DISTRIBUTED ITERATIVE CLUSTER LOCALIZATION IN WIRELESS SENSOR NETWORKS

OLCA ARDA ÇAKIROĞLU

IŞIK UNIVERSITY

2009

DISTRIBUTED ITERATIVE CLUSTER LOCALIZATION IN WIRELESS SENSOR NETWORKS

OLCA ARDA ÇAKIROĞLU

BS, Computer Engineering, 2006

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Engineering

IŞIK UNIVERSITY

2009

IŞIK UNIVERSITY

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

DISTRIBUTED ITERATIVE CLUSTER LOCALIZATION IN WIRELESS SENSOR NETWORKS

OLCA ARDA ÇAKIROĞLU

Assist. Prof. Cesim ERTEN (Thesis Supervisor)	Kadir Has University	_____
Assoc. Prof. Ercan SOLAK	Işık University	_____
Assist. Prof. Hasan Fehmi ATEŞ	Işık University	_____

APPROVAL DATE 07/01/2009

DISTRIBUTED ITERATIVE CLUSTER LOCALIZATION IN WIRELESS SENSOR NETWORKS

Abstract

We designed a distributed algorithm for iterative cluster localization. Because this algorithm is especially designed for large scale networks, increase in the number of sensor nodes has no inefficiency effect on sensor node performances. A node has the information regarding every node within some vertices which are in range of the node. In the presented techniques every node only focused on localizing itself, however in our algorithm every node can localize itself individually or they obtain localization information from other nodes. And we see that each node likely to localizes itself after it has localized other nodes in its cluster which obviously shows the contribution of coordinate sharing. Although our algorithm is mostly dependent on sharing information, we show the messaging overhead is quite reasonable.

KABLOSUZ ALGILAYICI AĞLARDA DAĞITIK YENİLEMELİ KÜMESEL YERELLEŞTİRME

Özet

Kablosuz algılayıcı ağları için dağıtık yenilemeli kümesel yerelleştirme algoritması geliştirdik. Bu algoritma büyük ağlara uygun bir şekilde dizayn edildiğinden, ağ içersindeki algılayıcı düğüm sayısının artışı düğümler üzerindeki iş yükünü arttırmayacaktır. Her düğüm diğer komşuları ile iletişim kurarak bulunduğu sınırlı bir ortam içerisindeki diğer düğümlerden haberdar olur ve bunları kendi kümesinde yereleştirebilmek üzere saklar. Şu ana kadar geliştirilen dağıtık yerelleştirme tekniklerinde düğümler sadece kendi kordinatlarını bulmak için işlemler yapıyorlardı, bizim sunduğumuz algoritma da ise düğümler kendi kordinatlarını kendileri hesaplayabilir veya başka düğümlerden bu bilgiyi elde edebilirler. Yerelleştirme yapan düğümlerin kendilerinden önce kendi kümeleri içerisindeki düğümleri yerelleştirdiklerini gördük. Bu da bize kordinat bilgilerini paylaşmanın yerelleştirmeye ne kadar çok yararlı olduğunu gösterdi. Kordinatların paylaşımı üzerine olan bir yerelleştirme algoritması olduğundan, bir düğüm yerelleştirme aşamasına gelmeden önce kendi kümesi içersinde bulunan düğümlerin pozisyon bilgilerini almış durumda olduğunu gösterdik. Mesajlaşma üzerine dayalı olan bir yerelleştirme tekniği olmasına rağmen, düğümler üzerindeki mesajlaşma yükünün makul seviyelerde olduğunu gördük.

Acknowledgements

This thesis was carried out in close collaboration with my supervisor Assist. Prof. Cesim Erten over the last 2 years. I would like to thank him for introducing me the field, giving advices, sharing his opinions and listening to mine. It is certain that this work will not be possible without his generous support.

I would thank Ömer Karataş for his friendship and the technical talks we had during the hours we spent on the project work. I would also thank Sevgi Dikmen and the people and friend working in Registrar's Office for their unconditional support.

To mom, dad, family and friends.

Table of Contents

Abstract **ii**

Özet **iii**

Acknowledgements **iv**

Table of Contents **vi**

List of Figures **vii**

1 Introduction **1**

2 Previous Work **3**

3 Preliminaries **6**

4 Distributed Iterative Cluster Localization **11**

4.1 Initial setup 13

4.2 Iterative Localization 14

4.3 Analysis of *DICL* 18

5 Experimental Setup and Results **19**

5.1 Random Network Generation and Parameters 19

5.2 Experiments and Discussion of Results 20

6 Conclusion **26**

References **27**

Curriculum Vitae **30**

List of Figures

3.1 Graph rigidity 6
3.2 Continuous deformations 7
3.3 Flip ambiguity and 3-connectivity 7
3.4 Flex ambiguity 8
3.5 Example of trilateration graphs 9
3.6 Finite localization 10

4.1 Forming C_u using each $Packet_{N_v^1}, v \in N_u^1$ 13
4.2 *IFL* in execution. 14

5.1 Random network localized with *DICL* 21
5.2 How r effects localization 22
5.3 Ratio of localized nodes through total number of nodes (n) . . . 22
5.4 Number of broadcasts per node 23
5.5 Amount of data broadcasted in units 23
5.6 Maximum possibilities per node 24
5.7 Total possibilities of each cluster. 25
5.8 Running time in seconds during *DICL*. 25

List of Algorithms

1 DICL - InitialSetup 12

2 DICL - Localize C_u 14

3 IFL 15

4 IFL - Sub procedures 16

Chapter 1
Introduction

Wireless sensor nodes are small electronic devices which basically consist of four main parts, a microprocessor, a RAM to store data in, a wireless communication device and a power supply. The cost of a node depends on its capabilities and its size. Each sensor node can be equipped with a sensing device to collect analogous information from the environment such as temperature, pressure, sound, vibration, motion and etc. Wireless sensor nodes are originally motivated by military applications such as battlefield surveillance. There are many applications and systems from other areas such as environment and habitat monitoring, weather forecast and health applications those require use of many sensor nodes organized as a network collectively gathering useful data; see [1] for a survey. In many such applications it is necessary to know the actual locations of the sensors. Sensor network localization is the problem of assigning geographic coordinates to each sensor node in given network. Although Global Positioning System (GPS) can determine the geographic coordinate of an object, a GPS device has to have at least four line of sight communication lines between different satellites in order to locate itself. Thus in cluttered space or indoor environments GPS may be ineffective. Disadvantages including the power consumption, cost and size limitations allow only a small number of nodes in a large scale network gave GPS. It is important to design methods that achieve localization with limited use of such systems.

The remainder of the thesis is organized as follows, in Chapter 2 we give summary on previous work done over localization problem. Chapter 3 describes necessary literature background over localization problem. In Chapter 4, we introduce our algorithm and analyze computational complex-

ity and messaging overhead. Chapter 5 gives the experimental results and in Chapter 6 conclusion and overview of the subject is described.

Chapter 2
Previous Work

There has been exist a widely studied literature on the study of Wireless Sensor Network (WSN) localization problem [2, 3, 4, 5, 6, 7, 8, 9]. In addition to that there have been many surveys conducted about WSN localization problem, some of which has been provided in [10, 4, 11, 1]. Due to the aim of having a deep understanding of existing localization techniques, we have classified them into three categories so that we have discussed *the process of the computational source supply*, *the types of information needed for localization* and finally, *the result produced by these techniques*. And Table 2.1 shows the list of several algorithms with their specific properties. The headers of columns are Algorithm, Computational Source Supplied, Information Used, Measurement Errors, Output, Require Anchor respectively to the column ordering in Table 2.1.

