On the Hardness of Approximating the MIN-HACK Problem

Ramkumar Chinchani, Duc Ha, Anusha Iyer, Hung Q. Ngo, and Shambhu Upadhyaya
Computer Science and Engineering,
201 Bell Hall
State University of New York at Buffalo,
Amherst, NY 14260, USA.
{rc27,ducha,aa44,hungngo,shambhu}@cse.buffalo.edu

Abstract

We show several hardness results for the MINIMUM HACKING problem, which roughly can be described as the problem of finding the best way to compromise a target node given a few initial compromised nodes in a network. We give several reductions to show that MINIMUM HACKING is not approximable to within $2^{(\log n)^{1-\delta}}$ where $\delta = 1 - \frac{1}{\log\log^c n}$, for any $c < 1/2$. We also analyze some heuristics on this problem.

1 Introduction

The MINIMUM COST HACKING problem (MIN-HACK for short), recently introduced in [4], can roughly be described as the problem of finding an optimal method to hack into a set of target nodes given an initial set of compromised nodes of a network represented by a directed graph G. The graph G is referred to as the *key challenge graph* for reasons explained later. For the hacker to traverse an edge $e = (u, v)$ of the graph, the hacker has to pay some price. The edge e to be visited has a *key challenge* k_e such as some authentication data, or records in a database. If the hacker possesses the key k_e, then he only needs to pay a small price c_2, otherwise he has to pay a larger price c_1 for traversing the edge. Each node v possesses some piece of information $\pi(v)$, which might be a key to traverse some edge in the graph. Hence, the more number of nodes the hacker is able to hack into, the more keys (information) he possesses, and thus the more likely he will be able to hack into the rest of the nodes with lower cost.

The MIN-HACK problem has been shown to be $\mathbf{NP}$-hard in [4]. In this paper, we address the question of how hard it is to approximate MIN-HACK to within some ratio. We will show that MIN-HACK is not approximable to within $g_c(n) := 2^{(\log n)^{1-\delta}}$, where $\delta = 1 - \frac{1}{\log\log^c n}$, for any $c < 1/2$. This negative result will be shown via three independent reductions, which are of independent interests, at least for pedagogical reason. The reductions are from MINIMUM LABEL COVER (MLC), from a PCP characterization of $\mathbf{NP}$ (mimicking an idea of Dinur and Safra [7]), and from the MINIMUM MONOTONE SATISFYING ASSIGNMENT (MMSA) problem. In fact, there is an hierarchy of MMSA numbered from MMSA$_i$, $i = 1, 2, \ldots$ Our reduction shows that MIN-HACK is on top of this hierarchy, namely for any $i \geq 4$,

$$\text{PCP} \ll \text{MMSA}_3 \ll \text{MLC} \ll \text{MMSA}_4 \ll \cdots \ll \text{MMSA}_i \ll \text{MIN-HACK},$$

where $\ll$ denote the relation "polynomially related approximation ratio." Although quite unlikely, if MIN-HACK can be approximated to within $g_c(n)$, then the hierarchy collapses. We have not been able to reduce any problem in "class 4" to MIN-HACK. (Class 4 consists of problems with non-approximability ratio n^δ like MAX-CLIQUE and COLORING [3].) We also analyze the approximation ratios of some common-sense heuristics for this problem.

The rest of the paper is organized as follows. Section 2 formally presents our problem and related background materials. Sections 3, 4, and 5 present three different reductions to show the hardness of MIN HACK. Section 6 gives our analysis of several Dijkstra's like heuristics for this problem.

2 Preliminaries

2.1 NP optimization problems, approximation algorithms, and approximation ratios

More detailed definitions and notations relating to optimization problems and approximation algorithms can be found in several books such as [8, 10, 11]. We briefly define related concepts here.

Following [5], an NP optimization problem Φ is a 4-tuple $(I, \text{sol}, \text{cost}, \text{OPT})$, where

- I is the set of polynomial-time recognizable instances of Φ.
- For each $x \in I$, $\text{sol}(x)$ is the set of feasible solutions for x. Feasible solutions are polynomially bounded in size and polynomial-time decidable.
- For each $x \in I, y \in \text{sol}(x)$, $\text{cost}(x, y)$ – a positive integer – is the objective value of the solution y. The objective function cost is polynomial-time computable.
- Lastly, $\text{OPT} \in \{\min, \max\}$ refers to the goal of the optimization problem: finding a feasible solution maximizing or minimizing the objective value.

The class of NP optimization problems is denoted by NPO. The class NPO PB consists of NPO problems whose objective functions are polynomially bounded.

We use $\Phi(x)$ to denote the optimal objective value of an instance x of problem Φ. Given a feasible solution y for an instance x of Φ, the approximation ratio of y is

$$R(x, y) := \max\left\{\frac{\text{cost}(x, y)}{\Phi(x)}, \frac{\Phi(x)}{\text{cost}(x, y)}\right\}.$$

Given a function $r : \mathbb{N} \to \mathbb{R}^+$, an $r(n)$-*approximation algorithm* A for Φ is a polynomial-time algorithm which, on input instance $x \in I$, returns a feasible solution y for x such that $R(x, y) \leq r(|x|)$. We will use $A(x)$ to denote $\text{cost}(x, y)$, the cost of the solution returned by A on x.

2.2 The MIN-HACK problem

In a recent paper [4], we have devised a model for computer system vulnerability assessment. The model can roughly be described as follows.

An instance of the model is constructed by a security analyst who is aware of the computational infrastructure. Before formally defining the model, let us first motivate its formulation.

Any physical entity on which some information or capability can be acquired is represented as a *vertex* of a graph, which shall be called the *key challenge graph*. Let V be the set of vertices. Typically, vertices are points in the network where some information may be gained such as a database server or simply any computer system whose resources can be used or misused.

Each piece of information or capability present at any vertex v is represented as a *key* called Key_v. Let $\mathcal{K}$ denote the set of keys. For example, records in a database, passwords stored on a computer, or computational resources of a computer can be represented as keys. When an attacker visits a vertex, he is empowered with this additional information or capability.

