Connected Identifying Codes for Sensor Network Monitoring

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

Identifying codes have been proposed as an abstraction for implementing monitoring tasks such as indoor localization using wireless sensor networks. In this approach, sensors' radio coverage overlaps in unique ways over each identifiable region, according to the codewords of an identifying code. While connectivity of the underlying identifying code is necessary for routing data to a sink, existing algorithms that produce identifying codes do not guarantee such a property. As such, we propose a novel polynomial-time algorithm called `ConnectID` that transforms *any* identifying code into a connected version that is also an identifying code and is provably at most twice the size of the original. We evaluate the performance of `ConnectID` on various random graphs, and our simulations show that the connected codes generated are actually at most 25% larger than their non-connected counterparts.

Index Terms

Localization, graph theory, approximation algorithms.

A version of this paper appeared in

N. Fazlollahi, D. Starobinski and A. Trachtenberg, *Connected Identifying Codes for Sensor Network Monitoring*, IEEE WCNC 2011, Cancun, Mexico.

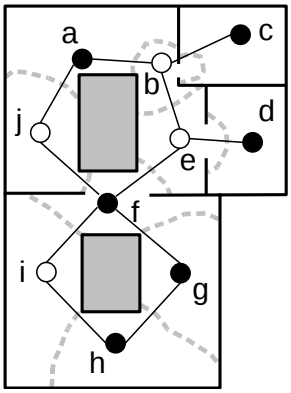

Fig. 1. An example building floor plan and connectivity graph of sensors located at positions marked by circles. The filled circles represent codewords of an identifying code for the sensor network connectivity graph. The dashed lines show the boundaries of distinguishable regions based on the radio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

I. Introduction

Sensor networks are widely used to monitor the environment. Thus, sensors gather ambient data and forward it to a sink for processing. Examples of important monitoring applications using sensor networks include identification of contamination source in water pipes [1], location detection [2–4], structural monitoring of buildings, bridges and air-crafts [5], patient monitoring [6] and tracking and monitoring of endangered animal species [7].

Identifying codes were introduced in [8] and later proposed for sensor network monitoring and particularly for location detection in indoor environments [2, 3]. In the method proposed in [2], sensors in a building are mapped to graph vertices. 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 are kept active while the rest can be put in energy-saving mode. The active sensors correspond to *codewords* of an identifying code in the graph. A target is located by the unique pattern of sensors within its radio range.

An example of indoor floor plan and the graph corresponding to sensor placement and sensor connectivity is depicted in Figure 1. Circles show positions of a sensors. 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). 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. When a target is placed at any of the regions marked by dashed lines, one can uniquely determine its location based on the identifying code. For instance, the set $\{a, c\}$ uniquely identifies the region surrounding position b.

In order to route data over a sensor network and sink sensor data to a processor for location detection processing, we need a *connected* network of active sensors. This important requirement has been ignored in previous work on identifying codes. Yet, if we only activate sensors that correspond to codewords of an identifying code and deactivate the rest, there is no guarantee that we achieve a connected network of active sensors. Therefore, although there exist various algorithms in the literature to create an identifying code for an arbitrary graph [2, 3, 9], none of them guarantees that the produced identifying code is connected.

In this work, we consider the problem of generating a connected identifying code for an arbitrary graph. This approach provides a framework for location detection in sensor networks with guaranteed routing connectivity between the sensors. In particular, we focus on building a connected identifying code out of an identifying code produced by one of the existing algorithms. Our goal is to add a minimum number of codewords so as to keep as many sensors as possible in energy-savings mode. Our contributions are the following:

- We propose a new polynomial-time algorithm called `ConnectID` that creates a connected identifying code from any identifying code for a general graph.

- We prove that `ConnectID` produces a connected identifying code with cardinality at most twice larger than that of the original identifying code. We further show that the bound is tight.
- Using the best known polynomial-time approximation algorithm for building an identifying code [3], we obtain an approximation ratio of $c \ln |V|$ with respect to the minimum connected identifying code, where $c > 0$ is a constant and $|V|$ is the number of vertices in the graph.
- We evaluate the performance of `ConnectID` in terms of the achieved identifying code cardinality through simulations on various random graphs. The simulations show that the size of the resulting code exceeds the size of the original code by a multiplicative factor of 1.25 or less (i.e., significantly smaller than 2).