Table 2.1: This table shows the information needed by algorithms, the requirements of the algorithms and the results produced by algorithms.

Alg.	CompSrcSup.	InfoUsed	MeasErr.	Output	R.A.
[8]	Distributed	Distance	Yes	Unique	No
[9]	Centralized	Distance	Yes	Finite	No
[2]	Distributed	Distance	Yes	Unique	No
[7]	Centralized	Distance	No	Unique	Yes
[5]	Centralized	Distance Distance and Angle	Yes	Unique	No
[3]	Distributed	Distance and Angle	Yes	Region	Yes
$DICL$	Distributed	Distance	No	Unique	Yes

First category on, the process of the computational source supply, can be divided into two subcategories as *centralized algorithms* and *distributed algorithms*. Centralized algorithms [12, 5, 9, 7] carry out the localization

on few number of specific central computers. It is possible to split centralized algorithms into three phases for a better understanding. First phase is collecting necessary information into the central computers which may require too many number of messaging through the network. After first phase is done, central computers have all information about network and start localizing the network. The third case is returning the localization data back to the sensor nodes which has same messaging overhead with the first phase. Despite of high messaging overhead, another disadvantage is that centralized approaches are not appropriate for big networks. On the other hand centralized approaches are able to localize more nodes than the distributed algorithms might do because they use all the information in the network. On distributed algorithms [8, 2, 3, 4] computational power is supplied by the sensor nodes. Unlikely to centralized approaches, distributed algorithms do not require too many messaging since each node knows about only some closer nodes in a specific range, but localization process may take some number of iterations for realization to converge. On the other hand, distributed algorithms are suitable for mobile sensor networks where sensor nodes are moving randomly while it is probably not possible for centralized approach.

The types of information needed for localization is the second category and this types of information are *connectivity*, *distance*, and *angle* [10, 13, 14]. Using connectivity information, a sensor node knows about neighboring nodes but it is not able to gather any further information such as distance. And localization using connectivity information is not preferable since it's resulting lack of solution accuracy. On the other hand, gathering connectivity information has no additional cost for sensor nodes hence it requires no additional hardware. Different methods are available for obtaining distance measurements between two sensor nodes. For instance, most known distance measurement methods are time difference of arrival (TDoA), time of arrival (ToA), and received signal strength indica-

tion (RSSI). Since it is impossible to measure exact distance in real world environment, these errors obliviously effects localization [14]. However, many assumptions made on simulating real world measurement errors are not consistent with characteristics of real measurement errors [14]. In addition to distance information, the method for measuring angle between two sensor nodes is known as *angle of arrival*, (AoA). Achieving angle information requires more complex hardware than distance information such as array of microphones or array of antennas [13].

Final categorization is the result produced by existing localization algorithms. A group of algorithms try to assign a single coordinate to each sensor node in network which is called *unique localization* [8, 2, 7, 5]. When it is not possible to localize nodes uniquely, possible positions where sensor node can be on are assigned to the nodes which is known as *finite localization* [9]. However, even it would be not possible to assign single or finite number of coordinates to a single node due to lack of information. [6] proposed a *region based* localization algorithm for assigning bounded regions to sensor nodes using noisy distance and angle measurements where each sensor nodes are expected to be somewhere in the associated region.

Chapter 3
Preliminaries

The sensor network localization problem can be converted into *graph realization problem*. Graph realization problem is based on reconstructing vertex positions using given distance information between adjacent vertices. Let $\mathbb{G}(V, E)$ be the graph corresponding to our physical network $\mathbb{N}$. Each vertex $v_i \in V = (v_1, ..., v_m, v_{m+1}, ..., v_n)$ corresponds to a specific physical sensor node i in $\mathbb{N}$. Vertices $v_1, ..., v_m$ corresponds to the nodes with known positions, called *anchors*. The rest of the vertices are the nodes with unknown positions. There exists an edge $(v_i, v_j) \in E$ if nodes i and j are within sensing range or both $i, j \leq m$. Each edge (v_i, v_j) where $v_i, v_j \in V$ and $i \neq j$ is associated with a real number, $d(v_i, v_j)$. That real number represents the Euclidean distance between the two nodes i, j. Formally, the graph realization problem is assigning coordinates to the vertices so that the Euclidean distance between any two adjacent vertices is equal to the real number associated with that edge [15, 16].

The graph realization problem has intrinsic connections with the graph rigidity theory. If we think of a graph in terms of bars and joints, a *rigid* graph means 'not deformable' or 'not flexible' [7]. In graph theory, definition of rigidity encompasses noncontinuously deformable graph. Formally, the rigidity of a graph can be characterized by Laman's condition [17]. A graph

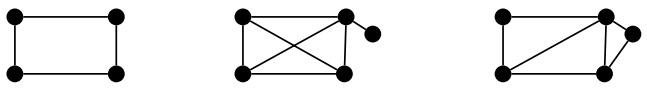
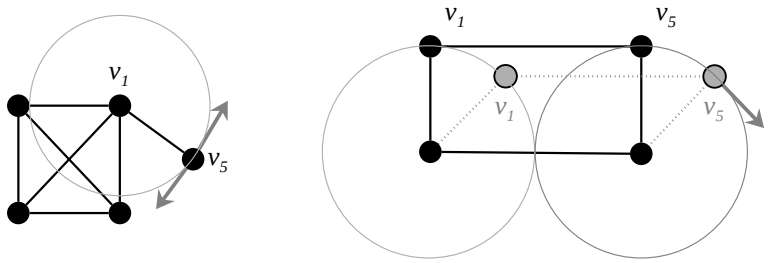
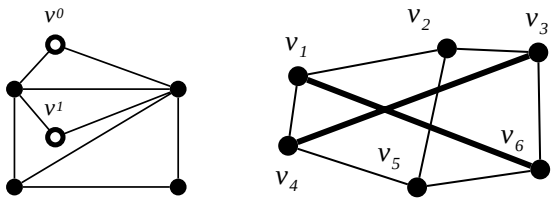


Figure 3.1: **(a)** Non-rigid, not enough edges **(b)** Non-rigid, edges are not well distributed **(c)** Rigid



(a) (b)

Figure 3.2: **(a)** v_5 can be any where on the circle having a radius of $d(v_1, v_5)$ centered on v_1 **(b)** With preserving distances positions of vertices can move



(a) (b)

Figure 3.3: **(a)** Flip ambiguity, v could be on v^0 or v^1. **(b)** 3-Connected

$\mathbb{G}(V, E)$ with $2n - 3$ edges where n is number of vertices in $\mathbb{G}$, is rigid in $\mathbb{R}^2$ if and only if no subgraph $\mathbb{G}'$ of $\mathbb{G}$ has more than $2n' - 3$ edges, where n' is the number of vertices in $\mathbb{G}'$ [18, 16]. Informally, a graph is rigid if it has a sufficient number of edges and its edges are 'well distributed' among its subgraphs; see Figure 3.1 for non-rigid and rigid graph examples.

