國 立 交 通 大 學

應用數學系

碩 士 論 文

一個用於無線網路中找出受限制的連通控制集的多項式時間近似演算法

A polynomial-time approximation algorithm for the constrained connected dominating set problem in wireless networks

研 究 生：黃思綸

指導教授：陳秋媛　教授

中 華 民 國 九 十 九 年 六 月

一個用於無線網路中找出受限制的連通控制集的多項式時間近似演算法

A polynomial-time approximation algorithm for the constrained connected dominating set problem in wireless networks

研 究 生：黃思綸　　Student：Szu-Lun Huang

指導教授：陳秋媛　　Advisor：Chiuyuan Chen

國 立 交 通 大 學

應 用 數 學 系

碩 士 論 文

A Thesis

Submitted to Department of Applied Mathematics

College of Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Applied Mathematics

June 2010

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 九 年 六 月

一個用於無線網路中找出受限制的連通控制集的多項式時間近似演算法

研究生：黃思綸　　指導老師：陳秋媛　教授

國 立 交 通 大 學

應 用 數 學 系

摘　要

在無線網路中，選出一些點來當做虛擬骨幹已經被探討超過二十年了。之前的許多研究都是在單位圓盤圖上，在這種模型中，所有的點的傳輸半徑都是相同的，而且兩點是相連的若且唯若這兩點的距離不超過一。先前的研究顯示連通控制集能夠當作虛擬骨幹來使用，在這方面已有許多的研究成果。在這篇論文中，我們提出了一個新的想法：有限制條件的連通控制集。這種連通控制集的特點就是：某些點基於特殊的理由，必須一定要在連通控制集中。例如，這些點的剩餘能源較多、或是落在比較重要的位置上。我們提出了一個 $O(n)$時間的演算法來建造有限制條件的連通控制集，這裡的 n 表示無線網路中的節點數。我們所提出的演算法的訊息複雜度是 $O(n)$，效益是 $8\cdot$(最小連通控制集的大小)$+3\cdot k$，其中 k 是有限制條件的點數。

關鍵詞：虛擬骨幹、單位圓盤圖、連通控制集、有限制條件的連通控制集、近似演算法。

中 華 民 國 九 十 九 年 六 月

A polynomial-time approximation algorithm for the constrained connected dominating set problem in wireless networks

Student: Szu-Lun Huang Advisor: Chiuyuan Chen

Department of Applied Mathematics

National Chiao Tung University

Abstract

In wireless ad hoc networks, selecting a set of nodes to form a virtual backbone has been investigated for more than two decades. It has been shown that a connected dominating set (CDS) can be used as a virtual backbone. There are many results for finding CDSs. In this thesis, we propose a new idea: a constrained connected dominating set (CCDS), which is a CDS having the property that some specified nodes must be included in it due to some special reason. For example, the specified nodes could be nodes with more remaining energy or nodes located at important locations. We propose a polynomial-time algorithm for constructing a CCDS; our algorithm works for all wireless networks and the message complexity of it is linear. When the given wireless network is a Unit Disk Graph, the performance ratio of our algorithm is $8|\text{MCDS}| + 3k$, where $|\text{MCDS}|$ is the size of a minimum connected dominating set (MCDS) and k is the number of constrained nodes.

Keywords: Virtual backbone; Unit Disk Graph; Connected dominating set; Constrained connected dominating set; Approximation algorithm.

誌 謝

能夠完成此篇論文首先最要感謝的就是陳秋媛老師，陳老師在我就讀研究所的期間，經常與我討論課業以及研究的方向，培養我獨立思考的能力。除了課業之外陳老師更常常關心我們的生活，以及未來的工作發展，很慶幸在研究所期間能有陳老師的教導。

同時也要感謝學長國元、鈺傑、松育、智懷、宗儒與信儀的照顧，在研究所的期間時時給予我課業上的指導以及許多有用的意見，亦感謝思賢、健峰、葉彬、育生、建文以及許許多多的研究同學和學弟妹們提供的一些研究方面的建議以及協助。

除此之外，我還要感謝我的家人對我的支持與鼓勵，我的未來一定不會辜負你們的期望。

Contents

List of Figures

1 Introduction

A wireless ad hoc network (or simply wireless network) consists of a set of nodes that communicate with each other without any physical infrastructure or centralized administration. A wireless network is usually modeled as a graph $G = (V, E)$, where V is called the *vertex set* and each vertex represents a node in network and E is called the *edge set* such that two vertices are adjacent by an edge if and only if their corresponding nodes in the network can communicate with each other. For convenience, in this thesis, vertices of a graph are also called nodes. When all the nodes have the same transmission range; the graph model of such a wireless network is usually called a *Unit Disk Graph* (UDG) [3]. In a UDG, there is a link between two vertices if and only if the distance between the two vertices is at most one.

Since a wireless network is lack of physical backbone, many researchers proposed the idea of using a virtual backbone to serve as a physical backbone. The usage of virtual backbone is very extensive. A virtual backbone can be used to: serve as an approximation of area coverage [1], reduce the energy consumption [2], and reduce the routing overhead [13]. For example, when a node wants to send a message, backbone nodes will calculate and decide the path that transmits the message from the source to the destination first. Then, this message only passes through nodes on the virtual backbone and matterless nodes will not hear or transmit the message. Besides, non-backbone nodes can stay in dormancy in order to save energy. The above example shows how a virtual backbone can be used to send a message. Consider the wireless network in Figure 1(a). Suppose its virtual backbone is the one shown in Figure 1(b). Suppose node a is the source and node g is the destination. The the message will be sent through nodes a, b, d, e, and g; see Figure 1(c).

