Connected Identifying Codes

Niloofar Fazlollahi, David Starobinski and Ari Trachtenberg
Dept. of Electrical and Computer Engineering
Boston University, Boston, MA 02215
Email: {nfazl,staro,trachten}@bu.edu

***Abstract*—We consider the problem of generating a *connected identifying code* for an arbitrary graph. After a brief motivation, we show that the decision problem regarding the existence of such a code is NP-complete, and we propose a novel polynomial-time approximation `ConnectID` that transforms *any* identifying code into a connected version of at most twice the size, thus leading to an asymptotically optimal approximation bound. When the input identifying code to `ConnectID` is robust to graph distortions, we show that the size of the resulting connected code is related to the best error-correcting code of a given minimum distance, permitting the use of known coding bounds. In addition, we show that the size of the input and output codes converge for increasing robustness, meaning that highly robust identifying codes are almost connected. Finally, we evaluate the performance of `ConnectID` on various random graphs. Simulations for Erdős-Rényi random graphs show that the connected codes generated are actually at most 25% larger than their unconnected counterparts, while simulations with robust input identifying codes confirm that robustness often provides connectivity for free.**

***Index Terms*—Identifying codes, localization, approximation algorithms, robustness, error correcting codes**

I. Introduction

An *identifying code* [3] for any given non-empty, connected graph $G = (V, E)$ is a subset $I \subseteq V$ of the vertices of the graph (called *codewords*) with the property that every vertex in the graph is adjacent to a *unique* and non-empty subset of I (known as the *identifying set* of the vertex). *Robust identifying codes* were introduced in [4] and proposed for applications to location detection in harsh environments, where the underlying graph topology may change because of addition or deletion of vertices or edges. An r-robust identifying code is thus one which remains an identifying code even if one adds or removes up to r vertices from *every* identifying set.

Identifying codes have been linked to a number of deeply researched theoretical foundations, including super-imposed codes [5], covering codes [3, 6], locating-dominating sets [7], and tilings [8–11]. They have also been generalized and used for detecting faults or failures in multi-processor systems [3], environmental monitoring [12, 13] and routing in networks [14].

Many of these applications actually assume some base connectivity between codewords implicitly requiring a connected identifying code, which we formally define in Section III-A. This issue is clearly seen in the application of identifying codes to RF-based localization in harsh environments [4, 15, 16]. In the method proposed in [4], sensors in a building are mapped to graph vertices, so that a pair of vertices is connected by an edge if the two corresponding physical sensors are within each other's communication range. Only a fraction of all sensors (those corresponding to codewords within the identifying code of the graph) are kept active while the rest can be put in energy-saving mode. A target is located by the unique pattern of sensors within its radio range.

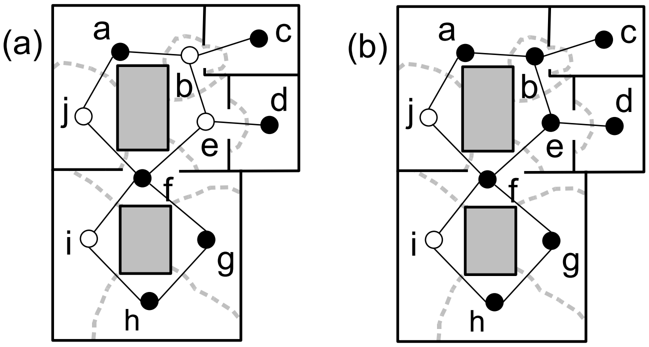

Fig. 1. (a) An example building floor plan and connectivity graph of sensors located at positions marked by circles. The filled circles represent codewords of an identifying code for the sensor network connectivity graph. The dashed lines show the boundaries of distinguishable regions based on the radio range of the active sensors. (b) Codewords of a connected identifying code for the same topology.

An example of an indoor floor plan and the graph corresponding to sensor placement and connectivity is depicted in Figure 1(a). Sensors that are within each other's radio communication range, like a and b, are connected by a graph edge (we assume connectivity between sensors is symmetrical), and filled circles a, c, d, f, g and h represent codewords of an identifying code for the sensor connectivity graph. Only the mentioned sensors actively monitor their surrounding for location detection, so that the location of a target placed at any of the regions marked by dashed lines can be uniquely determined based on the set of sensors that hear it. For instance, the set $\{\mathrm{a}, \mathrm{c}\}$ uniquely identifies the region surrounding position b.

In order to route data over a sensor network and transfer sensor data to a processor for location detection processing, one needs the network of active sensors to be *connected*, as shown in Figure 1(b). Yet, if one only activates sensors

Preliminary elements of the coding-theoretic aspects of this work were presented at ITA 2011 [1]. Preliminary applications of this work were presented in WCNC 2011 [2].

Copyright (c) 2011 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

This work was supported in part by the National Science Foundation under grants CCF-0729158, CCF-0916892, and CNS-1012910.

that correspond to codewords of an identifying code and deactivates the rest, there is no guarantee that one produces a connected network of active sensors. In fact, although there exist various algorithms in the literature for creating an identifying code for an arbitrary graph [4, 14, 17], none of these algorithms guarantee that the produced identifying code is connected.

Our approach focuses on building a connected (robust) identifying code out of an arbitrary given (robust) identifying code, with the goal of adding a minimum number of codewords to the original input identifying code and thereby keeping as many sensors as possible in energy-saving mode. We begin by proving that finding a connected identifying code is NP-complete, and by presenting a novel, efficient algorithm (called `ConnectID`) that produces a connected code of at most twice the cardinality of the arbitrary identifying code on which it is based. This translates to an asymptotically optimal $O(\log(|V|))$ approximation bound for our approach, when the original code is produced using the polynomial-time `rID` algorithm proposed in [14].

When the input to `ConnectID` is $r > 0$-robust, we show that the size of the resulting (connected) identifying code is upper bounded by the largest error-correcting codes of a given minimum Hamming distance. Moreover, the sizes of the input and output codes differ by a multiplicative factor of roughly $1 + \frac{1}{2r}$, meaning that they are asymptotically equal as robustness r increases. In other words, highly robust codes are almost connected, and this is confirmed by simulations on Erdős-Rényi random graphs.

This paper is organized as follows. We begin with a discussion of the related literature in Section II. In Section III we provide some background, including formal definitions of identifying codes in Section III-A and a brief review of existing algorithms for generating identifying codes in Section III-B. We then prove that the connected identifying code problem is NP-complete in Section IV. Section V presents our main algorithm `ConnectID`, starting with some models and notation in Section V-A, the core of our algorithm in Section V-B, some performance results in Section V-C, implementation details in Section V-D and complexity analysis in Section V-E. We present our numerical simulations in Section VI and conclude the paper in Section VII.

II. Related work

There is extensive theoretical work on identifying codes in the literature.

In [18, 19] identifying codes are proved to be NP-complete by reduction from the 3-satisfiability problem. Karpovsky et. al. [3] provide information theoretic lower bounds on the size of identifying codes over generic graphs and some specific graph topologies like meshes. The works in [5, 20–22] derive upper/lower bounds on size of the minimum identifying codes, with some providing graph constructions based on relating identifying codes to superimposed codes. The work in [22] focuses on random graphs, providing probabilistic conditions for existence together with bounds.

Many variants of identifying codes are defined and studied in the literature: a *robust* identifying code [4, 6] is resilient to changes in the underlying graph, a $(1, l \geq 0)$-identifying code [5, 20] uniquely identifies any subset of at most l vertices, a ρ-radius identifying code [3] uniquely identifies every vertex using the set of all codewords within distance ρ or less from the vertex, and a dynamic identifying code [6] is a walk whose vertices form an identifying code.

Identifying codes are also linked to superimposed codes [3, 5, 20–22], dominating sets [23], locating dominating sets [7], the set cover [13, 17] and the test cover problem [13, 17] and r-robust identifying codes are linked to error correcting codes with minimum Hamming distance $2r + 1$ [4] and the set r-multi-cover problem [17].

Of these, the locating dominating sets are closest in flavor to identifying codes, and indeed Suomela [23] links identifying codes and locating dominating sets to dominating sets and shows that it is possible to approximate both problems within a logarithmic factor, and that sub-logarithmic approximation ratios are intractable.

