Kernel Reference Set and Its Computation Algorithm

Ding-Yuan Bian[1], Qi-Wei Ge[2], Qian Zhu[3] and Qian-Ming Shao[4]
[1, 3, 4] Department of Communications Science and Engineering, Fudan University
Shanghai 200433, China
[2] Faculty of Education, Yamaguchi University
1677-1Yoshida, Yamaguchi 753-8513, Japan
E-mail: [1, 3, 4]{052021081,qianzhu,qmshao}@fudan.edu.cn
[2]gqw@inf.edu.yamaguchi-u.ac.jp

Abstract: This paper proposes a new concept of graphs: kernel reference set. A kernel reference set is a fraction of vertices in a graph such that if their positions are known, positions of all other vertices can be derived from information of the distance matrix. We present an algorithm to find kernel reference set of graphs. Because the problem of kernel reference set is firstly proposed in this paper, and the theoretical limit of computation complexity is unknown currently. So we implement our algorithm with C to do the simulation. From simulation results, we verified the validity of our algorithm and evaluate its performance. Inspired by anchor nodes in wireless sensor networks, kernel reference set could be used to analyze anchorbased localization schemes of WSNs applications such as coalmine security monitoring, etc.
Key words: *Kernel reference set, wireless sensor networks, anchor nodes, localization, graph matching*

1. Introduction

Wireless sensor networks (WSNs) [1] are a kind of concurrent systems which combine sensor, wireless communications, and computer technologies. A WSN is composed of a large number of sensor nodes that has the ability to gather, transmit, and process information.

Localization of sensor nodes is one of the most fundamental issues of WSNs and many localization algorithms have been proposed. One kind of localization schemes are based on anchor nodes whose physical position is known in advance. Locations of other sensor nodes are derived from anchor nodes, so performance of anchor-based localization algorithms highly depend on deployment of anchor nodes.

Many problems in WSNs can be analyzed by using graph theory. Generally, vertices in graphs present sensor nodes and edges represent communication ability between neighboring sensor nodes. Therefore, some problems of WSNs can be analyzed by graph models, such as connectivity, bottleneck nodes, etc. Nevertheless, there isn't a proper model for anchor nodes in graph theory up to now.

In this paper we propose a new concept of graph named "kernel reference set". A kernel reference set is a set of vertices in a graph which represent anchor nodes in WSNs. If position of all vertices in a kernel reference set is given, all other vertices' position could be derived from information of the distance matrix. In this paper, we propose an algorithm to find kernel reference set of a given graph and verify its validity through computational simulation.

This paper is organized in the following way: Section 2 give definition of kernel reference set. An algorithm to find it in a graph is proposed in section 3. We present results of simulations in section 4 and make a conclusion in section 5.

2. Definition of Kernel Reference Set

Some well know definitions in graph theory are used in this paper and we will use the terminology of Bondy & Murty [2]. Assume a graph G(V,E) is given, where V is non-empty vertexset and E is edge-set. Its distance matrix D is also known. (All graphs hereinafter are simple, connected, and unidirectional) But matching relations of vertices ID and rows/columns in D are unknown.

Definition 1: A kernel reference set is a set of vertices in G such that if matching relations of these vertices are given, all other vertices' matching relations could be derived from information of the distance matrix.

Obviously there could be more than one kernel reference set for one graph. And there must be at least one:V except any one of total n vertices. The meaning of kernel reference set is to represent anchor nodes in WSNs. The less the number of anchor nodes, the less the cost and complexity will the sensor networks be. So under the same condition, the kernel reference set with the minimum vertices number is the best.

Assume the graph G represents a WSN. Vertices are sensor nodes and edges represent communication ability between sensor nodes. The distance matrix D includes topology information of the WSN. Therefore, the matching relationship of vertices and rows/columns of D imply position of sensor nodes in the WSN. A kernel reference set in G corresponds to anchor nodes in the WSN.

3. Algorithm to Find Kernel Reference Set

The formulation of the kernel reference set finding problem is as follows:

Input: a graph G(E,V)
Output: a kernel reference set of the graph G with minimum vertices.

It is obvious that for a given graph G(V,E), there must be at least one kernel reference set (Any combination of N-1 vertices where there is N vertices in G), and there could be more than one selection of kernel reference set. So the goal of our algorithm is to find a kernel reference set with as less vertices as possible in reasonable time limit. Since anchor nodes are usually more complicated and expensive than other sensor nodes, this standard is appropriate.

The main idea of our algorithm is as follows: first construct an automorphic graph of G and name it G'. Then start to match vertices in G and G' according to information in their distance matrices D and D'. Matched vertices become reference vertices, and they could help to match other vertices. When no more matching could be derived and there are still unmatched vertices, one vertex in G is selected and its matching relationship is directed achieved. (G' is manually constructed so all matching between vertices in G and G' are actually known, but they are assumed unknown in the algorithm) This seemingly cheating process is call exposing. Exposed vertex also becomes reference vertex. The matching process continues until all vertices have been matched. All exposed vertices will constitute a kernel reference set of G.

