Adaptive Algorithms for Detecting Critical Links and Nodes in Dynamic Networks

Yilin Shen, Thang N. Dinh, My T. Thai
CISE Department, University of Florida, Gainesville, FL, 32611
Email: {yshen, tdinh, mythai}@cise.ufl.edu

***Abstract*—The assessment of network vulnerability is of great importance in the presence of unexpected disruptive events or adversarial attacks targeting on critical network links and nodes. However, it is extremely challenging to seek and safeguard against most destructive scenarios in dynamic networks where changes to their topologies are frequently introduced. In this paper, we propose CLA and CNA algorithms, to adaptively detect critical links and nodes in a dynamic network whose removals maximally destroy the network's functions, without recomputing from scratch. The effectiveness of our solutions is validated on various types of networks with different topology structures.**

I. Introduction

The rapid and exceptional growth of mobile networks has called for an accurate and continuous assessment of network vulnerability in the mobile environment, i.e., how much the network performance reduces in various cases of undesired disruptions, such as natural disasters, unexpected elements failures, or especially adversarial attacks. In a typical attacking point of view, an attacker would first exploit the network weaknesses, and then only need to update the targeted critical nodes or links with the change of network structure, whose corruptions bring the whole network down to its knees. For instance, an adversarial attack to any essential Internet hosts or clients, once successful, may cause tremendous breakdowns to millions of companies' websites and online services. Yet, the destruction to the network by using the same attacking scheme might be tremendously reduced due to the mobility and evolution of the network. Therefore, in order to continuously maintain the normal network functions at each time-slot in dynamic circumstances, it is of great importance to adaptively explore the network vulnerability, i.e., identify and update the proper crucial nodes and links in a timely manner.

There have been many studies proposing different metrics to account for the network vulnerability [2], [3], [10], [11], among which the degree of suspected nodes or edges [3], the average shortest path length [2], the global clustering coefficients [10], and the available number of compromised $s-t$ flows [11] appear to be the most popular and effective. Unfortunately, these mentioned measures do not seem to cast well for some particular kinds of network vulnerabilities, especially when network is dynamic and its fragmentation is of high priority, as depicted in Figure 1.

Let us consider a simple example in Figure 1 illustrating a small portion of the Internet, where nodes $v_1, v_2, \ldots, v_7$ are ISPs and the rest are consumers or transmission nodes. As revealed in this figure, at time-slot 0, any successful corruptive attacks to nodes v_4 and v_6 are sufficient to bring the whole network down to its knees with no satisfied customers. When the network structure changes at the next time-slot without links (v_4, v_9) and (v_6, v_9), the attacks to v_8 and v_{10} can better destroy the network by allowing less pairs of nodes to communicate with each other, as well as reduce the cost by the avoidance of attacking ISPs. In a different attacking strategy, regardless of the change of network structure at different time-slots, the removal of node v_7 or v_{14}, if the adversary was to use maximum degree centrality as the metric, does not appear to harm the network function because all customers are still satisfied. However, these removals also reduce the global clustering coefficients to almost 0 and increase the average shortest path to nearly 3. Besides, if the attacker uses the available number of compromised $v_1 - v_2$ flows, the destructions of nodes v_4 and v_7 will drop the flow to 1, and they still, unfortunately, cannot destroy the existence of the giant ISP component providing services to the (almost) whole network.

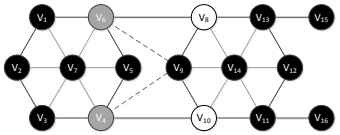


Fig. 1: At time-slot 0, since node v_4 and v_6 are connected to v_9 (dotted links), the corruptive attacks of v_4 and v_6 (grey nodes) can destroy the function of the whole network such that only less than 39% nodes connect each other. When the network structure changes at time-slot 1 without links (v_4, v_9) and (v_6, v_9), v_8 and v_{10} (white nodes) can be attacked to better destruct the network by only allowing 35% nodes to connect each other.

This example illustrates two important points that the other works are lack of: (1) In order to break down the network, we need to somehow control the balance among disconnected components while ensuring the nonexistence of giant components. One possible and effective way to do so is to measure the *total pairwise connectivity*, i.e., the number of connected node-pairs [7] in the network. Back to our example, at time-slot 1, a scrutiny look into the destructions of nodes v_8 and v_{10}, which we know can break down the network's function, reveals that they, indeed, reduce the network total pairwise connectivity to its greatest extent (35%). This great reduction, as a result, significantly leads to the malfunction of the whole network. (2) The set of critical links and nodes to destroy the network can be entirely different even the network structure only changes a little bit, especially using this effective new metric, total pairwise connectivity. As illustrated in our example, the set of critical nodes becomes v_8 and v_{10} at time-slot 1 although only two links are removed from the network. Instead of v_4 and v_6, the attacks of the two critical nodes can not only reduce the pairwise connectivity by 4% (from 39% to 35%), but also reduce the cost due to the avoidance of ISPs.