In a graph $G = (V, E)$, a subset I of V is called an *independent set* if no two vertices

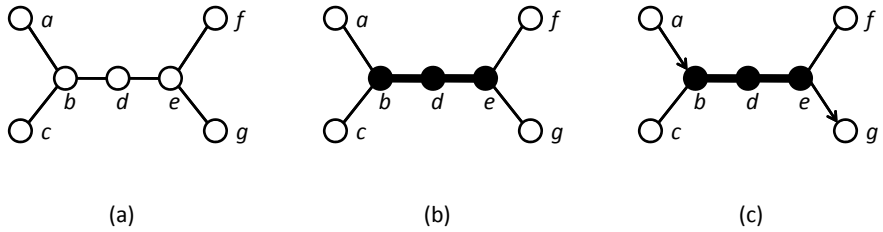

Figure 1: (a) A wireless network, (b) its virtual backbone (the subgraph induced by $\{b, d, e\}$), (c) transmitting a message from a to f by using the virtual backbone.

in I are adjacent. A *maximal independent set* (MIS) is an independent set such that no vertex out of this set could be added to this set to form an independent set with a larger cardinality. A subset D of V is called a *dominating set* if every vertex in G is in D or is adjacent to a vertex in D. A node in D is called a *dominator* and a node outside D is called a *dominatee*. A dominating set is a *connected dominating set* (CDS) if the subgraph induced by this set is connected. A minimum connected dominating set (MCDS) is the connected dominating set of minimum cardinality. The use of a virtual backbone implies that a virtual backbone essentially has the ability to collect and manage the topology of the entire network. Consequently, a CDS is applicable to be the role of a virtual backbone.

Since constructing and maintaining a virtual backbone impose control overhead onto the overall communication, the constructed backbone size should be as small as possible. A virtual backbone requires to be connected. Hence a MCDS is a good candidate. However, finding a MCDS in UDGs has been proven to be an NP-hard problem [3]. Therefore, many researchers consider finding a CDS with a small cardinality (i.e., an approximation solution) instead of a MCDS. Many approximation algorithms have been proposed (see Section 2). Because a maximal independent set (MIS) is itself a dominating set, many approximation algorithms for the MCDS problem begin with finding a maximal independent set first and then, add vertices to the MIS to make it become a CDS.

The purpose of this thesis is to introduce a new version of CDS, the *constrained*

connected dominating set (CCDS), which has the characteristic that some specified nodes must belong to the CDS. The specified nodes could be nodes with more remaining energy or nodes located at important locations in a city or nodes that could have larger degree (meaning that they could serve more nodes). As an application of CCDS, if some specified nodes have more energy than the other nodes, then it is reasonable to restrict these specified nodes to be in the CDS so that the given wireless network could have longer lifetime. As another application of CCDS, in a populous city, it is desirable to have some base stations at specified locations so that the quality of communication could be enhanced; it is reasonable to restrict these base stations to be in the CDS.

Due to the above reasons, we introduce the constrained connected dominating set. We will present a polynomial-time algorithm for computing a constrained connected dominating set. The message complexity of our algorithm is linear. When the given wireless network is a UDG, the performance ratio of our algorithm is $8|\text{MCDS}|+3k$, where $|\text{MCDS}|$ is the size of a MCDS and k is the number of constrained nodes. To know how good our algorithm is, we will compare it with the famous algorithm of Wu and Li [13] by simulations. For convenience, in the remaining part of this thesis, we use WL-algo to denote Wu and Li's algorithm and our-algo, our algorithm. Let the node with the maximum degree (meaning that this node could serve or dominate more nodes than the other nodes) be chosen as the unique constrained node. When the number of nodes in the given wireless network ranges from 40 to 200, the average number of nodes in the CDS generated by WL-algo and that generated by our-algo are shown in Figure 2. Let max and min denote the maximum and the minimum value of the simulation results. The simulation results show that $|max - min|$ could be as large as 115.12 for WL-algo and $|max - min|$ is only 2.34 for our-algo.

This thesis is organized as follows. Section 2 surveys some previous works. Section 3 states our algorithm. Section 4 proves performance ratio, time complexity, and mes-

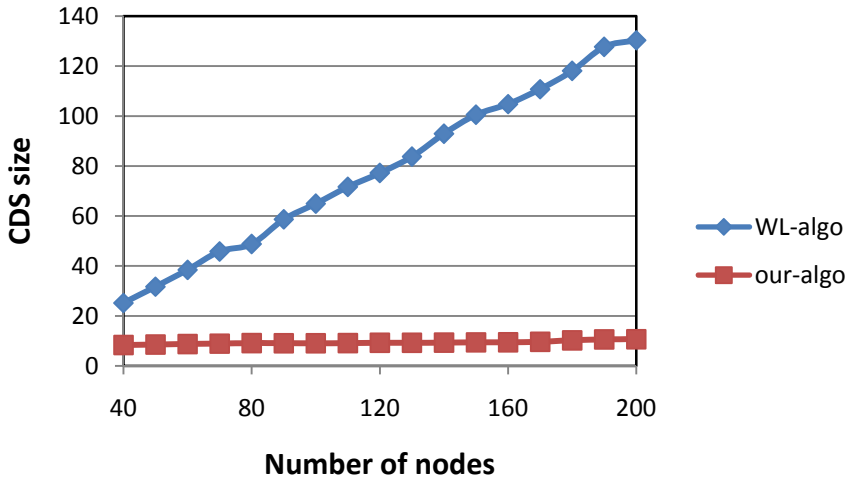

Figure 2: The effect of the number of nodes N. This figure shows that our-algo outperforms WL-algo since $|max - min| \approx 115.12$ for WL-algo and $|max - min| \approx 2.34$ for our-algo.

sage complexity of our algorithm. Section 5 contains the simulation results. Concluding remarks are given in the last section.

2 Related works

Throughout this thesis, n denotes the number of nodes in a given network. Many schemes have been proposed to construct a virtual backbone of a wireless network by using a CDS of the network [2, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14]. Since the size of a virtual backbone affects the routing overhead of the entire network, we take account of the size of the constructed CDS and the performance ratio of an approximation algorithm, which is defined to be the ratio of the worst-case size of the constructed CDS to the size of a MCDS of the given network topology.

In [5, 6, 9], Das et al. proposed distributed algorithms which first find an approximation solution D to the minimum dominating set problem and then connects the nodes in D to obtain a CDS. These schemes have performance ratio of $O(H(\triangle))$, where H is the

harmonic function and $\triangle$ is the maximum degree of the network.

In contrast to Das et al.'s scheme, in [13], Wu and Li proposed a localized algorithm (WL-algo) that directly finds a CDS and then prunes redundant nodes in the CDS. WL-algo is very simple: every node only has to collect the information of its two-hops neighborhood and a node u is added to the CDS whenever it has two nonadjacent neighbors. The performance ratio of WL-algo is $O(n)$.