Obviously if the graph is not rigid, infinite number of realizations are possible through continuous deformations; see Figure 3.2. However, even when the graph is rigid there may be ambiguities that give rise to more than one possible realization. There exist two types of ambiguities: *flip ambiguity* and *flex ambiguity*. Flip ambiguity occurs by partial reflection of a graph; see Figure 3.3a. Avoiding flip ambiguity requires the graph to be *3-Connected*. 3-Connectivity requires a graph not to get disconnected even after removal of any two vertices; see Figure 3.3b. Even if the graph

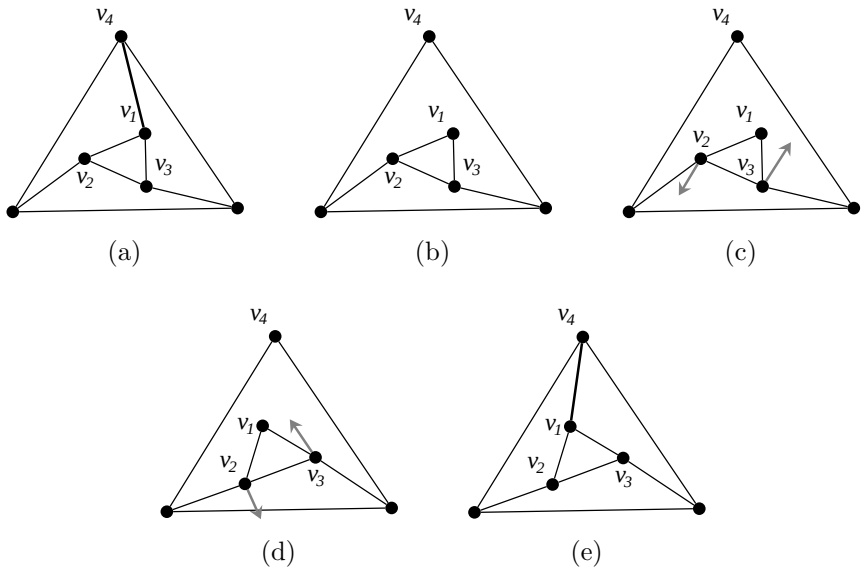
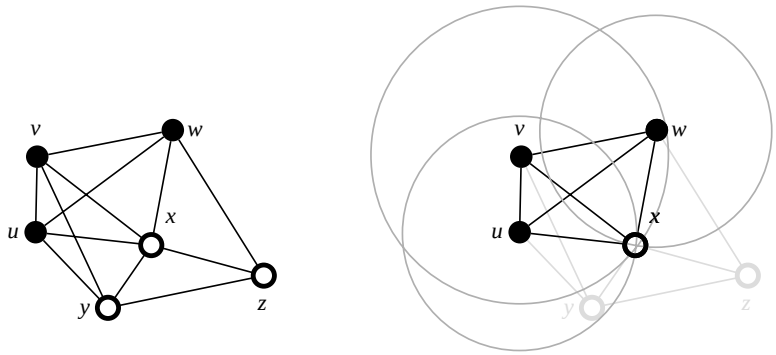


Figure 3.4: **(a)** Original **(b)** Edge (v_1, v_4) removed **(c)** Inner triangle rotated until **(d)** $|v_1, v_2|$ equals to $d(v_1, v_4)$. **(e)** (v_1, v_4) reinserted

is rigid it may not remain so after the removal of a single edge which gives rise to a flex ambiguity. The new graph can be flexed with continuous deformations arriving at a configuration to insert the removed edge with the same distance; see Figure 3.4. This type of an ambiguity is eliminated if the graph is *redundantly rigid*, i.e. it stays rigid upon removal of any single edge.

In order to formalize these ambiguities the term *globally rigid* is introduced [19]. Formally, it has been proven that a graph $\mathbb{G}(V, E)$ containing four or more vertices has unique realization in $\mathbb{R}^2$ if and only if it is globally rigid [16, 20, 21, 22, 19, 11]. A graph is globally rigid if and only if it is *3-connected* and *redundantly rigid* [16, 20]. Although a graph $\mathbb{G}(V, E)$ is globally rigid in $\mathbb{R}^2$, it is NP-Hard to find a unique realization of $\mathbb{G}(V, E)$. This problem has been correctly conjectured by [20, 16] and lately proven by [21, 22]. In spite of the NP-Hardness of the graph realization problem [11] even when the given graph is globally rigid, checking a graph for global rigidity has a polynomial time solution [23]. Furthermore it is possible to



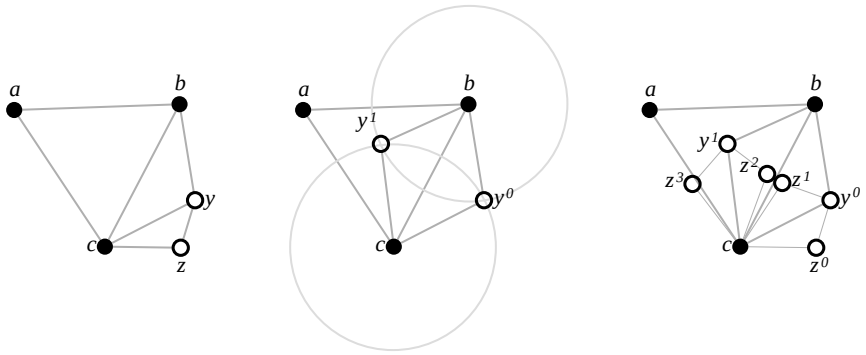
(a) (b)

Figure 3.5: **(a)** Trilateration graph, u, v, w corresponds initial K_3. **(b)** x has three uniquely localized neighbors, so the coordinate of x is the point three circles intersect.

extract globally rigid components of a graph in polynomial time if the graph itself is not globally rigid [18].

Although finding a unique realization of a globally rigid graph is NP-Hard, there exists an exceptional globally rigid graph class called *trilateration graphs* that is uniquely localizable in polynomial time. A graph is a trilateration graph if it has a trilateration ordering $\pi = \{u_1, u_2, ..., u_n\}$ where u_1, u_2, u_3 forms a K_3 and each u_i has at least three neighbor u_j that come before u_i in π, i.e. $j > i$. Figure 3.5 shows an example of a trilateration graph. A possible trilateration ordering for this graph is $\pi = \{u, v, w, x, y, z\}$. Because a node is uniquely localizable if it has 3 uniquely localized neighbors, every vertex in a trilateration ordering is localized successively since each vertex $u_i \in \pi, i > 3$ has three neighboring vertices come from earlier than u_i in ordering which makes u_i localizable with these three neighboring information. Figure 3.5b shows an example trilateration step for a single node, x which is trilaterated using u, v, w.

Bilateration graphs are defined similarly. A graph is a bilateration graph if is has a bilateration ordering $\pi = \{u_1, u_2, ..., u_n\}$ where u_1, u_2, u_3 forms a K_3 and each u_i for $i > 3$ has at least two neighbors that come before u_i in



(a) (b) (c)

Figure 3.6: **(a)** Bilateration graph, $\pi = \{a, b, c, y, z\}$ **(b)** y has flip ambiguity, two possible positions flipped through (b, c). **(c)** shows the possible positions of vertices after finite localization by fixing a, b, c.

π. We note that unlike trilateration graphs a bilateration graph may not be globally rigid therefore uniquely realizable. Instead bilateration graphs are finitely localizable i.e. fixing u_1, u_2, u_3 (to eliminate global transformations of the graph) we can find a finite set of possible locations for each $u_i \in \pi$ where $i > 3$.