Motivated by the effectiveness of the total pairwise connectivity metric in dynamic networks, we study the problems to detect and update a given number of network elements (links or nodes) all the time in order to assess the network

vulnerability by minimizing the total pairwise connectivity. A naive solution to these problems would try to repeatedly execute one of the available static methods [4], [13] to find new critical elements whenever the network changes. However, this strategy suffers from the major disadvantage, i.e., the huge consumption of time and computing resources, which delays the protection and allows the intruder to attack and destroy the network functions. In this paper, we propose two novel adaptive algorithms, *Critical Link Adaption* (CLA) and *Critical Node Adaption* (CNA) algorithms, to detect and update the critical elements in a timely manner, by investigating the underlying relations between the change of network structure and the network connectivity. Our CLA and CNA algorithms are validated on a wide range of networks with different scales and topologies, in comparison to the most effective approach to detect critical links and nodes in static networks.

Organizations: Section II introduces the network model and problem definition. We propose two novel adaptive algorithms, CLA and CLA, in Section III and the experimental results are illustrated in Section IV to validate the performance of these two algorithms. Related work is presented in section V and we conclude the whole paper in Section VI.

II. Network Model and Problem Definition

Network Model: Let $G_0 = (V_0, E_0)$ be the initial network at time-slot 0, with $|V_0| = n_0$ nodes and $|E_0| = m_0$ links. Denote S_0 as the initial set of critical elements (links or nodes). Two nodes i and j in the network are connected iff there exists at least one path from i to j. Moreover, let ΔG_T be the changes of the whole network after time-slot T, in which ΔV_T and ΔE_T are the sets of new (or removed) nodes and links. Then, the network at time-slot $T+1$ can be described as $G_T = (V_T, E_T)$ as a combination of the previous one together with the change, i.e., $G_{T+1} = G_T \cup \Delta G_T$. Then, a *dynamic network* $\mathcal{G}$ is a sequence of network snapshots evolving all the time: $\mathcal{G} = (G_0, G_1, \ldots, G_T, \ldots)$.

Problem Definition: Given an integer k and a dynamic network $\mathcal{G}$ having the initial network G_0 and network snapshots $G_1, G_2, \ldots$ obtained through a collection of network topology changes $\Delta G_1, \Delta G_2, \ldots$ all the time. The problem asks for adaptive algorithms to efficiently detect and update the set of k critical elements (links or nodes), by only utilizing the set of critical elements S_{T-1} at the previous time-slot, such that the total pairwise connectivity is minimized.

III. Adaptive Algorithms in Dynamic Networks

In this section, we propose two different adaptive algorithms, *Critical Link Adaption (CLA)* and *Critical Node Adaption (CNA)*, to detect critical links and nodes by handling various scenarios in the change of networks. Our CLA and CNA algorithms consist of two phases: 1) identifying the initial critical elements at time-slot 0; 2) updating and tracing the critical elements in a dynamic network.

First, in order to obtain the initial critical elements in the initial network G_0, we introduce the Integer Linear Programming (ILP) formulations to detect the top k critical links and nodes, as depicts in (1) and (2) respectively, by minimizing the total pairwise connectivity.

$$\begin{aligned} \min \quad & \sum_{i,j \in V} u_{ij} \\ \text{s.t.} \quad & u_{ij} + u_{jh} - u_{hi} \le 1 \qquad \forall i,j,h \in V \\ & \textstyle\sum_{(i,j)\in E}(1 - u_{ij}) \le k \\ & u_{ij} \in \{0,1\} \end{aligned} \tag{1}$$

$$\begin{aligned} \min \quad & \sum_{i,j \in V} u_{ij} \\ \text{s.t.} \quad & u_{ij} + u_{jh} - u_{hi} \le 1 \qquad \forall i,j,h \in V \\ & v_i + v_j + u_{ij} \ge 1 \qquad \forall (i,j) \in E \\ & \textstyle\sum_{i \in V} v_i \le k \\ & v_i \in \{0,1\}, u_{ij} \in \{0,1\} \end{aligned} \tag{2}$$

where the indicator variable u_{ij} is defined as

$$u_{ij} = \begin{cases} 1, & \text{if nodes } i \text{ and } j \text{ are connected} \\ 0, & \text{otherwise} \end{cases}$$

and the other indicate variable v_i is further defined as

$$v_i = \begin{cases} 1, & \text{if node } i \text{ is deleted} \\ 0, & \text{otherwise} \end{cases}$$

These two ILPs share the same objective to minimize the total pairwise connectivity. The first constraint in both ILPs imposes the triangular connectivity. That is, if node i and j are connected, node j and h are connected, node i and h have to be connected. The last constraints in (1) and (2) ensure the number of all deleted links and nodes to be at most k respectively. In terms of critical nodes, the second constraint is added in (2) to guarantee that at least one endpoint of a link has to be deleted if its two endpoints are disconnected in the optimal solution. Then, since $\sum_{i,j \in V} u_{ij}$ can be minimized when there are more pairs of nodes (i,j) satisfying $u_{ij} = 0$, we have their relaxed Linear Programming (LP) with $u_{ij} \in [0,1]$, $v \in [0,1]$ instead. Then, based on these two LP formulations, the initial set of critical elements can be detected using the HILPR algorithm [13], which is based on the idea of iteratively solving the LP and rounding and has been shown to be the most effective approach in the literature.