If there is a channel of access or communication between two physical entities which facilitates interaction, then a *directed edge* is created between the two corresponding vertices, pointing to the direction of the allowed interaction. Multiple channels of communication are possible, hence there can be multiple edges between two vertices. Let E be the set of edges.

The presence of a security measure or an enforced security policy protects the resources and allows only authorized interaction. This deterrence is represented as a *key challenge* on the corresponding channel of communication. An example of a key challenge is the password authentication required prior to accessing to a server.

If a user does not have the right key to the key challenge, then he incurs a significant *cost* in breaking or circumventing the security policy; legitimate access incurs only a smaller cost of meeting the key challenge. For example, when password authentication is used, if a user knows the password, he incurs little or no cost, while another user who doesn't know the password will incur a higher cost in breaking the password. The cost metric is a relative quantity signifying the amount of deterrence offered by one security measure over another. It has been abstracted as a non-negative integer for the purposes of our model. Figure 1 illustrates the building block of our model.

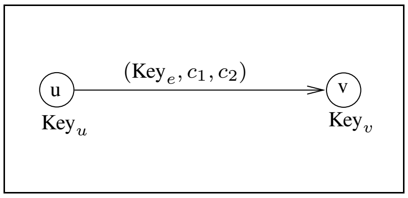

Figure 1: Basic building block of a key challenge graph. Key$_e$ is the key challenge of the edge $e = (u, v)$, c_1 is the cost one has to pay to traverse (u, v) without having Key$_e$, and $c_2 < c_1$ is the corresponding cost if the attacker has the key.

The starting point of an attack could be one or more vertices in the graph, which are assumed to be initially compromised by the attacker. Let V_s denote this set.

The target of an attack could also be one or more vertices in the graph. In case of multiple targets, the goal is to compromise all of them. Let the set of target vertices be denoted by V_t. An example of a target is a source code repository for a commercial product.

In what follows, we formalize the aforementioned concepts.

Definition 2.1 (Key Challenge Graph)**.** A *Key Challenge Graph* G is a tuple:

$$G = (V, E; \mathcal{K}, V_s, V_t, \kappa, \delta),$$

where V is the set of vertices, E is the set of directed edges (we allow multi-edges, i.e. G is a multi-graph), V_s is the initial set of compromised vertices, V_t is the set of target vertices ($V_t \cap V_s = \emptyset$), $\kappa : V \to \mathcal{K}$ is a function that assigns keys to vertices, $\delta : E \to \mathcal{K} \times \mathbb{N} \times \mathbb{N}$ is a function that assigns key challenges and costs to edges.

For instance, $\kappa(v_1) = k_0$ means that the key k_0 can be obtained at vertex v_1, $\delta(e_1) = (k_1, c_1, c_2)$ implies an assignment of a key challenge to edge e_1, which requires an attacker to produce the key k_1. If he cannot do so, then he incurs a cost c_1; otherwise, he incurs a smaller cost c_2. An adversary begins his attack at some point in the set of compromised nodes in the graph and proceeds by visiting more and more vertices until the target(s) is reached. At each visited vertex, the attacker adds the corresponding key to his collection of keys picked up at previous vertices. Once an attacker compromises a vertex, he continues to have control over it until the attack is completed. Therefore, any vertex appears exactly once in the attack description. While a trivial attack can be performed by visiting all reachable vertices until the target is reached, cost constraints occlude such free traversals.

Definition 2.2 (Attacks and successful attacks)**.** An *attack* is a finite and ordered sequence of a vertices $(v_1, v_2, \ldots, v_m)$ satisfying the following conditions:

1. for each $i \in \{1, \ldots, m\}$, there is some vertex $u \in V_s \cup \{v_1, \ldots, v_{i-1}\}$ such that $(u, v_i) \in E$,
2. $V_s \cap \{v_1, \ldots, v_m\} = \emptyset$,

The first condition is meant to say that the attacker must get to a new vertex v_i via a visited (or compromised) vertex u. A *successful attack* is an attack that contains all target nodes in V_t

The next important aspect of the model is the cost metric. Although an attack is defined exclusively in terms of vertices, the cost incurred by the attacker at a vertex is mainly dependent on the edge that he chooses to visit the vertex. We first define the cost of traversing an edge and then the cost of visiting a new vertex. The latter is the basic unit of cost metric in our model.

Definition 2.3 (Cost of Traversing an Edge)**.** Let V^* be the set of visited vertices so far, including the initially compromised vertices, i.e. $V_0 \subseteq V^*$. For $u \in V^*$ and $v \notin V^*$, the *cost of traversing the edge* $e = (u, v) \in E$, given that $\delta(e) = (k, c_1, c_2)$, is c_1 if $k \notin \{\kappa(w) \mid w \in V^*\}$; otherwise, it is c_2. (In general, $c_1 > c_2$.)

If $e = (u, v) \notin E$, for technical convenience we assume that $\delta(e) = (k, \infty, \infty)$, where k is some unique key no node has.

Definition 2.4 (Cost of Visiting a New Vertex)**.** Define V^* as above. The *cost of visiting a new vertex* $v \notin V^*$ is defined to be

$$\alpha(v, V^*) = \min\{\text{cost of traversing } (u, v) \mid u \in V^*\}. \tag{1}$$

The cost of an entire attack is measured as a sum of the effort required to compromise individual vertices by attempting to counter the key challenges on the edges with or without the keys that an attacker has already picked up.

Definition 2.5 (Cost of an attack)**.** The *cost of an attack* $(v_1, \ldots, v_m)$ is defined as:

$$\textsc{MinHack}(v_1, \ldots, v_m) = \sum_{i=1}^{m} \alpha(v_i, V_i), \tag{2}$$

where $V_i = V_s \cup \{v_1, \ldots, v_{i-1}\}$.

The MIN-HACK problem is the problem of finding a minimum cost successful attack, given a key challenge graph.