In [10], Stojmenovic et al. proposed an algorithm that is similar to WL-algo, but instead of using the information of two-hops neighborhood, more information has to collected. The scheme in [10] has performance ratio $O(n)$, which is the same as that of WL-algo. The size of the CDS generated by [10] is usually less than that of WL-algo; however, the message complexity of the algorithm in [10] is larger than that of WL-algo.

In [7, 8, 11, 12], the authors focused on UDGs. In [12], Alzoubi et al. proposed a scheme to find a MIS first and then connect the MIS to form a CDS; this scheme has performance ratio 8. The same authors proposed a distributed algorithm in [11] such that this scheme has a constant approximation ratio, with linear time complexity and linear message complexity. Later on, Gao et al. [7] proposed an improved distributed algorithm that has better message complexity than [11]. Li et al. [8] proposed a distributed algorithm that also finds a MIS first and then, uses a Steiner tree to connect all nodes to the MIS, and obtains a CDS; the performance ratio of this scheme is $(4.8 + \ln 5)$, which is the best ratio up to now.

3 An algorithm for constructing a constrained connected dominating set

In this section, we present our approximation algorithm for constructing the constrained connected dominating set in UDGs; we assume that the given UDG is connected.

To facilitate elaboration, we use colors to stand for states of a node. Four colors are used here:

- The color WHITE means that the node is unexplored.
- The color BLACK means that the node is a dominator.
- The color GRAY means that the node is a dominatee, that is, it has a BLACK node neighbor that dominates it.
- The color BLUE means that the node is a candidate for being a dominator.

Moreover, four notations are used here:

1. $c(u)$ denotes the color of node u.
2. $p(u)$ denotes the parent of node u in the tree that u belongs to.
3. $u \Downarrow$ [msg] means that node u receives a message [msg] from a neighbor.
4. $u \Uparrow$ [msg] means that node u broadcasts message [msg] to all its neighbors.

Our algorithm consists of two stages and the idea is as follows. In Stage 1 (the pseudo code is listed in Algorithm 1), each constrained node (say, A) broadcasts a message and tries to grow a tree; those nodes that receive a message from A become nodes in the tree grown from A. Notice that when Stage 1 ends, if T_A and T_B are two trees and there exists a node u in T_A and a node v in T_B such that $c(u) =$ BLACK and $c(v) =$ BLACK and u and v are adjacent, then T_A and T_B are considered as a tree, not two trees. After Stage 1, the CDS of each tree have been found. Therefore, the purpose of Stage 2 is to connect CDSs found in Stage 1 to form a CDS of the original network by adding some GRAY nodes (these GRAY nodes will become BLACK nodes after they are added to the CDS).

We now give the pseudo code of our algorithm. Note that when a node i sends a message, it will also include its ID, which is i. The input is the IDs of the constrained nodes. The output are nodes which are colored BLACK. Stage 1 is as follows.

Algorithm 1 Stage 1 of our-algo

1: **if** u is a constrained node **then** $c(u) \leftarrow$ BLACK; **else** $c(u) \leftarrow$ WHITE;

2: **if** u is a constrained node **then** $u \Uparrow$ [BLACK];

3: **if** $c(u)$ is WHITE and $u \Downarrow$ [BLACK] **then** { $p(u) \leftarrow v$, where v is the node with the maximum ID among all nodes that send [BLACK] to u; $c(u) \leftarrow$ GRAY; $u \Uparrow$ [GRAY]; }

4: **if** $c(u)$ is WHITE and $u \Downarrow$ [GRAY] **then** { $p(u) \leftarrow v$, where v is the node with the maximum ID among all nodes that send [GRAY] to u; $c(u) \leftarrow$ BLUE; $u \Uparrow$ [BLUE]; }

5: **if** $c(u)$ is BLUE and $u \Downarrow$ [BLACK] **then** { $c(u) \leftarrow$ GRAY; $u \Uparrow$ [GRAY]; }

6: **if** $c(u)$ is BLUE and $u \Downarrow$ [BLUE] and (u has the maximum ID among the senders of its one-hop BLUE neighbors) **then** { $c(u) \leftarrow$ BLACK; $u \Uparrow$ [BLACK]; u sends [INVITE] to $p(u)$; }

7: **if** $c(u)$ is GRAY and $u \Downarrow$ [INVITE] **then** { $c(u) \leftarrow$ BLACK; }

8: When all the nodes are colored with BLACK or GRAY, Stage 1 ends.

Figure 3 shows the change of states of a node when Stage 1 is executed. During the execution of Stage 1, a spanning forest will be constructed step by step. When a WHITE node receives [BLACK] or [GRAY] message for the first time, this WHITE node sets its parent node and is included to a tree in the forest. After the execution of Stage 1, the BLACK nodes in each tree form a CDS of that tree. Recall that the BLACK nodes are dominators and hence they are nodes in CDSs. If Stage 1 results in several CDSs instead of only one CDS, then stage 2 begins and connects CDSs (BLACK nodes in different trees) to form a CDS of the original network; this final CDS is clearly a CCDS since all the constrained nodes are in the final CDS.

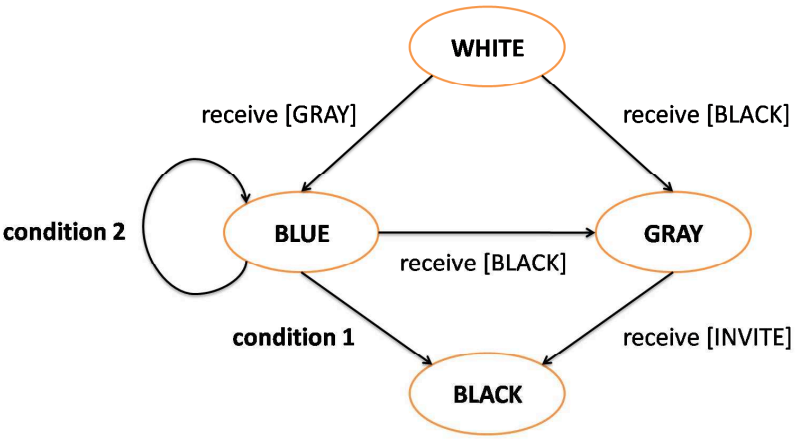

Figure 3: The change of states when Stage 1 is executed.