Next, before introducing the second phase of the CLA and CNA algorithms, we first take a scrutiny look into the events, which update the network by introducing or removing a set of links or nodes. As one can see, these events can be further decomposed as a sequence of node (or link) insertions (or removals), in which a single node (or a single link) is introduced (or removed) at a time-slot. That is, during the time-slot interval T and $T+1$, changes to the network can be viewed as a collection of *four simpler events* as follows:

1) Link e insertion: $G_t = G_{t-1}[E_{t-1} \cup \{e\}]$;
2) Link e removal: $G_t = G_{t-1}[E_{t-1} \setminus \{e\}]$;
3) Node v insertion: $G_t = G_{t-1}[V_{t-1} \cup \{v\}, E \cup e(v)]$;
4) Node v removal: $G_t = G_{t-1}[V_{t-1} \setminus \{v\}, E \setminus e(v)]$.

where $e(v)$ is the set of links incident to v.

In the rest of this section, we first focus on the second phase of our CLA and CNA algorithms to update the critical elements based on the above four simpler events in Section III-A and III-B. Section III-C further presents the refinement of our CLA and CNA algorithms to improve their performance in terms of running time, and the analysis of time complexity.

A. Critical Link Adaption (CLA) Algorithm

The CLA algorithm, Algorithm 1, focuses on updating the critical links in dynamic networks by utilizing the set of critical links at the previous time-slot. First, let us exploit *a vital property* for critical links, as depicted in Fig. 2. As can be seen, we observe that the set of critical links S consists of *core critical links* S^c and *margin critical links* S^m, where S^c is the set of links whose removal will disconnect large dense components and S^m is the marginal links (the links connect nodes of degree 1). There could exist some S^m when the remaining costs $k - |S^c|$ is not sufficient to separate any dense components. In this case, the removal of such marginal links S^m can decrease the total pairwise connectivity to the greatest extent. However, these margin links S^m are not real critical links which should be avoided to be selected as critical links at next time-slot. Our adaptive algorithms also take this important point into account besides the relations between different connected components. Then, we consider the four simpler events respectively.

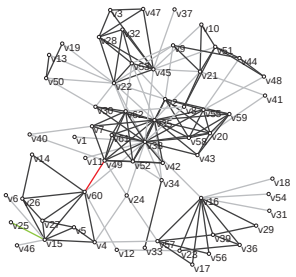


Fig. 2: Terrorist Network by Krebs with 50 critical edges (Gray). The removal of the gray critical edges separates the network into several dense components. The critical links (v_{49}, v_{60}) (red) and (v_{15}, v_{25}) (green) are representative of core critical links S^c and margin critical links S^m respectively.

1) Link Insertion: When a new link e arrives at time-slot t, we first define the *Single Link Fitness* (SLF) function $\mathcal{FL}_e$ to measure the contribution of the link e to the total pairwise connectivity in the graph G_{t-1}, as follows:

$$\mathcal{FL}_e(G_{t-1}) = \begin{cases} |C^{e^1}||C^{e^2}|, & C^{e^1} \neq C^{e^2} \\ 0, & \text{otherwise} \end{cases}$$

where C^{e^1} and C^{e^2} are the connected components which the two endpoints e^1 and e^2 of link e belong to respectively. Lemma 1 shows that the value of SLF function, called *SLF score*, can optimally reflect the contribution to total pairwise connectivity for a single link.

Lemma 1. *The removal of a link with maximum value of SLF function $\mathcal{FL}_e$ disconnects maximum number of node-pairs.*

Proof: Consider a link connecting the components C_i and C_j. The difference between the pairwise connectivity before and after removing e is

$$\binom{|C_i| + |C_j|}{2} - \binom{|C_i|}{2} - \binom{|C_j|}{2} = |C_i||C_j|$$

which is essentially the value of $\mathcal{FL}_e$. ∎

Then, we consider two cases: If the two endpoints of e belong to the same component in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$, the critical links keep the same. Otherwise, we calculate its SLF score and the SLF scores of all critical links in S_{t-1} on residual graph $G_{t-1}[E_{t-1} \setminus S_{t-1}]$. The new link e will be only selected as a new critical link if the original critical link $s \in S_{t-1}$ with minimum SLF score among all critical links in S_{t-1} has smaller SLF score than the new link e.