This paper is organized as follows. In Section II, we formally describe identifying codes and review the related work. In Section III-A we introduce our model and some of our notations. In Section III-B we present our proposed algorithm `ConnectID`. In Section III-C we provide performance analysis of `ConnectID` and its computation complexity. We provide our numerical results in Section IV and conclude the paper in Section V.

II. Background and related work

Assume we have a graph G with a set of vertices V and a set of edges E. Every vertex in V is either a *codeword* or a *non-codeword*. We denote I the set of vertices in V that are codewords. An *identifying set* for vertex $v \in V$ is the set of all codewords that are within distance one from v (this includes node v itself and all of its neighbors). If the identifying set for every vertex is unique, then we call I an *identifying code*. Every super-set of I is an identifying code [2]. We require that no identifying set be empty.

One can verify that for the graph and codewords shown in Figure 1, the identifying set for every vertex of the graph is unique, i.e., the identifying set for vertex a is $\{a\}$, for vertex b is $\{a, c\}$ and so on. Location of a target can be identified at every region using a look-up table that maps identifying sets to vertex IDs.

Ref. [2] suggests application of identifying code theory for indoor location detection. They present a greedy heuristic that creates an *irreducible* identifying code (i.e., no codeword is redundant) for an arbitrary graph. Ref. [3] introduces a more efficient algorithm for generating identifying codes based on a reduction to the *set covering* problem [10]. Accordingly, they prove an approximation ratio with respect to the minimum size identifying code that increases logarithmically with the number of vertices in the graph. They also suggest additional applications of identifying codes for node labeling and routing in the underlying sensor network. Both references implicitly assume that the sensor network can route data toward a sink, an assumption that may not hold in practice and provides the motivation for this work.

In [11, 12], the problem of computing a minimum identifying codes is proved to be NP-complete. Authors in [13] provide probabilistic existence thresholds for identifying codes in random graphs and upper and lower bounds on the minimum cardinality of identifying codes in a random graph. Ref. [14] considers identifying codes that are robust to failure of a bounded number of their codewords over various graph topologies. They also consider *dynamic identifying codes*. A dynamic identifying code is a walk in a graph whose vertices form an identifying code.

Other related graph abstractions include *dominating sets* and connected dominating sets. A dominating set is a subset of graph vertices such that every vertex is adjacent to at least one member of the dominating set. In [15] authors present and compare several heuristics for generating a connected dominating set for an arbitrary graph and provide a competitive performance bound. Reference [16] surveys the literature on connected dominating sets and reviews existing algorithms.

The results on connected dominating sets do not apply to connected identifying codes. Although every identifying code is a dominating set, not every dominating set is an identifying code. Thus, the optimal identifying code generally has larger cardinality than that of the optimal dominating set.

III. ALGORITHM ConnectID

A. Model and notations

We assume an undirected connected graph $G(V, E)$ (or G in short) where V is the set of nodes and E is the set of edges between the nodes. Assume $I \in V$ is the set of codewords of an identifying code in G and a super-set I_c of I is the set of codewords of a connected identifying code in G. The *redundancy ratio* of I_c vs. I is defined to be the ratio of the cardinality of I_c to that of I, i.e. $R = |I_c|/|I|$ where $R \geq 1$ denotes the redundancy ratio. It is desirable to have R as close as possible to one.

We define a *component of connectivity* (or a component in short) C of I in graph G to be a subset of codewords in I such that the sub-graph of G induced by this subset is connected, i.e., the graph $G'(C, E \cap (C \times C))$ is connected where $C \times C$ denotes all pairs of vertices in C. In addition, no codewords can be added to C while maintaining the connectivity of the induced graph G'. Back to the example of Figure 1, we have $I = \{a, c, d, f, g, h\}$. The components of connectivity for I are $C_1 = \{a\}$, $C_2 = \{c\}$, $C_3 = \{d\}$ and $C_4 = \{f, g, h\}$.

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 belonging to C_1 to a vertex x_2 belonging to C_2. A plain path consists of non-codeword vertices except for x_1 and x_2. The term *path* throughout this paper is more general than plain path and is not restricted to non-codewords. As an example, in Figure 1, $\{a, b, e, f\}$ and $\{a, j, f\}$ are the only plain paths between components C_1 and C_4. Note that path $\{a, j, f, e, d\}$ 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 $\texttt{dist}(C_1, C_2)$ and is defined to be the length in number of hops of the shortest plain path between C_1 and C_2. If there is no plain path between C_1 and C_2, then $\texttt{dist}(C_1, C_2) = \infty$. As an example, in Figure 1, $\texttt{dist}(C_1, C_2) = 2$, $\texttt{dist}(C_1, C_3) = 3$ and $\texttt{dist}(C_1, C_4) = 2$.