The following definition will be used in Stage 2. Let T_A and T_B be two tress obtained in Stage 1. T_A and T_B are called *neighboring trees* if there exists a node u in T_A and a node v in T_B such that u and v are adjacent. Two types of messages are used here:

- [ONE-HOP] message is the message that includes all one-hop neighbors of a node.
- [TWO-HOP] message is the message that includes all two-hop neighbors of a node.

Stage 2 works as follows. Every BLACK node (say, u) checks if there exists a BLACK node (say, v) in neighboring trees such that u and v are within three-hop distance. If such a BLACK node v exists, then the BLACK node u calculate the shortest path to each of such BLACK nodes v. Let T_A and T_B be two neighboring trees. A shortest path from T_A to T_B is the path with minimum internal nodes; if there are more than one shortest paths, the path is $u = x_0, x_1, x_2, \ldots, x_n = v$, where $x_1, x_2, \ldots, x_{n-1}$ are chosen by the following rule. For each $i = 0, 1, \ldots, n-1$, let U_i be the neighbors of x_i in all shortest x_i-v paths; then x_{i+1} is the node with largest ID among all nodes in U_i.

All nodes broadcast [ONE-HOP]. When a GRAY node u receives [ONE-HOP] information from another tree, u collects these [ONE-HOP] information and broadcasts

[TWO-HOP]. A GRAY node u may play the role of connecting two CDS's formed by BLACK nodes. A tree calculates shortest paths to neighboring trees and selects the shortest one (say, path P) among these paths. After P is selected, color all the GRAY nodes on the path with BLACK. Repeat the above process to connect neighboring trees until all BLACK nodes are in a connected tree. Stage 2 is as follows.

Algorithm 2 Stage 2 of our-algo

1: Every node $u \Uparrow$ [ONE-HOP];

2: **if** $c(u)$ is GRAY and u receives a [ONE-HOP] message from another tree **then** { u uses all the [ONE-HOP] messages that it receives to prepare its [TWO-HOP] message; $u \Uparrow$ [TWO-HOP]; }

3: **if** $c(u)$ is GRAY and u receives a [ONE-HOP] message from another tree **then** { u collect this information, $u \Uparrow$ [TWO-HOP]; }

4: **if** $C(u)$ is BLACK and $u \Downarrow$ [TWO-HOP] **then** { u knows that in a neighboring tree, there are BLACK nodes within three-hop distance; u calculates shortest paths to these BLACK nodes in a neighboring trees; }

5: A tree determines a shortest path to a neighboring tree; **if** a node u belongs to this shortest path and $c(u)$ is GRAY **then** $c(u) \leftarrow$ BLACK;

6: Repeat step 5 until all BLACK nodes form a CDS.

After Stage 2 is executed, a CCDS will be constructed and our-algo also ends. Here we use Figure 4(a)-(e) to show how our-algo works step by step. The network topology used here is the one used in [12]. In this figure, the numbers labeled on the nodes are the IDs. Consider Stage 1 of our-algo. At step 1, all constrained nodes are colored BLACK and all the other nodes are colored WHITE. Suppose the node with ID 5 and ID 6 are the constrained nodes; see Figure 4(a). The two constrained nodes broadcast the [BLACK] message. WHITE nodes 1, 3, 7, 8, 9, 10, 11 and 12 receive the [BLACK] message and therefore set their color to GRAY; see Figure 4(b). Clearly, Nodes 1, 3, 7, 8, 9, 10, 11

and 12 are dominated by the constrained nodes 5 or 6. Nodes 1, 3, 7, 8, 9, 10, 11 and 12 broadcast the [GRAY] message. Nodes 0, 2 and 4 receive the [GRAY] message; see Figure 4(c) and therefore set their color to BLUE. Nodes 0 has smaller ID than node 4 and therefore node 0 does nothing; nodes 2 has smaller ID than node 4 and therefore node 2 also does nothing; node 4 has the largest ID among all its one-hop BLUE neighbors; thus node 4 set its color to BLACK; see Figure 4(d) for an illustration. Clearly, node 4 dominates nodes 0 and 2. Node 4 broadcasts the [BLACK] message and sends the [INVITE] message to its parent (which is node 8); nodes 0 and 2 receive the [BLACK] message from node 4 and they set their color to GRAY; since node 8 receives the [INVITE] message, it sets its color to BLACK; see Figure 4(e). Since all nodes are colored BLACK or GRAY, Stage 1 ends. Furthermore, since all BLACK nodes form a connected graph, Stage 2 is not executed and our-algo ends.

See also Figure 5 for an example of the network topology and the constructed CCDS obtained by our-algo. The network consists of 100 nodes which are randomly scattered in a square of size $100 \times 100m^2$. The transmission range of each node is $25m$. The red solid lines connect the nodes of the CCDS obtained by our-algo.

4 Performance Analysis

In [12], Wan et al. have proved the following lemma.

Lemma 4.1. [12] *The size of any independent set in an unit disk graph $G = (V, E)$ is at most* $4|\text{MCDS}| + 1$.