There is also considerable work regarding generation of dominating sets and connected dominating sets [24, 25], but these results do not apply directly to connected identifying codes, since not every dominating set is an identifying code. In other words, the optimal identifying code generally has larger cardinality than that of the optimal dominating set.

Finally, identifying codes are also proposed for various applications. The authors in [4] suggest application of identifying code theory for indoor location detection. They introduce robust identifying codes and also present a heuristic which creates robust identifying codes for an arbitrary graph. The work in [14] uses the same technique for indoor location detection, although the authors introduce a more efficient algorithm for generation of robust identifying codes. They also suggest an additional application of identifying codes for efficient sensor labeling for data routing in the underlying sensor network. Both references implicitly assume that a sensor network can route location detection data toward a sink, which is not satisfied in those sensor networks where only vertices corresponding to codewords are active. Since we will use the algorithms in [4, 14] for generating an identifying code, we will review their techniques in more detail in Section III-B.

The work in [12] studies the problem of sensor placement in a network which may be a water supply network or an air ventilation system with potential contamination source(s) such that the contamination source is identified under either of the following constraints:

- *sensor-constrained version* where the number of sensors is fixed and the identification time has to be minimized,
- *time-constrained version* where the identification time is limited and the number of sensors has to be minimized.

The latter version of this problem is shown to be a variant of the identifying code problem [13].

III. Background on identifying codes

We begin with a formal description of identifying codes in Section III-A, followed in Section III-B with a review of two existing algorithms for generating them.

A. Definitions

Consider a graph with vertex set $V \neq \emptyset$ and edge set $E \neq \emptyset$ (we shall make this non-empty assumption throughout the text). We categorize every vertex in V as either a *codeword* or a *non-codeword*, with the set of codewords denoted $I \subseteq V$. For every vertex v in V, the *identifying set* is the set of vertices in I that are adjacent to v (including v itself, if it is a codeword), and it is denoted by $S_I(v)$. If the identifying set for every vertex is unique and non-empty, then we call I an *identifying code*. Note that every superset of I is an identifying code for the same graph [4]. As a simple example, the identifying sets for vertices a, b, and f in Figure 1(a) are, respectively, $\{\mathrm{a}\}$, $\{\mathrm{a}, \mathrm{c}\}$, and $\{\mathrm{f}, \mathrm{g}\}$, all of which are different.

An identifying code I over a given graph G is said to be connected if there exists a simple path in G between any two vertices of I, wherein all the vertices on the path belong to I. The code is r*-robust* if it remains an identifying code after we arbitrarily add or remove up to r vertices in V to (or from) every identifying set, i.e., $S_I(u) \triangle V_1 \neq S_I(v) \triangle V_2$ for every $V_1, V_2 \subset V$ such that $|V_1|, |V_2| \leq r$. The operator $\triangle$ is the symmetric difference operator, meaning that $A \triangle B$ includes all elements that are either only in set A or only in set B for any given pair of sets A and B. The *minimum symmetric difference* of an identifying code I, $d_{\min}(I)$, is defined to be the minimum Hamming distance between every pair of identifying sets, i.e., $d_{\min}(I) = \min_{u,v \in V, u \neq v} |S_I(u) \triangle S_I(v)|$. It is shown in [4] that an identifying code I is r-robust if and only if $d_{\min}(I) \geq 2r+1$, and that every superset of an r-robust identifying code I is also an r-robust identifying code.

B. Existing algorithms

Next, we briefly review two existing polynomial-time algorithms that generate an identifying code (if one exists) for an arbitrary graph. We refer the reader to the cited references [4, 14] for further details.

Algorithm `ID-CODE` introduced in [4] initially selects all vertices V in the input graph to be codewords, and then checks, one by one, whether each vertex can be removed from the code without losing the identifying property. This greedy algorithm produces an *irreducible* code, meaning that no codeword can be removed from it while still keeping it an identifying code, and it can be modified to yield r-robust codes by changing the greedy criterion accordingly.

Algorithm `rID` presented in [14] initially calculates the identifying set of every vertex, assuming that all vertices are codewords. Then it associates with every vertex v in V the set of vertex pairs which distinguish v, i.e., one vertex in the pair is adjacent to v and the other is not. The algorithm iteratively forms an identifying code by selecting the vertex that distinguishes the most pairs to be a codeword. Using a similar approximation to the *set cover* problem [26], the authors in [14, 17] prove that `rID` achieves a logarithmic approximation ratio upper bounded by $c_1 \ln |V|$ and lower bounded by $c_2 \ln |V|$ for some constants $c_1 > c_2 > 0$. They also show that this bound is tight unless $\text{NP} \subset \text{DTIME}\left(|V|^{O(\log \log |V|)}\right)$ [27]. A robust version of `rID` is also presented in [14] using a reduction to the set multi-cover problem [26].

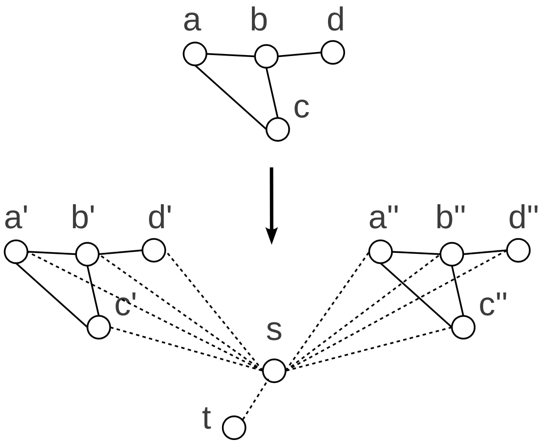

Fig. 2. Graph G with four vertices on top and constructed graph G^* with ten vertices. Vertex s connects vertices a', b', c' and d' in subgraph G' and vertices a'', b'', c'' and d'' in subgraph G'' by edges that are shown dashed.

IV. NP-completeness

Next we prove that deciding whether a connected identifying code with a certain number of codewords exists for any given graph is NP-complete.

Theorem 4.1: Given any non-empty graph G and an integer k, the decision problem of the existence of a connected identifying code with cardinality at most k in G is NP-complete.

Proof: We will prove the above statement with a polynomial-time reduction from the identifying code problem which is known to be NP-complete [18, 19, 28]. Specifically, we show that an identifying code with cardinality at most k exists in G if and only if there exists a connected identifying code with cardinality at most $2k+1$ in a specially generated graph G^*. In order to complete the proof, we need to show that any instance of a connected identifying code can be verified in polynomial time, a rather straightforward exercise that we omit.

Next, we explain our polynomial-time construction of the graph $G^*(V^*, E^*)$ from any non-empty graph $G(V, E)$. We begin by constructing two copies, $G'(V', E')$ and $G''(V'', E'')$, of G. The vertices of these graphs are connected through the isomorphic bijections $g' : V \to V'$ and $g'' : V \to V''$, having the property that $(u, v) \in E$ implies that $(g'(u), g'(v)) \in E'$ and $(g''(u), g''(v)) \in E''$. We combine G' and G'' with two new vertices s and t, the former connecting to all vertices V' and V'', and the latter connecting only to s. In other words, this new graph will be $G^*(V^*, E^*)$ with $V^* = V' \cup V'' \cup \{\mathrm{s}, \mathrm{t}\}$ and

$$E^* = E' \cup E'' \cup \{e_{\mathrm{s},v} | v \in V' \cup V''\} \cup \{e_{\mathrm{s},\mathrm{t}}\},$$

with $e_{i,j}$ denoting an edge between vertices i and j. Clearly the transformation from G to G^* is polynomial and takes $\Theta(|V| + |E|)$ time since $|V^*| = 2|V| + 2$ and $|E^*| = 2|E| + 2|V| + 1$. Figure 2 demonstrates our construction for a sample instance of G.

We next show that there exists an identifying code with cardinality $\leq k$ in G if and only if there exists a connected identifying code with cardinality at most $2k + 1$ in G^*.