Exposing is the key step in the algorithm. Even by randomly selecting an unmatched vertex to expose, the algorithm can always find a kernel reference set. After analyzing symmetrical properties of graphs, we find that the more unique the vertex is, the better to choose it to expose. An intuitional explanation to this rule is that if the exposed vertex has a unique position in the graph, it is more likely to help to distinguish other vertices. Specific and detailed definition of the exposing standard will be discussed later in this paper.

Algorithm to find a kernel reference set

Input: a graph G
Output: a kernel reference set of G
Step 1: Generate an automorphic graph of G, name it G'
Step 2: Generate distance matrices of G and G', name them D and D' respectively
Step 3: Sort rows in D and D' by dictionary order, name sorted distance matrices SD and SD' respectively
Step 4: Divide vertices in G and G' into classes according to SD and SD'
Step 5: While two vertices have identical distances to all other vertices, expose any of them
Step 6: While there is a class in G that only contains one unmatched vertex in G, match it to its counterpart in G' (The vertex has a unique position in the topology)
Step 7: For each unmatched vertex in G, construct a row vector that the ith element of this vector is the distance to the ith matched vertex. If there is a unique vector, its corresponding vertex could be matched. (The vertex has a unique position in the topology after ith vertices have been matched)
Go to Step 6
Step 8: While there is any unmatched vertex, choose a vertex in G to expose according the following rules:
(i) The class it belongs to has minimum number of unmatched vertices;
(ii) It has the maximum value of distances to other vertices.
Go to Step 6
Step 9: Construct a kernel reference set with all exposed vertices

Now we will give a simple example to demonstrate our algorithm.

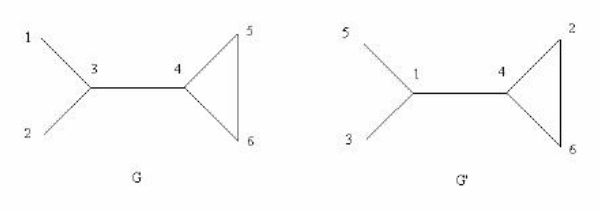

Fig. 1 Graph G and its automorphic graph G'

Figure 1 shows a graph G and its automorphic graph G'. From the two graphs we can easily get their distance matrices D and D' (as shown in Table 1 and Table 2).

Tabel 1: Distance matrix D of graph G

	1	2	3	4	5	6
1	0	2	1	2	3	3
2	2	0	1	2	3	3
3	1	1	0	1	2	2
4	2	2	1	0	1	1
5	3	3	2	1	0	1
6	3	3	2	1	1	0

Tabel 2: Distance matrix D' of graph G'

	1	2	3	4	5	6
1	0	2	1	1	1	2
2	2	0	3	1	3	1
3	1	3	0	2	2	3
4	1	1	2	0	2	1
5	1	3	2	2	0	3
6	2	1	3	1	3	0

The first row and the first column here are indices of vertices of graphs; they are not content of distance matrix. It should be noted that during the algorithm, vertices indices in graph G' are supposed to be unknown except when exposing. Only the n×n distance matrix G' is available.

Since matching relations of G and G' are unknown, sequence of elements in a row vector in D' cannot give any information at first. So we sort the distance matrices. The sorting is divided into two steps. First sort each row of the distance matrix in ascending order; then sort those row vectors that have been sorted internally in ascending order. i.e. Compare from the first column, sort all rows in ascending order, then compare the second column, the third….until the nth column.

Sorted distance matrices SD and SD' are shown as Table 3 and 4.

Tabel 3: Sorted distance matrix SD of graph G

3	0	1	1	1	2	2
4	0	1	1	1	2	2
5	0	1	1	2	3	3
6	0	1	1	2	3	3
1	0	1	2	2	3	3
2	0	1	2	2	3	3

Tabel 4: Sorted distance matrix SD' of graph G'

1	0	1	1	1	2	2
4	0	1	1	1	2	2
5	0	1	1	2	3	3
3	0	1	1	2	3	3
2	0	1	2	2	3	3
6	0	1	2	2	3	3

Observing SD and SD', we can find they are the same except the first column. Considering the first column is the vertices index which isn't a constitutional part of distance matrices. We can get the following property: sorted distance matrix of a graph and that of its automorhpic graph are the same.

Row vectors in the sorted distance matrix represent an unordered sequence of distance from the vertex to all other vertices in the graph. Considering them a type of invariant of vertices, we can divide vertices into different classes where vertices in different classes cannot be confused. According to the previously mentioned property, results of class dividing of G and G' must be the same except that vertices indices may be different.

Graph G (G') can be divided into 3 classes:
C_1= {v_3, v_4}, C_2= {v_5, v_6}, C_3= {v_1, v_2}, and
C_1'= {v_1', v_4'}, C_2'= {v_5', v_3'}, C_3'= {v_2', v_6'}

Vertices belong to different classes have different invariants, so they cannot be matched.