Lemma 4.2. *There are at most two GRAY nodes on a shortest path between two neighboring trees. Consequently, at most two GRAY nodes have to change their color to BLACK in order to connect two neighboring trees.*

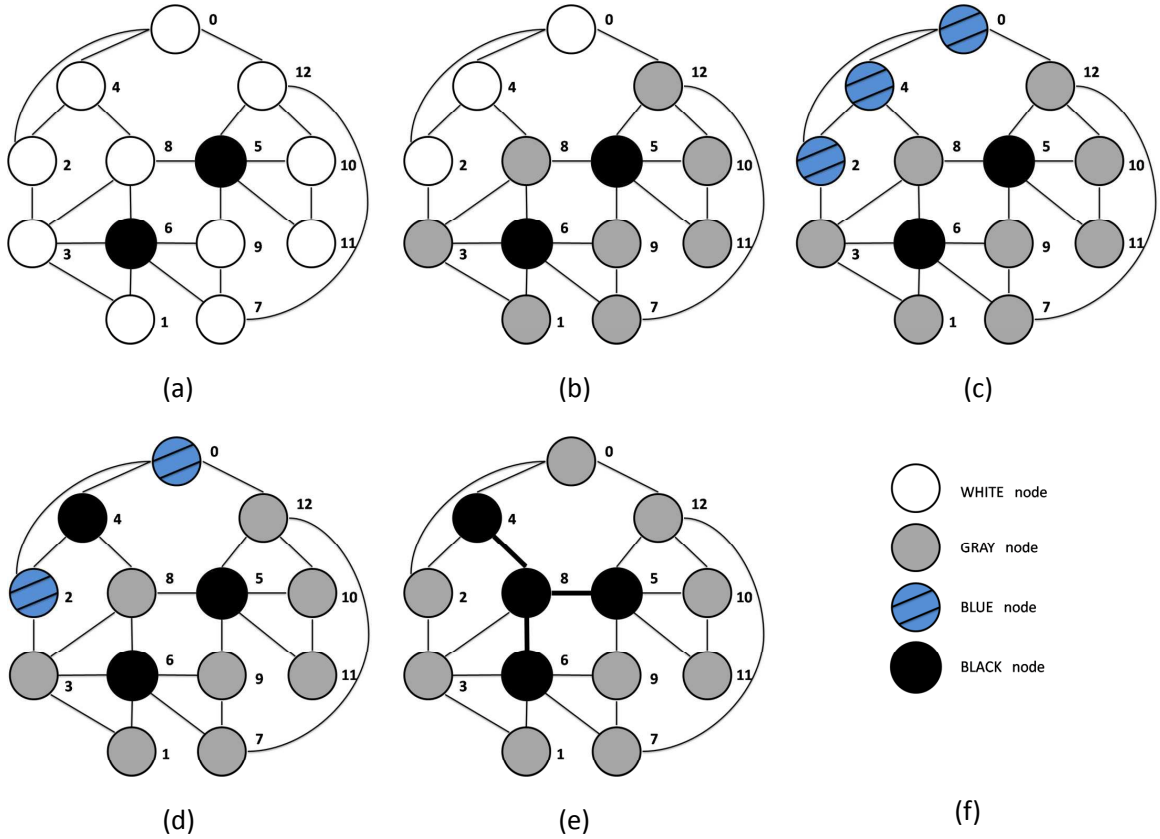

Figure 4: An example of our-algo.

Proof. This lemma follows from Step 4 of Stage 2. ∎

Lemmas 4.1 and 4.2 will be used in the following analysis.

Theorem 4.3. *For unit disk grpahs, the size of the connected dominating set generated by our-algo is at most* $8|\text{MCDS}| + 3k$*, where* k *is the number of constrained nodes.*

Proof. Let $\mathcal{D}$ denote the output of our-algo. Recall that our-algo outputs all the BLACK nodes. Since every constrained node is colored BLACK, initially, there are k BLACK nodes. In Stage 1, a node that is not a constrained node becomes a BLACK node only when it satisfies the constraints of step 6 or step 7. First consider step 6. Notice that the BLUE nodes that become BLACK nodes (in step 6) form an independent set. Suppose

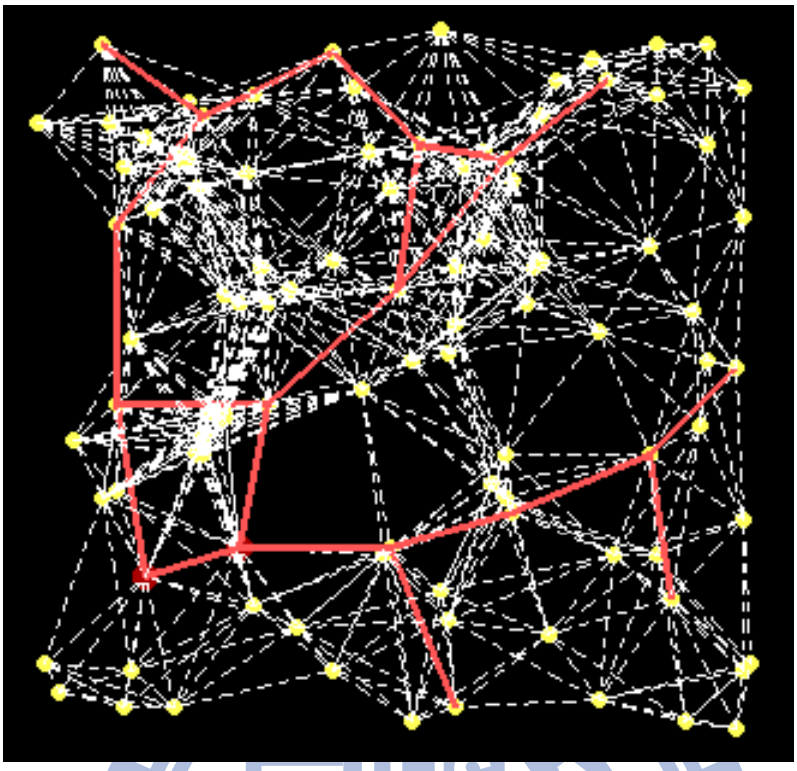

Figure 5: An example of the CCDS obtained by our-algo.

there are r such BLUE nodes. By Lemma 4.1, we have

$$r \leq 4|\text{MCDS}| + 1.$$

Now consider step 7. Notice that the GRAY nodes that become BLACK nodes (in step 7) are parents of the r BLUE nodes that become BLACK nodes in step 6. Therefore, step 6 and step 7 together contribute at most $2r$ BLACK nodes.