$\Longrightarrow$. Assume we have an identifying code I with cardinality at most k over graph G. Define $I' \subseteq V'$ to be the image of I under the mapping g', i.e., $I' = \{g'(v)|\ \forall v \in I\}$, and similarly $I'' = \{g''(v)|\ \forall v \in I\}$. Then $I^* = I' \cup I'' \cup \{\mathrm{s}\}$ is clearly connected because s is connected to all vertices. Moreover, since I' is an identifying code for G', every vertex in V' has a unique identifying set, and similarly for I''; these sets are all different because V' and V'' have no common vertices and empty identifying sets are not allowed. Altogether then, I^* is a connected identifying code with cardinality at most $2k+1$.
$\Longleftarrow$. Assume that we have a connected identifying code with cardinality at most $2k+1$ over graph G^*. This identifying code must contain the vertex s; otherwise, either the code is disconnected or G' or G'' have no codewords, meaning that there is an empty identifying set. Removal of s will result in k codewords in each of G' and G'' or $< k$ codewords within one of G' or G'' (WLOG, assume it is within G'). Since s is connected to all vertices in G^*, no pair of vertices can be identified using s. Therefore, the resulting codewords within G' will necessarily correspond to an identifying code for G, unless it contains an empty identifying set.

The vertex t serves to ensure that every vertex in G has a non-empty identifying set. If t is not a codeword, then no other vertex in G' can have the same identifying set $\{\mathrm{s}\}$. Thus, every vertex in G' must have a codeword neighbor that is not s. If t is a codeword, then there must be less than k codewords in G'. In this case, there may be a single vertex v in G' with identifying set $\{\mathrm{s}\}$, but adding v to the codewords of G' will produce a non-empty identifying code of size not larger than k for G. ■

V. ALGORITHM ConnectID

We next present and analyze our polynomial-time approximation algorithm for connected identifying codes.

A. Model and notations

We assume an undirected, connected graph $G(V,E)$ (or G in short) where V is the set of vertices and E is the set of edges between the vertices. We consider $I \in V$ to be the set of codewords of an identifying code in G and a superset $I_{\mathrm{c}} \supseteq I$ to be the set of codewords of a derived connected identifying code in G. The *redundancy ratio* $R = |I_{\mathrm{c}}|/|I| \geq 1$ relates the two quantities.

We define a *component of connectivity* (or a component) C of I in graph G to be a maximal subset of I such that the subgraph of G induced by this subset is connected, i.e., the graph $G'(C, E \cap (C \times C))$ is connected and any codeword added to C renders it unconnected. For the example of Figure 1(a), we have $I = \{\mathrm{a,c,d,f,g,h}\}$ with components of connectivity $C_1 = \{\mathrm{a}\}$, $C_2 = \{\mathrm{c}\}$, $C_3 = \{\mathrm{d}\}$ and $C_4 = \{\mathrm{f,g,h}\}$.

A *plain path* between components C_1 and C_2 is an ordered subset of vertices in V that forms a path in G connecting a vertex $x_1 \in C_1$ to a vertex $x_2 \in C_2$, with x_1 and x_2 being the only codewords in the path. By distinction, a *path* may include any number of codewords or non-codewords. In Figure 1(a), $\{\mathrm{a,b,e,f}\}$ and $\{\mathrm{a,j,f}\}$ are the only plain paths between components C_1 and C_4. On the other hand, $\{\mathrm{a,j,f,e,d}\}$ is not a plain path between C_1 and C_3 because f is a codeword.

The *distance* between a given pair of components, say C_1 and C_2, is denoted $\mathtt{dist}(C_1, C_2)$ and is defined to be the number of edges on the shortest plain path between C_1 and C_2. If there is no plain path between C_1 and C_2, then $\mathtt{dist}(C_1, C_2) = \infty$. In Figure 1(a), $\mathtt{dist}(C_1, C_2) = 2$, $\mathtt{dist}(C_1, C_3) = 3$ and $\mathtt{dist}(C_1, C_4) = 2$.

B. Algorithm description

We present algorithm ConnectID in the format of a function which receives the set of codewords of an identifying code I for a given graph G and returns the set of codewords I_{c} of a connected identifying code. For sake of clarity, we first present algorithm ConnectID informally.

In the initialization phase, function $\mathtt{ConnectID}(G, I)$ partitions the identifying code I into a set of N distinct components of connectivity $\{C_1, C_2, ..., C_N\}$ where $1 \leq N \leq |I|$. Note that every pair of components is connected by some path in G because of the connectivity of G. Define C to be a set that stores the growing connected identifying code, arbitrarily initialized to the set of codewords in one of the components, say C_1. In addition, $\widehat{C}$ is the set that stores all components whose codewords are not yet included in C. Therefore, $\widehat{C}$ is initialized to $\{C_2, ..., C_N\}$.

At every iteration, the algorithm first updates the distance $\mathtt{dist}(C, C_j)$ between C and every component C_j in $\widehat{C}$ (Section V-E will describe how to do this efficiently). It then extracts from $\widehat{C}$ the component C^* with minimum $\mathtt{dist}(C, C^*)$ (breaking ties arbitrarily). The algorithm selects as codewords all vertices on the shortest plain path connecting C and C^*, denoted $\mathtt{path}^*(C, C^*)$, and unites the codewords in C and C^* and $\mathtt{path}^*(C, C^*)$ into C. After this step, the algorithm examines whether there are any other components in $\widehat{C}$ which become connected to C via the newly selected codewords on $\mathtt{path}^*(C, C^*)$. We define $\Gamma \subseteq \widehat{C}$ to be the set of such components. If Γ is non-empty, C is united with the components in Γ and the members in set Γ are removed from set $\widehat{C}$. The iteration above is repeated until $\widehat{C}$ becomes empty.

At termination, the algorithm returns the connected set $I_{\mathrm{c}} = C$, which, as a superset of I, is necessarily an identifying code.

Below, is a more formal presentation of algorithm $\mathtt{ConnectID}(G, I)$:

```
Algorithm ConnectID$(G, I)$:
Initialization:
  1)  Partition $I$ into a unique set of components of connectivity $\{C_1, C_2, ..., C_N\}$ where $1 \leq N \leq |I|$.
  2)  Set $\widehat{C} \leftarrow \{C_2, ..., C_N\}$.
  3)  Set $C \leftarrow C_1$.
Iteration:
  7)  While $\widehat{C}$ is not empty,
  8)      Update $\mathtt{dist}(C, C_j)$ and $\mathtt{path}(C, C_j)$ for every
          $C_j \in \widehat{C}$ and set $C^* \leftarrow \arg\min_{C_j \in \widehat{C}} \mathtt{dist}(C, C_j)$.
  9)      Extract component $C^*$ from $\widehat{C}$.
 10)      Set $C \leftarrow C \cup C^* \cup \mathtt{path}^*(C, C^*)$.
 11)      Find the set $\Gamma \subseteq \widehat{C}$ of components that are connected
          to $C$.
```

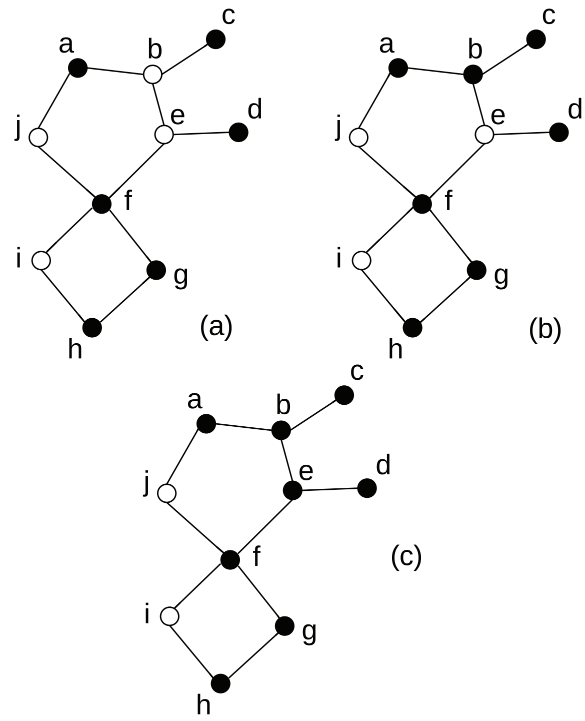

Fig. 3. Progress of ConnectID(G, I). The filled circles represent codewords of an identifying code I for the illustrated graph G. Initially, I is partitioned to components $C_1 = \{\mathrm{a}\}$, $C_2 = \{\mathrm{c}\}$, $C_3 = \{\mathrm{d}\}$ and $C_4 = \{\mathrm{f}, \mathrm{g}, \mathrm{h}\}$. We then set (a) $C = \{\mathrm{a}\}$ and $\widehat{C} = \{C_2, C_3, C_4\}$, (b) $C = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ and $\widehat{C} = \{C_3, C_4\}$, and (c) $C = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}, \mathrm{g}, \mathrm{h}\}$ and $\widehat{C} = \{\}$.

```
12)    If $\Gamma$ is not empty,
13)        For every component $C_j \in \Gamma$,
14)            Extract $C_j$ from $\widehat{C}$.
15)            Set $C \leftarrow C \cup C_j$.
16) Return $I_\mathrm{c} \leftarrow C$.
```