The matching process is simplified to distinguish vertices in the same class. For those vertices that have identical distance to all other vertices, the only method to distinguish them is exposing. For example, we can find that v_1 and v_2 have identical distance to all other vertices. If we choose to expose v1, then we get to know that v1is matched with v_5' in G'. Since C_1(C_1') only contains 2 vertices, we can naturally derive that v_2 in G is matched with v_3' in G'. Similarly, we can find that v_5 in G is matched with v_2' in G' and v_6 in G is matched with v_6' in G' by exposing v_5 or v_6.

The rest vertices could be matched in two ways:

(i) If there exists a class of G that only contains one unmatched vertex, it is distinctly matched to the vertex in the corresponding class in G';

(ii) Suppose k vertices of G have been matched, sort them in ascending order in term of vertex index. For each unmatched vertex in G compose a row vector that has k elements. Each element is the distance from the unmatched vertex to a reference vertex (in the same sequence as reference vertices). If there is a unique row vector, then the vertex it represents can be matched by finding the vertex in G' that has exact the same distances to all reference vertices. Take the example in Figure 1. Suppose we already know v_1 in G is matched with v_5' in G' and v_2 in G is matched with v_3' in G'. From the information in D, we can construct row vectors which indicate distance to reference vertices v_1 and v_2 for unmatched vertices in G as follows:

v_3 - [1, 1]; v_4 - [2, 2]; v_5 - [3, 3]; v_6 - [3, 3]

The first element is the distance to the first reference vertex v_1; the second element is the distance to the second reference vertex v_2. The vector [1, 1] is unique. So there must be only one vertex in G' whose distance to reference vertices of G' are also [1, 1]. From D' we can easily find v1 is the only vertex that satisfies the condition. So we can get that v_3 in G is matched with v_1' in G'. In the same way, all vertices of G and G' in Figure 1 can be matched. Two exposed vertices v_1 and v_5 constitutes a kernel reference set.

4. Computational Simulation

To verify the algorithm and evaluate its performance, we have implemented the algorithm with C language on a PC with a 2.8GHz Pentium IV processor and 512MB of RAM. As for testing graphs, we used graphs from a database of graphs designed for isomorphism and sub-graph isomorphism benchmarking [3]. It contains 72,800 couples of simple graphs belonging to different categories. We choose 2D and 3D regular meshing for testing because generally regular structures represent a worse case for graph matching.

The simulation program will first find a kernel reference set by using the algorithm we proposed. Then it will verify whether the matching of vertices generated by the program comply with the initial settings.

Evaluation is based on two criteria: the first is runtime cost; the second is how many vertices the kernel reference set contains. Some simulation results are showed in the following tables. All data are average value of 10 different couples of graphs.

Table 5: Average run time of 2D meshes

Size	8x8	9x9	10x10	14x14
Time(ms)	136	342	1237	18103

Table 6: Average run time of 3D meshes

Size	3x3x3	4x4x4	5x5x5	6x6x6
Time(ms)	10.5	79.4	1464	18879

From runtime we can see the algorithm is polynomial complexity. But the algorithm doesn't guarantee that the kernel reference set contains minimum number of vertices. Since 2D meshes and 3D meshes are regular structures, the limit of minimum number of reference vertices are 2 and 3 respectively. From data in table 7 and table 8, we can see simulation results are tightly close to theoretical limit. It partly verifies that the algorithm is efficient.

Table 7: Average number of vertices in the simulation results of 2D meshes

Size	8x8	9x9	10x10	14x14
Number of vertices	2.0	2.0	2.0	2.0

Table 8: Average number of vertices in the simulation results of 3D meshes

Size	3x3x3	4x4x4	5x5x5	6x6x6
Number of vertices	3.7	3.0	3.6	3.0

5. Concluding Remarks

In this paper we propose a new concept of graph: kernel reference set. We also give an algorithm with polynomial complexity to find a kernel reference set of a given graph. Although the algorithm doesn't guarantee the result contains minimum number of vertices, computational simulation shows it is quite efficient.

For applications where the topology of the WSN is known but position of individual sensor nodes are unknown, the concept of kernel reference set could be used for deciding deployment of anchor nodes. One example is lighting system in large stadiums. Lighting management system requires that each bulb (which combines a sensor unit) could be controlled individually. However, it's very hard to require that each bulb be installed in a specified socket. With help of some bulbs with anchor nodes (a kernel reference set), other bulbs could be installed randomly. By letting all nodes communicate with adjacent nodes and gathering this information, a distance matrix could be generated. Therefore, position of all nodes could b achieved.

References

[1] C. Shen, C. Srisathapornphat, and C. Jaikeno, "Sensor Information Networking Architecture and Applications," *IEEE Pers. Commun*., vol 8, no. 4, pp. 52-59, 2001

[2] J.A. Bondy, and U.S.R. Murty, *Graph Theory with Applications*, Macmillan, 1976

[3] P. Foggia, C. Sansone, M. Vento, "A Database of Graphs for Isomorphism Benchmarking," *Proc. of 3rd LAPR TC-15 Workshop on Graph-based Representations in Pattern Recognition. Ischia*, pp. 188-19, 2001