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 of a connected identifying code I_c. First, we present algorithm ConnectID informally:

In the initialization phase, function $\texttt{ConnectID}(G, I)$ 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.

We define C to be a set that stores the growing connected identifying code. It is initialized to the set of codewords in one of the components, say C_1. We define $\widehat{C}$ to be a 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, we first update the distance $\texttt{dist}(C, C_j)$ between C and every component C_j in $\widehat{C}$. Then, we extract from $\widehat{C}$ the component C^* with minimum $\texttt{dist}(C, C^*)$ (breaking ties arbitrarily). We assign as codewords all vertices on the shortest plain path connecting C and C^* denoted $\texttt{path}^*(C, C^*)$. Then, we unite the codewords in C and C^* and $\texttt{path}^*(C, C^*)$ to form a single larger component, again called C. After this step, we examine if there are any other components in $\widehat{C}$ which become connected to C via the newly selected codewords on $\texttt{path}^*(C, C^*)$. We define $\Gamma \subseteq \widehat{C}$ to be the set of such components. If Γ is non-empty, we unite C with the components in Γ and extract them from $\widehat{C}$. In Section III-C, we briefly explain how to efficiently compute $\texttt{dist}(C, C_j)$ and $\texttt{path}^*(C, C_j)$ for every component C_j in $\widehat{C}$. We repeat the iteration explained above until $\widehat{C}$ becomes empty. At termination, we return $I_c = C$. Note that $I \subseteq I_c$ and therefore, I_c is an identifying code.

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

Algorithm $\texttt{ConnectID}(G, I)$**:**

Initialization:

1) Partition I into a unique set of components of connectivity $\{C_1, C_2, ..., C_N\}$ where $1 \leq N \leq |I|$.

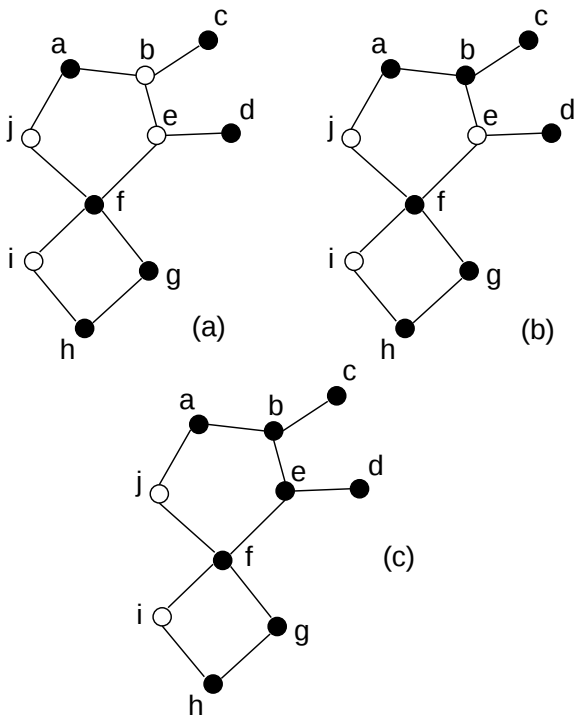

Fig. 2. Progress of $\texttt{ConnectID}(G, I)$. The filled circles represent codewords of an identifying code I for the illustrated graph G (a) initially, I is partitioned to components $C_1 = \{a\}$, $C_2 = \{c\}$, $C_3 = \{d\}$ and $C_4 = \{f, g, h\}$. We set $C = \{a\}$ and $\widehat{C} = \{C_2, C_3, C_4\}$ (b) $C = \{a, b, c\}$ and $\widehat{C} = \{C_3, C_4\}$ (c) $C = \{a, b, c, d, e, f, g, h\}$ and $\widehat{C} = \{\}$.

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 $\texttt{dist}(C, C_j)$ and $\texttt{path}(C, C_j)$ for every $C_j \in \widehat{C}$ and set $C^* \leftarrow \arg\min_{C_j \in \widehat{C}} \texttt{dist}(C, C_j)$.
9) Extract component C^* from $\widehat{C}$.
10) Set $C \leftarrow C \cup C^* \cup \texttt{path}^*(C, C^*)$.
11) Find the set $\Gamma \subseteq \widehat{C}$ of components that are connected to C.
12) If Γ 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 $I_c \leftarrow C$.