Example. Figure 3 shows the progress of ConnectID(G, I) after every iteration for the same graph and the same input identifying code as shown in Figure 1(a). The vertices in black are codewords. Assume that at initialization we have: $C_1 = \{\mathrm{a}\}$, $C_2 = \{\mathrm{c}\}$, $C_3 = \{\mathrm{d}\}$ and $C_4 = \{\mathrm{f}, \mathrm{g}, \mathrm{h}\}$. In Figure 3(a) we set $C = C_1$ and $\widehat{C} = \{C_2, C_3, C_4\}$. At first iteration, after we calculate the distance between C and all components in $\widehat{C}$ at line 8, we have: $\mathtt{dist}(C, C_2) = 2$, $\mathtt{dist}(C, C_3) = 3$, $\mathtt{dist}(C, C_4) = 2$. At line 9, we extract one component with minimum dist from $\widehat{C}$, which may be C_2 or C_4. Assume that we select C_2. Then, we unite C and C_2 and vertex b at line 10. Hence, $C = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ as illustrated in Figure 3(b). There are no components in $\widehat{C}$ that are connected to C at this stage, i.e., $\Gamma = \{\}$, and we return back to line 7. We update distances and paths again: $\mathtt{dist}(C, C_3) = 2$ and $\mathtt{dist}(C, C_4) = 2$. We extract the component with minimum dist, which may be C_3 or C_4. Assume that we extract C_3 at line 9. Hence, we unite C and C_3 and vertex e and obtain $C = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}\}$. Then, we examine the only component remaining in $\widehat{C}$ which is C_4 to see if it is now connected to C. We get $\Gamma = C_4$ and we unite C and C_4 at line 15. Finally, in Figure 3(c) we have $C = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}, \mathrm{f}, \mathrm{g}, \mathrm{h}\}$ which is the connected identifying code I_c output by the algorithm.

Algorithm ConnectID resembles Prim's algorithm for constructing the minimum spanning tree of a graph [29], but exhibits some fundamental differences. For example, Prim's algorithm selects an edge with minimal weight at every iteration and finally spans every vertex in the graph. However, ConnectID selects a path with the shortest length at every iteration and finally spans all components, which may not include all vertices in the graph.

C. *Performance results*

In this section, we first prove two properties of any identifying code I. These properties are invariably true at every iteration of ConnectID. Based on this, we prove our main result, that is, that algorithm ConnectID produces a connected identifying code whose size is tightly bounded with respect to the input identifying code. Finally, we provide a performance analysis for the connected robust identifying code achieved by ConnectID when the input identifying code is robust.

Lemma 5.1: Consider any identifying code I that is partitioned into a set of components of connectivity $P = \{C_1, ..., C_{|P|}\}$ over graph G. If $|P| > 1$, then every component C_i in P is at most three hops away from some other component C_j in P where $j \neq i$.

Proof: By the definition presented in Section III-A for an identifying code, every non-codeword vertex in G is adjacent to at least one codeword in I. Since the graph is connected, every pair of components in P should be connected by at least one path. Consider the shortest path connecting component C_i in P to component C_k in P where $k \neq i$. The second node on this path (the node at the first hop) is obviously not a codeword because otherwise it would be included in C_i. The third node on this path (the node at the second hop) is either a codeword belonging to a component C_j in P or is a non-codeword adjacent to some component C_j. Component C_j should be different from C_i because otherwise the selected path from C_i to C_k will not be the shortest. ■

Lemma 5.2: Every vertex in graph G that is adjacent to a component C_i with cardinality one in P is adjacent to at least one other component C_j in P where $j \neq i$.

Proof: This property follows from the uniqueness of the identifying sets. The identifying set of the single codeword belonging to component C_i is itself. If any non-codeword that is adjacent to C_i is not adjacent to at least one other component C_j where $j \neq i$, then it will have the same identifying set as the single codeword in C_i which contradicts the definition of an identifying code. ■

Corollary 5.3: Consider any identifying code I that is partitioned into a set of components of connectivity $P = \{C_1, ..., C_{|P|}\}$ over graph G. If $|P| > 1$, then every component C_i in P with cardinality one is at most two hops away from some other component C_j in P where $j \neq i$.

Lemmas 5.1 and 5.2 hold for every identifying code I over graph G. Therefore, they are true right after the initialization of algorithm ConnectID. Since at every iteration, we add one or more codewords and do not remove any codeword, the set of codewords in C and in every component of $\widehat{C}$ forms an identifying code. Hence, Lemmas 5.1 and 5.2 invariably hold after every iteration.

Next, we provide the overall analysis of our algorithm which is based on Lemmas 5.1 and 5.2.

Theorem 5.4: Assuming I is an identifying code for graph G and I_{c} is the identifying code created by algorithm $\mathtt{ConnectID}(G,I)$, we have:

i) I_{c} is a connected identifying code.

ii) The total number of codewords, $|I_{\mathrm{c}}|$ generated by algorithm $\mathtt{ConnectID}(G,I)$ is at most $2|I|-1$. Furthermore, this bound is tight.

Proof:

i) Clearly, C remains a component of connectivity throughout. The while loop starting at line 7 necessarily terminates when $\widehat{C}$ is empty. Since every component extracted from $\widehat{C}$ unites with C at line 10 or line 15, at termination of the while loop $I \subseteq C$, implying that $I_{\mathrm{c}} = C$ is an identifying code.

ii) At every iteration of ConnectID, we unite C with at least one component denoted C^* in $\widehat{C}$ and add at most two codewords according to Lemma 5.1. If the newly merged component C^* has cardinality one, then either C^* is two hops away from C or according to Lemma 5.2, the non-codeword on $\mathtt{path}^*(C,C^*)$ that is adjacent to a codeword in C^*, is also adjacent to at least one other component C_i in $\widehat{C}$. In the latter case, after the union at line 10, C_i becomes connected to C and unites with C at line 15. Thus, we are adding at most two codewords on $\mathtt{path}^*(C,C^*)$. Overall, we select at most one new vertex as codeword for every codeword in $I \setminus C_1$ where $\setminus$ denotes the usual set difference operator. Thus, the cardinality of the resulting identifying code $|I_{\mathrm{c}}|$ is at most $|I| + |I \setminus C_1| \leq 2|I| - 1$ codewords when $\mathtt{ConnectID}(G,I)$ terminates.

This bound is tight. Consider a ring topology with $2k$ vertices (k being a positive integer). The optimal identifying code (i.e., that with minimum cardinality) consists of k interleaved vertices, whereas the connected identifying code for this graph and the mentioned input identifying code must necessarily contain all but one vertex, i.e., $|I_{\mathrm{c}}| = 2k-1$. ■

Corollary 5.5: The redundancy ratio $R = |I_{\mathrm{c}}|/|I|$ of the connected identifying code I_{c} achieved by $\mathtt{ConnectID}(G,I)$ is at most two for any given graph G.

If the input identifying code I to $\mathtt{ConnectID}(G,I)$ is an identifying code achieved by the algorithm in [14], then we have $|I| \leq c\,|I^*| \ln|V|$ where $c > 0$ is a constant, I^* is the identifying code with minimum cardinality for graph G and $|V|$ is the number of vertices in graph G. We define I_{c}^* to be the connected identifying code with minimum cardinality in graph G. Since $|I_{\mathrm{c}}^*| \geq |I^*|$, we have the following corollary.