2) Link Removal: When a link e is removed, we only need to consider the case when the removed link e belongs to S_{t-1}. Otherwise, it will not greatly affect the functionality of the network. In this case, we will need to identify more critical links. The idea is to solve the relaxed LP of (1) on the residual graph $G_{t-1}[E_{t-1} \setminus S^c_{t-1} \cup \{e\}]$ with $k_t = |S^m_{t-1} \cup \{e\}|$ such that the margin critical links can be redetermined. The new set of critical links are the union of S^c_{t-1} and the k_t links having smallest u^*_{ij}.

3) Node Insertion: When a new node v is inserted, it is interesting to update the set of critical links since we need to consider both its incident links (new links) and the original critical links S_{t-1}. Let $e(v)$ be the set of links connecting to v. We first divide $e(v)$ into a collection of subsets $\mathcal{E}(v) = \{E_1(v), E_2(v), \ldots, E_d(v)\}$ which connects v to the d different connected components $C_1, \ldots, C_d$ in the residual graph $G_{t-1}[E_{t-1} \setminus S_{t-1}]$. That is, all endpoints in $E_i(v)$ incident to v belong to the same connect component in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$. Moreover, it is clear that $E_i(v) \subseteq e(v)$, $E_i(v) \cap E_j(v) = \emptyset$ for all $i \neq j$ and $\bigcup_i E_i(v) = e(v)$. Consider two connected components C_i and C_j in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$. We define *Pairwise Components Fitness* (PCF) function $\mathcal{FL}_{ij}$ in G_{t-1} to measure the contribution of the set of links between them to total pairwise connectivity, as follows:

$$\mathcal{FL}_{ij}(G_{t-1}) = \begin{cases} \frac{|C_i||C_j|}{|T_{ij}|}, & |T_{ij}| > 0 \\ 0, & \text{otherwise} \end{cases}$$

where T_{ij} is the set of links with one endpoint in component C_i and the other endpoint in C_j. Clearly, by minimizing the value of PCF function, called *PCF score*, we can use a small number of links to disconnect more node-pairs.

In the algorithm, the idea is to iteratively replace a subset of links incident to the new node v with other original critical links in S_{t-1} until no new links have higher PCF score than the original critical links. To do this, in each iteration, we consider using all links in some $E_i(v)$ to replace randomly $|E_i(v)|$ original critical links between some two connected components C_i and C_j having the minimum PCF score among all pairs of components in graph $G_{t-1}[E_{t-1} \setminus S_{t-1}]$ if the PCF score of $E_i(v)$ is larger than $\mathcal{FL}_{ij}$ and the size of T_{ij} is larger than $|E_i(v)|$, where T_{ij} is the critical links connecting connected components C_i and C_j. To calculate the PCF score of $E_i(v)$, we need to find out the two components connected by $E_i(v)$. Specifically, one component is clearly C_i and the other is the union of all other connected components incident to v, which is $\bigcup_{l=1,\ldots,i-1,i+1,\ldots,d} C_l$. In other words, the PCF score of $E_i(v)$ can be obtained on the graph $G_{t-1}[E_{t-1} \cup \{\bigcup_{l=1,\ldots,i-1,i+1,\ldots,d} E_l(v)\}]$.

4) Node Removal: When a node v is removed, we consider all links incident to node v. Let $e^s(v)$ be the set of links not only incident to v but also belonging to the set of original critical links S_{t-1}, i.e., $e^s(v) = N(v) \cap S_{t-1}$. Similar as the case of link removal, instead of only one removed link, we take into account the set $e^s(v)$ and solve the relaxed LP of (1) on the residual graph $G_{t-1}[V_{t-1} \setminus \{v\}, E_{t-1} \setminus S^c_{t-1}]$ with $k_t = |S^m_{t-1} \cup e^s(v)|$ such that the margin critical links can be

redetermined. The new set of critical links are the union of S^c_{t-1} and the k_t links having smallest u^*_{ij}.

Algorithm 1: CLA Algorithm

Input : Graph $G_{t-1} = (V_{t-1}, E_{t-1})$ at time-slot $t-1$, an integer k, the set of critical links S_{t-1} at time-slot $t-1$
Output: The set of critical links S_t at time-slot t