2.3 The MINIMUM LABEL COVER$_p$ (MLC$_p$) problem

Given $p \in \mathbb{N}^+$, an instance to the MLC$_p$ problem consists of:

(i) a bipartite graph $G = (U \cup V; E)$,

(ii) two sets B_1 and B_2 of labels, one for U and one for V, and

(iii) for each edge $e \in E$, a relation $\Pi_e \subseteq B_1 \times B_2$ which defines admissible pairs of labels for that edge.

A *labelling* (f_1, f_2) is a pair of functions: $f_1 : U \to 2^{B_1}$ and $f_2 : V \to 2^{B_2} \setminus \{\emptyset\}$. Basically, a labelling associates a set of labels to each $u \in U$, and a non-empty set of labels to each $v \in V$. A labelling *covers* an edge $e = (u, v) \in E$ if, for every label b_2 assigned to v, there is some label b_1 assigned to u such that $(b_1, b_2) \in \Pi_e$. A labelling is a *complete cover* (or *complete label cover*) if it covers all edges. Let $U = \{u_1, \dots, u_n\}$. The *L_p-cost* of a labeling (f_1, f_2) is the L_p-norm of the vector $(|f_1(u_1)|, \dots, |f_1(u_n)|) \in \mathbb{Z}^n$. Specifically,

$$L_p\text{-cost}(f_1, f_2) = (|f_1(u_1)|^p + \cdots + |f_1(u_n)|^p)^{1/p}.$$

And, $L_\infty\text{-cost}(f_1, f_2) = \max\{|f_1(u_i)| : 1 \le i \le n\}$. The MLC$_p$ *problem* is to find a complete cover with minimum L_p-cost.

It is not necessary to assign more than one label to any vertex $v \in V$. If a complete label cover assigns multiple labels to some vertex $v \in V$, then the labeling obtained by removing all but one label from v is also a complete covering. Consequently, henceforth we can impose the condition that vertices in V get only one label each.

Hardness results for this problem were devised in [2,7,9]. The current best result is that of [7], which says that MLC$_p$ is NP-hard to approximate to within $\Theta(2^{(\log n)^{1-\delta}})$, where $\delta = (\log\log n)^{-c}, \forall c < 1/2$.

2.4 The Minimum Monotone Satisfying Assignment (MMSA) problem

A *monotone boolean formula* F is a boolean formula over the basis $\{\wedge, \vee\}$, i.e. F uses only binary connectives $\wedge$, and $\vee$. Equivalently, F is a rooted binary tree where each internal node is labeled with either $\wedge$ or $\vee$ and has exactly two children, and each leaf is a boolean variable.

The Minimum Monotone Satisfying Assignment (MMSA) was considered in [1] in relation to the problem of finding the length of a propositional proof. The MMSA problem is the problem of finding a truth assignment satisfying a monotone boolean formula which minimizes the number of TRUE variables. The problem was shown to be at least as hard as Label-Cover.

For each positive integer i, MMSA$_i$ is the restriction of MMSA to formulas of depth i. For instance, MMSA$_3$'s instances are monotone boolean formulas of the form AND of OR's of AND's. Following [7], let $\ll$ denote the relation "polynomially related approximation-ratio". The authors showed that, for all $i \ge 4$,

$$\text{PCP} \ll \text{MMSA}_3 \ll \text{MLC} \ll \text{MMSA}_4 \ll \cdots \ll \text{MMSA}_i \tag{3}$$

3 Reducing MMSA to Min-Hack

We need to quote the following result from Dinur and Safra [7].

Theorem 3.1. *It is NP-hard to approximate* MMSA$_3$ *to within a ratio of* $\Theta(g_c(n))$ *for any* $c < 1/2$, *where*

$$g_c(n) := 2^{(\log n)^{1-\delta}}, \ \ \delta = 1 - \frac{1}{\log\log^c n}. \tag{4}$$

Thus, MMSA in general is not approximable to within $g_c(n)$, assuming P $\neq$ NP. We now describe a reduction from MMSA to Min-Hack so that the approximation ratio is preserved (with a linear blow-up in input size).

Given a monotone boolean formula $\phi(x_1, \dots, x_m)$. It is useful to think of ϕ as a full binary tree T, each of whose internal node is either an AND or an OR, and each of whose leaves is labeled with one of the variables $x_1, \dots, x_m$. We give a name to each internal node of T (beside the label AND or OR) to distinguish all nodes of T.

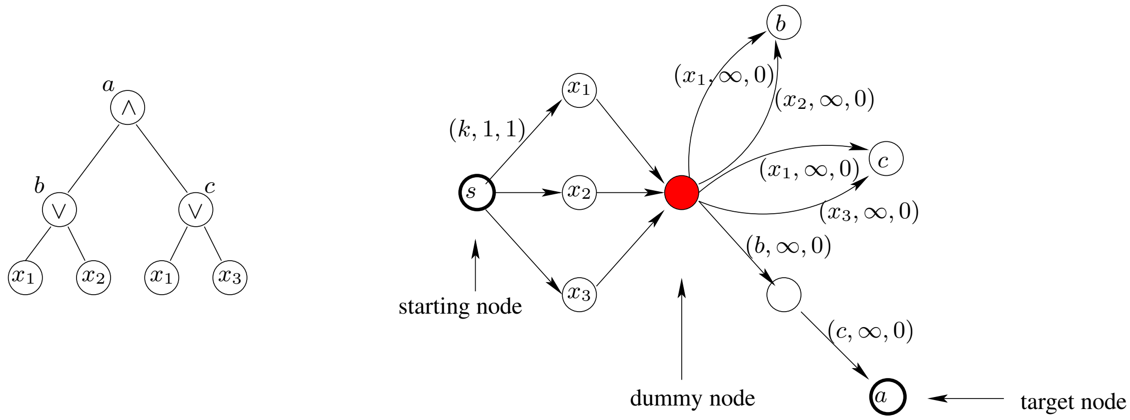

Figure 2: Example of the reduction from MMSA to MIN-HACK

The key challenge graph $G_\phi = (V, E; \mathcal{K}, V_s, V_t, \kappa, \delta)$ is constructed from ϕ as follows. (Refer to Figure 2 for an illustration of the reduction.)

