國立交通大學

應用數學系

碩士論文

用於無線感測網路中達到較高覆蓋率的新適應定位演算法

New Adaptive Localization Algorithms That Achieve Better Coverage for Wireless Sensor Networks

研究生：林紹鈞

指導教授：陳秋媛　教授

中華民國一零二年六月

用於無線感測網路中達到較高覆蓋率的新適應定位演算法

New Adaptive Localization Algorithms That Achieve Better Coverage for Wireless Sensor Networks

研 究 生：林紹鈞　　Student：Shao-Chun Lin

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 2013

Hsinchu, Taiwan, Republic of China

中 華 民 國 一 零 二 年 六 月

用於無線感測網路中達到較高覆蓋率的新適應定位演算法

研究生：林紹鈞　　指導老師：陳秋媛 教授

國 立 交 通 大 學

應 用 數 學 系

摘 要

許多無線感測網路的應用(例如:森林火災的偵測、環境監控、軍事應用、野生動物的追蹤)都需要位置資訊來偵測並且記錄事件發生的地點。如何獲取感測器的位置資訊變成一個重要的問題，並且通常被稱為定位問題。利用全球定位系統(GPS)是一個可能的解決方案，但是並不實際。其原因在於一個無線感測網路經常包含數千個感測器，以致於為每個感測器裝上 GPS 的價格太過昂貴而難以實現這個解決方案。在文獻[3]中，黃等人研究了一個新的最佳化問題，最小花費定位問題(minimum cost localization problem)，這個問題透過利用最小數量的錨點來定位出所有感測器的位置。本篇論文的目的是對於無線感測網路提出可調式演算法的概念。我們的演算法比文獻[3]的演算法簡易，並且考慮了所有文獻[3]的演算法所考慮的情形；實驗數據也顯示我們的演算法有比較好的覆蓋率。

關鍵詞：無線感測網路、定位、演算法、剛性、全域剛性、三角定位。

New Adaptive Localization Algorithms That Achieve Better Coverage for Wireless Sensor Networks

Student: Shao-Chun Lin Advisor: Chiuyuan Chen

Department of Applied Mathematics

National Chiao Tung University

Abstract

Many applications in wireless sensor networks (such as forest fires detection, environment monitoring, military, and wildlife tracking) require position information to detect and record events. How to obtain the position information of the sensors become an important problem and is usually called the localization problem. Using Global Positioning System (GPS) is a possible solution for localization but is impractical. The reason is that a wireless sensor network usually has thousands of sensors and it is too expensive to equip every sensor a GPS. In [3], Huang et al. studied a new optimization problem, minimum cost localization problem, which aims to localize all sensors by using the minimum number of anchors. The purpose of this thesis is to propose adaptive algorithms for the localization problem. Our algorithms are simpler than the algorithms in [3] and cover all the cases in the algorithms in [3]; simulation results also show that our algorithms have better coverage.

Keywords: wireless sensor network, localization, algorithm, rigidity, globally rigid, trilateration, triangulation.

誌　　謝

首先，碩士能夠畢業，我要感謝我的父母，若不是他們不停的支援我，做我的後盾，我也不可能這麼快樂的念大學、研究所。接著非常感謝我的指導老師陳秋媛老師，很感恩她這兩年來對我耐心的指導以及付出，才能夠讓我知道自己有哪些地方需要改進，並且在這兩年不斷的訓練與成長。

另外，在碩士的兩年內，非常感謝修課的老師在我學習的期間教導我莫大幫助的知識，包括了陳秋媛教授、翁志文教授、傅恆霖教授、符麥克教授、滕元翔老師，除此之外，系辦的張慧珊姊姊、張麗君姊姊、陳盈吟姊姊也都在我需要幫忙時，給了我必要的協助。

在研究的路上，常常感到許多挫折，除了老師們的支持之外，同學們也給了不少的幫助，尤其是邱鈺傑學長、林武雄學長、袁智龍學長、王奕倫學長、呂竺晏、趙彥丞、蕭禕廷、林立庭、徐志杰、王稟鈞、陳珈惠、吳惠敏、姚欣豪等同學，都在我報告以及課堂學習中，給我非常有用的建議，並提醒我的缺失，若不是他們的幫助，我也不可能順利的完成我的碩士論文；在他們的身上，也看到了許多值得學習的優點，能夠和這些同學在同一個團體奮鬥，是我引以為榮的。

Contents

List of Figures

1 Introduction

Wireless sensor networks (WSNs) have many applications such as forest fires detection, environment monitoring, military, and wildlife tracking. In many applications, the sensors (nodes) need to know their positions in order to detect and record events. In this thesis, we use the terms *sensor* and *node* interchangeably. The process of computing the positions of the nodes is called *localization*. This thesis focus on the localization problem. Here we require a fine-grain location awareness although some researchers considered a coarse-grain location awareness; see [3].

It is not difficult to see that one way to solve the localization problem is to equip each sensor a GPS. Since a wireless sensor network usually has thousands of sensors, this approach is too expensive and therefore impractical. Novel approaches have been proposed to choose some sensors and equip them with GPS and the other sensors determine their positions by means of the Euclidean distances to their neighbors using different distance ranging methods (such as radio signal strength or time difference of arrival).

A wireless sensor network is usually modeled as a graph $G = (V, E)$ and many often, a *unit disk graph* is used, meaning that each node has the same transmission range. Before going further, we introduce some terminologies. A graph G is *generic* if it has no three nodes collinear. A graph G is *grounded* if $uv \in E$ implies that the distance between u, v can be measured or estimated via wireless communication. In this thesis, we assume the wireless sensor network forms a generic, grounded, unit disk graph; see also [8].

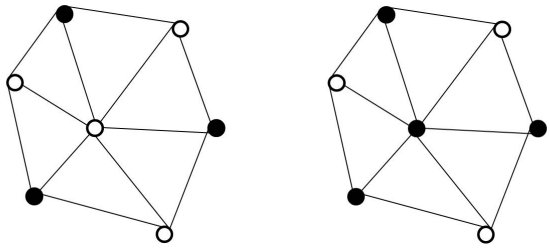
Two nodes are *neighbors* if they are adjacent. For convenience, let $N(v)$ denote the set of neighbors of a node v and call $N(v)$ the *neighborhood* of v. Many researchers call a node which is equipped with GPS an *anchor* or *anchor node* or *beacon*. In this thesis, however, for clarity, a node which is equipped with GPS is called an *initial-anchor*; a node which is not an initial-anchor and has determined its location is called an *obtained-anchor*. Both initial-anchors and obtained-anchors are called *anchors*. For each node v, only the

distances between u, v where $u \in N(v)$ and the positions of anchors in $N(v)$ can be used to determine the location of v. For convenience, let S denote the set of all initial-anchors and call S the *initial-anchor set*. The cardinality of S is denoted as $|S|$.

Trilateration is the most basic approach for localization and it has been used for thousands of years. This approach uses the known positions of anchors and the measured distance to the anchors. To determine the location of a node using trilateration alone, this node needs to hear from at least three anchors. That is, a sensor v computes the number of anchors in its neighborhood $N(v)$ and if v has three anchors in $N(v)$, then the location of v can be determined. In many localization algorithms, *iterative trilateration* (or *multilateration*) is used to localize nodes. In iterative trilateration, nodes measure distances to their neighbors and share their position information with their neighbors to collaboratively compute their positions. If a sensor whose position has been determined, it can act as a new anchor to localize other nodes by sharing its position with its neighbors. This iterative process continues until no nodes can be further localized.

How to find a good initial-anchor set S for a wireless sensor network is obviously an interesting problem. Do notice that finding an initial-anchor set for trilateration is *not* equivalent to finding a 3-dominating set. A 3-dominating set can serve as an initial-anchor set for trilateration but an initial-anchor set for trilateration needs not to be a 3-dominating set. The following figure shows an example S with $|S| = 3$ and a 3-dominating set requires four nodes.

A graph G is *localizable* if the position of each node can be uniquely determined. A formulation of the localization problem is: given a graph G and an initial-anchor set S, determine if G is localizable by using S. We say S is *feasible* if the position of each node in G can be uniquely determined with S being the initial-anchor set. Some researchers do not ask for 100% of sensors being localized. In this situation, *coverage* is considered and is defined to be the percentage of sensors that can be localized.



(a) (b)

Figure 1: (a) A initial-anchor set S for trilateration, which is not a 3-dominating set. (b) A 3-dominating set for the graph in (a) requires at least four nodes.

Most localization algorithms focus on determining whether a wireless sensor network is localizable and/or how to localize as many nodes as possible with a "static" initial-anchor set; see [1, 5, 7, 13, 11, 12, 14, 15, 17]. Recently, in [8], Huang et al. studied a new optimization problem, minimum cost localization problem, which aims to localize all sensors using the minimum number of anchors given the distance measurements. The algorithm proposed in [8] is called *sweep* and it first applies trilateration; when no nodes can be further localized, it switches to consider pairs of sensors u, v such that u, v are neighbors and they can cooperate to obtain their positions. After no such u, v pairs can be found, the algorithm switches back to trilateration again, and so on. We will describe the algorithm *sweep* in detail in Sections 2 and 3.

This thesis considers the same problem as in [8], i.e., the minimum cost localization problem. The main contribution of this thesis is to propose an adaptive algorithm to choose an initial-anchor set S. For convenience, let S and S' denote the initial-anchor sets generated by our algorithm and by the algorithm in [8], respectively. Simulation results show that:

1. Our algorithm always has a better coverage than the algorithm in [8].

2. For dense graphs, S always has a smaller cardinality than S'.

3. For sparse graphs, the cardinalities of S and S' are very similar.

This thesis is organized as follows. In Section 2, we give an overview of related works. The main result is presented in Section 3. Simulations and comparisons are given in Section 4. Concluding remarks are given in the final section.

2 Related works

In this section, we review related works with respect to localization and this section consists of three subsections. In Subsection 2.1, we give results of "rigidity theory". In Subsection 2.2, we review the results of coarse-grain localization. In Subsection 2.3, we briefly describe previous results of fine-grain localization.

2.1 Rigidity theory

In [6], Hendrickson proved that if G has a unique realization in R^d, then G is $(d+1)$-connected and redundantly rigid (defined later). He also conjectured that every realization of a $(d+1)$-connected and redundantly rigid graph in R^d is unique. Hendrickson's conjecture is true for $d = 1$ but was disproved by Connelly for $d \geq 3$ in [4]. In [9], Jackson and Jordsan resolved the remaining open case by showing that Hendrickson's conjecture is true for $d = 2$. Jackson and Jordsan also deduced that every realization of a 6-connected graph as a two-dimensional generic framework is a unique realization.

A graph G is *redundantly rigid* in R^d if deleting any edge of G results in a graph which is rigid in R^d. It is a long story to give a formal definition for "rigid"; therefore we will adopt the following famous theorem, which was proposed by Laman.

Theorem 2.1. [2, 10] *Laman's Condition*

A graph $G = (V, L)$ with n vertices is rigid in R^2 if and only if L contains a subset E consisting of $2n - 3$ edges with the property that, for any nonempty subset $E' \subset E$, the number of edges in E' cannot exceed $2n' - 3$, where n' is the number of vertices of G which are endpoints of edges in E'.

Notice that in many literatures (for example [2, 9]), if a grounded graph has a unique localization solution then it will is called *globally rigid*. In [9], Jackson and Jordsan presented a method to determine if a graph has a unique localization solution.

Theorem 2.2. [2, 9] *A grounded graph G is globally rigid in R^2 if and only if either G is a complete graph on at most three vertices or G is 3-connected and redundantly rigid.*

The following theorem gives a sufficient condition for a graph to be globally rigid. Note that in [1], Ferruccio et al. tried to characterize a series of graphs which can be easily localized and they proved that a 6-connected graph can be localized by trilateration with three initial-anchors.

Theorem 2.3. [9] *If a graph G is 6-connected, then G is globally rigid in R^2.*

Intuitively, if a graph is non-rigid, then it may have an infinite number of localization solutions. When a graph is rigid, it has a finite number of localization solutions. When a graph is globally rigid, it has a unique localization solution. We use Figure 2 to illustrate the definition of rigidity. The graph in Figure 2(a) is non-rigid, in Figure 2(b) is rigid since it satisfies Laman's Condition, in Figure 2(c) is redundantly rigid since if we deleted any edge then the resultant graph is still rigid. Note that the graph in Figure 2(c) is also globally rigid since it is 3-connected and redundantly rigid.

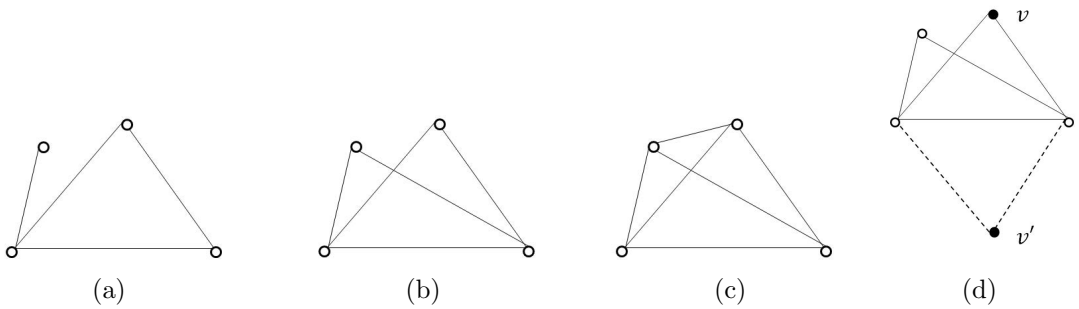


Figure 2: Examples of a (a) non-rigid, (b) rigid, (c) redundantly rigid (also globally rigid) graph. (d) A rigid graph may have two localization solutions.