if *a link e is inserted* **then**
  if *two endpoints of e belong to different component in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$* **then**
    Calculate the SLF score of e and all critical links in S_{t-1} in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$;
    Replace e as a critical link with the link s with minimum SLF score in S_{t-1} if the SLF score of e is larger than that of s;
  end
end
if *a link e is removed* **then**
  if $e \in S_{t-1}$ **then**
    Solve the relaxed LP of (1) on the residual graph $G_{t-1}[E_{t-1} \setminus S^c_{t-1} \cup \{e\}]$ with $k_t = |S^m_{t-1} \cup \{e\}|$;
    $S_t \leftarrow$ the union of S^c_{t-1} and the k_t links having smallest u^*_{ij};
  end
end
if *a node v is inserted* **then**
  while $\exists$ *two connected components C_i and C_j such that $\mathcal{FL}_{ij}$ is minimum* **and** *the size of T_{ij} (number of links between C_i and C_j) is larger than $|E_i(v)|$* **do**
    Replace all links in some $E_i(v)$ as critical links with randomly $|E_i(v)|$ original critical links in T_{ij};
  end
end
if *a node v is removed* **then**
  $e^s(v) \leftarrow N(v) \cap S_{t-1}$;
  Solve the relaxed LP of (1) on the residual graph $G_{t-1}[V_{t-1} \setminus \{v\}, E_{t-1} \setminus S^c_{t-1}]$ with $k_t = |S^m_{t-1} \cup e^s(v)|$;
  $S_t \leftarrow$ the union of S^c_{t-1} and the k_t links having smallest u^*_{ij};
end

B. Critical Node Adaption (CNA) Algorithm

The CNA algorithm, Algorithm 2, focuses on updating the critical nodes in dynamic networks by utilizing the set of critical nodes at the previous time-slot. First, we consider two fitness functions $\mathcal{FN}_e$ and $\mathcal{FN}_v$ with respect to a link e and a node v in G_{t-1} to measure the contribution to the pairwise connectivity for a link e and a node v as follows:

• *Link Endpoint Fitness* (LEF) function $\mathcal{FN}_e$:

$$\mathcal{FN}_e(G_{t-1}) = \begin{cases} \max\{(|C^{e^1}|-1)|C^{e^2}|, & \\ |C^{e^1}|(|C^{e^2}|-1)\}, & C^{e^1} \neq C^{e^2} \\ |C^{e^1}|-1, & \text{otherwise} \end{cases}$$

where C^{e^1} and C^{e^2} are the connected components which the two endpoints e^1 and e^2 of link e belong to respectively. The underlying reason to define $\mathcal{FN}_e$ is two-fold: (1) similar as Lemma 1, when $C^{e^1} \neq C^{e^2}$, the contribution of one endpoint of the link e can be optimally reflected by $\mathcal{FN}_e$ since we consider the maximum between that of two endpoints. (2) when the link e belongs to one connected component, the contribution of either of its endpoint is exactly $|C^{e^1}|-1$.

• *Single Node Fitness* (SNF) function $\mathcal{FN}_v$:

$$\mathcal{FN}_v(G_{t-1}) = \begin{cases} \sum_{1\le i<j\le d(v)} |C_i||C_j| & \\ +\sum_{1\le i\le d(v)} |C_i|, & d(v) > 1 \\ |C|-1, & \text{otherwise} \end{cases}$$

where $d(v)$ is the number of connected components containing $N(v)$ in G_{t-1} after removing v. Then, Lemma 2 shows that the contribution of the node v to total pairwise connectivity using $\mathcal{FN}_v$.

Lemma 2. *The removal of a node with maximum value of SNF function $\mathcal{FN}_v$ disconnects maximum number of node-pairs.*

Proof: Consider a node connecting the components $C_1, \ldots, C_{d(v)}$, the difference between the pairwise connectivity before and after removing v is

$$\binom{\sum_{1\le i<j\le d(v)} |C_i|+1}{2} - \sum_{1\le i\le d(v)} \binom{|C_i|}{2} = \sum_{1\le i<j\le d(v)} |C_i||C_j| + \sum_{1\le i\le d(v)} |C_i|$$

which is essentially the value of $\mathcal{FN}_v$. ∎

Finally, the values of these two fitness functions are referred to as *LEF score* and *SNF score* respectively.

1) Link Insertion: When a link e arrives at time-slot t, we consider two cases: The critical links keep the same if the two endpoints of e belong to the same component in $G_{t-1}[V_{t-1} \setminus S_{t-1}]$ or at least one endpoint is the original critical node in S_{t-1}. Otherwise, we calculate the LEF score for the new link e. The endpoint of e determining the LEF score, i.e., e^1 if $(|C^{e^1}|-1)|C^{e^2}| > |C^{e^1}|(|C^{e^2}|-1)$ and e^2 otherwise, will be selected as a new critical node if the original critical node $s \in S_{t-1}$ with minimum SNF score among all critical nodes in S_{t-1} has smaller SNF score than the LEF score of e.

2) Link Removal: When a link e is removed, the set of critical nodes is kept if either of its endpoints is a critical node at time-slot $t-1$. Otherwise, we calculate the LEF score for the new link e on graph $G_{t-1}[V_{t-1} \setminus S_{t-1}]$ when none of these two endpoints is originally critical nodes, or we have only one non-critical node in time-slot $t-1$. Likewise, the endpoint of e determining the LEF score, i.e., e^1 if $(|C^{e^1}|-1)|C^{e^2}| > |C^{e^1}|(|C^{e^2}|-1)$ and e^2 otherwise, will be selected as a new critical node if the original critical node $s \in S_{t-1}$ with minimum SNF score among all critical nodes in S_{t-1} has smaller SNF score than the LEF score of e.