The key set $\mathcal{K}$ is the set of all labels of nodes in T, along with three dummy keys s, k, and k'. The three dummy keys are not the key challenge of any edge in G_ϕ.

The compromised node set V_s consists of just a node s. We abuse notations a little bit here by naming this node with the key it has.

The target node set V_t consists of a node t whose key is the label of the root of T. The idea is that, getting to this node is the same as verifying that ϕ is satisfied by some truth assignment.

The graph G_ϕ consists of two main stages: the *truth assignment* and the *verification* stages.

The truth assignment stage consists of m edges from s to m nodes with keys $x_1, x_2, \dots, x_m$. One always has to pay a cost of 1 to get to any of these nodes. These edges have key challenge k (a dummy key). Getting to node x_i corresponds to assigning x_i to be TRUE. Hence, the number of keys from $\{x_1, \dots, x_k\}$ the attacker gets is the same as the number of TRUE variables in a truth assignment for ϕ. For convenience, all nodes x_1 to x_m are connected to a dummy node which has key k', another dummy key not used anywhere else. The cost to get to the dummy node is always 0. The key challenges of edges leading to the dummy node is, again, k.

The verification stage is designed to make sure that the combination of keys that the attacker gets from the first stage corresponds to a truth assignment satisfying ϕ. There are two types of components for this stage: the AND components, and the OR components. For every internal node a with children b and c of T, we construct a component corresponding to a.

If a is an AND node, then connect the dummy node to a node with key a via another auxiliary node. See Figure 3 for an illustration. The idea is that, to get the key a, one needs the keys of both of its children b and c, otherwise one pays a price of ∞. Node that, one can use a cost of $m + 1$ to represent ∞. This point will become more obvious later.

If a is an OR node, then to get the key a we only need either key b or key c (see Figure 4).

Recall a note earlier saying that ∞ can be represented by $m + 1$. The proof of the following lemma asserts this point.

Lemma 3.2. *For any positive integer p. The monotone boolean formula ϕ has a truth assignment with at most p TRUE variables if and only if there is a successful attack on the graph G_ϕ with cost at most p.*

Proof. Note that a true assignment with all m variables assigned to TRUE would satisfy ϕ, and that a successful attack on G_ϕ of cost m can always be found (the attacker can get all keys including corresponding to a postorder traversal of T). Hence, if $p \geq m + 1$, then the theorem is trivially true. We

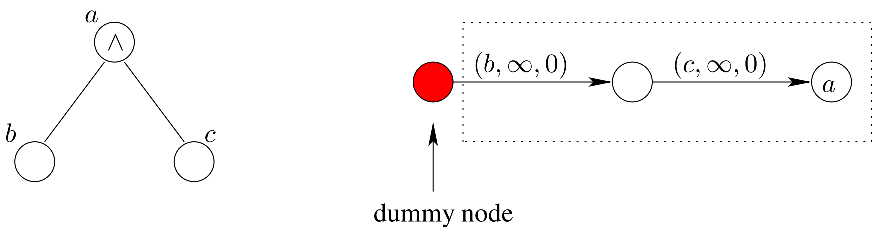

Figure 3: AND component

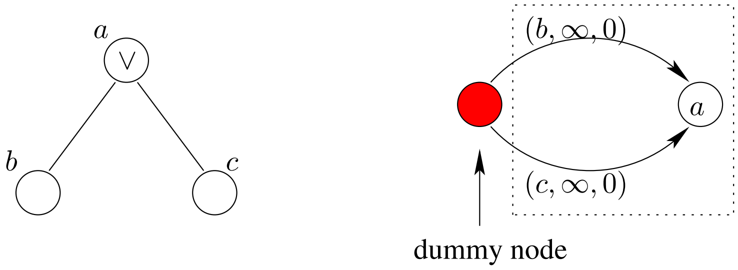

Figure 4: OR component

assume $p \leq m$.

For the forward direction, assume ϕ has a truth assignment of cost p. Let X be the set of TRUE variables in this assignment. The attack in G_ϕ begins by getting all keys corresponding to variables in X with cost $p = |X|$. The internal nodes' keys can then be obtained with zero cost since ϕ is satisfied by assigning all variables in X to be TRUE. The backward direction is similar. $\square$

Theorem 3.3. *For every positive constant $c < 1/2$, it is NP-hard to approximate the* MIN-HACK *problem to within a ratio of* $\Theta(g_c(n))$.

Proof. This follows directly from Lemma 3.2 and Theorem 3.1. $\square$

We now have an "upper bound" for the hierarchy of MMSA mentioned in relation (3):

$$\text{PCP} \ll \text{MMSA}_3 \ll \text{MLC} \ll \text{MMSA}_4 \ll \cdots \ll \text{MMSA}_i \ll \text{MIN-HACK} \tag{5}$$

Consequently, if MIN-HACK is approximable to within (some constant times) $g_c(n)$, then the hierarchy collapses after MMSA_4.

4 Reducing LABEL-COVER to MIN-HACK

In this section, we present a reduction from MLC_1 to MIN-HACK which preserves the approximation ratio. Since MLC_1 was shown to be not approximable to within $g_c(n)$ for any $c < 1/2$ (unless P = NP) [7], this reduction gives another proof of the hardness result for MIN-HACK.

Consider an instance of MLC_1 consisting of a bipartite graph $G = (U \cup V, E)$, label sets B_1, B_2, and relation $\Pi_e \subseteq B_1 \times B_2$ for each edge $e \in E$. We shall construct in polynomial time an instance of MIN-HACK

$$G_{mh} = (V_{mh}, E_{mh}; \mathcal{K}, V_s, V_t, \kappa, \delta)$$

such that there is a complete covering for G_{mh} of cost at most c if and only if there is a feasible attack on G_{key} of cost at most c, for any positive integer c. Let $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_m\}$, $m_1 = |B_1|$, and $m_2 = |B_2|$.

Note that MLC$_1$ has the L_1-cost, i.e. the cost of a labeling is the total number of labels at all vertices in U, counting multiplicities. Without loss of generality, we assume that G_{mh} is feasible, namely the labeling (f_1, f_2) where $f_1(u) = B_1$, for all $u \in U$, and

$$f_2(v) \in \bigcap_{(u,v) \in E} \{b_2 \mid (b_1, b_2)\}, \ \ \forall v \in V.$$

This trivial labeling has cost mm_1.