Corollary 5.6: If the input identifying code I to $\mathtt{ConnectID}(G,I)$ is an identifying code achieved by the algorithm in [14], then the cardinality of the connected identifying code I_{c} achieved by ConnectID is at most $c'\,|I_{\mathrm{c}}^*| \ln|V|$ where $c' > 0$ is a constant.

Robustness analysis. The properties of ConnectID ensure that it produces a connected *robust* code if it is given a robust code as an input. Next, we combine the results of the algorithm with well-known coding theoretic bounds to derive bounds on the cardinality of connected robust identifying codes. We show that as robustness increases, the resulting codes are increasingly connected.

Before presenting our analysis, we present our notations. Recall our notation that an r-robust identifying code I over graph G can be partitioned into connected components $P = \{C_1, \ldots C_{|P|}\}$. We define $S_{\min}(I)$ (or just $S_{\min}$ in short) to be the minimum non-unitary cardinality of a component in P, i.e.,

$$S_{\min} = \min_{j \text{ s.t. } C_j \in P \text{ and } |C_j|>1} |C_j|.$$

Our upper bound on the cardinality of I_{c} depends on $S_{\min}$, for which we shall provide lower bounds later in this section.

Lemma 5.7: Given an $r \geq 1$-robust identifying code I with connected components $P = \{C_1, \ldots, C_{|P|}\}$, there may be at most one component C_i with cardinality one.

Proof: We prove the Lemma by contradiction. Suppose there are at least two components with cardinality one. Then, the Hamming distance between the identifying sets of the single codeword in the two components is two. This contradicts our assumption that I is r-robust for $r \geq 1$ since the minimum symmetric difference of I, $d_{\min}(I)$, should be at least $2r+1$. ■

The following theorem is based on Lemma 5.7.

Theorem 5.8: The connected identifying code I_{c} produced by $\mathtt{ConnectID}(G,I)$ from an r-robust identifying code I over graph G satisfies

$$|I_{\mathrm{c}}| \leq \left(1 + \frac{2}{S_{\min}}\right) |I| - \frac{2}{S_{\min}}.$$

Proof: If I is connected, the bound follows trivially. Otherwise, I consists of at least two components. Therefore, $\widehat{C}$ and C in ConnectID are initially not empty. Based on Lemma 5.7 there is at most one component with cardinality one. Three scenarios are possible:

(i) Component C is initialized to the only component with cardinality one. In this case, every component in $\widehat{C}$ has cardinality at least $S_{\min}$ and there are $|I|-1$ codewords not in C. Hence, $\widehat{C}$ contains at most $(|I|-1)/S_{\min}$ components initially. Using a similar reasoning as in Theorem 5.4 based on Lemma 5.1, ConnectID adds at most two codewords per every component that is initially in $\widehat{C}$. Therefore, we have $|I_{\mathrm{c}}| \leq |I| + 2(|I|-1)/S_{\min}$.

(ii) There is a component with cardinality one in $\widehat{C}$ at initialization. In this case, there are at most $|I| - S_{\min}$ codewords not in C initially. We add at most one codeword for the component with cardinality one in $\widehat{C}$ based on Lemma 5.2. There are at most $(|I| - S_{\min} - 1)/S_{\min}$ other components in $\widehat{C}$ initially. Therefore, we add at most $2(|I| - S_{\min} - 1)/S_{\min}$ codewords plus one codeword for the component with cardinality one to $|I|$. The overall cardinality of I_{c} is at most $|I| + 2(|I|-1)/S_{\min} - 1$ in this case.

(iii) There is no component with cardinality one. In this case, there are at most $(|I| - S_{\min})/S_{\min}$ components in $\widehat{C}$ initially and we add at most two codewords per every component in $\widehat{C}$. Therefore, we have $|I_{\mathrm{c}}| \leq |I| + 2|I|/S_{\min} - 2$.

Case (i) leads to the largest upper bound on $|I_{\mathrm{c}}|$ among the three cases. ■

The following lemma relates $S_{\min}$ to the $r \geq 1$-robust identifying code of minimum possible size.

Lemma 5.9: The value of $S_{\min}$ is lower bounded by the minimum size of an $r \geq 1$-robust identifying code with more than one codeword.

Proof: We are given an r-robust identifying code that is partitioned to a set of components P. For every component C_j in P, the identifying set for every codeword in C_j consists of a unique subset of codewords in C_j. The minimum symmetric difference between the identifying sets of the codewords in C_j must be at least $2r+1$, i.e., $d_{\min}(C_j) \geq 2r+1$. Therefore, the codewords in C_j form an r-robust identifying code for the subgraph induced by C_j in G. Hence, the size of C_j has to be at least as large as the size of the minimum possible r-robust identifying code. Since $S_{\min}$ is greater than one by definition, the lemma follows. ■

Based on Lemma 5.9, we next relate $S_{\min}$ to the size of a minimum error-correcting code. Recall that the characteristic vector of a set is the binary vector whose i-th bit is 1 if and only if the i-th element of a given universe (in this case, the set of vertices in the graph) is in the set. Note that the characteristic vectors of the identifying sets of an r-robust identifying code I form a binary r-error correcting code of length $|I|$. The reverse does not necessarily hold because of the limitations imposed on identifying codes by the graph structure.

We can now form a relationship between $S_{\min}$ and the coding-theoretic function $A(n,d)$ denoting the maximal size of a (binary) code of length n and minimum distance d. This leads us to our theorem linking bounds on connected identifying codes and error-correcting codes, and allows the application of coding-theoretic upper bounds to connected identifying codes.

Theorem 5.10: Given an $r = \frac{d-1}{2}$-robust identifying code, it holds that

$$S_{\min} \geq \min_{2 \leq n \leq A(n,d)} n.$$

Proof: For any given $r \geq 0$-robust identifying code I with n codewords over an arbitrary graph G, we know from [4] that $d_{\min}(I) \geq 2r+1$, meaning that an r-robust identifying code with n codewords over any given graph G exists *only if* an r-error correcting code exists with length n and size $A(n, d = 2r+1) \geq n$.

Let $n_{\min} = \arg\min_{n \geq 2}(n \leq A(n,d))$. If $n_{\min} = 2$, then $S_{\min} \geq n_{\min}$ trivially since $S_{\min}$ is an integer strictly larger than one. Otherwise for $n_{\min} > 2$, it must be that $n' > A(n', d)$ for every n' such that $1 < n' < n_{\min}$. This implies that $S_{\min} \geq n_{\min}$, proving the theorem. ■

Thus, $S_{\min}$ is bounded by the smallest n for which $A(n,d) \geq n$. We can sharpen this result with the Plotkin bound [30] and the following lemma.

Lemma 5.11: For all $2 \leq n \leq 2d-1$ and odd $d \geq 3$,

$$A(n,d) < n.$$

Proof: The Plotkin bound states that $A(n,d) \leq 2\left\lfloor \frac{d+1}{2d+1-n} \right\rfloor$ for odd $d > \frac{n-1}{2}$. In order for the right-hand side of the inequality to be greater or equal to n, it must be that:

$$\begin{aligned} \frac{d+1}{2d+1-n} &\geq n/2 \\ n^2 - (2d+1)n + 2d + 2 &\geq 0. \end{aligned}$$

For $2 \leq n \leq 2d-1$ and odd $d \geq 3$, this inequality has no feasible solution. ■

The following lemma follows directly from Lemma 5.11.

Lemma 5.12: For odd $d \geq 3$, $S_{\min} \geq 2d$.

Combining Theorem 5.8 with Lemma 5.12 we have the following simple bound on the size of a connected code generated by our algorithm.

Corollary 5.13: If the input identifying code I to $\texttt{ConnectID}(G, I)$ is an r-robust identifying code for graph G, where $r \geq 1$, we have,

$$|I_c| \leq \left(1 + \frac{1}{2r+1}\right) |I| - \frac{1}{2r+1} .$$

We observe that with increase of r, $S_{\min}$ increases and the upper bound on $|I_c|$ gets closer to $|I|$. This implies that for larger robustness r, I tends to be more connected and we usually require fewer additional codewords to make it connected. Furthermore, according to Corollary 5.13 for large values of robustness r, $|I_c|$ tends to $|I|$. Note, on the other hand, that connectivity does not necessarily imply robustness, as one can observe from Figure 1(b).