3) Node Insertion: When a node v is inserted, we calculate its SNF and the SNF scores of all original critical nodes in S_{t-1} on residual graph $G_{t-1}[V_{t-1} \setminus S_{t-1}]$. Similarly, the new node v will only be selected as a new critical node if the original critical node $s \in S_{t-1}$ with minimum SNF score among all critical nodes in S_{t-1} has smaller SNF score than that of the new node v.

4) Node Removal: When a node v is removed, we only consider the case that v is an original critical node at time-slot $t-1$. Otherwise, it will not greatly affect the functionality of the network. In this case, since the number of critical nodes reduces to $k-1$ at time-slot t, we will need to identify one more critical node. The idea is to solve the relaxed LP of (1) on the residual graph $G_{t-1}[V_{t-1} \setminus S_{t-1}]$ with $k_t = 1$. The new set of critical links are the union of S_{t-1} and the link having largest v^*_i.

C. Refinement in CLA / CNA and Time Complexity Analysis

Furthermore, we improve our CLA and CNA algorithms discussed above in Section III-A and III-B in terms of running time, and analyze their time complexity.

1) Running Time Refinement: We note that both of the LP formulations (1) and (2) have $O(n^3)$ constraints owing to the triangle inequality constraints. To improve the running time of our algorithm during solving the LP, we further apply the

Algorithm 2: CNA Algorithm

Input : Graph $G_{t-1} = (V_{t-1}, E_{t-1})$ at time-slot $t-1$, an integer k, the set of critical nodes S_{t-1} at time-slot $t-1$
Output: The set of critical nodes S_t at time-slot t

if *a link e is inserted* **then**
  if *two endpoints of e belong to different components in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$* **and** *Neither of them is in S_{t-1}* **then**
    Calculate the LEF score of e and SNF score of all critical nodes in S_{t-1} in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$;
    Replace the endpoint of e determining the LEF score as a critical node with the node s with minimum SNF score in S_{t-1} if the LEF score of e is larger than the SNF score of s;
  end
end
if *a link e is removed* **then**
  if *both of endpoints of e are not in S_{t-1}* **then**
    Calculate the LEF score of e and SNF score of all critical nodes in S_{t-1} in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$;
    Replace the endpoint of e determining the LEF score as a critical node with the node s with minimum SNF score in S_{t-1} if the LEF score of e is larger than the SNF score of s;
  end
end
if *a node v is inserted* **then**
  Calculate the SNF score of v and all critical nodes in S_{t-1} in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$;
  Replace v as a critical node with the node s with minimum SNF score in S_{t-1} if the SNF score of e is larger than the SNF score of s;
end
if *a node v is removed* **then**
  if $v \in S_{t-1}$ **then**
    Solve the relaxed LP of (2) on the residual graph $G_{t-1}[V_{t-1} \setminus S_{t-1}]$ with $k_t = 1$;
    $S_t \leftarrow$ the union of S_{t-1} and the link having largest v_i^*;
  end
end

constraint pruning technique in [6] to eliminate the inactive constraints. Particularly, in both LP (1) and (2), instead of using the triangle inequality constraints, we introduce the new LP_{nc} with the following alternative constraints:

$$u_{ij} + u_{jh} - u_{hi} \leq 1, \quad h \in N(i) \cup N(j)$$

Clearly, the number of total constraints in LP_{nc} can be substantially reduced to $O(mn)$, which depends on the number of links m. According to [6], LP_{nc} has the same objective value of the LP formulation (1) (or (2)).

2) Running Time Analysis of the second phase in CLA and CNA: Consider the link removal in CLA and the node removal in CLA and CNA. As discussed above, our algorithms only need to solve LP in the residual graph, which contains much less constraints due to the substantial decrease of the number of links m. In terms of the link insertion of CLA, the running time is $O(|\mathcal{C}|^2 \log |\mathcal{C}|)$ ($\mathcal{C}$ is the set of connected components in $G_{t-1}[E_{t-1} \setminus S_{t-1}]$), derived from the calculation and sorting of PCF scores for component-pairs. The running time of node insertion in CLA is $O(|\mathcal{C}|^2 \max\{d, \log |\mathcal{C}|\})$ due to the sorting of PCF scores and the comparison of them between $|\mathcal{E}(v)| = d$ component-pairs. The running time of link insertion, link removal and node insertion in CNA is $O(k \log k)$ due to the sorting of the SNF scores for k critical nodes.