The construction of G_{mh} consists of two main components (see Figure 5): (i) the *label acquisition*

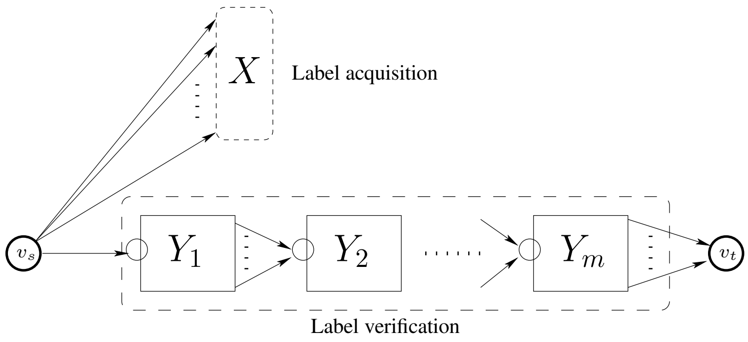

Figure 5: Overview of the reduction from MLC$_1$ to MIN-HACK

component corresponds to assigning labels to the vertices and paying the corresponding costs, and (ii) the *label verification* component corresponds to asserting that the labeling is valid.

The initial compromised vertex is v_s, i.e. $V_s = \{v_s\}$, and the target vertex is v_t, i.e. $V_t = \{v_t\}$. Abusing notation, we shall also use the name of a node to denote the key it possesses. (Nodes in our reduction will have different keys.)

We first define the vertices in the label acquisition component (see Figure 6). There is a vertex

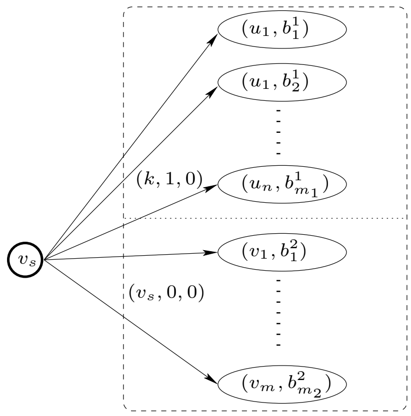

Figure 6: The label acquisition component

(u_i, b_j^1) (whose key is (u_i, b_j^1)) for each $u_i \in U$, $b_j^1 \in B_1$, and (v_i, b_j^2) for each $v_i \in V$, and for each $b_j^2 \in B_2$. The idea is that, if vertex (w, b) is visited in an attack, then b is in the set of labels for w, where $w \in U \cup V$, and $b \in B_1 \cup B_2$. The edges in the label acquisition component are all of the form $(v_s, (w, b))$, where $w \in U \cup V$ and $b \in B_1 \cup B_2$. For each $v_i \in V$, the edges $e = (v_s, (v_i, b_j^2))$ have $\delta(e) = (v_s, 0, 0)$, namely we can assign any label b_j^2 to v_i for "free." On the other hand, for each $u_i \in U$, the edge $e = (v_s, (u_i, b_j^1))$ has $\delta(e) = (k, 1, 0)$, where k is some elusive key no vertex possesses. The idea is that adding a label b_j^1 to the set of labels of a vertex u_i imposes a cost of 1.

With the verification component we aim to verify that, for every $v \in V$, we have picked up some label $b_2 \in B_2$ for v such that, for every edge $(u, v) \in E$, $\Pi_{(u,v)}$ contains (b_1, b_2) for some $b_1 \in B_1$ for which we do have key (u, b_1), namely b_1 was chosen to be in the label set of u. Thus, the verification component consists of a series of smaller components Y_i, each of which is meant to verify the above fact for v_i.

For any $v_j \in V$, the component Y_j consists of the following vertices (see Figure 7):

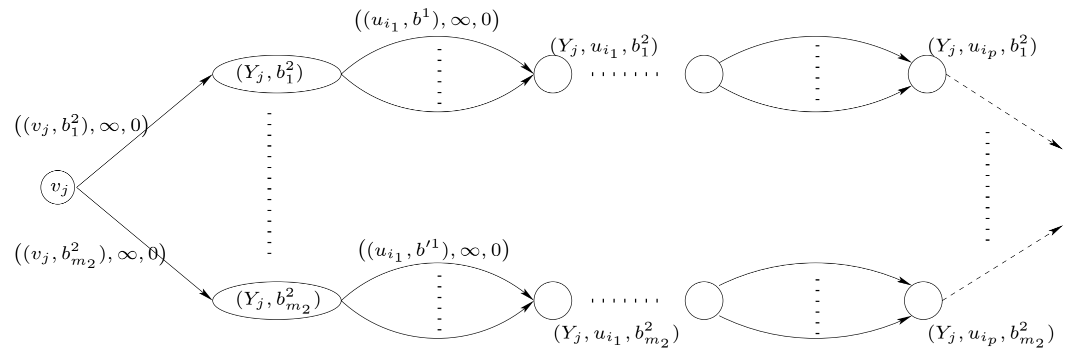

Figure 7: The label verification component Y_j

(i) the starting vertex v_j to enter the component,

(ii) a vertex (Y_j, b_l^2) for every label $b_l^2 \in B_2$, and

(iii) a vertex (Y_j, u_i, b_l^1) for every label $b_l^1 \in B_1$ and every $u_i \in U$ such that $(u_i, v_j) \in E$.

The edge set for component Y_j is defined as follows. There is an edge $(v_j, (Y_j, b_l^2))$ for every label $b_l^2 \in B_2$, where

$$\delta((v_j, (Y_j, b_l^2))) = \left((v_j, b_l^2), mm_1 + 1, 0\right).$$

The idea is that, in order to visit (Y_j, b_l^2) we must have picked up a label b_l^2 for vertex v_j, otherwise we would be paying a very high cost. In the picture, we use ∞ to denote $mm_1 + 1$.