From the above, the localization problem can be transformed into the problem of determining the globally rigidity of a graph, i.e., determining whether a graph is 3-connected and redundantly rigid. It is not difficult to check if a graph is 3-connected. As for the redundantly rigidity, in [7], Hendrickson proposed a polynomial-time algorithm (for convenience, call this algorithm *H-Algorithm*) to determine the redundantly rigidity of a graph. Note that when *H-Algorithm* was proposed, it is not known that this algorithm could be used to solve the localization problem. Until Jackson and Jordsan proved Theorem 2.2 in [9], Connelly [5] combined the results of [7] and [9] (i.e., *H-Algorithm* and Theorem 2.2) and proposed an algorithm to determine if a graph is localizable. Connelly even re-proved the correctness of *H-Algorithm*. For convenience, we call Connelly's method *C-Algorithm*. C-Algorithm is centralized and is a divide-and-conquer algorithm.

The rigidity theorem has two major drawbacks for the localization problem: (i) It characterizes the graphs which can be localized by exactly three anchors, but it does not characterize graphs which can be localized by four or more anchors. (ii) It provides a theoretical method (the answer is only "Yes" or "No"), not an algorithmic method, for the localization problem. Note that a wireless sensor network may not have a certain structure and its corresponding graph may not globally rigid. Thus it is very natural to ask: "How to localize a graph which is not globally rigid?" In [17], Yang et al. tried to use the wheel graph structure to solve the localization problem for a graph which is not necessarily globally rigid. They pointed out that if a wheel structure (in the given graph) contains three or more anchors, then this wheel structure is localizable since a wheel is 3-connected and redundantly rigid (hence is globally rigid). It is also natural to ask: "If there exist more than three anchors, then what kind of graphs can be localized?". Several literatures have ever discussed the question.

2.2 Coarse-grain localization

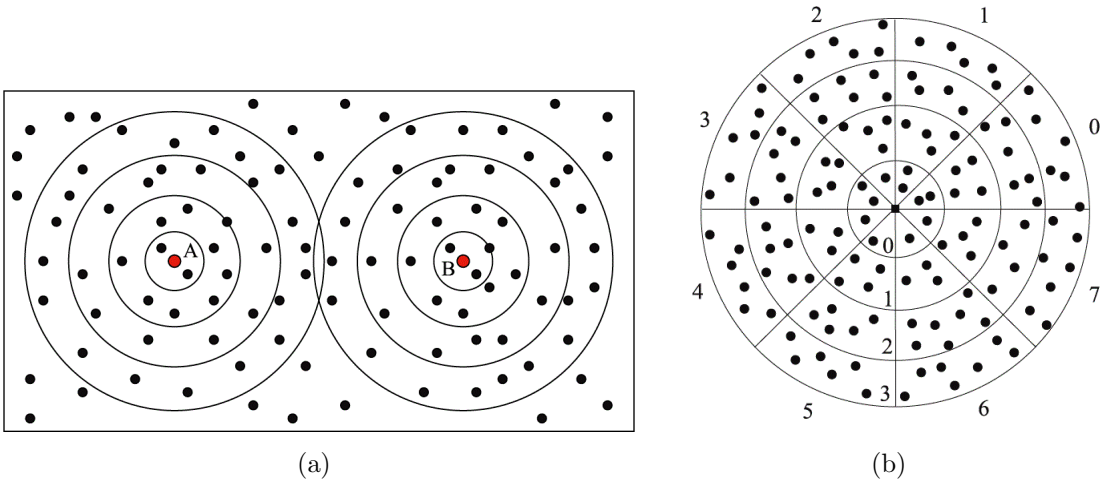
In some applications, a wireless sensor and actor network (WSAN) is considered, which consists of massively and randomly deployed tiny sensors and a few *actors* that organize the sensors in their vicinity into a short-lived *actor-centric* network to support a specific mission. After deployed, these tiny and low-cost sensors are unaware of their location and are unattended. Actors are mobile to collect the sensed data from sensors. Each actor is equipped with better processing capabilities, higher transmission power to send broadcasts for a distance, and a longer battery life than the sensors.

In a WSAN, a coarse-grain location awareness is sufficient with a trade-off: that an coarse-grain location awareness is lightweight, but the resulting positioning accuracy is only a rough approximation of the exact geographic location. *Training* is referred to the task of allowing each sensor to acquire a coarse-grain location. Wadaa et al. [16] first proposed a training protocol in which each actor trains sensors in its vicinity, namely, the *actor-region*, to associate these sensors with coarse-grain coordinates related to the actor. Recently, in [3], Barsi et al. proposed an energy-efficient training protocol. After training, each sensor in the actor-region will acquire two coordinates: the *corona* and the *sector* to which it belongs. See Figure 3 for an illustration. A training protocol provides for free a *clustering* of the sensors and a virtual infrastructure, where a *cluster* consists of all sensors having the same coordinates.

More precisely, a training protocol imposes a virtual coordinate system onto the WSAN by establishing:

1. *Coronas*: The actor-region is divided into k coronas $C_0, C_1, \ldots, C_{k-1}$ determined by k concentric circles of radii $r_1 < r_2 < \cdots < r_k$ centered at the actor.

2. *Sectors*: The actor-region is divided into h equiangular sectors $S_0, S_1, \ldots, S_{h-1}$, originated at the actor, each having a width of $\frac{2\pi}{h}$ radians.



(a) (b)

Figure 3: (a) A wireless sensor and actor network with two actors A and B. (b) A trained subnetwork. The source of this figure is [3].

2.3 Fine-grain localization

Different from the coarse-grain localization, a fine-grain localization determines the position of each sensors. This position may be related to the earth or a virtual coordinate system on the given network. In a localization algorithm, a sensor can acquire its position by two methods: (i) using GPS and (ii) by calculation (for example, using trilateration).

We now give the details of trilateration. In this localization algorithm, a sensor (say, v) computes the number of anchors in $N(v)$. If this number is ≥ 3, then v uses three of the anchors (say, a_1, a_2, a_3) to compute its own position. First use the distances a_1v and a_2v to construct two possible positions for v. Then use the distance a_3v to determine which of the two possible positions of v is correct. Notice that trilateration will fail if the three anchors a_1, a_2, a_3 are collinear. Thus to use trilateration, the given graph has to be a generic graph.

As mentioned in the previous section, in [8], Huang et al. considered the minimum cost localization problem. They proposed an algorithm called *sweep*. This algorithm first applies trilateration; when no nodes can be further localized, it switches to consider pairs

of sensors u, v such that u, v are neighbors and they can cooperate to obtain their positions. After no such u, v pairs can be found, the algorithm switches back to trilateration again, and so on. More precisely, when u, v satisfy the following three conditions, they can cooperate to fulfill localization: (i) u and v are neighbors (hence the distance between them is known), (ii) $N(u)$ contains two anchors (say, a_1 and a_2) and $N(v)$ also contains two anchors (say, a_3 and a_4), and (iii) $|\{a_1, a_2\} \cap \{a_3, a_4\}| \leq 1$ (meaning that the anchors used by u cannot be identical to those used by v). It is not difficult to see that if u, v satisfy (ii), then both u and v have two possible positions. If u, v do not satisfy (iii), then there are at least two possible pairs of positions and hence cannot be uniquely localized. If u, v satisfy (iii), then it is possible that only one pair of positions match the distance between u and v, which can be obtained if u, v satisfy (i).

Some of the algorithms for the localization problem are centralized, while the others are distributed. In centralized algorithms, some nodes have a more powerful computation ability; hence these nodes can gather more information of the given graph such as the maximum and the average degree of nodes. While in distributed algorithms, all sensors have a limited power and computation ability; hence sensors can only gather the information within the vicinity.

In [11], Langendoen and Reijers compared three distributed algorithms that handle the localization problem when there are noises. One of the algorithms uses the bounding box approach to approximate the position of sensors. Another one uses the average distance approach to compute the distance between pairs of sensors, and then use this distance information to compute the position of sensors. The remaining one calculates the distance between anchors and sensors and uses the standard least square method to compute approximate positions; due to the the existence of noise, solutions may not exist.

We briefly describe the difference between algorithms that assume noise-free and algorithms that allow the existence of noise. The first difference is that: for the latter

algorithms, the estimated distances may not be accurate due to the existence of noise. These algorithms have to use the distance information (which may not be accurate) to obtain positions which are closest to the actual positions. The second difference is that: for the former algorithms, the positions of anchors are only shared with one-hop neighbors, but for the latter algorithms, the positions of anchors are shared within two-hop or three-hops or n-hop neighbors for some n. In [13], an n-hop multilateration algorithm allowing the existence of noise was proposed and the positions of anchors are shared within n-hop neighbors. Note that there are various methods for estimating the distance between sensors and anchors within n-hop neighbors.

There are three representative algorithms that can localize a graph when there exist noises; see [14, 15, 12]. In [11], Langendoen and Reijers provided a comparison between these three algorithms. As was mentioned before, there are various methods for estimating the distance between sensors and anchors within n-hop neighbors. Each of the algorithms in [14, 15, 12] provides an interesting method to estimate the distance between sensors and anchors within n-hop neighbors. Before ending this subsection, we briefly describe the methods used in [14, 15, 12].

In [14], Savarese et al. proposed a method called DV-hop. In DV-hop, a sensor (say, v) computes the number k of hops between an anchor (say, a) and itself. Then, the distance between v and a is estimated by the average distance for one hop times k.

In [15], Savvides et al. proposed a method and it was called Sum-Dist in [11]. In Sum-Dist, anchors initiate a flood. More precisely, Sum-dist starts its process from the anchors, which send messages including the anchor id, anchor position, and a path length 0. Each sensor that receives the message adds its measured range to the path length. To avoid the flooding of messages, Savvides et al. limited the flooding range. At the end of the above process, most sensors will get several anchors' positions and several estimated distances.

One drawback of DV-hop is that it fails for highly irregular network topologies, where the variance in actual hop distances is very large. In [12], Niculescu and Nath have proposed another method, called Euclidean, which is based on the local geometry of the nodes around an anchor. Again, anchors initiate a flood, but forwarding the distance is more complicated than [14, 15]. In particular, when a node has received messages from two neighbors that know their distance to the anchor, and to each other, it can calculate the distance to the anchor. We use Figure 4 to describe the method Euclidean. This figure shows a node (i.e., Self) that has two neighbors n1 and n2 with distance estimates (a and b) to an anchor (i.e., Anchor). Note that Self is not a neighbor of Anchor and currently does not have a distance estimate to Anchor. Self and Self' are actually the two possible positions for the same node. The nodes n1, n2, and Self (or n1, n2, and Self') form a K_3 and therefore the distances c, d, and e are known. In this figure, *r1* and *r2* denote the distance estimates of Self to Anchor and Self' to Anchor, respectively. The method Euclidean uses *neighbor vote* or *common neighbor* to determine which of *r1* and *r2* is correct.

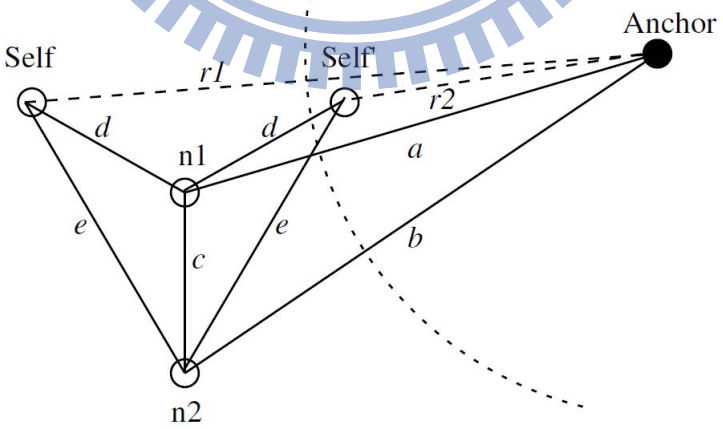


Figure 4: An illustration of the Euclidian method; this figure is from [11].

After each node obtains a distance estimate, an algorithm is needed to calculate the

position of each node. We now introduce two algorithms to calculate positions. The two algorithms are called *Lateration* and *Min-max*. In *Lateration*, each distance estimate contributes a constraint; using all the constraints, one can use the standard least square method to obtain an approximate position of a node.