D. Implementation

Our implementation relies on well-known data structures and algorithms, as may be found in a standard text [29]. Its main data structure is the *disjoint-set*, which is used to maintain components of connectivity. For our purpose, every disjoint set will store a connected component of codewords as a linked list, with all the codewords of a component maintaining a link to a common representative.

Populating these data structures requires the use of a connected components algorithm, such as that of Hopcroft and Tarjan based on the BFS or DFS [31] requiring an overall $O(|V| + |E|)$ time. We next describe how to use this data structure to calculate the distance $\texttt{dist}(C, C_j)$ and the shortest plain path $\texttt{path}(C, C_j)$ between component C and every component C_j in $\widehat{C}$ as needed in line 8 of $\texttt{ConnectID}$.

Starting at any codeword of component C, we run an optimized two-stage Breadth First Search (BFS). To begin, we select an arbitrary codeword in C to be the source (with distance metric 0). In the first stage, we visit and finish all codewords in component C without updating our distance metric. In the second stage, we visit and finish other vertices not in C as we increment the distances. The motivation behind a two-stage BFS is to finish all codewords at distance zero from the source, i.e., codewords in C, before the rest of the vertices. In order to engineer the BFS in two stages, we use two BFS queues. The first queue stores the visited but unfinished codewords in C. The second queue stores the rest of the vertices that are visited but unfinished.

In the first stage of BFS, when we visit a non-codeword adjacent to a codeword in C, we insert it into the second queue, and we do not extract any vertex from the second queue until we finish all codewords in C in the first stage, i.e., we empty the first queue. In the second stage, the BFS continues the search starting from the non-codewords in the second queue. All codewords outside C are considered leaf vertices, i.e., we do not visit their adjacent vertices. This is because we are only interested in plain paths. While running the BFS, we maintain an estimate of the distance $\texttt{dist}(C, C_j)$ between C and every component C_j in $\widehat{C}$, initialized to infinity. Every time BFS visits a codeword, it finds the component to which it belongs using the $\texttt{find}-\texttt{set}$ primitive (of the disjoint-set data structure) and updates the estimate of $\texttt{dist}(C, C_j)$ accordingly (i.e., keeping the smaller value seen so far). It also stores the codeword that achieved the smaller distance since this will be used to find the shortest plain path $\texttt{path}(C, C_j)$ upon termination. We also maintain the component C^* with minimum $\texttt{dist}(C, C^*)$ during the BFS process. In this way, there will be no additional processing to find the component with minimum distance from C.

Computation of the distances and the shortest plain paths between C and the components in $\widehat{C}$ described above is no more than that of the standard BFS upon which it is based, i.e., $O(|V| + |E|)$, since we exercise a constant overhead per node during the traversal.

E. Complexity analysis

We next consider the worst case running time of ConnectID. The initialization phase takes $O(|E|)$ time: we remove all non-codewords and incident edges from the graph, run connected-components to partition the result, and then set up $\widehat{C}$ (as a linked list) and C. The iteration part of algorithm ConnectID can run in $O(N|E|)$ time as follows.

The while loop (starting at line 7) iterates at most N (which is $O(|E|)$) times, and at least one component is extracted from $\widehat{C}$ per iteration. Within the loop, each iteration requires the calculation of $\texttt{dist}(C, C_j)$ and $\texttt{path}^*(C, C_j)$ at line 8 requires $O(|V|+|E|)$ time as described in Section V-D. Line 9 takes the $O(1)$ needed to delete from a linked list, since we have already identified the component C^*. Lines 10 and 15 require $O(|V|)$ time, since Lemma 5.1 assures only a constant number of calls to the disjoint-set unionprimitive. Line 11 requires the algorithm to run $\texttt{find}-\texttt{set}$ (i.e., constant-time) on all neighbors of vertices on $\texttt{path}^*(C, C^*)$, of which there are $O(|E|)$. For each of the components found, a union operation is used, giving a net total of $O(|V|)$ unions over the life of the iteration loop. Altogether, the computational complexity of ConnectID is $O(N|E|)$, which is $O(|V||E|)$ since $N \leq |V|$.

VI. Numerical results

In this section, we evaluate the performance of ConnectID on two types of random graphs: Erdős-Rényi random graphs and regular random graphs, i.e., graphs with random arrangement of edges such that every node will have a fixed degree. It should be noted that even though geometric random graphs are appropriate for modeling the outdoor communication range of wireless sensors, they are less practical for indoor or harsh environments for which applications of identifying codes have been proposed [4], and we have thus not included them. Indeed, geometric random graphs generally do not possess identifying codes [14], although there are ways to get around this problem by removing a few indistinguishable vertices from the graph.

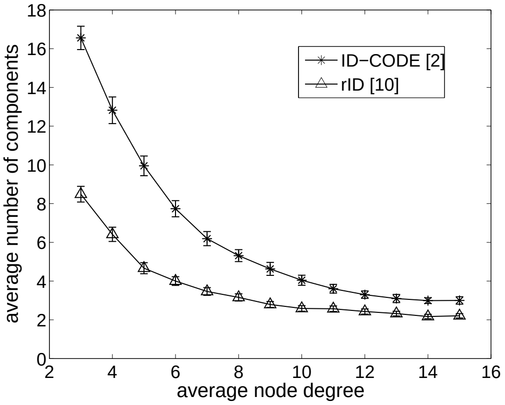

Fig. 4. Average number of components of connectivity for the identifying codes produced by ID-CODE [4] and by rID [14] over 100-node Erdős-Rényi random graphs and varying average node degree.

In order to generate an identifying code for a given graph instance, we use the two existing algorithms rID [14] and ID-CODE [4] that we briefly reviewed in Section III-B. As we will see, the identifying codes generated by rID and ID-CODE are often disconnected.

Our metrics are the following: the number of components of connectivity for each of the identifying codes, the cardinality of the identifying codes generated by algorithm ID-CODE and algorithm rID, the cardinality of the connected identifying codes generated by ConnectID for each of the two identifying codes and the corresponding redundancy ratios. We have measured the mentioned metrics over at least 100 graph instances and plotted the empirical means and 95% confidence intervals.

A. Erdős-Rényi random graphs

We consider two scenarios, either we fix the number of vertices in the graph and change the average node degree, or we fix the average node degree and change the graph size (i.e., the number of graph vertices). We finally present results for connected robust identifying codes.

Figures 4, 5 and 6 correspond to random graphs with 100 nodes and average node degree ranging from 3 to 15.

Figure 4 shows the average number of components of the identifying codes produced by ID-CODE and by rID. We expect lower redundancy with fewer components. If there is a single component, the identifying code is connected. We observe that algorithm rID produces fewer components of connectivity than algorithm ID-CODE on average. We also observe that the average number of components decreases as the average node degree increases and equals about 2 when the average node degree equals 15. This is reasonable since the connectivity between vertices (and codewords) increases with the average node degree.

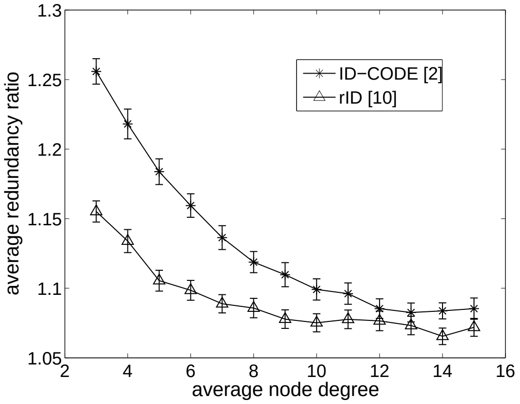

Fig. 5. Average redundancy ratio of the connected identifying codes generated by `ConnectID` for input identifying codes from `ID-CODE` [4] and from `rID` [14] over 100-node Erdős-Rényi random graphs and varying average node degree.

