cdot \mathrm{OPT}_c \ .$$

Here, the first and third inequalities are implied by the definition of $\tilde{c}$, and by the observation that U cannot contain any vertex v with $c(v) > C_{\max}$. The second inequality is implied by the optimality of U w.r.t. $\tilde{c}$. The final inequality holds since $C_{\max} \leq \mathrm{OPT}_c$.

References

[1] R. Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *Journal of Algorithms*, 39(2):137–144, 2001.

[2] R. Bar-Yehuda, G. Flysher, J. Mestre, and D. Rawitz. Approximation of partial capacitated vertex cover. *SIAM Journal on Discrete Mathematics*, 24(4):1441–1469, 2010.

[3] N. H. Bshouty and L. Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, pages 298–308, 1998.

[4] B. Caskurlu, A. Gehani, C. Ç. Bilgin, and K. Subramani. Analytical models for risk-based intrusion response. *Computer Networks*, 57(10):2181–2192, 2013.

[5] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

[6] G. N. Frederickson and D. B. Johnson. Finding k-th paths and p-centers by generating and searching good data structures. *Journal of Algorithms*, 4(1):61–80, 1983.

[7] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53(1):55–84, 2004.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Co., New York, NY, USA, 1979.

[9] A. Gehani. *Support for Automated Passive Host-based Intrusion Response*. Ph.D. Thesis, Duke University, 2003.

[10] A. Gehani. Performance-sensitive real-time risk management is NP-hard. In *Workshop on Foundations of Computer Security, affiliated with the 19th IEEE Symposium on Logic in Computer Science*, 2004.

[11] A. Gehani and G. Kedem. Real-time access control reconfiguration. In *International Infrastructure Survivability Workshop, Affiliated with the 25th IEEE International Real-Time Systems Symposium (RTSS)*, 2004.

[12] A. Gehani and G. Kedem. Rheostat: real-time risk management. In *7th International Symposium on Recent Advances in Intrusion Detection*, 2004.

[13] D. S. Hochbaum. The t-vertex cover problem: Extending the half integrality framework with budget constraints. In *Proceedings of the 1st International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 111–122, 1998.

[14] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[15] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[16] J. Mestre. A primal-dual approximation algorithm for partial vertex cover: Making educated guesses. *Algorithmica*, 55(1):227–239, 2009.

[17] N. B. of Standards. In *1st Computer Security Risk Management Model Builders Workshop*, Martin Marietta, Denver, Colorado, May 1988.

[18] N. B. of Standards. In *2nd Computer Security Risk Management Model Builders Workshop*, AIT Corporation, Ottawa, Canada, June 1989.

[19] N. B. of Standards. In *3rd Computer Security Risk Management Model Builders Workshop*, Los Alamos National Laboratory, Santa Fe, New Mexico, August 1990.

[20] N. B. of Standards. In *4th Computer Security Risk Management Model Builders Workshop*, University of Maryland, College Park, Maryland, August 1991.

[21] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–440, 1991.