In *Min-max*, a node (say, v) that needs to be localized will compute a region that it belongs to; this region is actually a rectangle and is called a *bounding box*. Figure 5 shows that v is localized by using (i) the positions of three anchors (called Anchor1, Anchor2, and Anchor3 in this figure) and (ii) the three estimated distances between Anchor1 and v, Anchor2 and v, and Anchor3 and v. For convenience, denote the three estimated distances as d_1, d_2, d_3, respectively. Let the positions of Anchor1, Anchor2, and Anchor3 be (a_1, b_1), (a_2, b_2), and (a_3, b_3), respectively. Let (x, y) denote the position of v. Then x lies in the intersection of the following three intervals:

$$[a_1 - d_1, a_1 + d_1],\ [a_2 - d_2, a_2 + d_2],\ [a_3 - d_3, a_3 + d_3].$$

The values

$$min_x = \max\{a_1 - d_1, a_2 - d_2, a_3 - d_3\} \text{ and } max_x = \min\{a_1 + d_1, a_2 + d_2, a_3 + d_3\}$$

are calculated to bound the x-coordinate of v. Similarly, y lies in the intersection of the following three intervals:

$$[b_1 - d_1, b_1 + d_1],\ [b_2 - d_2, b_2 + d_2],\ [b_3 - d_3, b_3 + d_3].$$

The values

$$min_y = \max\{b_1 - d_1, b_2 - d_2, b_3 - d_3\} \text{ and } max_y = \min\{b_1 + d_1, b_2 + d_2, b_3 + d_3\}$$

are calculated to bound the y-coordinate of v. The above process also explains why this algorithm called *Min-max*. After calculating min_x, max_x, min_y, and max_y, the algorithm chooses $(\frac{min_x + max_x}{2}, \frac{min_y + max_y}{2})$ as the estimated position of v; this is the node labeled "est. pos." in Figure 5.

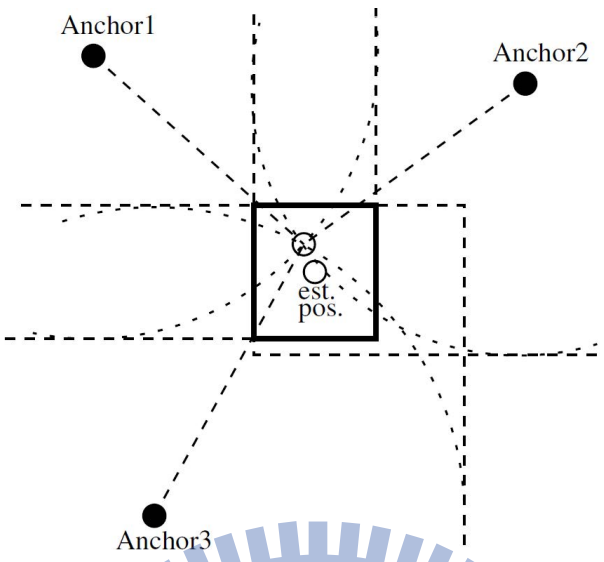


Figure 5: A illustration of the *Min-max* method; this figure is from [11].

3 The main result

This section gives the main result of this thesis. In particular, an algorithm called *Main* is proposed; see below. Throughout this section, $v.ann$ denotes the number of anchors in $N(v)$.

Algorithm *Main* uses the Boolean variable $v.localized$ to indicate whether a node v has been localized. Initially, all nodes v have $v.localized = false$. Once v is localized, $v.localized$ is set to *true*. Algorithm *Main* uses the following theorem to choose its initial-anchor set S; see lines 5-10 in *Main*.

Theorem 3.1. *Let $G = (V, E)$ be a graph that models a given wireless sensor network and S be any feasible initial-anchor set of G. For all $v \in V$ with degree $deg(v) \leq 2$, we have $v \in S$. In other words, any feasible initial-anchor set must contain all nodes with degree ≤ 2.*

Proof. Suppose to the contrary that S is a feasible initial-anchor set and there is a node $v \notin S$ such that $deg(v) \leq 2$. Since S is feasible, v must be localized. However, if $deg(v) = 1$, then v has an infinite number of possible positions, which is a contradiction;

Algorithm 1 *Main*

Require: A wireless sensor network $G = (V, E)$.
Ensure: A localization for G and the initial-anchor set S that is used in the localization.

```
1:  S = ∅;
2:  for each v ∈ V do
3:      v.localized = false;
4:  end for
5:  for each v ∈ V do
6:      if v.degree ≤ 2 then
7:          S = S ∪ {v};
8:          v.localized = true;
9:      end if
10: end for
11: run Localization-Phase;
12: while there exists any u such that (u.localized == false) do
13:     v = the result of AnchorChoose-Phase;
14:     S = S ∪ {v};
15:     run Localization-Phase;
16: end while
```

if $deg(v) = 2$, then v has at least two possible positions, which is also a contradiction. □

Then, Algorithm *Main* runs the *Localization-Phase*; see line 11 and Subsection 3.1. After the *Localization-Phase*, if there exits any node which is not localized, then Algorithm *Main* runs *AnchorChoose-Phase* to choose a node v as a new initial-anchor and add v to the initial-anchor set S; see lines 13-14 in *Main*. After v is added to S, Algorithm *Main* runs *Localization-Phase* again. Notice that we have proposed an adaptive algorithm for *AnchorChoose-Phase* and we will give the details in Subsection 3.2. The while-loop in lines 12-16 repeats until all nodes have been localized. When Algorithm *Main* stops, the initial-anchor set S is feasible.

3.1 Localization-Phase

Notice that the *Localization-Phase* can be implemented in many ways. In this thesis, we give three localization algorithms (called *Trilateration*, *Sweep2*, and *Rigid*) and each of them can be used in the *Localization-Phase*.

3.1.1 Trilateration

The pseudocode of *Trilateration* is as follows.

Algorithm 2 *Trilateration*

Require: A wireless sensor network $G = (V, E)$.
Ensure: G with those nodes of it that can be localized by trilateration.

```
1: for each v ∈ V such that (v.localized == false) do
2:     compute v.ann;
3: end for
4: for each v ∈ V such that (v.localized == false) do
5:     if v.ann ≥ 3 then
6:         use three anchors in N(v) to determine the position of v;
7:         v.localized = true;
8:         run Trilateration;
9:     end if
10: end for
```

3.1.2 Sweep2

We have mentioned the algorithm *sweep* that was proposed in [8] in Subsection2.3. In [8], algorithm *sweep* is implemented by either *Greedy-Sweep-1* or *Greedy-Sweep-2*; thus there are two cases. Notice that *Greedy-Sweep-2* actually contains all the statements in *Greedy-Sweep-1*. We find that it is unnecessary to have both *Greedy-Sweep-1* and *Greedy-Sweep-2*. Therefore, in this thesis, we modify algorithm *sweep* and propose a new algorithm called *Sweep2*. Our algorithm *Sweep2* is more general and it does not need to distinguish between the two cases in *sweep*. Similar to *Trilateration*, algorithm *Sweep2* will compute the number of anchors in a node's neighborhood. If there are two neighboring nodes u, v such that both $u.annandv.ann$ equal to 2, then it is possible to localize u, v at once. Let the two possible positions for u be u_1, u_2 and the two possible positions for v be v_1, v_2. Then algorithm *Sweep2* can check if only one of the four distances u_1v_1, u_1v_2, u_2v_1, and u_2v_2 equals to the distance uv. If this is true, then u, v can be localized at once. The pseudocode is now given below.

Algorithm 3 *Sweep2*

```
1: for each v ∈ V such that (v.localized == false) do
2:     Compute v.ann;
3: end for
4: run Trilateration;
5: for each v ∈ V such that (v.localized == false) do
6:     if v.ann == 2 then
7:         for each u ∈ N(v) do
8:             if (u.ann == 2) and (u.localized == false) then
9:                 compute the two possible positions for u and denote them as u_1, u_2;
10:                compute the two possible positions for v and denote them as v_1, v_2;
11:                compute the distances u_1v_1, u_1v_2, u_2v_1, u_2v_2, and uv;
12:                if ∃! distance (say, u_iv_j) in {u_1v_1, u_1v_2, u_2v_1, u_2v_2} equals to uv then
13:                    use the distances u_iv_j and uv to determine the positions of u, v;
14:                    u.localized = v.localized = true;
15:                    run Sweep2;
16:                end if
17:            end if
18:        end for
19:    end if
20: end for
```

In *Sweep2*, two nodes u, v each has only two anchors in their neighborhood but u, v try to cooperate to obtain their positions. The idea of *Sweep2* can be generalized to obtain *Sweep3*, *Sweep4*, ..., *Sweepk*. In general, *Sweepk* means k nodes try to cooperate to obtain their positions.

3.1.3 Rigidity

In this subsubsection, we propose a localization algorithm based on the rigidity theory; call this algorithm *Rigidity*. When algorithm *Main* runs, all nodes with degree ≤ 2 will be added into the initial-anchor set. Suppose *Main* uses *Rigidity* in its *Localization-Phase*. Then, each initial-anchor broadcasts a "Hello" message to all of its neighbors. Each node v (including the anchors) that receives a "Hello" message run the algorithm *Trilateration* at first and then uses *C-Algorithm* to find a rigid sub-graph H in $N[v]$ if there exists a such graph. Algorithm *Rigidity* computes the position of a node by a method similar to

the one used in *Sweepk*. This solution will be unique since the subgraph H is globally rigid. The pseudocode of algorithm *Rigidity* is as follows.

Algorithm 4 *Rigid*

```
1: for each v ∈ V do
2:     if v.localized == true then
3:         v broadcasts a "Hello" message to all of its neighbors;
4:     end if
5: end for
6: for each v ∈ V which receives the "Hello" message do
7:     run Trilateration;
8:     use C-Algorithm to find a globally rigid graph in N[v], containing at least non-
       anchor node, say this graph as H;
9:     if there exists three anchors in H then
10:        use the Sweepk to localize each u ∈ H;
11:        for each u ∈ H and u.localized == false do
12:            u.localized = true;
13:            broadcast a "Hello" message to all of its neighbors;
14:        end for
15:    end if
16: end for
```

Figure 6 shows how *Rigid* algorithm works. First, u finds there are three anchors in $N(u)$ and uses *Trilateration* to localize itself (see Figure 6(a) and (b)). Then, u will determine whether $N[u]$ contains a globally rigid graph with at least one non-anchor node. *Rigid* finds there exists a wheel graph which is globally rigid in $N[v]$ and computes the position of each node. Finally, all nodes of the wheel graph are localized (see Figure 6(c)).

3.2 AnchorChoose-Phase

In [8], Huang et al. also proposed a method to choose initial-anchors; for convenience, we call it *HuangChoose*. In this subsection, we will propose an adaptive algorithm for choosing anchors. We first give our observations and motivations. After the execution of line 11 in algorithm *Main*, there may exist nodes that have not been not localized yet. In order to localize all of the nodes, more initial-anchors are needed. How to choose a node which is not localized as an initial-anchor is obviously an important issue.

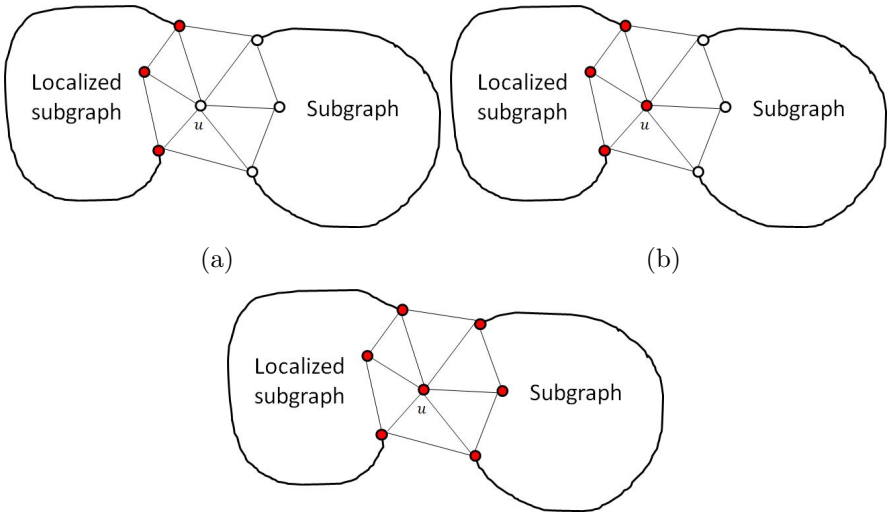


Figure 6: Example of Rigid Algorithm.

Intuitively, among the non-localized nodes, a node v with the maximum degree should be chosen since v makes the largest number of nodes have one more anchor in their neighborhoods. However, simulation results show that nodes that have large degrees usually locate in some regions of the given graph. Therefore, it is very likely that the chosen anchors are from the same region of the graph and could not help with localizing nodes in the other regions. To overcome this difficulty, we will use an adaptive algorithm to choose initial-anchors. Let $v.degree$ denote the degree of a node v. More precisely, instead of using $v.degree$, algorithm *AdpativeChoose* will use $(v.degree - v.ann)$. For instance, if $v.degree$ is 8 and $v.ann$ is 3, then algorithm *AdpativeChoose* will treat $v.degree$ as 5. A node v with the maximum $(v.degree - v.ann)$ will be chosen as an initial-anchor. The pseudocode is as follows.