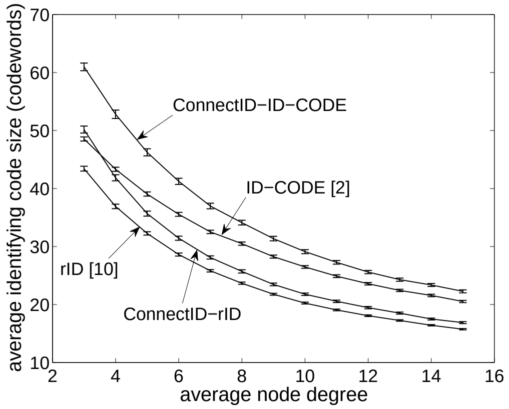

Fig. 6. Average cardinality of the input identifying codes from `ID-CODE` [4] and from `rID` [14] and average cardinality of the connected identifying codes generated by `ConnectID` in both cases for 100-node Erdős-Rényi random graphs and varying average node degree.

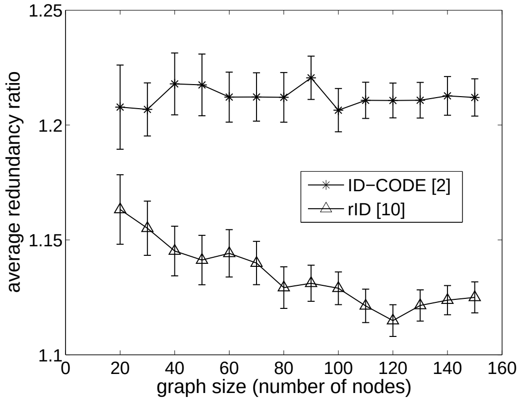

Fig. 7. Average redundancy ratio of the connected identifying codes generated by `ConnectID` for Erdős-Rényi random graphs of increasing size and the input identifying codes from `ID-CODE` [4] and from `rID` [14]. The average degree of the graphs is kept fixed to four.

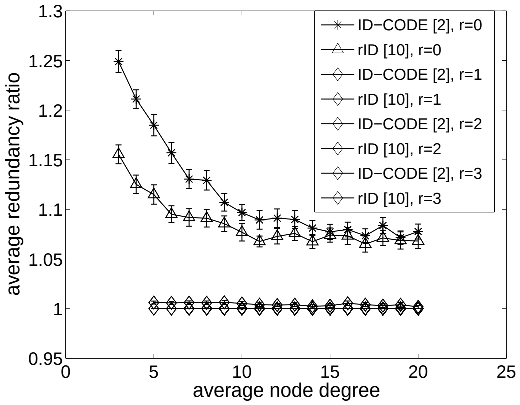

Fig. 8. Average redundancy ratio of the connected identifying codes generated by `ConnectID` for input identifying codes from `ID-CODE` [4] and from `rID` [14] with various degrees of robustness ranging from 0 to 3. The underlying graphs are 100-node Erdős-Rényi random graphs. All curves with diamond markers are almost overlapping.

Figure 5 shows the average redundancy ratio of `ConnectID`, when the input identifying codes are generated by `ID-CODE` and by `rID`. As one can expect, based on the results of Figure 4, `rID` leads to a smaller redundancy ratio than that of `ID-CODE`. In both cases, the average redundancy ratio decreases as the average node degree increases and approaches a value quite close to 1 for an average node degree of 15. The average redundancy ratio achieves its highest value (i.e., slightly above 1.25) for `ID-CODE` and an average node degree of 3.

Figure 6 compares the cardinality of the connected identifying codes generated by `ConnectID` with the cardinality of identifying codes generated by `ID-CODE` and by `rID`. As previously shown in Figure 5, we observe that the cardinality of the connected identifying code is far smaller than twice that of the input identifying code. We also observe that the cardinality of all four identifying codes decreases with the average node degree. We conclude that for Erdős-Rényi random graphs, algorithm `rID` not only generates a smaller identifying code compared to `ID-CODE` to begin with, but also its resulting connected identifying code is significantly smaller for all examined average node degrees.

Figure 7 depicts the average redundancy ratio for graphs with average node degree of 4 and number of vertices ranging from 20 to 150. According to the figure, for `rID`, the redundancy ratio of the connected identifying code decreases (at least initially) with the size of the graph. The redundancy ratio does not change significantly for `ID-CODE`.

Figure 8 depicts the redundancy ratios for connected identifying codes from `ConnectID` when the input identifying code is 0-robust, 1-robust, 2-robust or 3-robust. The graph size is fixed to 100 vertices and the average node degree varies. Except for the case of 0-robust input, we obtain redundancy ratios of about one. This implies that robust identifying codes are often connected for Erdős-Rényi random graphs.

B. *Regular random graphs*

Next, we evaluate the performance of `ConnectID` over regular random graphs with 100 nodes and changing node degrees. Figure 9 depicts the redundancy ratios for connected identifying codes from `ConnectID` when the input identifying code is 0-robust, 1-robust, 2-robust or 3-robust. As for Erdős-Rényi random graphs, we observe that robust identifying codes

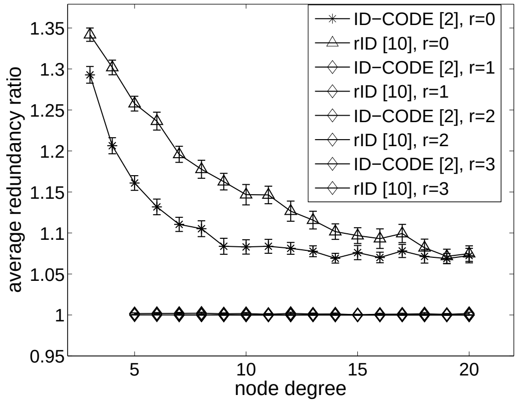

Fig. 9. Average redundancy ratio of the connected identifying codes generated by `ConnectID` for input identifying codes from `ID-CODE` [4] and from `rID` [14] with various degrees of robustness ranging from 0 to 3. The underlying graphs are 100-node regular random graphs. All curves with diamond markers are almost overlapping.

for regular random graphs tend to be connected.

VII. Conclusion and future work

In this work, we addressed the problem of guaranteeing the connectivity of identifying codes, a problem relevant to joint monitoring and routing in sensor networks. We showed, by reduction from the identifying code problem, that the decision problem regarding the existence of a connected identifying code is NP-complete. We introduced algorithm `ConnectID` that produces a connected identifying code by adding codewords to any given identifying code for an arbitrary graph. The cardinality of the resulting connected identifying code is upper bounded by $2|I| - 1$ where $|I|$ is the cardinality of the input identifying code. We proved that the mentioned bound is tight and proposed an efficient implementation for `ConnectID` with polynomial time complexity that grows as the product of the number of edges in the graph and the number of vertices in the graph.

Motivated by the application of robust identifying codes in monitoring harsh environments where sensors may fail and the connectivity is unreliable [4], we extended our analysis to the case where the input identifying code to `ConnectID` is r-robust which leads to a connected r-robust identifying code. By applying the theory of r-error correcting codes, we derived upper bounds on the cardinality of the resulting connected identifying code that depend on the robustness r and the cardinality of the input identifying codes $|I|$. Our results prove that as r becomes large, the redundancy ratio tends to one, meaning that robustness implies connectivity.

We numerically evaluated the redundancy ratio of `ConnectID`. Our simulation results for Erdős-Rényi random graphs and regular random graphs showed that this quantity is generally far below the theoretical bound of two. When the input identifying code is robust, the redundancy ratio is close to one (i.e., the input identifying code is connected or almost connected).

This paper opens several directions for further research. For instance, one could explore different approaches for constructing a connected identifying code. Thus, instead of first constructing an identifying code and then connecting it, once could try to build connected identifying codes from scratch. Also, rather than bounding the redundancy ratio as done in this paper, one could devise algorithms that provide performance guarantees on the minimum number of redundant vertices (codewords) needed, e.g., using Steiner tree heuristics [32, 33]. That said, the results of our paper show that one cannot expect much gain if the input identifying code is robust. Finally, one could investigate extensions of our work to the problem of constructing connected identifying codes with robustness not only in identification but also in connectivity, i.e., the generated identifying code remains connected in the event of failure of a bounded number of graph vertices.

Acknowledgments

We would like to thank the anonymous referees for their very thorough review of the paper, and especially for identifying and filling a small but important hole in Theorem 4.1 and suggesting Lemma 5.12 based on the Plotkin bound.

References

[1] N. Fazlollahi, D. Starobinski, and A. Trachtenberg, “Connecting identifying codes and fundamental bounds,” in *Proc. Information Theory and Applications*, February 2011.