Let $\Gamma(v) = \{u_{i_1}, \ldots, u_{i_p}\}$ be the set of neighbors of v_j in G. There is an edge in Y_j of the form $((Y_j, b_l^2), (Y_j, u_{i_1}, b^1))$ for every label b^1 such that $(b^1, b_l^2) \in \Pi_{(u_{i_1}, v_j)}$. Thus, there are as many copies of this edge as there are such b^1. The copy of the edge corresponding to b^1 has key challenge (u_{i_1}, b^1); otherwise, a price of ∞ is to be paid. The idea is that, in order to walk from (Y_j, b_l^2) to (Y_j, u_{i_1}, b_l^2), we need to have one of the keys (u_{i_1}, b^1), which means the labeling has covered the edge $(u_{i_1}, v_j) \in E$. If it so happens that there is no such b^1, we put between (Y_j, b_l^2) and $(Y_j, u_{i_1}, b^1))$ an edge with the elusive key challenge k, and the "infinity" cost $mm_1 + 1$. That completes the verification that edge (u_{i_1}, v_j) is covered. We need to also check that $(u_{i_2}, v_j), \ldots, (u_{i_p}, v_j)$ are covered. This is done by serially connecting similar components, one for each $u_{i_2}, \ldots, u_{i_p}$.

This construction can clearly be done in polynomial time, and it is easy to verify that G has a complete covering of cost at most a if and only if G_{mh} has a feasible attack of cost at most a. Specifically, when $a \geq mm_1 + 1$ (the infinity cost) we use the trivial attack corresponding to the trivial cover which assigns B_1 to each vertex in U. When $a \leq mm_1$, we use the trivial correspondence between a complete labeling and a successful attack as laid out in the construction of the MIN-HACK instance.

5 Reducing PCP to MIN-HACK

In this section, we give a direct proof that MIN-HACK is not approximable to within $g_c(n)$ for any $c < 1/2$ by using a PCP characterization of NP with almost polynomially small error probability [6]. This PCP characterization can be summarized in the following theorem, which we quote from [7].

Theorem 5.1 (Dinur et al. [6])**.** *Let $c < 1/2$ be arbitrary and $D \leq \log\log^c n$. Let $\Psi = \{\psi_1, \ldots, \psi_n\}$ be a system of boolean constraints over variables $X = \{x_1, \ldots, x_m\}$ such that each constraint depends on D variables, and each variable takes values in a field $\mathcal{F}$ of size $O(2^{(\log n)^{1-1/O(D)}})$. Then, it is NP-hard to decide between the following two cases:*

Yes: There is an assignment to the variables such that all $\psi_1, \ldots, \psi_n$ are satisfied.

No: No assignment can satisfy more than $O(1)/|\mathcal{F}|$ fraction of the ψ_i.

The general strategy to prove a hardness result for MIN-HACK is to show that if the MIN-HACK is approximable to within a certain ratio, then it is possible to distinguish between the YES and the NO instances of the boolean constraint satisfaction problem mentioned in Theorem 5.1. The idea is to find a "gap-preserving" reduction from Ψ to an instance of MIN-HACK. We shall follow the line of Dinur and Safra [7]. The following reduction shows that the three problems MLC$_p$, MMSA, and MIN-HACK are closely related.

Suppose we are given an instance $\Psi = \{\psi_1, \ldots, \psi_n\}$ as in Theorem 5.1 with m variables $X = \{x_1, \ldots, x_m\}$. For each $x \in X$, let Ψ_x denote the set of all constraints ψ which depend on x. For each constraint ψ_j, let $x_{j_1}, \ldots, x_{j_D}$ denote all the variables that ψ_j depends on.

An attack graph G_{mh} can be constructed with more or less the same format as the reduction from MLC. Figures 8 and 9 are almost self-explanatory. The graph G_{mh} has v_s as the initially compromised

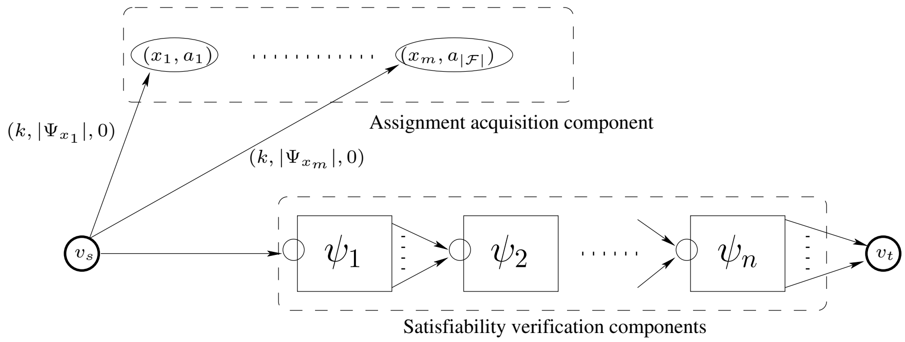

Figure 8: Overview of the reduction from PCP to MIN-HACK

vertex, v_t the target vertex, and two types of components: the *assignment acquisition component* and a series of *satisfiability verification components*, one for each ψ_j, $1 \leq j \leq n$. (The edges or nodes with no labels are dummy edges and nodes, with dummy keys and costs zeros.)

The assignment acquisition component consists of vertices of the form (x_i, a_l), for all $x_i \in X$ and $a_l \in \mathcal{F}$. As usual, we use nodes' labels to also denote the keys the nodes have. To get the key (x_i, a_l), the attacker has to pay the price of $|\Psi_{x_i}|$.

In the satisfiability verification component for ψ_j (see Figure 9), there are a number of parallel paths, one for each assignment $r : X \to \mathcal{F}$ which satisfies ψ_j. In order to get through this verification component for ψ_j, the attacker needs to have at least one complete set of keys $(x_{j_1}, r(x_{j_1}), \ldots, (x_{j_D}, r(x_{j_D})$ for some assignment r that satisfies ψ_j.