Figure 7 shows the performance of algorithm *AdpativeChoose*. In Figure 7(a), nodes colored with black are anchors and the others are non-anchors and therefore non-localized. As shown in Figure 7(b), a tradition algorithm will choose u as the next initial-anchor since u has the maximum degree. However, as shown in Figure 7(c), algorithm *AdpativeChoose*

Algorithm 5 *AdpativeChoose*

1: **for** each $v \in V$ such that ($v.localized == false$) **do**
2: compute the degree $v.degree$;
3: $v.degree = v.degree - v.ann$;
4: **end for**
5: find v with the maximum $v.degree$ which is not localized;
6: **return** v;

will choose v as next initial-anchor since v has the maximum ($v.degree - v.ann$). It is not difficult to check that v is a better choice than u; see Figure 7(d).

Algorithm *AdpativeChoose* is a greedy strategy. It is obvious that there are other greedy strategies; for example, *HuangChoose* is another greedy strategy. How to choose the most effective initial-anchor is still an open problem and is our future work. We will compare *AdpativeChoose* with *HuangChoose* later; see Section 4.

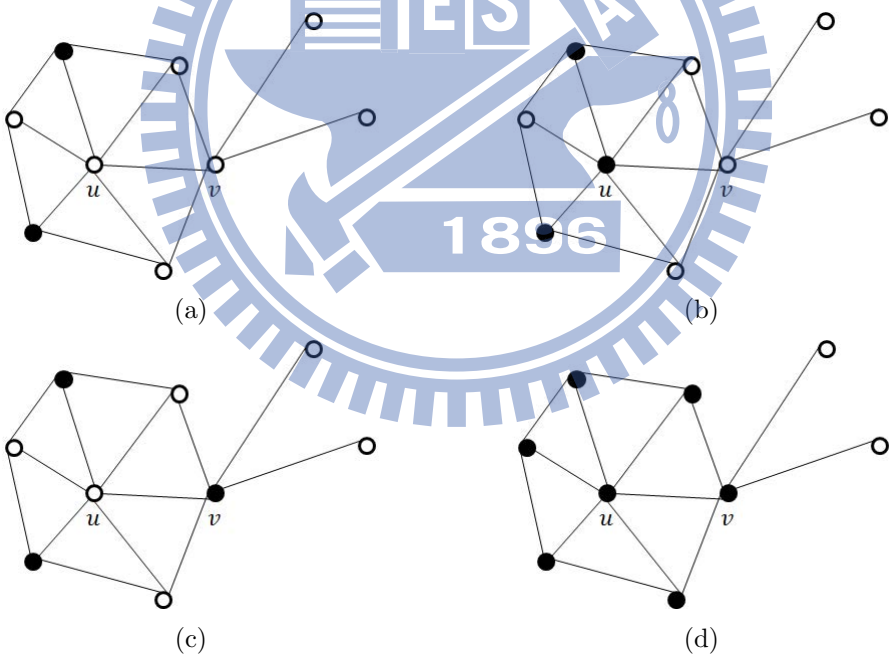


Figure 7: (a) A subgraph of the original network. (b) A strategy without adaptivity: a node with the maximum degree is chosen as the next initial-anchor; here u is chosen. (c) The strategy with adaptivity: a node with the maximum ($v.degree - v.ann$) is chosen as the next initial-anchor; here v is chosen. (d) The contribution of v; those nodes colored black are localized.

4 Simulation results

In this section, we provide our simulation results. For convenience, in the remaining figures, *LocalTri* means *Trilateration*. According to our simulations, the performance of *Sweep3* is almost the same as *Sweep2*. However, the computation of *Sweep3* is much more complicated than that of *Sweep2*. Therefore, we will not provide the results of *Sweep3*, ..., *Sweepk*. In our simulations, we will compare the following six combinations of *Localization-Phase* + *AnchorChoose-Phase*:

LocalTri + *AdpativeChoose*, *LocalTri* + *HuangChoose*, *LocalTri* + *MaxDegree*,

Sweep2 + *AdpativeChoose*, *Sweep2* + *HuangChoose*, and *Sweep2* + *MaxDegree*.

Notice that *MaxDegree* will choose a node v such that $v.localized$ = false and $\deg(v)$ is the maximum among all such nodes.

This section is divided into six subsections and is organized as follows. In Subsection 4.1, we introduce three parameters (*IAF*, *COVERAGE*, and *CP*) for evaluating the performance of the four combinations of *Localization-Phase* + *AnchorChoose-Phase*; our comparisons are based on the values of the three parameters (*IAF*, *COVERAGE*, and *CP*). In Subsection 4.2, we compare the four combinations with respect to the graph in [8]. In Subsection 4.3, we compare the six combinations with respect to very sparse graphs. In Subsection 4.4, we compare the six combinations with respect to sparse graphs. In Subsection 4.5, we compare the six combinations with respect to dense graphs.

Before going further, we describe the four simulation environments that we will use: the graph in [8], very sparse graphs, sparse graphs, and dense graphs. Note that all the graphs used in our simulations are unit disk graphs. The first simulation environment is the graph in [8], which is a graph with 199 nodes randomly generated in a 1200m $\times$ 1000m rectangle region and the transmitting range of each node is set to 80m. The second simulation environment contains 200 very sparse graphs: each graph is with 200 nodes

randomly generated in a 1200m × 1000m rectangle region and the transmitting range of each node is set to 70m. The third simulation environment contains 200 sparse graphs: each graph is with 200 nodes randomly generated in a 800m × 600m rectangle region and the transmitting range of each node is set to 70m. The fourth simulation environment contains 200 dense graphs: each graph is with 200 nodes randomly generated in a 800m × 600m rectangle region and the transmitting range of each node is set to 100m.

4.1 The three parameters considered in our simulations

In this subsection, we introduce three parameters (*IAF*, *COVERAGE*, and *CP*) for evaluating the performance. In the remaining part of this thesis, we will use *Anchor* to denote the set of nodes that know their positions, that is, *Anchor* contains initial-anchors and those nodes that have been localized. Let G be the given graph, n be the number of nodes in G, and $\mathcal{A}$ be a combination of *Localization-Phase* + *AnchorChoose-Phase*.

The first parameter *IAF* is the cardinality of an initial-anchor set, i.e.,

$$IAF_{\mathcal{A}}(G) = |S|$$

where S is an initial-anchor set. The second parameter *COVERAGE* is the percentage of nodes that know their positions, i.e.,

$$COVERAGE_{\mathcal{A}}(G) = \frac{|Anchor|}{n}.$$

It is obvious that if an initial-anchor set is feasible, then *COVERAGE* will achieve 100%. When a localization algorithm is running, nodes can be added into the initial-anchor set; therefore we use S_c to denote the current initial-anchor set. The third parameter *CP* is used to show the contribution of the current initial-anchor set and we define

$$CP_{\mathcal{A}}(G) = \frac{|Anchor|}{|S_c|}.$$

Do notice that we provide simulation results for *CP* only for the graph in [8].

4.2 Simulation results for the graph in [8]

The graph in [8] is shown in Figure 8.

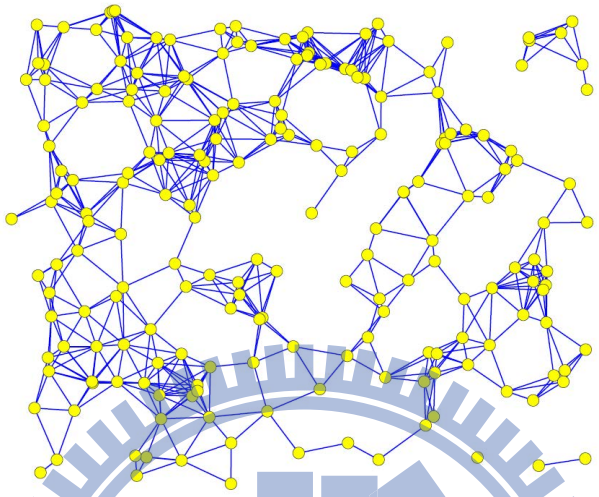


Figure 8: The graph in [8].

By [8],

$$IAF_{LocalTri+HuangChoose}(G) = 42 \text{ and } IAF_{Sweep2+HuangChoose}(G) = 33.$$

If our adaptive algorithm *AdpativeChoose* is used to choose anchors, then

$$IAF_{LocalTri+AdaptiveChoose}(G) = 40 \text{ and } IAF_{Sweep2+AdaptiveChoose}(G) = 31.$$

To sum up,

$$40 = IAF_{LocalTri+AdaptiveChoose}(G) < IAF_{LocalTri+HuangChoose}(G) = 42$$

and

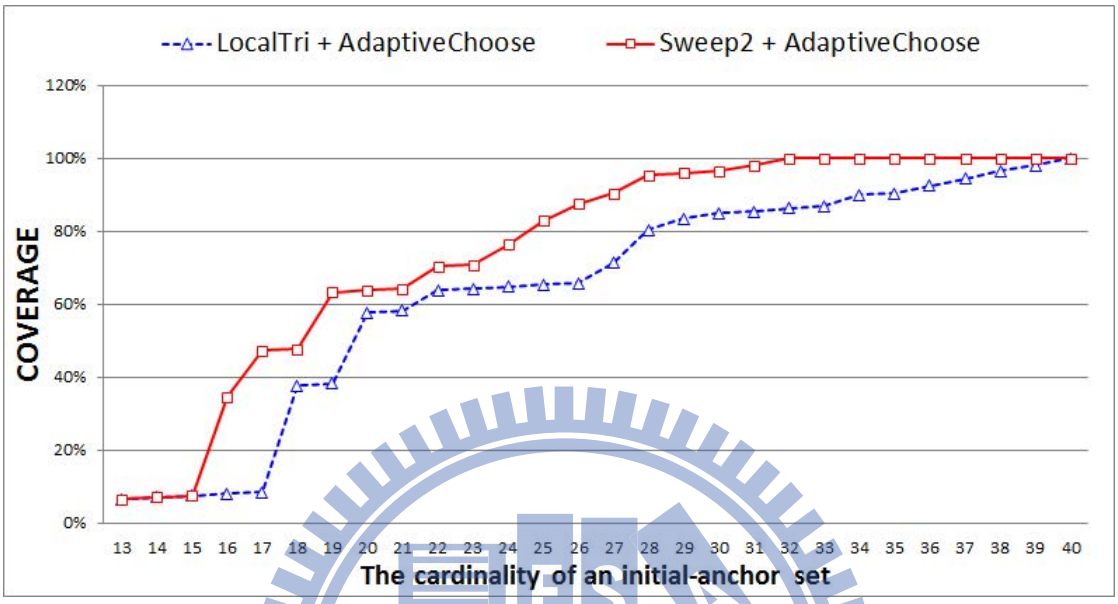
$$31 = IAF_{Sweep2+AdaptiveChoose}(G) < IAF_{Sweep2+HuangChoose}(G) = 33.$$

Thus *AdaptiveChoose* has a better (i.e., smaller) *IAF* than *HuangChoose*.

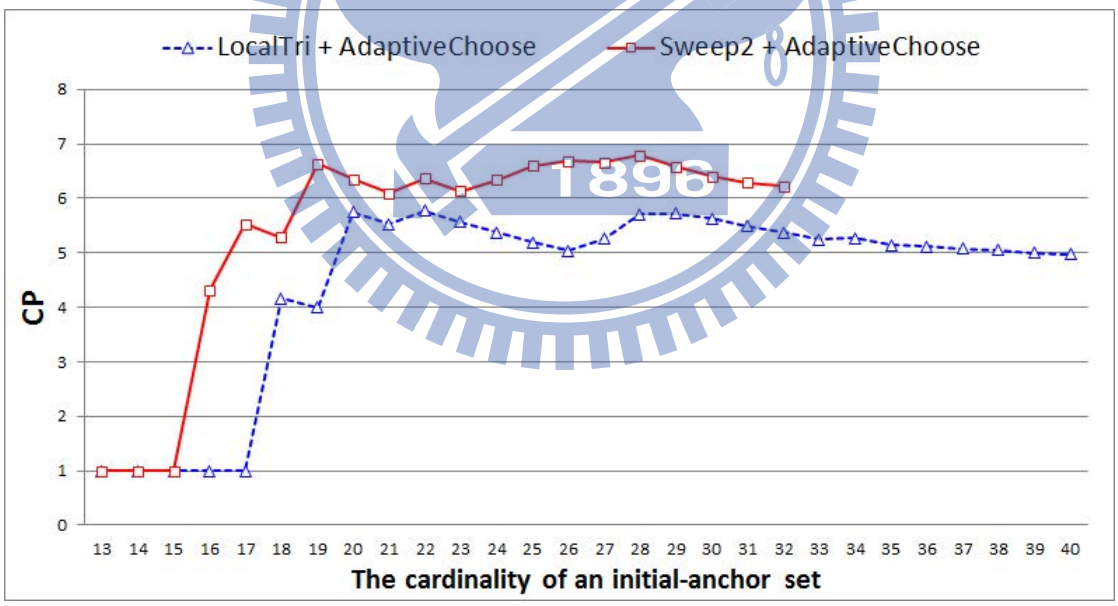
Figure 9(a) shows *COVERAGE* of *LocalTri* + *AdpativeChoose* and *Sweep2* + *Adpative-Choose*. It can be seen that when the cardinality of an initial-anchor set is 15, the curve of

Sweep2 + *AdpativeChoose* suddenly rises up and *Sweep2* + *AdpativeChoose* outperforms *LocalTri* + *AdpativeChoose*.