Example. Figure 2 shows the progress of $\texttt{ConnectID}(G, I)$ for the same graph and the same input identifying code as shown in Figure 1. The vertices in black are codewords. The figure shows the progress of $\texttt{ConnectID}(G, I)$ after every iteration. Assume at initialization we have: $C_1 = \{a\}$, $C_2 = \{c\}$, $C_3 = \{d\}$ and $C_4 = \{f, g, h\}$. In figure 2(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: $\texttt{dist}(C, C_2) = 2$, $\texttt{dist}(C, C_3) = 3$, $\texttt{dist}(C, C_4) = 2$. At line 9, we extract one component with minimum $\texttt{dist}$ 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, $C = \{a, b, c\}$ as illustrated in figure 2(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: $\texttt{dist}(C, C_3) = 2$ and $\texttt{dist}(C, C_4) = 2$.

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 $C = \{a, b, c, d, e\}$. 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 2(c) we have $C = \{a, b, c, d, e, f, g, h\}$ which is the connected identifying code I_c output by the algorithm.

C. Performance analysis

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, algorithm ConnectID produces a connected identifying code whose size is tightly bounded with respect to the original identifying code. Finally, we briefly discuss the running time of ConnectID.

Lemma 3.1: 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 another component C_j in P where $j \neq i$.

Proof: By the definition presented in section II 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 3.2: 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 3.3: 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 another component C_j in P where $j \neq i$.

Lemma 3.1 and 3.2 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 vertices in C and in every component of $\widehat{C}$ forms an identifying code. Hence, Lemmas 3.1 and 3.2 invariably hold after every iteration.

Theorem 3.4: Assuming I is an identifying code for graph G and I_c is the identifying code created by algorithm ConnectID(G, I), we have:

i) I_c is a connected identifying code.
ii) The total number of codewords generated by algorithm ConnectID(G, I) is at most $2|I| - 1$. Furthermore, this bound is tight.

Proof:

i) Clearly, C is a component of connectivity at initialization and it remains connected after every iteration of function ConnectID. The while loop starting at line 7 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 $I \subseteq C$. This implies $I_c = C$ is an identifying code. We prove by contradiction that the while loop must terminate. Assume $\widehat{C}$ is not empty at some iteration of the while loop. Then C will be at distance of at most three hops from at least one component, say C_j, in $\widehat{C}$ because Lemma 3.1 holds at every iteration. As a result, ConnectID will assign C_j a finite dist(C, C_j) and extract it from $\widehat{C}$ for union with C. Hence, $\widehat{C}$ eventually becomes empty.

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 3.1. If the newly merged component C^* has cardinality one, then either C^* is two hops away from C or according to Lemma 3.2, the non-codeword on $\texttt{path}^*(C, C^*)$ 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 $\texttt{path}^*(C, C^*)$ per at least two components C^* and C_i. Overall, we assign at most one new vertex as codeword for every codeword in $I \setminus C_1$. Thus, the cardinality of the resulting identifying code $|I_c|$ is at most $2|I| - 1$ codewords when $\texttt{ConnectID}(G, I)$ terminates.

This bound is tight. Consider a ring topology with $2k$ nodes (k being a positive integer). The optimal identifying code (i.e. that with minimum cardinality) consists of k interleaved vertices and the minimum cardinality of a connected identifying code for this graph and the mentioned input identifying code is $|I_c| = 2k - 1$. ■

Corollary 3.5: The redundancy ratio $R = |I_c|/|I|$ of the connected identifying code I_c achieved by $\texttt{ConnectID}(G, I)$ is at most two for any given graph G.

Corollary 3.6: If the input identifying code I to $\texttt{ConnectID}(G, I)$ is an identifying code achieved by the algorithm in [3], then the cardinality of the connected identifying code I_c achieved by ConnectID is at most $c\ |I_c^*| \ln |V|$ where $c > 0$ is a constant, I_c^* is the connected identifying code with minimum cardinality in graph G and $|V|$ is the number of vertices in graph G.