Since there are only k constrained nodes, after Stage 1 is executed, there are at most k trees. Thus at most $k-1$ shortest paths are needed to connect the trees to form a CDS. By Lemma 4.2, each shortest path contributes at most two BLACK nodes. Thus $2(k-1)$

Table 1: Performance ratio with respect to the size of k.

k (the number of constrained nodes)	$\leq \frac{\lvert\text{MCDS}\rvert}{3}$	$\leq \frac{\lvert\text{MCDS}\rvert}{6}$	$\leq \frac{\lvert\text{MCDS}\rvert}{9}$
performance ratio	9	$8\frac{1}{2}$	$8\frac{1}{3}$

BLACK nodes are sufficient to connect all the trees. From the above, we have

$$\begin{aligned} |\mathcal{D}| &\leq \underbrace{k}_{\text{constrained nodes}} + \underbrace{r+r}_{\text{step 6 and step 7 of Stage 1}} + \underbrace{2(k-1)}_{\text{Stage 2}} \\ &\leq k + 2(|\text{MCDS}|+1) + 2(k-1) \\ &= 8|\text{MCDS}| + 3k. \end{aligned}$$

■

We have the following corollary. (See also Table 1.)

Corollary 4.4. *For unit disk graphs, when the number of constrained nodes is at most* $\frac{|\text{MCDS}|}{3}$*, the performance ratio of our-algo is 9.*

Theorem 4.5. *Our-algo has message complexity* $O(n)$ *and time complexity* $O(n)$*, where* n *is the number of nodes in the entire network.*

Proof. There are four types of messages involved in Stage 1 and two types of messages in Stage 2. Moreover, each node broadcasts each type of messages at most once. So the message complexity of our-algo is $O(n)$. We now analyze the time complexity. In Stage 1, each node broadcast each type of messages at most once; since each broadcast takes unit time, Stage 1 takes $O(n)$ time. Similarly, in Stage 2, each node broadcast each type of messages at most once; since each broadcast also takes unit time, Stage 2 takes $O(n)$ time. Notice that the number of independent ONE-HOP neighbors of each node is at most 5 and the number of independent TWO-HOP neighbors of each node is at most 18. Consequently, a shortest path connecting two neighboring trees could be found in

constant time. Since at most $k-1$ shortest paths have to be found and $k \leq n$, Stage 2 takes $O(n)$ time. Thus the time complexity for our-algo is $O(n)$. ∎

5 Simulation

In this section, we perform simulation to evaluate the performance of our-algo and WL-algo [13]. In addition, we assume that only one constrained node is chosen and it is the node with the maximum degree, meaning that it could cover (or serve) more nodes than the other nodes.

In our simulation, all nodes are scattered randomly in an $A \times A$ square area where A is the length of the square edge; and all nodes have the same transmission range R, meaning that each node can directly communicate with nodes within the distance R. Moreover, only connected graphs are considered and counted in our simulation. There are three major parameters that affect the size of a generated CDS: the number of nodes N of the network, the transmission range R, and the area size $A \times A$. For each setting (N, R, A), 1000 connected UDGs will be generated and the average of the sizes of the 1000 generated CDSs will be the simulation result for this setting. Let *max* and *min* denote the maximum and the minimum value of the simulation results, respectively.

5.1 The effect of the number of nodes N

In this subsection, we fix the transmission range R at $400m$, the area size at $1000 \times 1000m^2$ and we vary the number of nodes N from 40 to 200 with the increment of 10. Figure 2 shows the simulation results of WL-algo and our-algo. From Figure 2, we observe that for WL-algo, the size of the generated CDS is in proportion to the number of nodes;

however, for our-algo, the size of the generated CDS increases very slowly. In particular,

$$|max - min| = \begin{cases} 115.12 & \text{for WL-algo} \\ 2.34 & \text{for our-algo} \end{cases}.$$

We also observe that that both of WL-algo and our-algo, the size of the generated CDS increase as the number of nodes increase. This is because when the number of nodes increases, the number of nodes that have to be dominated increases.

5.2 The effect of the transmission range R

In this subsection, we fix the number of nodes N at 30, the area size at $1000 \times 1000m^2$ and the transmission range R varies from $250m$ to $600m$ with the increment of $50m$. Figure 6 shows the simulation results of WL-algo and our-algo. From Figure 6, we observe that

$$|max - min| = \begin{cases} 13.26 & \text{for WL-algo} \\ 12.12 & \text{for our-algo} \end{cases}.$$

Although the value of $|max - min|$ is very close for WL-algo and our-algo, the size of the CDS generated by our-algo is only about 49% of that generated by WL-algo. Figure 6 shows that the size of CDS decreases as the transmission range increases. This is because when the transmission range increases, the number of nodes that could be covered (hence dominated) by a node increases and therefore the size of the CDS could be decreased.

5.3 The effect of the area size

In this section, we fix the number of nodes N at 30, the transmission range at $400m$ and the area size varies from $400 \times 400m^2$ to $1400 \times 1400m^2$ with the increment of $100m$ in the length of the square area. Figure 7 shows the simulation results of WL-algo and our-algo. From Figure 7, we observe that

$$|max - min| = \begin{cases} 22.52 & \text{for WL-algo} \\ 12.03 & \text{for our-algo} \end{cases}.$$

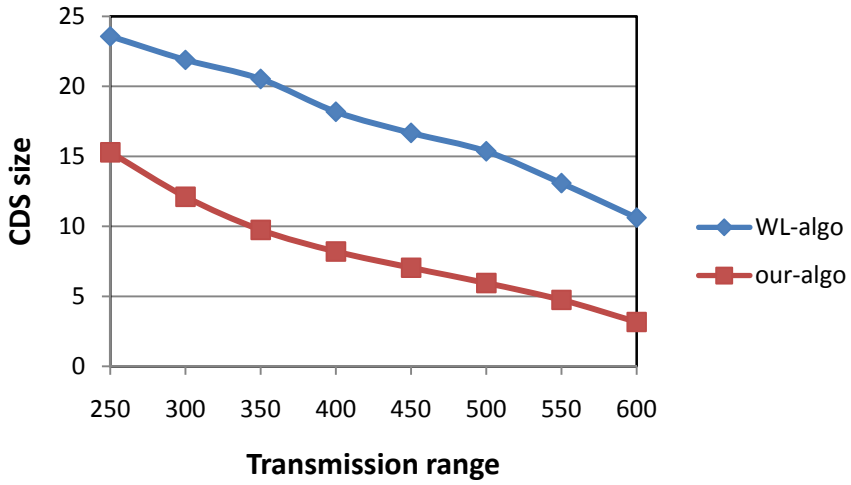

Figure 6: The effect of the transmission range R.