Figure 9(b) shows the *CP* of *LocalTri* + *AdpativeChoose* and *Sweep2* + *AdpativeChoose*. Notice that a local maximum *CP* occurs when the curves in Figure 9(a) has a local maximum. *CP* decreases after some initial-anchors are used, indicating that the remaining un-localized nodes may have certain structures which are difficult to be localized.



(a)



(b)

Figure 9: (a) *COVERAGE* and (b) *CP* of *LocalTri* + *AdpativeChoose* and *Sweep2* + *AdpativeChoose*.

4.3 Simulation results for very sparse graphs

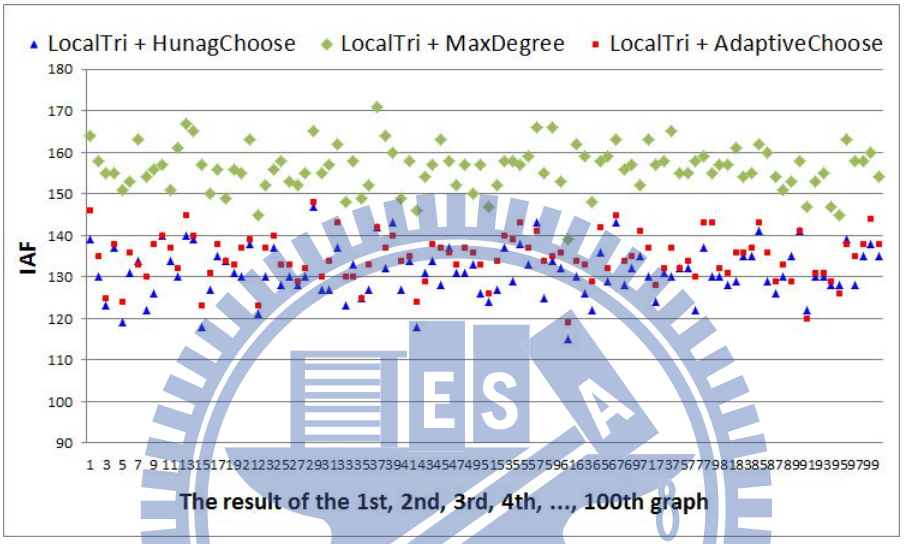
We first compare *IAF*. Figure 10 shows that when *IAF* is considered, *AdaptiveChoose* performs almost equal to *HuangChoose* and in only very few cases, *AdaptiveChoose* performs worse than *HuangChoose*. In fact, the average *IAF* of *LocalTri* + *AdpativeChoose* is 134.63, *LocalTri* + *MaxDegree* is 156 and that of *LocalTri* + *HuangChoose* is 131.12. Moreover, the average *IAF* of *Sweep2* + *AdpativeChoose* is 122.47, *LocalTri* + *MaxDegree* is 147 and that of *Sweep2* + *HuangChoose* is 119.56.

To consider *COVERAGE*, we restrict the cardinality of an initial-anchor set to be at most 100 or at most 120. Figure 11 shows the *COVERAGE* value when the cardinality of an initial-anchor set is restricted to at most 100. If shows that when *LocalTri* is used, *AdaptiveChoose* performs almost equal to *HuangChoose* and in only very few cases, *AdaptiveChoose* performs better than *HuangChoose*; however, when *Sweep2* is used, *AdaptiveChoose* performs almost equal to *HuangChoose* and in only very few cases, *AdaptiveChoose* performs worse than *HuangChoose*. In fact, the average *COVERAGE* of *LocalTri* + *AdpativeChoose* is 57.80%, *LocalTri* + *MaxDegree* is 57.80% and that of *LocalTri* + *HuangChoose* is 57.73%. Moreover, the average *COVERAGE* of *Sweep2* + *AdpativeChoose* is 68.06%, *Sweep2* + *MaxDegree* is 68.10% and that of *Sweep2* + *HuangChoose* is 65.64%.

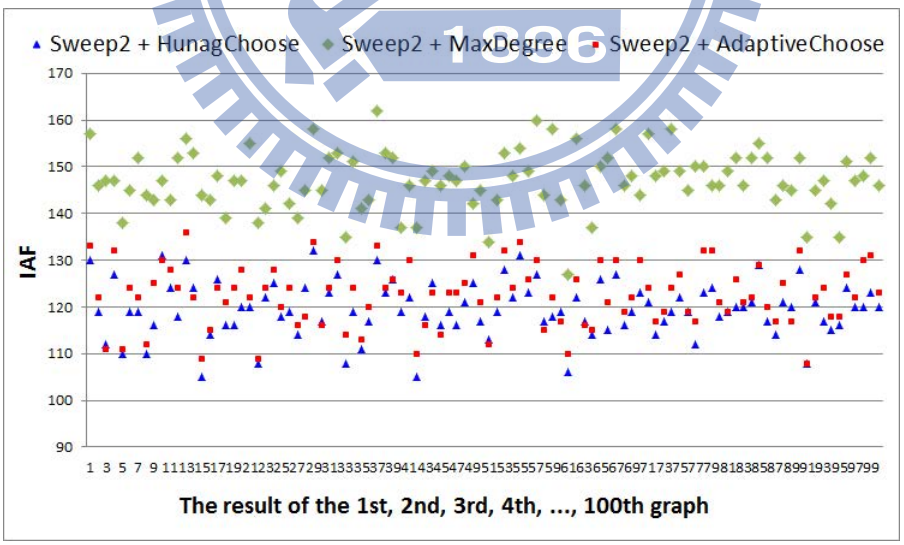
Figure 12 shows the *COVERAGE* value when the cardinality of an initial-anchor set is restricted to at most 120. It shows that when *LocalTri* is used, *AdaptiveChoose* performs almost equal to *HuangChoose* and in many cases, *AdaptiveChoose* performs better than *HuangChoose*; however, when *Sweep2* is used, *AdaptiveChoose* performs almost equal to *HuangChoose* and in only very few cases, *AdaptiveChoose* performs worse than *HuangChoose*. In fact, the average *COVERAGE* of *LocalTri* + *AdpativeChoose* is 57.87%, *LocalTri* + *MaxDegree* is 57.87% and that of *LocalTri* + *HuangChoose* is 57.92%. Moreover, the average *COVERAGE* of *Sweep2* + *AdpativeChoose* is 68.06%, *Sweep2* + *MaxDegree*

is 68.10% and that of *Sweep2* + *HuangChoose* is 65.64%.

To sum up, for very sparse graphs, we find that *AdaptiveChoose* performs almost equal to or performs better than *HuangChoose* when *LocalTri* is used in the *Localization-Phase* and *AdaptiveChoose* performs almost equal to *HuangChoose* when *Sweep2* is used in the *Localization-Phase*.

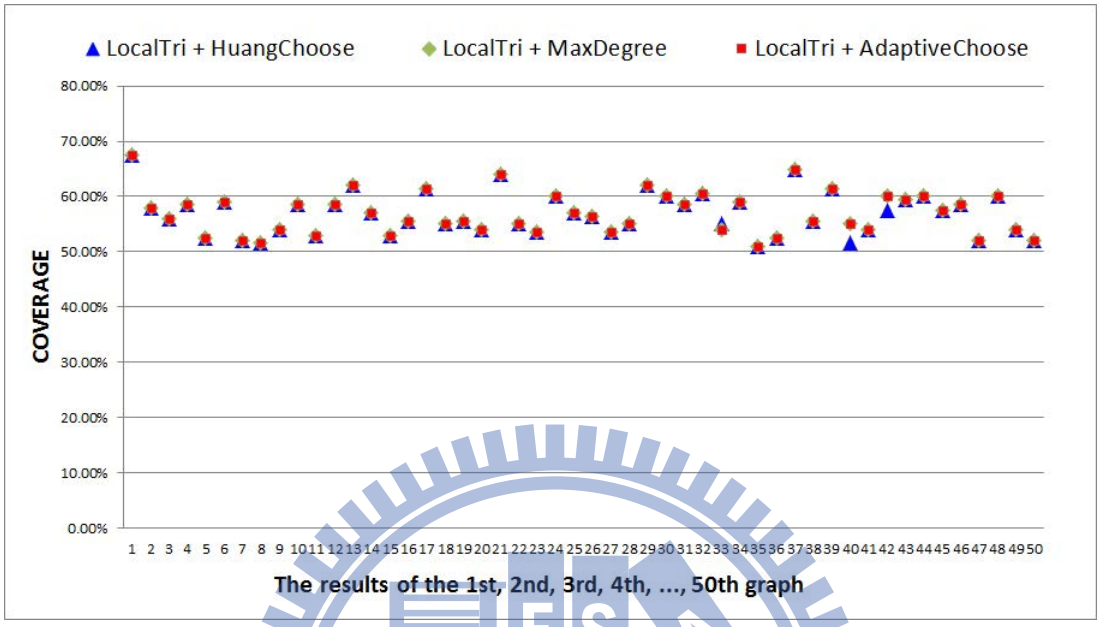


(a)

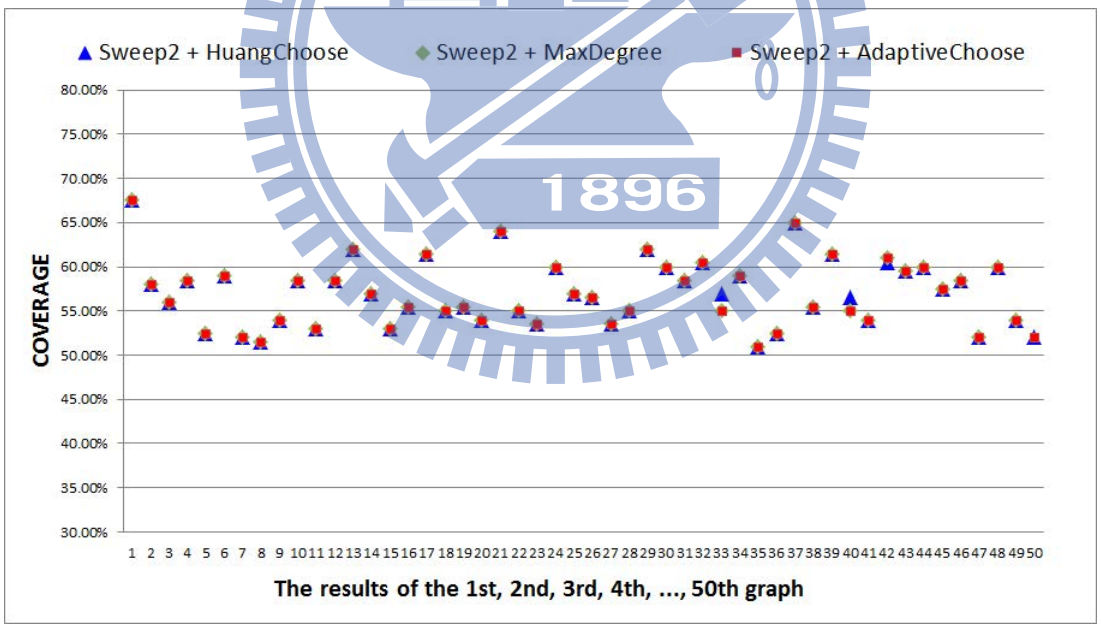


(b)

Figure 10: *IAF* for very sparse graphs. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.