Chapter 4
Distributed Iterative Cluster Localization

Distributed Iterative Cluster Localization (*DICL*) relies on two low level primitives: A reliable internode distance measurement and internode communication. There are several techniques for obtaining distance between two sensor nodes such as TDoA, RSSI, ToA [10, 13, 14]. However *DICL* does not make any assumptions on how the distance information is gathered and thus it is not dependent on a specific technology. We assume that the gathered distance information is error-free. We also assume that the communication between sensor nodes is done through broadcasting and a broadcasted data is transmitted by all neighboring nodes of the broadcaster. We also note that, each sensor node has a unique identification number. Let k_u is the number of neighboring nodes of u, hence $k_u = |N_u|$ and where each $v \in N_u$ is adjacent to u. Let $k_u^i = |N_u^i|$ where N_u^i contains nodes in i^{th} level where,

$$
\begin{aligned}
N_u^0 &= \{u\} \\
N_u^1 &= N_u \\
N_u^i &= \left(\bigcup_{j=1}^{k_u^{i-1}} N_{v_j} \right) \setminus (N_u^{i-1} \cup N_u^{i-2}), \quad 2 \leq i \leq r, v_j \in N_u^{i-1}
\end{aligned}
$$

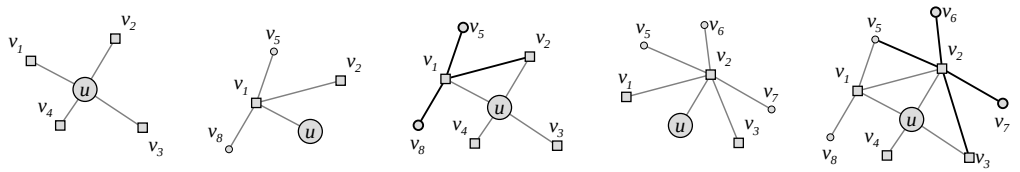
DICL consists of two main phases: *Initial Setup* and *Iterative Localization*. During initial setup, each sensor node broadcasts and gathers distance and coordinate information between its neighbors. Using this information each node u constructs a cluster centered at u and this cluster is localized in the next phase Iterative Localization.

The iterative localization phase is iterative by nature. Each cluster cen-

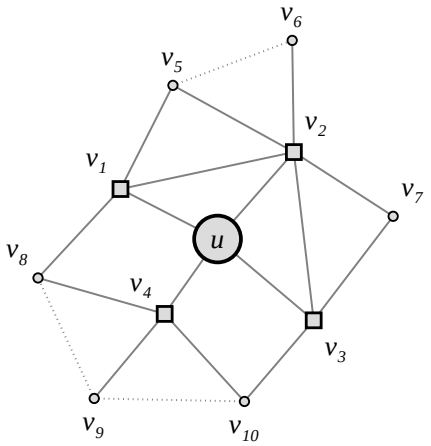
ter localizes its cluster based on its current knowledge of the cluster which consists of the internode distance and known positions. It then shares information regarding newly found unique positions within its cluster with its neighbors. Finally it collects analogous information from its neighbors. This process of cluster localization, sharing and gathering of new information is repeated at each iteration. Although our localization method takes its roots from [9, 8], several important features make it unique. In [8] the idea of clusters are proposed where each sensor node has its own cluster. Each cluster needs two-hop information to be constructed which is equivalent to C_u^2 in our problem. Although [8] collects equal amount of information to *DICL* running on C_u^2 does, it does not maximize utilization of collected data. In addition, only trilateration parts of each cluster is localized with [8] which cause less nodes to be localized with [8] than *DICL* would. Similar to [9] the localization at the cluster-level uses the concept of bilaterations and finite localization. One important difference is that although [9] is a centralized localization algorithm, *DICL* is distributed. [9] needs whole network for localization, and it only localizes one single bilateration component in the network. We introduce the concept of incremental finite localization to achieve the localization at the cluster-level. Rest of this chapter provides the details of the localization algorithm executed by each node u in $\mathbb{N}$.

Algorithm 1 Executed at each $u \in V$ in a distributed fashion. Collects necessary information from its neighbors and constructs its cluster C_u^r.

```
1: procedure INITIAL SETUP
2:     for i ← 0, r do
3:         Broadcast Packet_{N_u^i}
4:         Gather Packet_{N_v^i}, v ∈ N_u
5:     Construct C_u^r
6:     LOCALIZE C_u^r
```



(a) (b) (c) (d) (e)

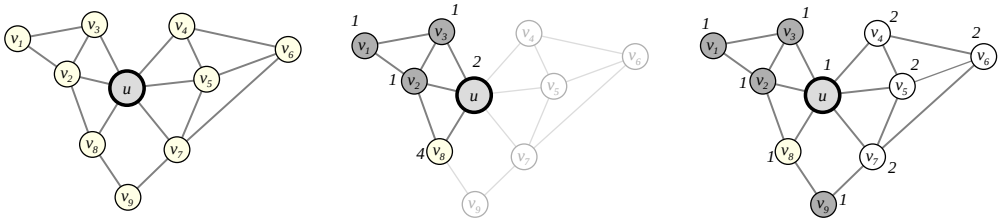


(f)

Figure 4.1: **(a)** $N_u \cup u$. **(b)** $N_{v_1} \cup v_1$. **(c)** $N_u \cup N_{v_1}$. **(d)** $N_{v_2} \cup v_2$. **(e)** $N_u \cup N_{v_1} \cup N_{v_2}$. **(f)** Final C_u except dashed edges

4.1 Initial setup

Algorithm 1 describes the initial setup phase for node u. (u_{xcoord}, u_{ycoord}) indicates the global coordinates of node u. Initially the coordinates of a non-anchor node is (*null, null*). In order to construct a cluster, C_u^r, there should be r pair of broadcasting and gathering. Each iteration u broadcasts a packed $Packet_{N_u^i}$ where,



(a) (b) (c)

Figure 4.2: The italic numbers near vertices indicate number of possible positions of vertices. **(a)** is C_u. **(b)** Localized C_u using v_1, v_2, v_3 as anchors. **(c)** Incremental localization on **(b)** with given anchor v_9.

$$
\begin{aligned}
Packet_u &= [u_{xcoord}, u_{ycoord}] \\
Packet_{N_u} &= [Packet_u, (d(u, v_1), ..., d(u, v_{k_u}))],\ v_i \in N_u, 1 \leq i \leq k_u \\
Packet_{N_u^i} &= [Packet_{N_{v_1}}, ..., Packet_{N_{v_{k_u^i}}}],\ 0 \leq i \leq r, v_j \in N_u^i, 1 \leq j \leq k_u^i
\end{aligned}
$$

And after each broadcast, u gathers each $Packet_{N_v^i}, v \in N_u^i$, it constructs the cluster C_u^r; see Figure 4.1. Finally, u starts the second phase of *DICL*, iterative localization.

Algorithm 2 Executed by each $u \in V$ in a distributed fashion. Localization procedure in order to localize C_u.

```
1: procedure LOCALIZE C_u
2:     loop
3:         A_u ← Gather each Packet_{A_v}, v ∈ N_u
4:         if any new anchor exists in A_u then
5:             INCREMENTALFINITELOCALIZATION(A_u)
6:             A_u = {v_1, ..., v_{|A_u|}}
7:             Each v ∈ A_u is an anchor and have not been broadcasted.
8:             Broadcast Packet_{A_u}
```

4.2 Iterative Localization

The goal of localization procedure is localizing as possible as vertices to share them with neighboring nodes, Algorithm 3 shows the pseudocode for

localization procedure. This localization process stops at some iteration point where there are no new information to handle. More specifically, at each iteration u finitely localizes C_u^r using recently discovered set of anchors, A_u. And we note that every finitely or uniquely localized node is appended in a list named L. Since finite localization is an incremental process, each time Algorithm 2 calls *IFL* procedure it continues its localization from where it was left; see Algorithm 3.