Lemma 5.2 (Completeness)**.** *If Ψ is satisfiable, then there is a successful attack on G_{mh} with cost nD.*

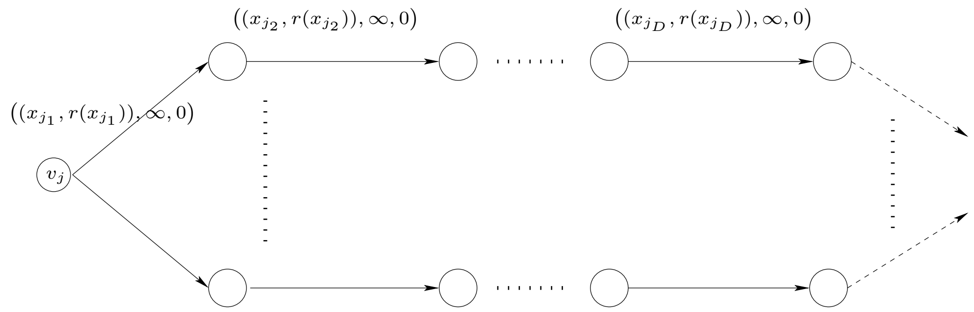

Figure 9: The satisfiability verification component for ψ_j

Proof. Let r be some assignment which satisfies Ψ. A successful attack can be constructed by first grabbing all the keys $(x_i, r(x_i))$, $i = 1, \ldots, m$. The total cost of getting these keys is $\sum_{i=1}^{m} |\Psi_{x_i}| = nD$, since each constraint is dependent on D variables.

To get to v_t, the attacker can then get through the components of the ψ_j by following the path corresponding to r in each ψ_j. $\square$

Lemma 5.3 (Soundness)**.** *Consider any $c < 1/2$, and let $g = g_c(n)$. If there is a successful attack on G_{mh} with cost at most gnD, then there is an assignment satisfying a $1/(2(2Dg)^D)$ fraction of constraints in Ψ.*

Proof. For each $x \in X$, let

$$A(x) = \{a \in \mathcal{F} \mid \text{the attacker visited node } (x, a)\}.$$

Note that visiting (x, a) incurs a cost of $|\Psi_x|$. Hence, the cost of the entire attack is

$$\sum_{x \in X} |\Psi_x||A(x)| \leq gnD. \tag{6}$$

Consider the probability distribution on X where every elements of X are chosen by first uniformly choose a constraint ψ at random, and then choose a variable x that ψ depends on at random. The probability of picking a particular x is $\frac{|\Psi_x|}{n}\frac{1}{D} = \frac{|\Psi_x|}{nD}$. Hence, relation (6) implies

$$\mathrm{E}_x[|A(x)|] = \sum_{x \in X} \frac{|\Psi_x|}{nD}|A(x)| \leq g.$$

Call a variable x *bad* if $|A(x)| > 2Dg$. By Markov inequality,

$$\operatorname*{Prob}_x[|A(x)| > 2Dg] \leq \operatorname*{Prob}_x[|A(x)| > 2D\mathrm{E}_x[|A(x)|]] < \frac{1}{2D}.$$

In other words, the probability of hitting a bad variable in this distribution is at most $1/(2D)$. We thus have

$$\begin{aligned}
\frac{1}{2D} &> \operatorname*{Prob}_{\psi \in \Psi, x \in \psi}[x \text{ is bad}] \\
&= \operatorname*{Prob}_{\psi \in \Psi}[\psi \text{ contains a bad variable}] \cdot \operatorname*{Prob}_{x \in \psi}[x \text{ is bad} \mid \psi \text{ contains a bad variable}] \\
&\geq \operatorname*{Prob}_{\psi \in \Psi}[\psi \text{ contains a bad variable}]\frac{1}{D}
\end{aligned}$$

Consequently, at least half of the ψ contains no bad variable.

To this end, define a random assignment $\bar{r} : X \to \mathcal{F}$ by assigning to x some $\bar{r}(x) \in A(x)$ uniformly. Since the attack was successful, for each ψ_j the attacker must have gotten through one of the parallel paths in the verification component for ψ_j corresponding to some assignment r that satisfies ψ_j. The probability that $r(x) = \bar{r}(x)$ for all $x \in \psi_j$ is $\prod_{x \in \psi_j} \frac{1}{|A(x)|}$, which is at least $1/(2Dg)^D$ for the ψ_j which do not contain bad variables. Combined with the fact that at least half of the ψ_j do not contain bad variables, we conclude that there is some assignment that satisfies a $1/(2(2Dg)^D)$ fraction of Ψ as desired. □

A PCP proof of Theorem 3.3. For any $c < 1/2$, we want to show that there is no $g_c(n)$ approximation for MIN-HACK, unless P=NP. Pick any c' such that $c < c' < 1/2$. Pick D and $\mathcal{F}$ in Theorem 5.1 such that $D = O(\log \log^{c'} n)$ and $|\mathcal{F}| = \Theta(g_{c'}(n))$. With these parameters, it is easy to see that $1/(2(2Dg)^D) > O(1)/|\mathcal{F}|$.

Consider the construction of G_{mh} described above. We will show that, if there is a g-approximation algorithm for MIN-HACK, then the algorithm can also be used to decide the YES and NO instances of the constraint satisfaction problem.

Given an instance Ψ of the constraint satisfaction problem. The strategy is to run the g-approximation algorithm on the instance G_{mh} constructed from g and report YES iff the answer is at most gnD. Clearly, if Ψ is a yes-instance, then the answer is at most gnD because, by the completeness lemma the optimal solution is at most nD. On the other hand, by the soundness lemma the approximation algorithm returns an answer at most gnD only when a fraction of $> O(1)/|\mathcal{F}|$ constraints of Ψ are satisfied, implying Ψ is a yes instance. □

6 Analysis of some heuristics for MIN-HACK

In this section, we analyze two variations of Dijkstra's like heuristics for MIN-HACK, which are being implemented and evaluated as part of our research project. To simplify presentation, we shall assume that there is only one initial compromised node (i.e. $V_s = \{s\}$) and one single target (i.e. $V_t = \{t\}$). The lower bounds on the approximation ratios hold for the general case, nevertheless.