Moreover, the size of the CDS generated by our-algo is only about 45% of that generated by WL-algo. Figure 7 shows that the size of CDS increases as the area size increases. This is because when the area size increases, the network becomes more and more sparse; so the number of neighbors of a node decreases, a node can cover less nodes, and consequently more nodes have to be in the CDS in order to dominate the entire network.

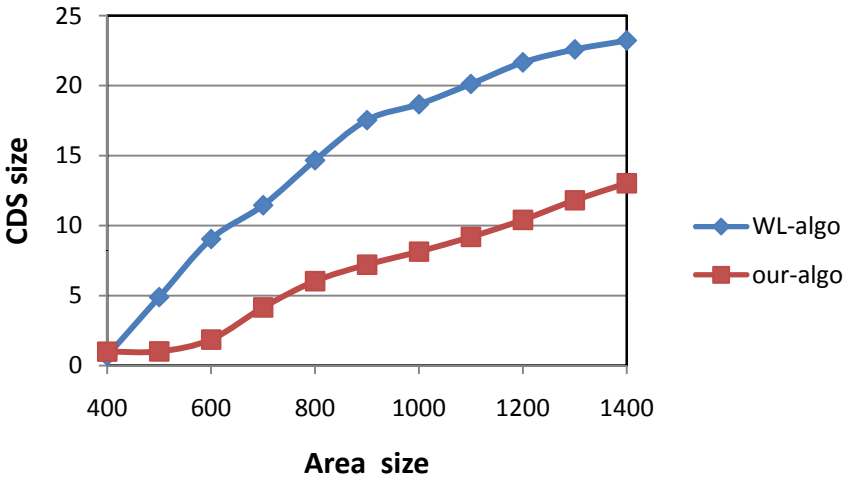

Figure 7: The effect of the area size.

6 Concluding remarks

In this paper, we have introduced a new idea for the connected dominating set: the *constrained connected dominating set*; which ensures that some specified nodes must be in the connected dominating set. The specified nodes could be nodes with more remaining energy or nodes located at important locations in a city or nodes that could larger degree (meaning that they could serve more nodes). We have proposed an algorithm with linear time complexity and linear message complexity; the performance ratio of our algorithm is $8|MCDS| + 3k$, where k is the number of constrained nodes. Theoretical analysis and simulation results shows that our algorithm is pretty good.

References

[1] J. Carle and D. Simplot-Ryl, Energy-efficient area monitoring for sensor networks, IEEE Computer Society 37 (2) (2004) 40-16.

[2] B. Chen, K. Jamieson, H. Balakrishnan and R. Morris, Span: an energyefficient coordination algorithm for topology maintenance in ad hoc wireless networks, ACM Wireless Networks 8 (5) (2002) 481-494.

[3] B. N. Clark, C. J. Colbourn and D. S. Johnson, Unit disk graphs, Discrete Mathematics 86 (1990) 165-177.

[4] F. Dai and J. Wu, On constructing k-connected k-dominating set in wireless networks, IEEE International Parallel & Distributed Processing Symposium (2005) 947-958.

[5] B. Das and V. Bharghavan, Routing in Ad-Hoc Networks Using Minimum Connected Dominating Sets, Proc. of International Conference on Communication (1997) 376-380.

[6] B. Das, R. Sivakumar and V. Bhargavan, Routing in ad hoc networks using a spine, Proc. of ICCCN (1997).

[7] B. Gao, Y. Yang and H. Ma, A new distributed approximation algorithm for constructing minimum connected dominating set in wireless ad hoc networks, International Journal of Communication System 18 (8) (2005) 734-762.

[8] Y. Li, M. T. Thai, F. Wang, C.-W. Yi, P.-J. Wan and D.-Z. Du, On greedy construction of connected dominating sets in wireless networks, Special issue of Wireless Communications and Mobile Computing (WCMC) (2005).

[9] R. Sivakumar, B. Das and V. Bharghavan, An improved spine-based infrastructure for routing in ad hoc networks, Proc. of IEEE Symposium on Computers and Communications (1998).

[10] I. Stojmenovic, M. Seddigh and J. Zunic, Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks, Proc. of IEEE Hawaii International Conference on System Sciences (2001).

[11] P.-J. Wan, K. M. Alzoubi and O. Frieder, Maximal independent set, weakly connected dominating set, and induced spanners for mobile ad hoc networks, International Journal of Foundations of Computer Science 14 (2) (2003) 287-303.

[12] P.-J. Wan, K. M. Alzoubi and O. Frieder, Distributed construction of connected dominating set in wireless ad hoc networks, Mobile Networks and Applications 9 (2) (2004) 141-149.

[13] J. Wu and H. Li, On calculating connected dominating set for efficient routing in ad hoc wireless networks, Proc. of Proceedings of the 3rd ACM International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (1999) 7-14.

[14] R. Xie, D. Qi, Y. Li and J. Z. Wang, A novel distributed MCDS approximation algorithm for wireless sensor networks, Wireless communications and mobile computing 9 (2009) 427-437.