[2] ——, “Connected identifying codes for sensor network monitoring,” in *Proc. IEEE WCNC*, Cancun, Mexico, 2011.

[3] M. G. Karpovsky, K. Chakrabarty and L. B. Levitin, “On a new class of codes for identifying vertices in graphs,” *IEEE Transactions on Information Theory*, vol. **44**, no. 2, pp. 599–611, March 1998.

[4] S. Ray, D. Starobinski, A. Trachtenberg and R. Ungrangsi, “Robust location detection with sensor networks,” *IEEE JSAC (special Issue on fundamental performance limits of wireless sensor networks)*, vol. **22**, no. 6, pp. 1016–1025, August 2004.

[5] S. Gravier and J. Moncel, “Construction of codes identifying sets of vertices,” *The Electronic Journal of Combinatorics*, vol. **12**, no. 1, 2005.

[6] I. Honkala and M. Karpovsky and L. Levitin, “On robust and dynamic identifying codes,” *IEEE Transactions on Information Theory*, vol. **52**, no. 2, pp. 599–612, February 2006.

[7] I. Honkala and A. Lobstein, “On identifying codes in binary Hamming spaces,” *Journal of Combinatorial Theory, Series A*, vol. **99**, no. 2, pp. 232–243, August 2002.

[8] G. Cohen, I. Honkala, A. Lobstein, and G. Zémor, “New bounds for codes identifying vertices in graphs,” *Electronic Journal of Combinatorics*, vol. **6** , 1999.

[9] I. Charon, I. Honkala, O. Hudry and A. Lobstein, “General bounds for identifying codes in some infinite regular graphs,” *Electr. J. Comb.*, vol. **8**, no. 1, 2001.

[10] I. Charon, O. Hudry, and A. Lobstein, “Identifying codes with small radius in some infinite regular graphs,” *Electr. J. Comb.*, vol. **9**, no. 1, 2002.

[11] I. Charon, I. Honkala, O. Hudry, and A. Lobstein, “The minimum density of an identifying code in the king lattice,” *Discrete Mathematics*, vol. **276**, no. 1-3, pp. 95–109, 2004.

[12] T. Berger-Wolf, W. Hart and J. Saia, “Discrete sensor placement problems in distribution networks,” *Mathematical and Computer Modelling*, vol. **42**, no. 13, pp. 1385–1396, December 2005.

[13] M. Laifenfeld, A. Trachtenberg, and T. Berger-Wolf, “Identifying codes and the set cover problem,” in *Proceedings of the 44th Annual Allerton Conference on Communication, Control, and Computing*, September 2006.

[14] M. Laifenfeld, A. Trachtenberg, R. Cohen and D. Starobinski, “Joint monitoring and routing in wireless sensor networks using robust identifying codes,” *Springer Journal on Mobile Networks and Applications (MONET)*, vol. **14**, no. 4, pp. 415–432, August 2009.

[15] S. Ray, R. Ungrangsi, F. De Pellegrini, A. Trachtenberg and D. Starobinski, “Robust location detection in emergency sensor networks,” in *Proc. INFOCOM*, San Francisco, CA, April 2003.

[16] M. Laifenfeld, A. Trachtenberg, and D. Starobinski, *Robust Localization Using Identifying Codes*. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing, 2009, pp. 321–347.

[17] M. Laifenfeld and A. Trachtenberg, “Identifying codes and covering problems,” *IEEE Transactions on Information Theory*, vol. **54**, no. 9, pp. 3929–3950, September 2008.

[18] I. Charon, O. Hudry and A. Lobstein, “Identifying and locating-dominating codes: NP-completeness results for directed graphs,” *IEEE Transactions on Information Theory*, vol. **48**, no. 8, pp. 2192–2200, August 2002.

[19] ——, “Minimizing the size of an identifying or locating-dominating code in a graph is NP-hard,” *Theoretical Computer Science*, vol. **290**, no. 3, pp. 2109–2120, 2003.

[20] S. R. T. Laihonen, *Codes identifying sets of vertices*. Berlin, Germany: Springer-Verlag, 2001, vol. 2227, in Lecture Notes in Computer Science.

[21] Y. Ben-Haim and S. Litsyn, “Exact minimum density of codes identifying vertices in the square grid,” *SIAM Journal on Discrete Mathematics*, vol. **19**, no. 1, pp. 69–82, 2005.

[22] A. Frieze, R. Martin, J. Moncel, M. Ruszinkó and C. Smyth, “Codes identifying sets of vertices in random networks,” *Discrete Mathematics*, vol. **307**, no. 9-10, pp. 1094–1107, May 2007.

[23] J. Suomela, “Approximability of identifying codes and locating-dominating codes,” *Information Processing Letters*, vol. **103**, no. 1, pp. 28–33, 2007.

[24] S. Guha and S. Khuller, “Approximation algorithms for connected dominating sets,” *Algorithmica*, vol. **20**, no. 4, pp. 374–387, April 1998.

[25] J. Blum and M. Ding and A. Thaeler and X. Cheng, *Connected dominating set in sensor networks and MANETs*. Norwell, MA: In Handbook of combinatorial optimization. Kluwer Academic Publishers, 2004.

[26] U. Feige, “A threshold of ln(n) for approximating set cover,” *Journal of the ACM*, vol. **45**, no. 4, pp. 634–652, 1998.

[27] C. H. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1993.

[28] G. Cohen, I. Honkala, A. Lobstein and G. Zémor, “On identifying codes,” in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 56, 2001, pp. 97–109.

[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[30] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. The Netherlands: North-Holland, 1977, vol. 16.

[31] J. Hopcroft and R. Tarjan, “Efficient algorithms for graph manipulation,” *Communications of the ACM*, vol. **16**, no. 6, pp. 372 – 378, June 1973.

[32] M. Karpinsky and A. Zelikovsky, “New approximation algorithms for the Steiner tree problem,” Electronic Colloquim on Computational Complexity (ECCC), Tech. Rep. TR95-030, 1995.

[33] G. Robins and A. Zelikovsky, “Tighter bounds for graph Steiner tree approximation,” *SIAM Journal of Discrete Math*, vol. **19**, no. 1, pp. 122–134, 2005.

PLACE
PHOTO
HERE

Niloofar Fazlollahi received her Ph.D. (2011) and MS (2007) in Computer Systems Engineering from Boston University, and her B.S. in Electrical Engineering from Sharif University of Technology, Tehran, Iran. She is currently a senior routing systems software engineer at Ericsson R&D, San Jose, CA.

PLACE
PHOTO
HERE

David Starobinski received his Ph.D. in Electrical Engineering (1999) from the Technion-Israel Institute of Technology. In 1999-2000 he was a visiting postdoctoral researcher in the EECS department at UC Berkeley. In 2007-2008, he was an invited Professor at EPFL (Switzerland). Since September 2000, he has been with Boston University, where he is now a Professor.

Dr. Starobinski received a CAREER award from the U.S. National Science Foundation (2002), an Early Career Principal Investigator (ECPI) award from the U.S. Department of Energy (2004), the best paper award at the WiOpt 2010 conference, and the 2010 BU ECE Faculty Teaching Award.

He is currently an Associate Editor of the IEEE/ACM Transactions on Networking. His research interests are in the modeling, performance evaluation, and security of high-speed, wireless, and sensor networks.

PLACE
PHOTO
HERE

Ari Trachtenberg received his Ph.D. (2000) and MS (1996) in Computer Science from the University of Illinois at Urbana/Champaign, and his B.S. from MIT (1994) in math with CS. He is currently an Associate Professor of Electrical and Computer Engineering at Boston University, where he has been since September 2000. Dr. Trachtenberg has worked with the Johns Hopkins Center for Talented Youth, Hewlett Packard Laboratories, and MIT Lincoln Lab. He is currently a Kern Faculty Fellow (2012), was a BU College of Engineering Innovative Engineering Education Faculty Fellow (2009), and has received the ECE award for excellence in teaching (2003), an NSF CAREER (2002), Kuck Outstanding Thesis at UIUC (2000), and Mavis Memorial Fund scholarship (1999). His research interests include applications of error correcting codes to networking, algorithms, and security.