Algorithm 3 Finite localization procedure for given set of anchors.

```
1: procedure IncrementalFiniteLocalization(Anchors[ ])
2:     Let L keeps all uniquely or finitely localized vertices.
3:     for all a ∈ Anchors do
4:         Let v equals to a_vertex and p equals a_coord
5:         if |v.positions| = 0 then
6:             Append p to v.Positions and v to L
7:         else
8:             for all p_v ∈ v.Positions do
9:                 if p_v ≠ p then RecursivelyRemove(p_v)
10:    π ← FindBilaterationOrdering(L)
11:    for all v ∈ π do
12:        Ancestors ←{Neighbors of v in L}
13:        Bilaterate(v, Ancestors[1], Ancestors[2])
14:        for i ← 3, |Ancestors| do
15:            UpdateBilateration(v, Ancestors[i])
```

Each time *IFL* runs, given set of nodes are set as anchors; see Algorithm 3. There are two cases, v could be a finitely localized node or an unlocalized node. When v is an unlocalized node, the given coordinate p is set as vertex's position. If v is finitely localized, each position $v^i \in v.positions$ where $v^i \neq p$ is removed, since one of the positions in $v.positions$ has to be the real coordinate of v, there will be always a position $v^i = p$ is left.

After *IFL* sets all given nodes as anchors, a bilateration ordering $\pi = \{s_1, s_2, ..., s_{|\pi|}\}$ is generated where all finitely and uniquely localized nodes in previous iterations are in π initially. And then, main part of the localization will start localizing nodes in π successively starting from the first unlocalized node. Since every node $v \in \pi$ has at least two neighbors $a, b \in L$,

Algorithm 4 Sub procedures for *IFL*.

procedure BILATERATE(vertex v, vertex a, vertex b)
  Set all positions of a and b as **invalid**
  for all $p_a \in a.Positions$ **do**
    for all $p_b \in b.Positions$ **do**
      if CONSISTENT(p_a, p_b) **then**
        $(p_{v_1}, p_{v_2}) \leftarrow$ CIRCLEINTERSECTION(p_a, $d(a, v)$, p_b, $d(b, v)$)
        $p_{v_1}.Ancestors \leftarrow p_{v_2}.Ancestors \leftarrow$ MERGEANCS(p_a, p_b)
        Append pv_1 and p_{v_2} to $v.Positions$
        Set p_a and p_b as **valid**
  Append v to L
  RECURSIVELYREMOVE **invalid** positions of a and b

procedure UPDATEBILATERATION(vertex v, vertex a)
  Set all positions of a and b as **invalid**
  for all $p_a \in a.Positions$ **do**
    for all $p_v \in v.Positions$ **do**
      if CONSISTENT(p_a, p_b) **AND** $|p_v, p_a| = d(v, a)$ **then**
        Append p_a to $p_v.Ancestors$ ($p_v.Ancestors[a] \leftarrow p_a$)
        Set p_a and p_b as **valid**
  RECURSIVELYREMOVE **invalid** positions of v and a

procedure CONSISTENT(position p_a, position p_b)
  for all $v \in V$ **do**
    if $p_a.Ancestors[v] \neq null$ **AND** $p_b.Ancestors[v] \neq null$ **then**
      if $p_a.Ancestors[v] \neq p_b.Ancestors[v]$ **then**
        return **false**
  return true

procedure MERGEANCS(position p_a, position p_b)
  Array $Ancestors[\ ]$
  for all $v \in V$ **do**
    $Ancestors[v] \leftarrow p_a.Ancestors[v]$
    if $Ancestors[v] = null$ **then**$Ancestors[v] \leftarrow p_b.Ancestors[v]$
  $Ancestors[a] \leftarrow p_a$ and $Ancestors[b] \leftarrow p_b$
  return Ancestors

procedure RECURSIVELYREMOVE(position p_a)
  RECURSIVELYREMOVE each p_v where $p_a \in p_v.Ancestors$
  Delete p_a

v is *bilaterated* once; see Algorithm 4. Bilateration of v, iterates every position pair a^i b^j, $a^i \in a.positions, b^j \in b.positions$ and checks against *consistency* where two positions a^i and b^j are not consistent if and only if $s^c \in a^i.ancestors$ while there exists another position of s, $s^h \in b^j.ancestors$, $i \neq j, c \neq h$; see Algorithm 4. If a^i, b^j pair is consistent, two circles, one is centered on a^i having a radius of $d(a, v)$ and other is centered on b^j with a radius of $d(b, v)$ are intersected. The intersected points are assigned as new positions of v which are v^x and v^{x+1} where $x + 1 = |v.positions|$ after ancestors of v^x and v^{x+1} are generated with *merge ancestor* procedure; see Algorithm 4.

After bilateration of v, each position of a and b which marked as *invalid* is removed. Remove operation is a recursive procedure and removal of a position $s^i \in s.positions$ removes each position c where $s^i \in c.ancestors$ recursively; see Algorithm 4. When a position does not satisfy the distance to its any of neighbors it is marked as invalid since each position has to have a connection between any single position of each neighboring node.

After v is bilaterated, it is updated with each remaining neighbor of v; see Algorithm 4. *Update bilateration* procedure, takes recently localized two nodes v and a where v was just bilaterated, and checks whether each possible position pair of $v^i \in v.positions$ and $a^j \in a.positions$ is consistent and satisfies the geometric distance. If both conditions are satisfied for v^i and a^j, a^j becomes an ancestor of v^i. Update bilaterate procedure removes all positions of a and v which are marked as invalid.

After *IFL* is executed, A_u is cleared and then filled with nodes where each $v \in A_u$ is an anchor and not broadcasted before. And those nodes are packed and then broadcasted to the neighbors in the form of the $Packet_{A_u}$ where,

$$Packet_{A_u} = [Packet_{v_1},\ ...,\ Packet_{v_{|A_u|}}] \text{ where } v_i \in A_u, 1 \leq i \leq |A_u|$$

4.3 Analysis of *DICL*

We analyze the messaging overhead, computational complexity and the space requirements of *DICL*. There are two performance measurements for messaging overhead, number of broadcasts and total amount of broadcasted information. We define k as mean of the average degree of each sensor node in $\mathbb{N}$, $k = (k_{v_1} + k_{v_2} + ... + k_{v_n})/n$. As *DICL* is described into two phases, analysis is done with the same way. Before analyzing Algorithm 1 the size of packets used is defined where $Packet_u$ has a size of $O(1)$, size of $Packet_{N_u}$ is $O(k)$ and $Packet_{N_u^i}$ is expected to be $O(k^i), 1 \leq i \leq r$. Hence number of broadcasts in Algorithm 1 is $O(k^r)$ and total amount of packets broadcasted is $O(k^r)$. In the Iterative Localization phase each recently discovered anchor is broadcasted at most once in $Packet_{A_u}$. The number of nodes in a cluster C_u^r is bounded by $O(k^r)$, therefore total size of all packets broadcasted in the network is $O(k^r)$ which is same as the first phase. As a result, the total messaging size is $O(k^r)$ which is relatively linear to big values of n. In terms of running time and memory requirements, a single execution of *IFL* takes $O(2^{k^r})$ in the worst case. Since *IFL* has exponential running time and memory requirements in cluster level, assuming cluster sizes are constant each *IFL* requires constant time. And experimental results also shows us that non of the r and k value pair reached any exponential running time, memory requirement nor messaging overhead.