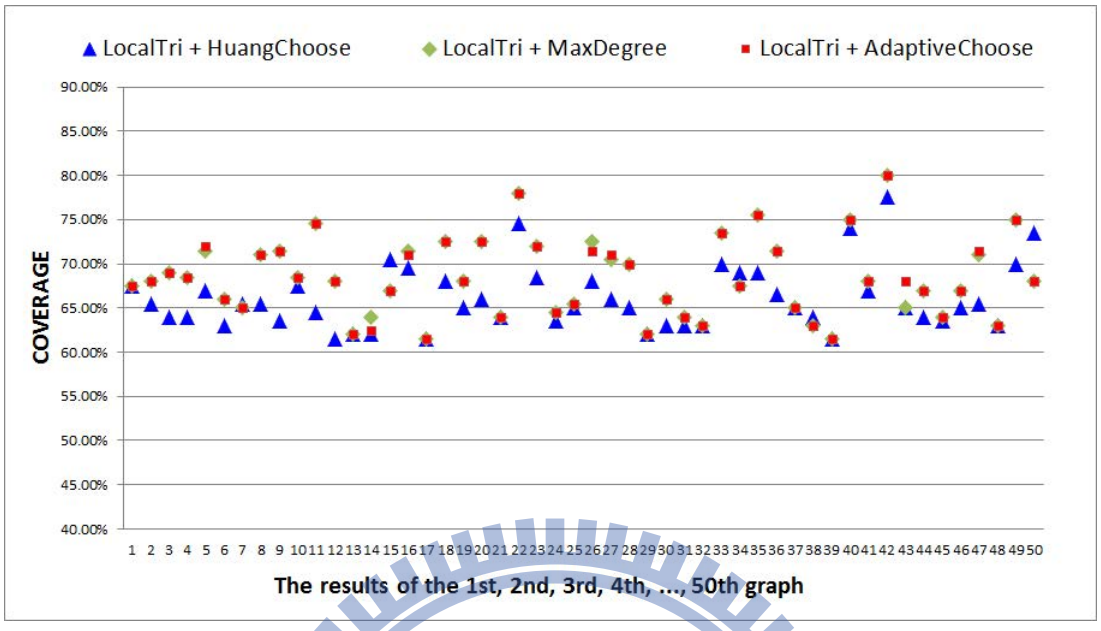


(a)

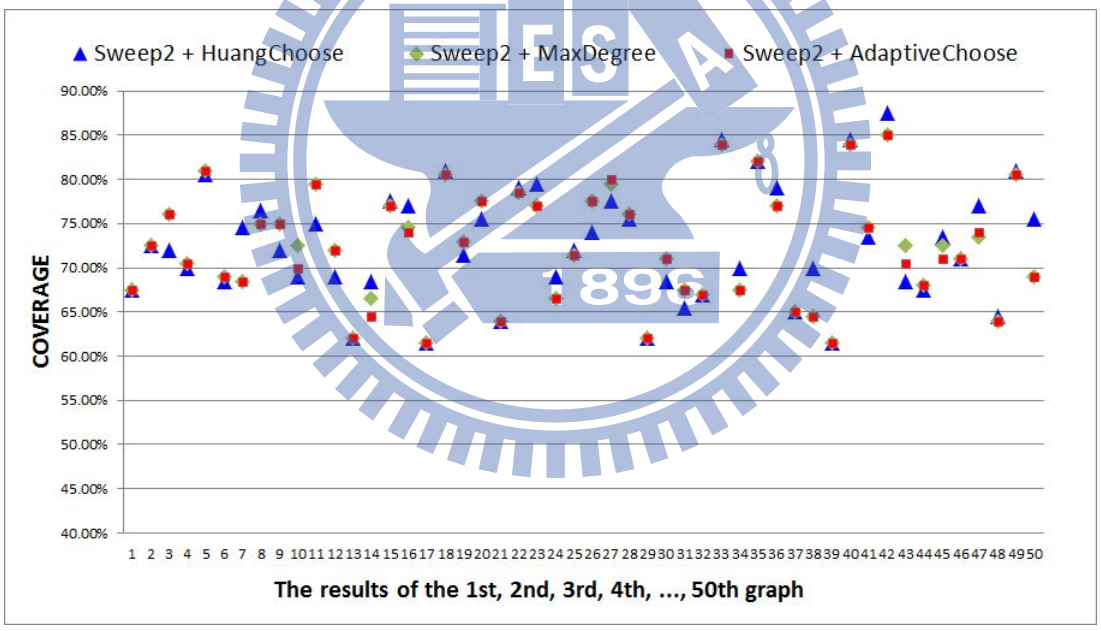


(b)

Figure 11: *COVERAGE* for very sparse graphs when the cardinality of an initial-anchor set is at most 100. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.



(a)



(b)

Figure 12: *COVERAGE* for very sparse graphs when the cardinality of an initial-anchor set is at most 120. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.

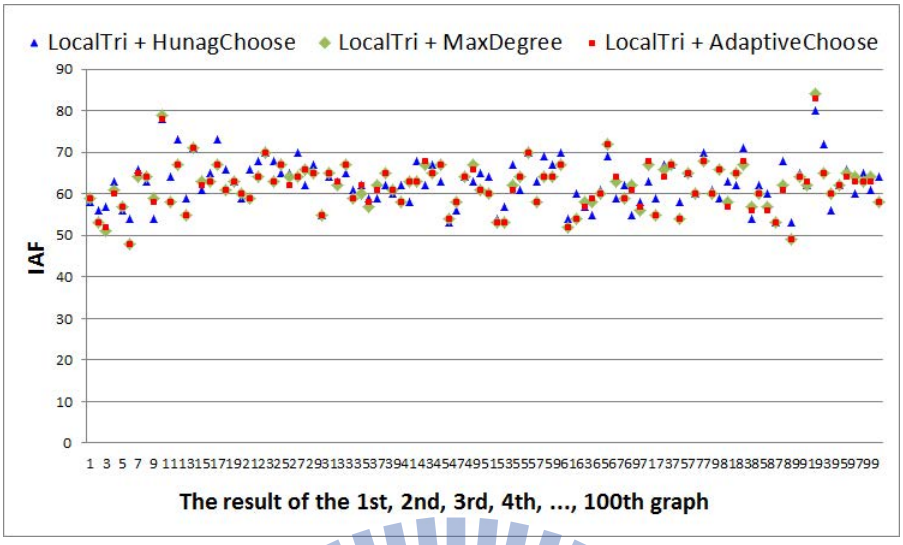
4.4 Simulation results for sparse graphs

We first compare *IAF*. Figure 13 shows that when *IAF* is considered, *AdaptiveChoose* performs almost equal to *HuangChoose*. In fact, the average *IAF* of *LocalTri* + *Adpative-Choose* is 61.585, *LocalTri* + *MaxDegree* is 61.625 and that of *LocalTri* + *HuangChoose* is 62.26. Moreover, the average *IAF* of *Sweep2* + *AdpativeChoose* is 50.36, *Sweep2* + *MaxDegree* is 51.005 and that of *Sweep2* + *HuangChoose* is 49.855.

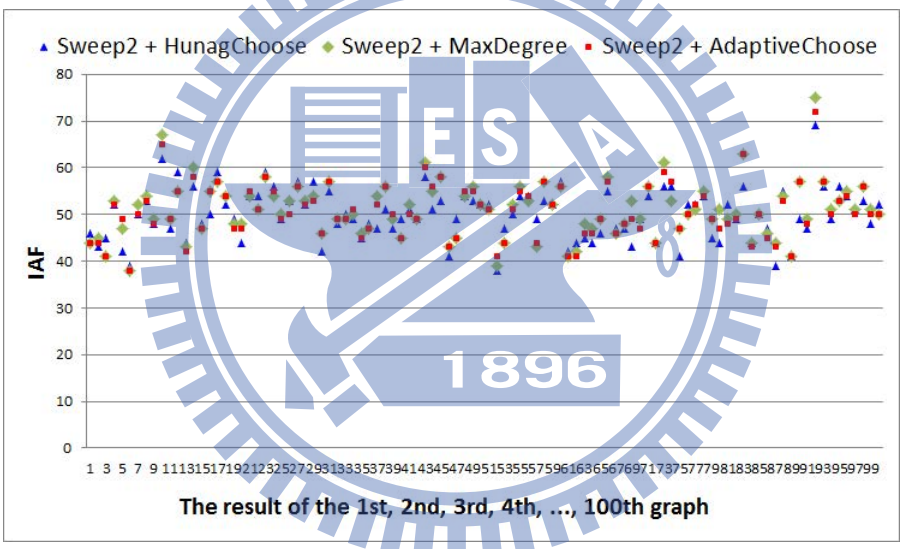
To consider *COVERAGE*, we restrict the cardinality of an initial-anchor set to be at most 30 or at most 50. Figure 14 shows the *COVERAGE* value when the cardinality of an initial-anchor set is restricted to at most 30. If shows that no matter which one of *LocalTri* and *Sweep2* is used, *AdaptiveChoose* performs better than *HuangChoose* in most cases. In fact, the average *COVERAGE* of *LocalTri* + *AdpativeChoose* is 48.66%, *LocalTri* + *MaxDegree* is 48.75% and that of *LocalTri* + *HuangChoose* is 24.26%. Moreover, the average *COVERAGE* of *Sweep2* + *AdpativeChoose* is 60.23%, *Sweep2* + *MaxDegree* is 60.47% and that of *Sweep2* + *HuangChoose* is 49.69%.

Figure 15 shows the *COVERAGE* value when the cardinality of an initial-anchor set is restricted to at most 50. It shows that when *LocalTri* is used, *AdaptiveChoose* performs better than *HuangChoose* in most cases; however, when *Sweep2* is used, *AdaptiveChoose* performs almost equal to *HuangChoose*. In fact, the average *COVERAGE* of *LocalTri* + *AdpativeChoose* is 88.58%, *LocalTri* + *MaxDegree* is 88.70% and that of *LocalTri* + *HuangChoose* is 76.64%. Moreover, the average *COVERAGE* of *Sweep2* + *AdpativeChoose* is 97.78%, *Sweep2* + *MaxDegree* is 97.58% and that of *Sweep2* + *HuangChoose* is 97.08%.

To sum up, for sparse graphs and when *IAF* is considered, *AdaptiveChoose* performs better than *HuangChoose* when *LocalTri* is used in the *Localization-Phase* and *Adaptive-Choose* performs worse than *HuangChoose* when *Sweep2* is used in the *Localization-Phase*. For sparse graphs and when *COVERAGE* is considered, no matter which one of *LocalTri* and *Sweep2* is used, *AdaptiveChoose* performs better than *HuangChoose* in most cases.

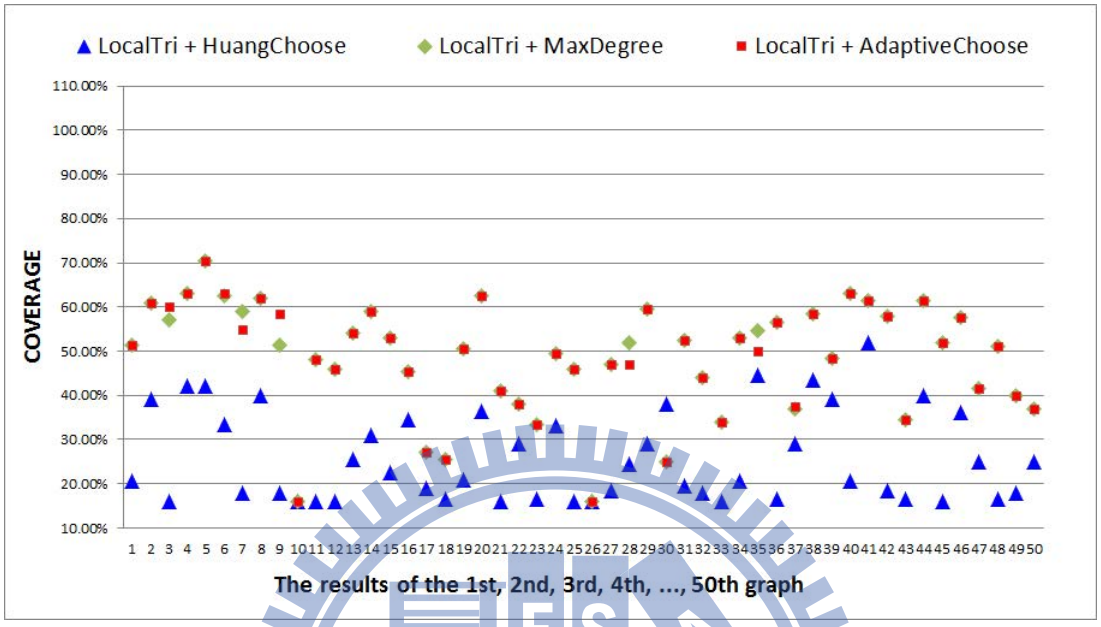


(a)

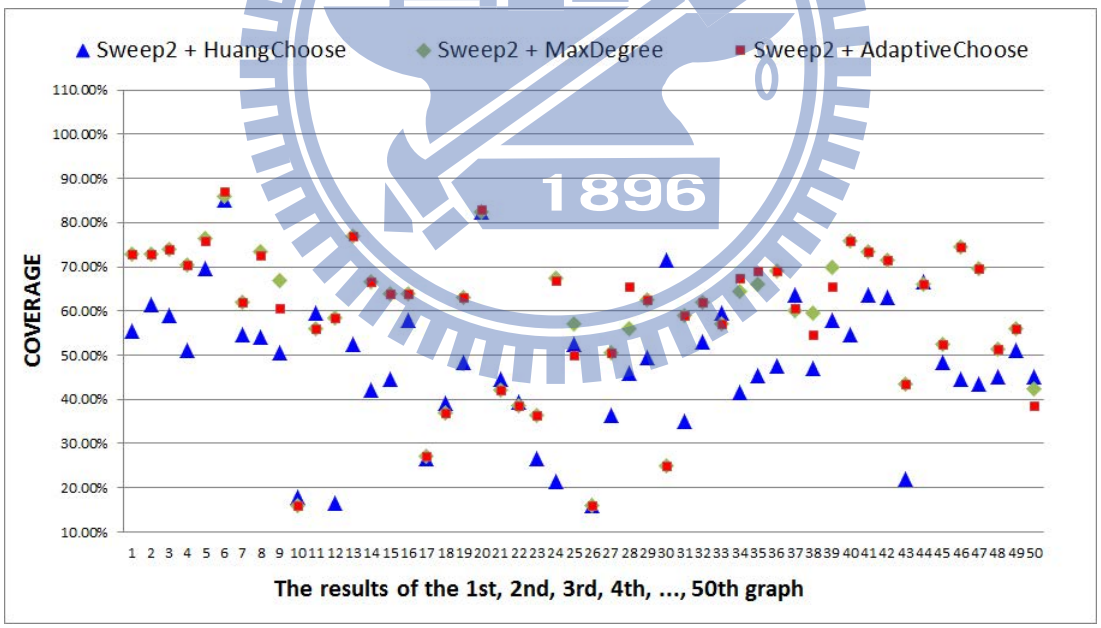


(b)

Figure 13: *IAF* for sparse graphs. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.