The first obvious idea is to mimic Dijkstra's algorithm for the s-t shortest path problem. We start with a set $S = \{s\}$, and keep adding into S the cheapest possible vertex we can reach (using both the cost function and the combination of keys we get so far). In the process, each newly added vertex v has information about which sequence of vertices was used to get to v. We use VER-SEQ(v) to denote the sequence of vertices leading to v. Initially, VER-SEQ(v) = NIL (the empty sequence) for all vertices. Let cost(v) denote the current estimated cost of getting to v, all of which are ∞ except for the cost of s being 0. After **all** vertices are reached, we take the sequence leading to t as the final answer. Note that we cannot stop after t is reached, because there might be a cheaper sequence to t if we are willing to go further. The pseudo code for this idea is as follows.

DIJKSTRA-BASED HEURISTIC

$S \leftarrow \emptyset$, cost$(s) \leftarrow 0$
cost$(v) \leftarrow \infty$ for all $v \in V - \{s\}$
VER-SEQ$(v) \leftarrow$ NIL for all $v \in V$
while $|S| \neq |V|$ **do**
  Choose v from $V - S$ with smallest cost(v)
  $S \leftarrow S \cup \{v\}$
  Let C be the collection of keys in VER-SEQ$(v) \cup \{v\}$
  for each $u \in V - \{v\}$ such that $(v, u) \in E$ **do**
    Let $(k, c_1, c_2) \leftarrow \delta(v, u)$

if $u \notin$ VER-SEQ(v) and $k \in C$ and cost$(v) + c_2 <$ cost(u) **then**
cost$(u) \leftarrow$ cost$(v) + c_2$
VER-SEQ$(u) \leftarrow$ [VER-SEQ$(v), v$] // adjoining v at the end of VER-SEQ(v)
end if
if $u \notin$ VER-SEQ(v) and $k \notin C$ and cost$(v) + c_1 <$ cost(u) **then**
cost$(u) \leftarrow$ cost$(v) + c_1$
VER-SEQ$(u) \leftarrow$ [VER-SEQ$(v), v$]
end if
end for
end while
Return VER-SEQ(t)

Proposition 6.1. *The algorithm above does not approximate the optimal solution to within any ratio.*

Proof. Consider a graph G consisting of a directed path $s, v_2, \dots, v_{n-2}, t$, and another directed edge (s, v_1). All edges from on the path has c_1-cost $a \in \mathbb{Z}^+$, c_2-cost zero, and key challenge $\kappa(v_1)$. The edge (s, v_1) has c_2-cost equal to 0 and key challenge $\kappa(s)$. It is obvious that the optimal solution has cost 0, while the algorithm above returns the sequence $[s, v_2, \dots, v_{n-2}, t]$ of cost $(n-1)a = (n-1)2^{\lg a}$. □

The problem is clear. The sequence VER-SEQ(v) does not allow us to take advantage of the fact that we might be able to wander away from v and come back to v with zero cost. This idea can easily be incorporated into the algorithm, namely each time we add in a new vertex v into S, we also check around v to see if any vertex can be added for free. If this is the case, we can modify VER-SEQ(v) accordingly. Thus, the sequence VER-SEQ(v) may include v more than once. Let us call this strategy MODIFIED DIJKSTRA HEURISTIC-1.

Proposition 6.2. *The algorithm* MODIFIED DIJKSTRA HEURISTIC-1 *has at least an exponential approximation ratio.*

Proof. This modified heuristic will be able to detect a cost 0 attack to t. However, it will not be able to detect that there might be a small cost "sacrifice" we can make.

Consider the example build as in the previous proposition. The only difference is that the c_2-cost of (s, v_1) is now 1. The same solution is still returned. The input size of the problem is $\Theta(n + \lg a)$. The ratio between the solution returned by the new heuristic and the optimal solution ($(n-1)2^{\lg a}$) is exponential in terms of the input size. □

One may be asking, "what if we look around v up to a radius k and keep all possible combinations, including the ones with cost greater than 0?" Since there are up to $k!\binom{n}{k} = n(n-1)\dots(n-k+1)$ combinations of ways to get from v, the radius k can only be a constant for the algorithm to have polynomial running time. The same conclusion holds about the exponential approximation ratio. We simply have put vertex v_1 in the example above at a distance $k+1$ from s.

References

[1] M. ALEKHNOVICH, S. BUSS, S. MORAN, AND T. PITASSI, *Minimum propositional proof length is NP-hard to linearly approximate*, J. Symbolic Logic, 66 (2001), pp. 171–191.

[2] S. ARORA, L. BABAI, J. STERN, AND Z. SWEEDYK, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, J. Comput. System Sci., 54 (1997), pp. 317–331. 34th Annual Symposium on Foundations of Computer Science (Palo Alto, CA, 1993).

[3] S. Arora and C. Lund, *Hardness of approximation*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS Publishing Company, 1997, pp. 399–346.

[4] R. Chinchani, A. Iyer, H. Q. Ngo, and S. Upadhyaya, *Towards a theory of insider threat assessment*, in Proceedings of the International Conference on Dependable Systems and Networks (DSN 2005, Yokohama, Japan), IEEE, 2005.

[5] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan, *Structure in approximation classes*, SIAM J. Comput., 28 (1999), pp. 1759–1782 (electronic).

[6] I. Dinur, E. Fischer, G. Kindler, R. Raz, and S. Safra, *PCP characterizations of NP: towards a polynomially-small error-probability*, in Annual ACM Symposium on Theory of Computing (Atlanta, GA, 1999), ACM, New York, 1999, pp. 29–40 (electronic).

[7] I. Dinur and S. Safra, *On the hardness of approximating label-cover*, Inform. Process. Lett., 89 (2004), pp. 247–254.

[8] D. S. Hochbaum, ed., *Approximation Algorithms for NP Hard Problems*, PWS Publishing Company, Boston, MA, 1997.

[9] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. Assoc. Comput. Mach., 41 (1994), pp. 960–981.

[10] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley Publishing Company, Reading, MA, 1994.

[11] V. V. Vazirani, *Approximation algorithms*, Springer-Verlag, Berlin, 2001.