Appendix

```
// File Name: Constrained_CDS.cpp
// Author:  黃思綸
// Email Address: aoc5000@msn.com
// Description: This program iconstructs a constrained connected
//                    dominating set for a unit disk graph.
// Input: An adjancy matrix; this matrix represents an unit disk graph
//           and the entries of this matrix are generated randomly.
// Output: The size of the generated constrained connected dominating set.

#include<iostream>
#include<cstdlib>
#include <iomanip>
#define area 1000   // size area
#include <queue>
#define radius 300; //  點的傳送半徑
using namespace std;

const int SZ=30;      //  點的數量

//  黑點廣播，收到的白點變成灰色
void step1(int node_color[SZ], int matrix[SZ][SZ],int parent[SZ]);

//  灰點廣播，收到的白點變藍色
void step2(int node_color[SZ], int matrix[SZ][SZ],int parent[SZ]);

//  藍點廣播，比較一步內所有藍點的 ID 大小，若是一步內最大的變黑，
//  並且送[INVITE]給自己的 parent
void step3(int node_color[SZ], int matrix[SZ][SZ],int parent[SZ]);

//  判別產生的圖是否連通
int connect(int arr[][SZ]);

//  找出最大 degree 的點
int max_degree(int *degree);

//  每個點的座標(二維)
```

```
struct coordinate
{
	int x;
	int y;
} a[SZ];

// 隨機產生每點的座標
void in(int n,int area_size);

int main()
{
	// 點數
	int n=SZ;

	// simulation 次數
	int times=0;

	// 亂數種子
	srand( time(NULL) );

	// 紀錄已經做的次數,要做 1000 次
	double times_CDS=0;

	// 每次產生的 CDS SIZE 相加,爲了要取平均
	double sum_CDS=0;

	// 點的傳輸半徑
	int r=radius;

	// 兩點距離
	int length=0;

	cout << "Area" << setw(16) << ": " << area << " X " << area << endl;
	cout << "Transmission range: " << r << endl;
	cout << "Number of nodes" << setw(5) << ": " << SZ << endl;

	int matrix[SZ][SZ];
	int i=0,j=0,k=0;
```

```
bool flag=0;

// 紀錄每個點的 degree
int degree[SZ]={0};

// 紀錄點的顏色  0:WHITE 1: GRAY 2: BLUE 3: BLACK
int node_color[SZ]={0};

// constrained node ID
int constrained;

// 紀錄點的 parent
int parent[SZ];

// 計算 CDS SIZE
int cds_size=0;

do
{
	in(n,area);

	// 設定圖的 adjacency matrix
	for(i=0;i<SZ;i++)
		for(j=i+1;j<SZ;j++)
		{
			length=(a[i].x-a[j].x)*(a[i].x-a[j].x)+(a[i].y-a[j].y)*(a[i].y-a[j].y);
			if(length<=(r*r))
			{
				matrix[i][j]=1;
				matrix[j][i]=1;
			}
			else
			{
				matrix[i][j]=0;
				matrix[j][i]=0;
			}
		}
```

```
// 若此圖是連通才往下做，否則略過這次產生的圖
if(connect(&matrix[0]))
        continue;

// 圖是連通則次數加一
times_CDS++;

// 設定每點的 degree
for(i=0;i<SZ;i++)
{
        degree[i]=0;
        for(j=0;j<SZ;j++)
        {
                if(matrix[i][j]!=0)
                        degree[i]++;
        }
}

// 挑出最大 degree 的點當做 constrained
constrained=max_degree(degree);

// 設定每 constrained node 爲 BLACK,其餘爲 WHITE
for(i=0; i<SZ; i++)
{
        node_color[i]=0;
        if(i==constrained)
        {
                node_color[i]=3;

                // 設定 constrained node 的 parent 爲自己
                parent[i]=constrained;
        }
}

// 演算法開始
do
{
```

```
            flag=0;
            step1( &node_color[0], &matrix[0], &parent[0]);
            step2( &node_color[0], &matrix[0], &parent[0]);
            step3( &node_color[0], &matrix[0], &parent[0]);

            // 判別圖是否還有 WHITE 或 BLUE,有的話設爲 1
            for(i=0; i<n; i++)
                if(node_color[i]==0 || node_color[i]==2)
                    flag=1;

            if(flag==0)
                break;
        }while(flag);

        cds_size=0;

        for(i=0;i<SZ;i++)
            if(node_color[i]==3)
                cds_size++;

        sum_CDS+=cds_size;
    }while(times_CDS<1000);

    // 最後的平均值
    double average_size=sum_CDS/times_CDS;

    cout << "Total times" << setw(9) << ": " << times_CDS << endl;
    cout << "Average size" << setw(8) << ": " << average_size << endl;

    system("pause");
    return 0;
}

void step1(int node_color[],int matrix_1[][SZ],int parent[])
{
    for(int i=(SZ-1); i>=0; i--)
    {
            if(node_color[i]==3)
```

```
                {
                    for(int j=0; j<SZ; j++)
                    {
                        if(matrix_1[i][j]==1)
                        {
                            if(node_color[j]==0)
                            {
                                node_color[j]=1;
                                parent[j]=i;
                            }
                            if(node_color[j]==2)
                                node_color[j]=1;
                        }
                    }
                }
        }
}

void step2(int node_color[], int matrix_1[][SZ],int parent[])
{
        for(int i=(SZ-1); i>=0; i--)
            if(node_color[i]==1)
                for(int j=0; j<SZ; j++)
                    if(matrix_1[i][j]==1 && node_color[j]==0)
                    {
                        node_color[j]=2;
                        parent[j]=i;
                    }
}

void step3(int node_color[], int matrix_1[][SZ],int parent[])
{
        bool flag;
        for(int i=0;i<SZ;i++)
        {
                flag=1;
                if(node_color[i]==2)
                {
```

```
                    flag=0;
                    for(int j=0; j<SZ; j++)
                        if(matrix_1[i][j]==1 && node_color[j]==2 && (i<j))
                            flag=1;

                    if(flag==0)
                    {
                        node_color[i]=3;
                        node_color[parent[i]]=3;
                    }
                }
            }
}

int max_degree(int *degree)
{
        int i=0;
        int j=0;
        int max=0;
        for(i=0;i<SZ;i++)
            if(*(degree+i)>=max)
            {
                max=*(degree+i);
                j=i;
            }
        return j;
}

int connect(int arr[][SZ])
{
      int x=0;
      int i=0,j=0;
      int tree_size=0;
      queue<int> qq;
      int color[SZ]={0}; // 0 means unexplored 1 means explored
      qq.push(x);
      bool flag;
```

```
        do
        {
                flag=1;
                for(i=0;i<SZ;i++)
                        if(arr[qq.front()][i]==1)
                        {
                                if(color[i]==0)
                                {
                                        color[i]=1;
                                        qq.push(i);
                                        tree_size++;
                                }
                        }
                 qq.pop();
                 if(tree_size==SZ||qq.empty())
                        flag=0;
        }while(flag==1);

        if(tree_size==SZ)
                return 0;
        else
                return 1;
}

void in(int n, int area_size)
{
        for(int i=0;i<n;i++)
        {
                a[i].x=rand()/32768.0*area_size;
                a[i].y=rand()/32768.0*area_size;
        }
}
```