Next, we briefly analyze the running time of ConnectID. In order to partition the input identifying code, we remove non-codeword vertices and the edges incident on them and use a connected components algorithm on the remaining graph, for example the algorithm by Hopcroft and Tarjan based on the Breadth First Search (BFS) or Depth First Search (DFS) [17, 18]. We maintain the components of connectivity using a *disjoint-set* data structure [18]. We calculate the distances dist and the shortest plain paths $\texttt{path}^*$ between C and the components in $\widehat{C}$ using a shortest path calculation algorithm based on a modified two-stage BFS which visits the codewords in C prior to other vertices and assigns them distance zero. Using the above data structure, the overall computational complexity of ConnectID is $O(N|E|)$ where $N \leq |V|$ is the total number of components after the initialization.

IV. Numerical results

We evaluate the performance of ConnectID on various instances of Erdos-Renyi random graphs. In order to generate an identifying code for a given graph instance, we use two existing algorithms [2, 3]. Throughout this section, we denote by ID-CODE our implementation of the algorithm presented in [2] and denote by rID our implementation of the algorithm in [3]. As we will see, the identifying codes generated by rID and ID-CODE are often disconnected.

We first use graphs with 100 vertices and change the average degree of the vertices from 3 to 15. We generate 100 instances of graph per every value of average degree. For every graph instance, we measure the following metrics: the cardinality of the identifying code generated by algorithm ID-CODE and algorithm rID, the number of components of connectivity for each of the identifying codes, the cardinality of the connected identifying code generated by ConnectID for each of the two identifying codes and the corresponding redundancy ratio. Our measurements are averaged over 100 instances. We present the empirical mean in Figures 3, 4 and 5. The error bars show 95% confidence intervals.

Figure 3 shows the average number of components of the identifying codes produced by ID-CODE and by rID. We expect lower redundancy when we have fewer components. If the number of components is 1, 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 reaches about two when the average node degree equals 15. This is reasonable since the connectivity between vertices (and codewords) increases with the average node degree.

Figure 4 shows the average redundancy ratio of ConnectID with input identifying codes generated by ID-CODE and by rID. As can be expected based on the results from Figure 3, we obtain a smaller

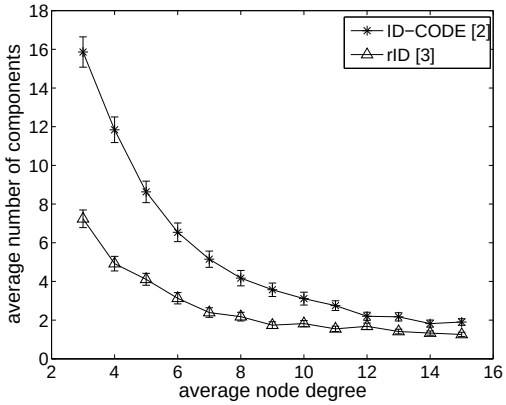

Fig. 3. Average number of components of connectivity for the identifying codes produced by ID-CODE [2] and by rID [3] over 100-node random graphs and varying average node degree.

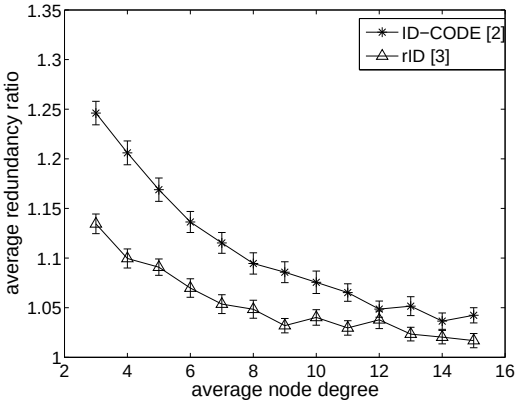

Fig. 4. Average redundancy ratio of the connected identifying codes generated by ConnectID for input identifying codes from ID-CODE [2] and from rID [3] over 100-node random graphs and varying average node degree.

redundancy ratio using algorithm rID and the average redundancy ratio decreases as the average node degree increases. Note that the redundancy ratio is close to one at average node degree of 15 for both algorithms and is about 1.25 for ID-CODE at its highest value when average node degree equals to 3.

Figure 5 compares the cardinality of the connected identifying code generated by ConnectID with the

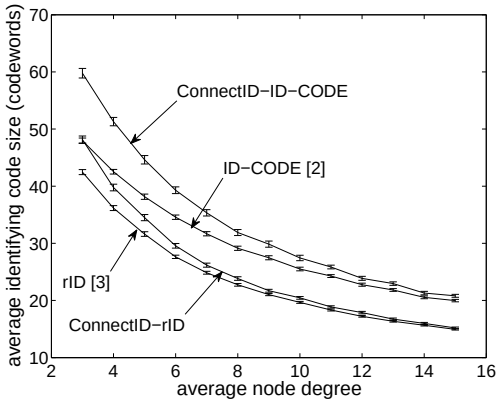

Fig. 5. Average cardinality of the input identifying codes from ID-CODE [2] and from rID [3] and average cardinality of the connected identifying codes generated by ConnectID in both cases for 100-node random graphs and varying average node degree.