Chapter 5
Experimental Setup and Results

The implementation was coded in C++ using LEDA library [24]. Because DICL is a distributed algorithm, a discrete event simulation system has been designed to test DICL. Experiments are performed on a computer with the configuration of AMD X2 3800+ of CPU and 3GB of RAM.

5.1 Random Network Generation and Parameters

Preliminaries mentions about the conversion of the network $\mathbb{N}$ to a $\mathbb{G}(V, E)$ therefore, generating a random network is equivalent to generating a random graph $\mathbb{G}$. Random Graph Generator (*RGG*) takes some sort of inputs and generates a random graph $\mathbb{G}$ which is then fed to *Network* simulation class where all experiments are simulated. The parameters *RGG* needs are: *number of nodes in network* and *average degree*. Number of nodes in network (n) corresponds the number of physical sensor nodes in $\mathbb{N}$. Second parameter is average degree (k) where expected number of neighbors of each node is k each node's expected number of neighbors is k and the average degree of $\mathbb{G}$ equals to k as it is also defined in Chapter 4. In order to generate a random graph, *RGG* first generates n random nodes in plane with random number generation method of LEDA [24] where each axis of a point is bounded with 500. After that, while range value of sensor nodes is increasing, each sensor node is connected other nodes in that range. This process iteratively continues until average degree of graph equals to k.

We note that since seed values are used for each configuration, all experiments are clearly reproducible in any platform. We generated seed sequence for experiments from a generator seed "DICL"that is 3425.

We have generalized *DICL* for variable r value which varies between 2-8

in experiments to determine optimum r value. And k differs between 4 and 16. We fix n at 100 because performance of *DICL* is only related with k and r not n. Each unique configuration is repeated 10 times with different seed values taken from seed sequence successively.

5.2 Experiments and Discussion of Results

We select our performance measures in order to analyze and construe localization, messaging performances, running time and space requirements. For a given n, we compute *average of localized node ratio* (lnr) as follows $lnr = ln/n$ where ln is the number of uniquely localized nodes in $\mathbb{N}$. The second and third performance measurements are *average broadcast count per node* (bc) and *average broadcasting amount per node* (ba) which are closely related with each other. bc counts number of messages in $\mathbb{N}$ broadcasted and takes its average by dividing it n. Because the size of each broadcasted packet differs specifically at the second phase, only counting bc does not show bandwidth usage per node therefore we compute the total size of broadcasted packets and take its average through $\mathbb{N}$. The fourth indicator *average time per node* (at) is the average running time required by *DICL* on a specific node. Last two performance measures are related to highlight how much space required by *DICL*. *Maximum possibilities per node* (mp), let mp_u is the maximum value of $|v.positions|$ where $v \in C_u^r$, then $mp = \underset{v_i \in \mathbb{N}}{\text{Max}}(mp_{v_i})$. Hence simulations may take too much time to run for bigger values of r, we bound $mp_u, u \in C_u^r$ at 1024 to prevent number of possibilities for each node not to grow exponentially. And finally, *Average of total possibilities per cluster* (tp) is the average of $\sum_{i=1}^{n} tp_{v_i}$ for all $v_i \in \mathbb{N}$.

Figure 5.1 is a visual illustration of the performance of *DICL*. The *LNR* values for r =1,2,3 for a specific random network which is included in experiments also are shown. We note that r =1, *DICL* works like analogous iterative trilateration. Even for a small value of $r = 3$ 80% of the network is localized. It is also important to note that the number of iterations per

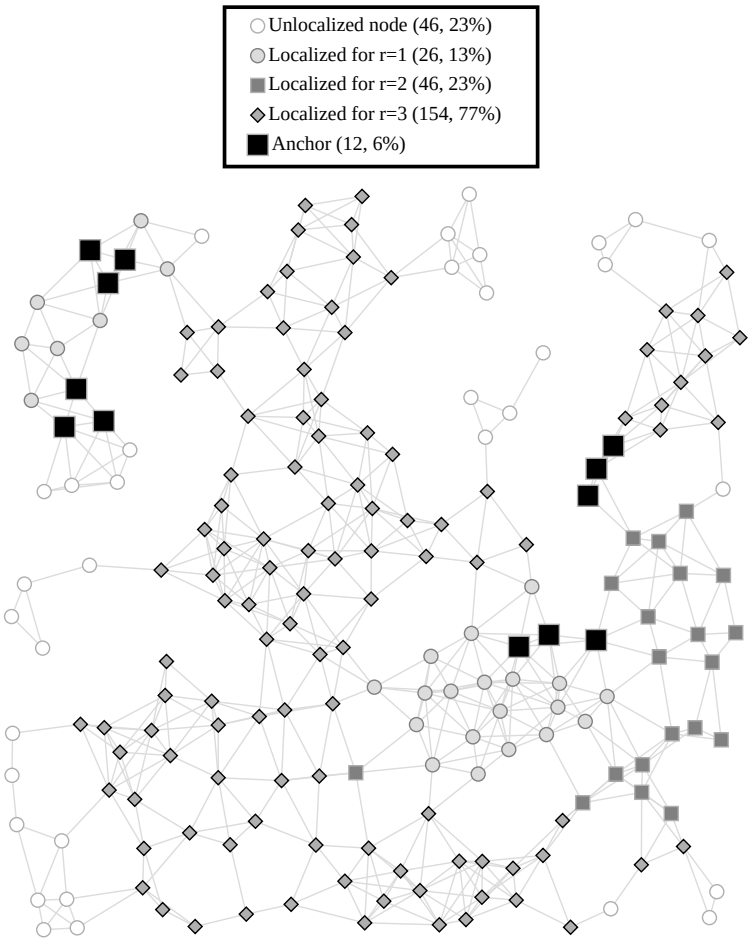


Figure 5.1: Random network localized with *DICL*. n =200, k =6.

node is almost 3 and *MP* is limited with 128 where $r = 3$. Which shows *DICL* provides a high localization ratio even if the network is sparse (k =6) and at the same time it is amenable for distributed implementation.

Figure 5.3 shows the lnr values for different r and k values. As r increases, lnr grows as expected however for $r > 5$ localization is not effected anymore. *DICL* localizes the graph shown in Figure 5.2 only for values of $r > 4$. Thus, the reason why bigger values of r can not localizes additional vertices is that occurrence possibility of such partial graphs are very low.