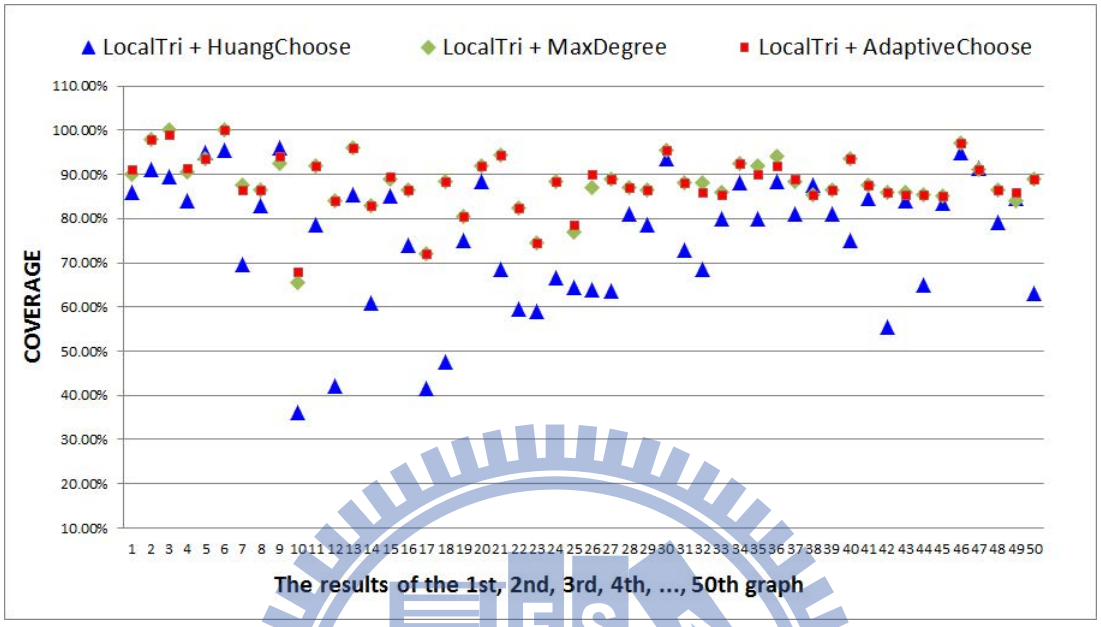


(a)

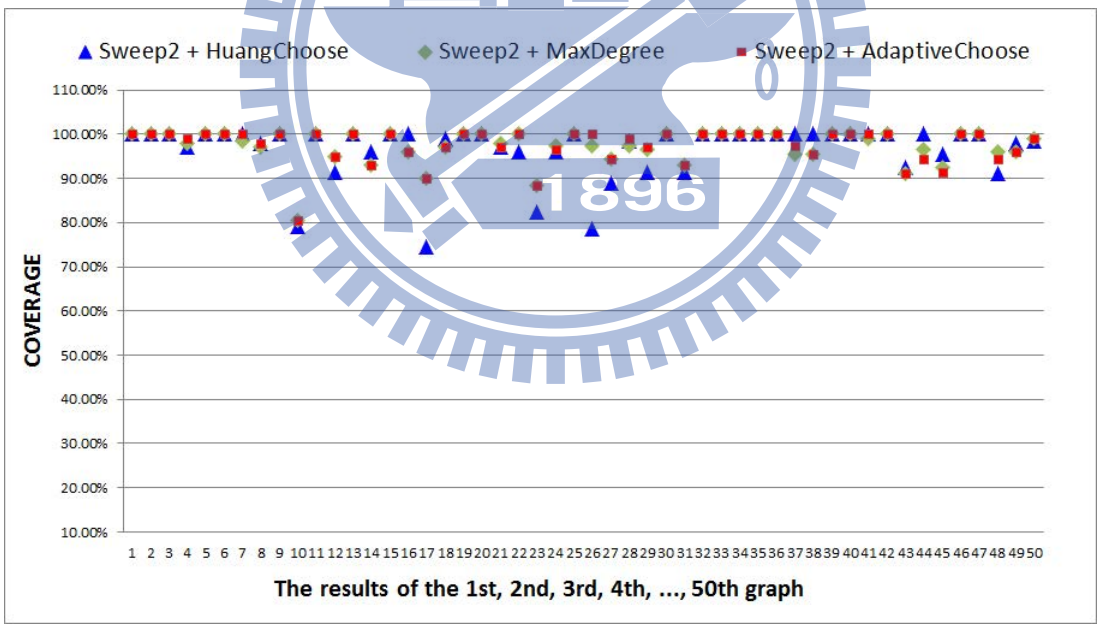


(b)

Figure 14: *COVERAGE* for very sparse graphs when the cardinality of an initial-anchor set is at most 30. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.



(a)



(b)

Figure 15: *COVERAGE* for very sparse graphs when the cardinality of an initial-anchor set is at most 50. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.

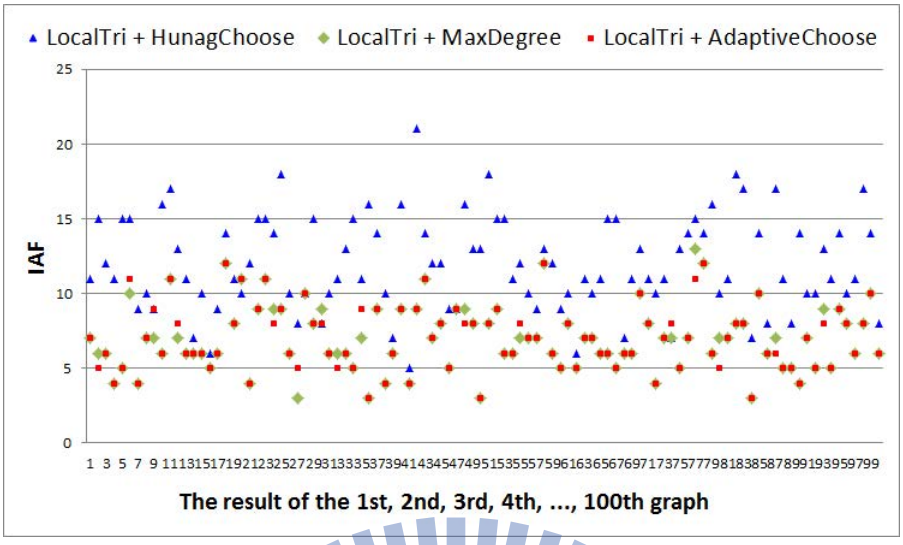
4.5 Simulation results for dense graphs

We first compare *IAF*. Figure 16 shows that when *IAF* is considered, *AdaptiveChoose* performs better than *HuangChoose*. In fact, the average *IAF* of *LocalTri* + *AdpativeChoose* is 6.915, *LocalTri* + *MaxDegree* is 6.875 and that of *LocalTri* + *HuangChoose* is 11.72. Moreover, the average *IAF* of *Sweep2* + *AdpativeChoose* is 5.81, *Sweep2* + *MaxDegree* is 5.795 and that of *Sweep2* + *HuangChoose* is 6.79.

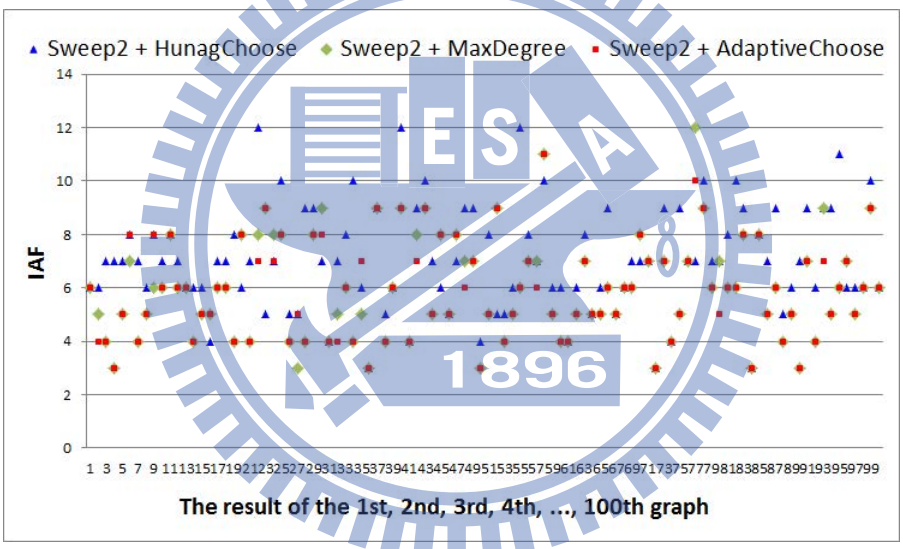
To consider *COVERAGE*, we restrict the cardinality of an initial-anchor set to be at most 5 or at most 10. Figure 17 shows the *COVERAGE* value when the cardinality of an initial-anchor set is restricted to at most 5. If shows that no matter which one of *LocalTri* and *Sweep2* is used, *AdaptiveChoose* performs better than *HuangChoose* in most cases. In fact, the average *COVERAGE* of *LocalTri* + *AdpativeChoose* is 80.49%, *LocalTri* + *MaxDegree* is 82.09% and that of *LocalTri* + *HuangChoose* is 12.19%. Moreover, the average *COVERAGE* of *Sweep2* + *AdpativeChoose* is 85.50%, *Sweep2* + *MaxDegree* is 87.31% and that of *Sweep2* + *HuangChoose* is 68.59%.

Figure 18 shows the *COVERAGE* value when the cardinality of an initial-anchor set is restricted to at most 10. It shows that no matter which one of *LocalTri* and *Sweep2* is used, *AdaptiveChoose* performs better than *HuangChoose* in most cases. In fact, the average *COVERAGE* of *LocalTri* + *AdpativeChoose* is 99.92%, *LocalTri* + *MaxDegree* is 99.88% and that of *LocalTri* + *HuangChoose* is 70.36%. Moreover, the average *COVERAGE* of *Sweep2* + *AdpativeChoose* is 99.99%, *Sweep2* + *MaxDegree* is 99.97% and that of *Sweep2* + *HuangChoose* is 99.69%.

To sum up, for dense graphs and when *IAF* is considered, no matter which one of *LocalTri* and *Sweep2* is used, *AdaptiveChoose* performs better than *HuangChoose*. For dense graphs and when *COVERAGE* is considered, we also have the same result.

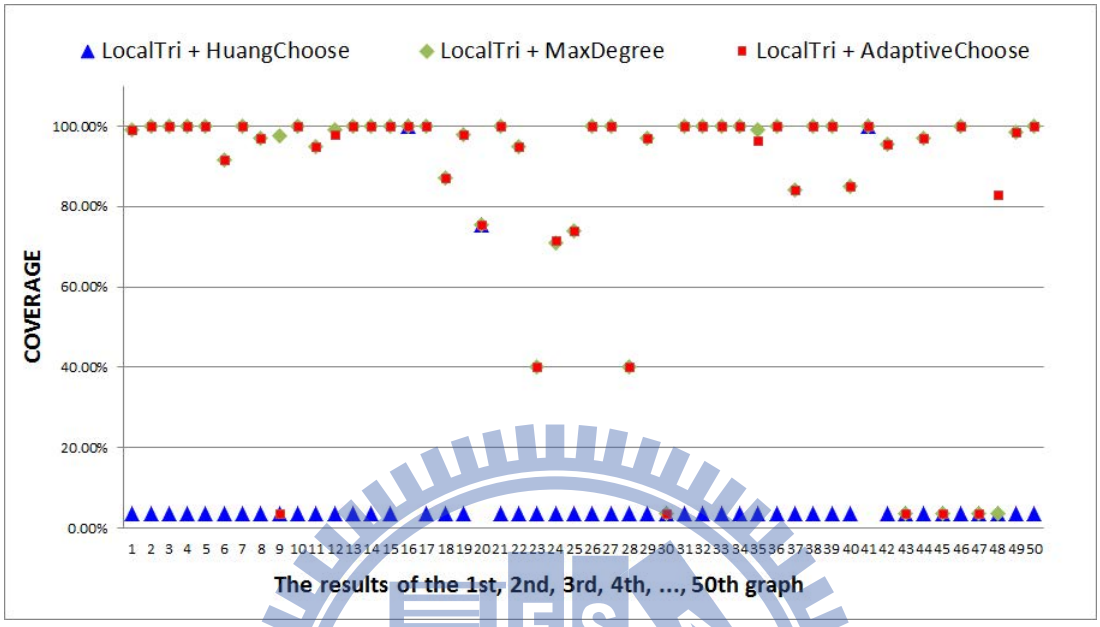


(a)