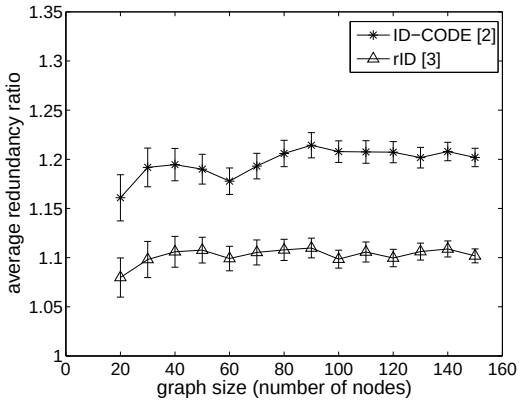

Fig. 6. Average redundancy ratio of the connected identifying codes generated by `ConnectID` for random graphs of increasing size and the input identifying codes from `ID-CODE` [2] and from `rID` [3]. The average degree of the graphs is kept fixed to four.

cardinality of identifying codes generated by `ID-CODE` and by `rID`. As also shown in Figure 4, 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. 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 6 depicts the average redundancy ratio for Erdos-Renyi random graphs with fixed average node degree of four and number of vertices ranging from 20 to 150. Samples are averaged over 100 graph instances as before. According to the figure, with the increase in size of the graph while keeping the average node degree fixed, the redundancy ratio increases slightly. However, it remains almost fixed for graphs with 90 or more vertices. As before, the redundancy ratio is lower for `rID` compared to `ID-CODE`.

V. Conclusion and future work

In this work, we addressed the problem of guaranteeing the connectivity of identifying codes when applied to location detection and routing in sensor networks. 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 have proved that the mentioned bound is tight and that `ConnectID` runs in polynomial time, i.e., at most the product of the number of graph edges and the number of graph vertices

We numerically evaluated the redundancy ratio of `ConnectID`. Redundancy ratio is the ratio of cardinality of the resulting connected identifying code by `ConnectID` to that of the input identifying code. Our simulation results showed that the resulting connected identifying code by `ConnectID` achieves a redundancy ratio of at most 1.25 for all examined cases. This is far below the theoretical bound of two. We used two different algorithms to generate the input identifying code. As one can expect, the achieved redundancy ratio decreases with the average node degree. Also, as one fixes the average node degrees and increases the graph size, the redundancy ratio remains almost unchanged. The analysis of this behavior remains an interesting area for future work.

Acknowledgment

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

REFERENCES

[1] 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.

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

[3] 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.

[4] K. Chakrabarty, S. S. Iyengar, H. Qi and E. Cho, "Grid coverage for surveillance and target location in distributed sensor networks," *IEEE Transactions on computers*, vol. **51**, no. 12, pp. 1448–1453, December 2002.

[5] N. Xu, S. Ranfwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan and D. Estrin, "A wireless sensor network for structural monitoring," in *Proc. the ACM Conference on Embedded Networked Sensor Systems (Sensys04)*, Baltimore, MD, November 2004.

[6] C. R. Baker, et. al., "Wireless sensor networks for home health care," in *Proc. the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW)*, Ontario, Canada, May 2007, pp. 832–837.

[7] E. Biagioni and K. Bridges, "The application of remote sensor technology to assist the recovery of rare and endangered species," *International Journal of High Performance Computing Applications*, vol. **16**, p. 315324, August 2002.

[8] M. G. Karpovsky, K. Chakrabarty and L. B. Levitin, "A new class of codes for identification of vertices in graphs," *IEEE Transactions on Information Theory*, vol. **44**, no. 2, pp. 599–611, March 1998.

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

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

[11] 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.

[12] ——, "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.

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

[14] I. Honkala, 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.

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

[16] J. Blum, M. Ding, A. Thaeler and X. Cheng, *Connected dominating set in sensor networks and MANETs*. Kluwer Academic Publishers, in: Du D-Z, Pardalos P (eds) Handbook of combinatorial optimization.

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

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