IV. Performance Evaluation

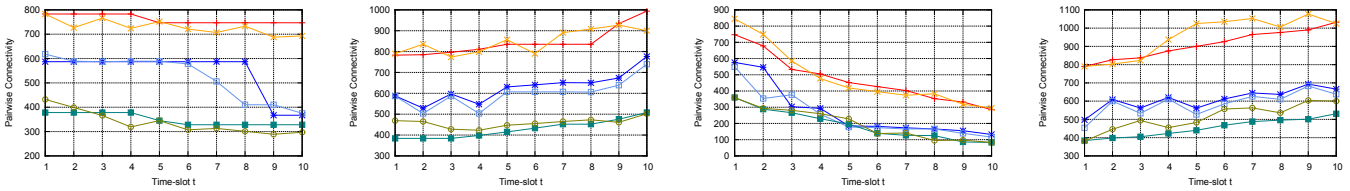
In this section, we evaluate the performance of our adaptive CLA and CNA algorithms by comparing them with HILPR algorithm [13] in each static snapshot, which has been shown to be the most effective to detect critical links and nodes in static networks. The simulation is implemented using the CPLEX optimization suite from ILOG, which includes the implementation of simplex method [8], a branch & bound algorithm and advanced cutting-plane techniques [14]. We test on several different network topologies, described as follows:

1) The real terrorist network compiled by Krebs [9] with 62 nodes and 153 links, which reflects the relationship between the terrorists involved in the terrorism attacks of Sep. 11, 2001. This experiment attempts to evaluate the performance of CLA and CNA on a real network.
2) Waxman network topology, a widely-accepted Internet model, is generated by the well-known BRITE [12].
3) Power-law network topology, generated by Barabási generator [1], has been discovered as one of the most remarkable properties in many large-scale networks.

In order to keep the similar density as the real terrorist network and also show the comparison with optimal solutions, we use the instance with 70 nodes and 140 links. We generate 100 instances for both Waxman and power-law models and show the average results. Moreover, at each time-slot, we randomly update the network by (1) randomly choose at most three existing links or nodes to remove; (2) randomly choose at most three non-existing links to insert; (3) insert at most three new nodes and randomly select at most three existing nodes to be its neighbors for each new node. Our experiment shows the average results of 20 testings in consecutive 10 time-slots.

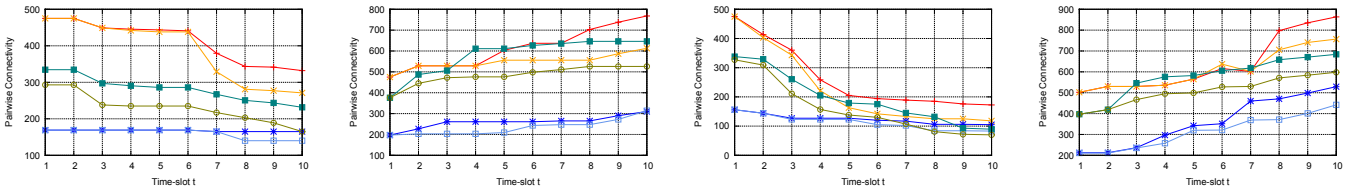
Fig. 3 reports the performance comparison between CLA and HILPR in all three networks. In Waxman network, power-law network and terrorist network, we select the number of critical links k to be 50, 30 and 30 respectively according to their different vulnerability tolerances. As revealed in Fig. 3(a) and Fig. 3(c), when a link or a node leaves the network, the gap between our CLA algorithm and HILPR algorithm is very slight (less than 5%) since the original core critical links usually keep the same when the network topology slightly changes, in the meanwhile, our CLA algorithm reconsiders all margin critical links at the previous time-slot. In the case of link and node insertions, as shown in Fig. 3(b) and Fig. 3(d), we are surprised to see that our CLA algorithm even outperform the static HILPR algorithm in Waxman network. The underlying reason is that the connected components in residual Waxman network are not dense enough such that the new incoming links or nodes are likely to make the dense components more ambiguous, which affects the solution quality of HILPR algorithm. The performance of CLA in terrorist networks and power-law networks is very close to HILPR algorithm since their heterogeneities usually allow them to have clear dense components in the residual networks.

Fig. 4 reports the performance comparison between CNA and HILPR in all three networks. Here, in the Waxman network, power-law network and terrorist network, we select the number of critical nodes k to be 15, 10 and 10 respectively, again according to their different vulnerability tolerances. Fig. 4(a) and Fig. 4(b) show that our CNA algorithm has a relatively worse performance (still less than 20% and 10% worse than HILPR) in Waxman networks because the Waxman network does not have clear dense subgraphs. In additional, our CNA algorithm performs much better on power-law networks and terrorist network due to the existence of large size dense connected components. Fig. 4(a) and Fig. 4(c) illustrate the performance of CNA in the case of node removal and node insertion. Note that there exists a small gap between the performance of CNA and HILPR algorithms after a certain time-slot, especially in Waxman networks. This is because some critical



(a) Link Removal (b) Link Insertion (c) Node Removal (d) Node Insertion

Fig. 3: The Performance Evaluation Of CLA against the HILPR Algorithm For Critical Link Detection



(a) Link Removal (b) Link Insertion (c) Node Removal (d) Node Insertion

Fig. 4: The Performance Evaluation Of CNA against the HILPR Algorithm For Critical Node Detection

nodes at previous time-slot become less important, i.e., the change of network topology makes them no longer connect big components.