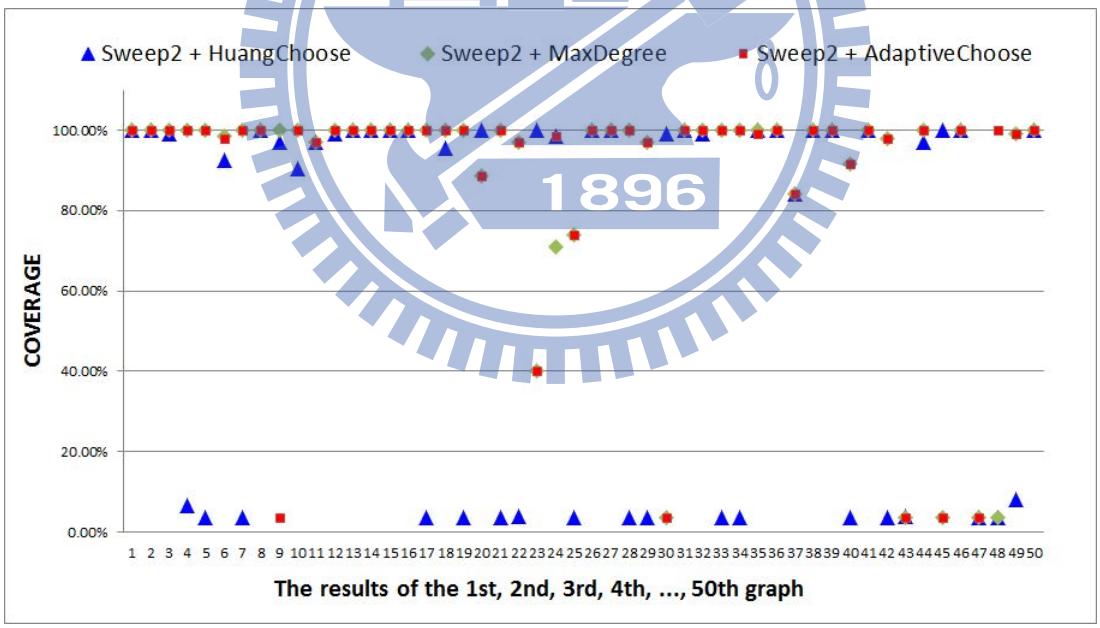


(b)

Figure 16: *IAF* for dense graphs. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.

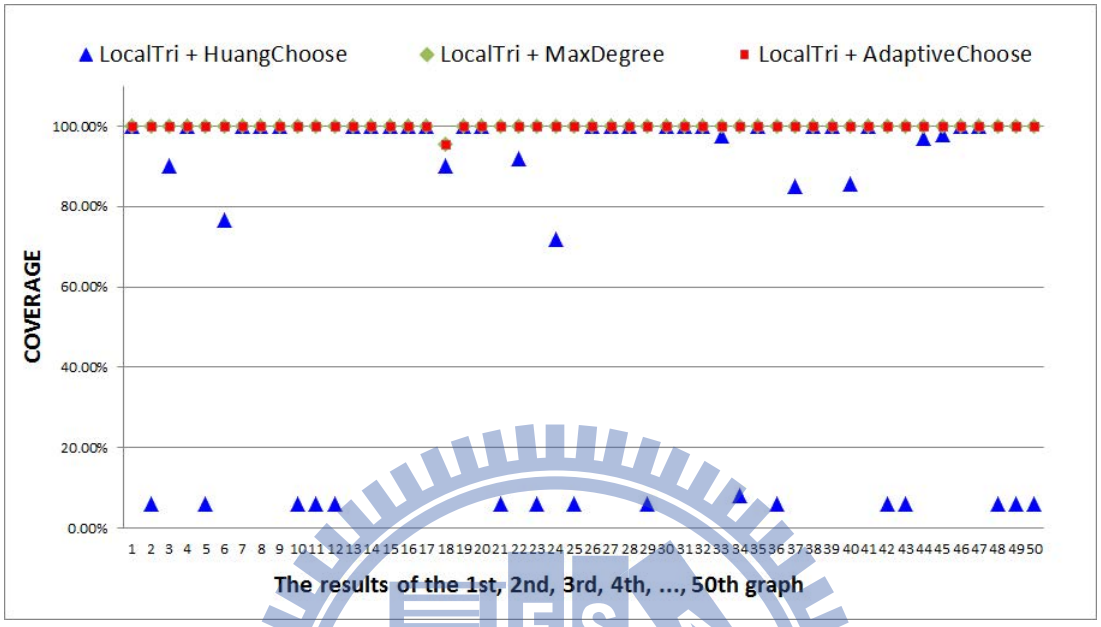


(a)

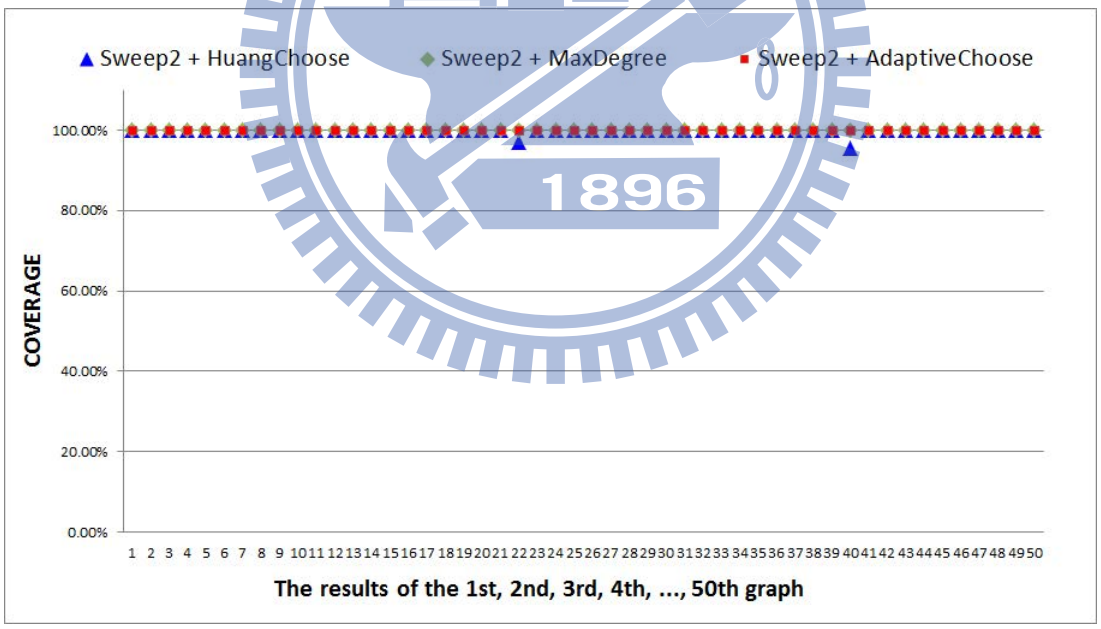


(b)

Figure 17: *COVERAGE* for very sparse graphs when the cardinality of an initial-anchor set is at most 5. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.



(a)



(b)

Figure 18: *COVERAGE* for very sparse graphs when the cardinality of an initial-anchor set is at most 10. (a) *LocalTri* is used in the *Localization-Phase*. (b) *Sweep2* is used in the *Localization-Phase*.

5 Concluding remarks

Many applications in wireless sensor networks require position information to detect and record events. In [8], Huang et al. studied the minimum cost localization problem, which aims to localize all sensors by using the minimum number of anchors. The purpose of this thesis is consider the same problem but using an adaptive algorithm. In particular, we propose an adaptive algorithm called *AdaptiveChoose* to choose initial-anchors. Our algorithms are simpler than the algorithms in [3] and cover all the cases in the algorithms in [8]; Simulation results show that *AdaptiveChoose* usually outperforms *HuangChoose* for dense graphs and *AdaptiveChoose* performs almost the same as *HuangChoose* for sparse or very sparse graphs. The overview of the simulation is shown in Figure 19. The first column is the simulation environment. The first row shows the two parameters that we consider. The algorithm with better result will be shown in the corresponding fields.

A=LocalTri	IAF	COVERAGE
G	AdaptiveChoose	No data
Vary Sparse	HuangChoose ≈AdaptiveChoose	The same
Sparse	The same	MaxDegree ≈AdaptiveChoose
Dense	MaxDegree ≈AdaptiveChoose	MaxDegree ≈AdaptiveChoose

(a)

A=Sweep2	IAF	COVERAGE
G	AdaptiveChoose	No data
Vary Sparse	HuangChoose ≈AdaptiveChoose	The same
Sparse	HuangChoose ≈AdaptiveChoose	AdaptiveChoose ≈MaxDegree
Dense	MaxDegree ≈AdaptiveChoose	MaxDegree ≈AdaptiveChoose

(b)

Figure 19: (a) *LocalTri*, (b) *Sweep2* is used in the Localization-Phase. In this figure, "≈" means the outcomes of the two methods differ in 2% and the better one will be shown first.

In the coming future, we would like to consider the followings problems. How to combine *AdpativeChoose* and *HuangChoose* to obtain better results? Given a certain initial-anchor set, determine what kind of graphs are localizable. It is known that a globally rigid graph can be localized by using three anchors; what will happen if more anchors are used? Design a distributed version of *AdaptiveChoose*. We have tried to

generalize algorithm *Sweep2* to be *Sweepk*. In *Sweep2*, two nodes try to cooperate to obtain their positions; in general, in *Sweepk*, k nodes try to cooperate to obtain their positions. Although we found that *Sweep3* does not improve *Sweep2*, we conjecture that *Sweepk* does improve *Sweep2* whenever $k \geq 4$. Finally, it is interesting to generalize the algorithm *Rigid* to *Rigidk*, meaning that instead of using one-hop neighborhoods, k-hop neighborhoods are used.

References

[1] B.D.O. Anderson, P.N. Belhumeur, T. Eren, D.K. Goldenberg, A.S. Morse, W. Whiteley, and Y.R. Yang, Graphical properties of easily localizable sensor networks, *Wireless Networks*, vol. 5 (2009) 177-191.

[2] J. Aspnes, T. Eren, D.K. Goldenberg, A.S. Morse, F. Yang, R. Yang, B.D.O. Anderson, and P.N. Belhumeur, A theory of network localization, *IEEE Transactions on Mobile Computing*, vol. 5, no. 12 (2006) 1663-1678.

[3] F. Barsi, A.A. Bertossi, C. Lavault, A. Navarra, S. Olariu, M.C. Pinotti, and V. Ravelomanana, Efficient Location Training Protocols for Heterogeneous Sensor and Actor Networks, *IEEE Transactions on Mobile Computing*, vol. 10, no. 3 (2011) 377-391.

[4] R. Connelly, On generic global rigidity, applied geometry and discrete mathematics, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 4, American Mathematical Society, Providence, RI (1991) 147-155.

[5] R. Connelly, Generic global rigidity, in: *Discrete & Computational Geometry*, vol. 33, (2005) 549-563.

[6] B. Hendrickson, Conditions for unique graph realizations, *SIAM Journal on Computing*, vol. 21, no. 1 (1992) 65-84.

[7] B. Hendrickson, The molecule problem: exploiting structure in global optimization, *SIAM Journal on Optimization*, vol. 5, no. 4 (1995) 835-857.

[8] M. Huang, S. Chen, and Y. Wang, Minimum cost localization problem in wireless sensor networks, *Ad Hoc Networks*, vol. 9 (2011) 387-399.

[9] B. Jackson and T. Jordsan, Connected rigidity matroids and unique realizations of graphs, *Journal of Combinatorial Theory*, Series B, vol. 94 (2005) 1-29.

[10] G. Laman, On graphs and rigidity of plane skeletal structures, *Journal of Engineering Mathematics*, vol. 4, no. 4 (1970) 331-340.

[11] K. Langendoen and N. Reijers, Distributed localization in wireless sensor networks: a quantitative comparison, *Computer Networks*, vol. 43 (2003) 499-518.

[12] D. Niculescu and B. Nath, Ad-hoc positioning system, in: *IEEE GlobeCom* (2001) 2926-2931.

[13] A. Pal, The n-hop multilateration primitive for node localization problems, *Mobile Networks and Applications*, vol. 8 (2003) 443-451.

[14] C. Savarese, K. Langendoen, and J. Rabaey, Robust positioning algorithms for distributed ad-hoc wireless sensor networks, in: *USENIX Technical Annual Conference*, Monterey, CA (2002) 317-328.

[15] A. Savvides, H. Park, and M. Srivastava, The bits and flops of the N-hop multilateration primitive for node localization problems, in: *First ACM International Workshop on Wireless Sensor Networks and Application (WSNA)*, Atlanta, GA (2002) 112-121.

[16] A. Wadaa, S. Olariu, L. Wilson, M. Eltoweissy, and K. Jones, Training a wireless sensor network, *Mobile Networks Applications*, vol. 10 (2005) 151-168.

[17] Z. Yang, Y. Liu, and X. Li, Beyond trilateration: on the localizability of wireless ad hoc networks, *IEEE/ACM Transcactions on Networking*, vol. 18, no. 6 (2010) 1806-1814.