The total number of broadcasts per node during experiments are shown in Figure 5.4. Since first phase of algorithm is constant which broadcasts always r times in order to construct C_u^r, the irregularity in the plot caused by

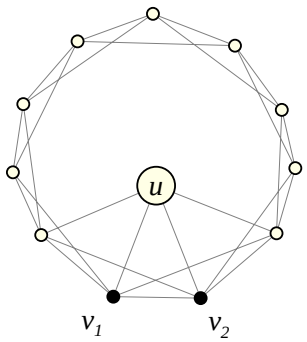


Figure 5.2: Globally rigid bilateration graph, u covers all edges and vertices in this graph for $r \geq 4$, for $r < 4$ this graph can not be localized with *DICL*

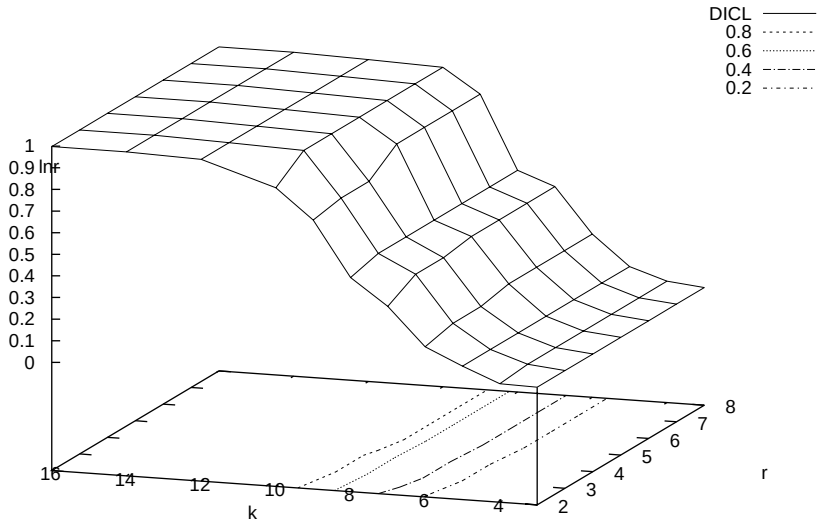


Figure 5.3: The ratio of localized nodes with *DICL* to n

the broadcasts in the Iterative Localization phase. Number of broadcasts on second phase depends on how many anchors are localized in each iteration which changes each iteration. There are more message overhead between $6 \leq k \leq 10$. For sparse networks when $k < 6$, the messaging overhead is low since not that many nodes are localized to be broadcasted in the first place. In contrast when $k > 10$ each localization iteration uniquely localizes many nodes at once therefore requires fewer broadcasts. However as can be verified in Figure 5.5, the *BA* values indicating the broadcast size per node

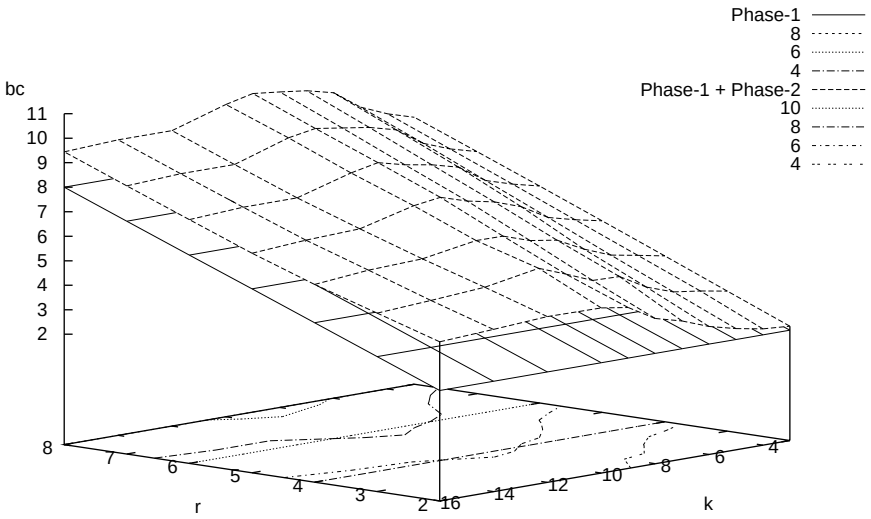


Figure 5.4: Average number of broadcasts per node

grows proportionally in terms of k and r.

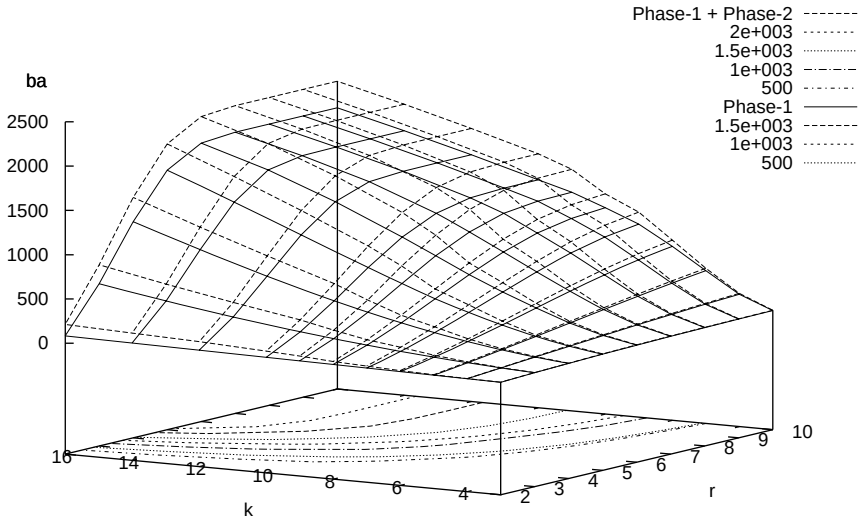


Figure 5.5: Average amount of data broadcasted in units

Figure 5.6 shows the number of finite positions per node during execution of *IFL*. The peak values are reached at $7 \leq k \leq 10$ for almost all r values, since low connectivity does not enable too many bilaterations, therefore possible locations, whereas high connectivity leads to unique lo-

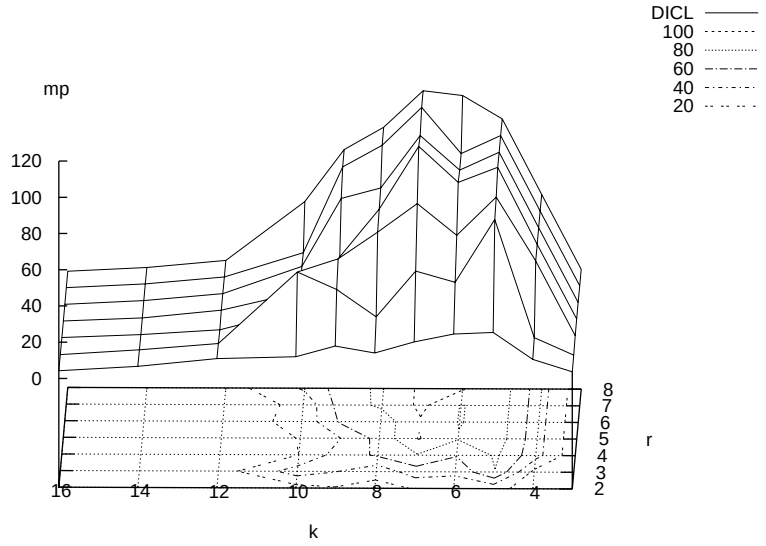


Figure 5.6: Maximum possibilities per node

calization too quickly. Similar reasoning could apply to TP except that cluster size plays an important role as well in this case; see Figure 5.7. The excepted cluster size is k^r, therefore for large values of k ($k \geq 9$), TP is constant or grows slightly even though MP decreases. High connectivity leads to ease of unique localization but also gives rise to large clusters, which seem to cancel out each others' affects in terms of space requirements of a sensor node.