Finally, the comparison of the running time between our adaptive algorithms and HILPR algorithm shows that time taken for solving the LP formulations in HILPR algorithm is much longer than our critical elements updating procedure and thus, may prevents this method to complete in a timely manner. In particular, at each time-slot, our CLA and CNA approaches take at most 20 seconds to complete all the tasks whereas HILPR [13] requires more than 3 minutes (even longer in Waxman networks) to solve the LP formulations iteratively k times. In that delay time, the network might be attacked by some intruders and thus, the solution may not be effective. These above experimental results confirm the robustness and efficiency of our approaches in dynamic networks.

V. Related Work

Many existing works on network vulnerability assessment mainly focus on the centrality measurements [5], including degree, betweenness and closeness centralities, average shortest path length [2], global clustering coefficients [10]. Due to the failures to assess the network vulnerability using above measurements, Arulselvan *et al.* [4] first proposed the total pairwise connectivity as an effective measurement and developed an effective heuristic algorithm. The β-disruptor problem was further defined by Dinh *et al.* [7] to find a minimum set of nodes or links whose removal degrades the total pairwise connectivity to a desired degree, along with pseudo-approximation algorithms. Later on, Shen *et al.* [13] further investigate the theoretical hardness detect both critical links and nodes on UDGs and PLGs as well as an effective HILPR algorithm. Unfortunately, these works only focus on detecting critical links and nodes in static networks, which are not efficient to apply onto dynamic networks.

VI. Conclusion

In this paper, we study the problems to detect and update critical links and nodes all the time for vulnerability assessment in dynamic networks based on the metric of total pairwise connectivity. We provided two adaptive algorithms, CLA and CNA, to investigate the relations between the change of network and its connectivity adaptively, along with the refinement technique. The experiments show the effectiveness of our algorithms on different network topologies in terms of both solution quality and time complexity.

Acknowledgement

This work is partially supported by NSF Career Award 0953284, DTRA, Young Investigator Award, Basic Research Program HDTRA1-09-1-0061.

References

[1] http://networkx.lanl.gov/reference/generated/networkx.generators.random_graphs.barabasi_albert_graph.html.
[2] R. Albert, I. Albert, and G. L. Nakarado. Structural vulnerability of the north american power grid. *Phys. Rev. E*, 69(2), Feb 2004.
[3] R. Albert, H. Jeong, and A. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.
[4] A. Arulselvan, C. W. Commander, L. Elefteriadou, and P. M. Pardalos. Detecting critical nodes in sparse graphs. *Comput. Oper. Res.*, 36:2193–2200, July 2009.
[5] S. P. Borgatti and M. G. Everett. A graph-theoretic perspective on centrality. *Social Networks*, 28(4):466 – 484, 2006.
[6] T. Dinh and M. Thai. Precise structural vulnerability assessment via mathematical programming. In *Milcom '11*, pages 1351 –1356, nov. 2011.
[7] T. N. Dinh, Y. Xuan, M. T. Thai, P. M. Pardalos, and T. Znati. On new approaches of assessing network vulnerability: hardness and approximation. *IEEE/ACM Trans. Netw.*, 20(2):609–619, Apr. 2012.
[8] F. S. Hillier and G. J. Lieberman. *Introduction to operations research, 4th ed.* Holden-Day, Inc., San Francisco, CA, USA, 1986.
[9] V. E. Krebs. Uncloaking terrorist networks. *First Monday*, 7(4), 2002.
[10] Luciano, F. Rodrigues, G. Travieso, and V. P. R. Boas. Characterization of complex networks: A survey of measurements. *Advances in Physics*, 56(1):167–242, Aug 2007.
[11] T. C. Matisziw and A. T. Murray. Modeling s-t path availability to support disaster vulnerability assessment of network infrastructure. *Computers & Operations Research*, 36(1):16 – 26, 2009. Part Special Issue: Operations Research Approaches for Disaster Recovery Planning.
[12] A. Medina, A. Lakhina, I. Matta, and J. W. Byers. BRITE: An approach to universal topology generation. In *MASCOTS*, page 346. IEEE Computer Society, 2001.
[13] Y. Shen, N. P. Nguyen, Y. Xuan, and M. T. Thai. On the discovery of critical links and nodes for assessing network vulnerability. *IEEE/ACM Trans. Netw.* Accepted with revision.
[14] L. A. Wolsey. *Integer Programming*. John Wiley, New York, 1998.