Actual running times for each node is shown in the Figure 5.8 in terms of seconds. Since the total time that spent for localization by each node is directly dependent on size of C_u^r. For all values of r, the changes on k affects running time similarly which means the discriminant parameter is k not r in terms of running times. In addition, growth in Figure 5.8 seems to be similar to that of TP for $k \leq 9$.

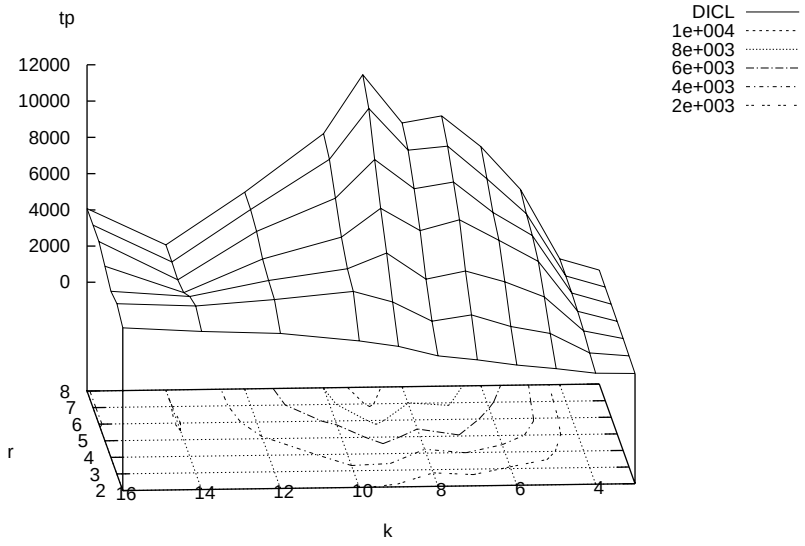


Figure 5.7: Total possibilities of each cluster contain end of localization

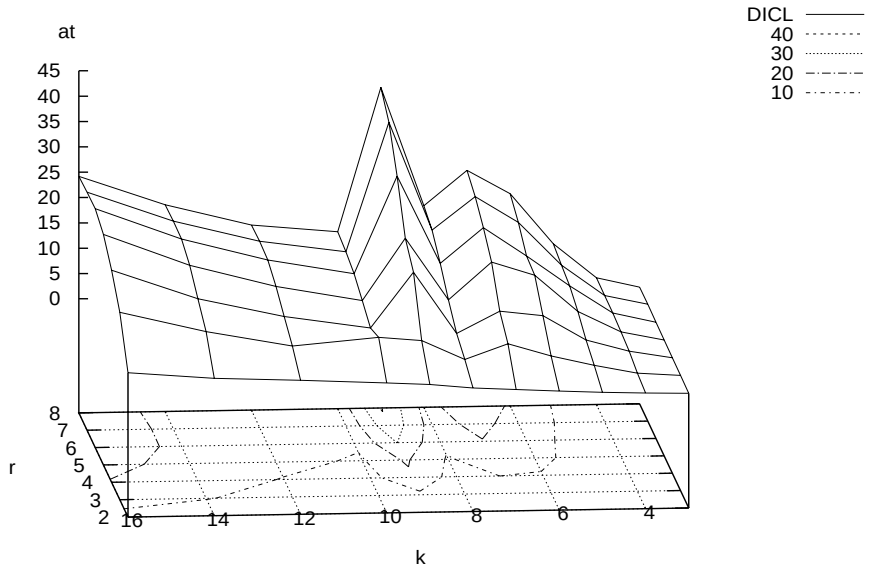


Figure 5.8: Running time in seconds during *DICL*

Chapter 6
Conclusion

We have designed and simulated a Distributed Iterative Cluster Localization algorithm for wireless sensor networks. And the experimental results show that *DICL* has a success on localization of large networks. It offers to localize a network with reasonable memory requirement and messaging overhead in feasible running time even if it has exponential running time in theory. An important direction for future is to generalize the localization framework to handle error in measurements. And merging two localization phases together which leads localization to start while C_u^r is constructing. Since we have shown that *DICL* is successful in localization even in sparse networks, more realistic parameters could be considered such as transmission errors, transmission delays and transmission rates. It would also be useful to conduct experiments to analyze the efficiency of model when the network is employed in various regions with randomly placed obstacles.

References

[1] Y. Sankarasubramaniam I. Akyildiz, W. Su and E. Cayirci. A survey on sensor networks, 2002. [cited at p. 1, 3]

[2] E. Demaine N. B. Priyantha, H. Balakrishnan and S. Teller. Anchor-free distributed localization in sensor networks. Technical report, 2003. [cited at p. 3, 4, 5]

[3] A. Basu and J. S. B. Mitchell. Abstract distributed localization using noisy distance and angle information. [cited at p. 3, 4]

[4] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Comput. Networks*, 43(4):499–518, November 2003. [cited at p. 3, 4]

[5] S. G. Kobourov A. Efrat, D. Forrester and C. Erten. A force-directed approach to sensor localization. In *In 13th Symposium on Graph Drawing (GD.* [cited at p. 3, 5]

[6] J. S. B. Mitchell A. Basu, J. Gao and G. Sabhnani. Distributed localization using noisy distance and angle information. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 262–273, New York, NY, USA, 2006. ACM. [cited at p. 3, 5]

[7] Q. Shi F. Niu, Kyperountas and J. Huang. Rigid body based location technology for ad hoc sensor networks. *Networking, Sensing and Control, 2006. ICNSC '06. Proceedings of the 2006 IEEE International Conference on*, pages 926–931, April 2006. [cited at p. 3, 5, 6]

[8] D. Rus D. Moore, J. Leonard and S. Teller. Robust distributed network localization with noisy range measurements. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 50–61, New York, NY, USA, 2004. ACM. [cited at p. 3, 4, 5, 12]

[9] M. Cao D. K. Goldenberg, P. Bihler and J. Fang. Localization in sparse networks using sweeps. In *In Proc. of ACM MobiCom*, pages 110–121, 2006. [cited at p. 3, 5, 12]

[10] J. Hightower and G. Borriello. Location systems for ubiquitous computing. *Computer*, 34(8):57–66, 2001. [cited at p. 3, 4, 11]

[11] D. K. Goldenberg A. S. Morse W. Whiteley Y. R. Yang B. D. O. Anderson J. Aspnes, T. Eren and P. N. Belhumeur. A theory of network localization. *IEEE Transactions on Mobile Computing*, 5(12):1663–1678, dec 2006. [cited at p. 3, 8]

[12] L. Hu and D. Evans. Localization for mobile sensor networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 45–57, New York, NY, USA, 2004. ACM. [cited at p. 3]

[13] B. Fidan G. Mao and B. D. O. Anderson. Wireless sensor network localization techniques. *Comput. Netw.*, 51(10):2529–2553, 2007. [cited at p. 4, 5, 11]

[14] A. Woo F. Jiang K. Whitehouse, C. Karlof and D. Culler. The effects of ranging noise on multihop localization: an empirical study. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 10, Piscataway, NJ, USA, 2005. IEEE Press. [cited at p. 4, 5, 11]

[15] J. Aspnes, D. Goldenberg, and Y. R. Yang. On the computational complexity of sensor network localization. In *In Proceedings of First International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, pages 32–44, 2004. [cited at p. 6]

[16] B. Hendrickson. Conditions for unique graph realizations. *SIAM J. Comput*, 21:65–84, 1992. [cited at p. 6, 7, 8]

[17] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4:331–340, 1970. [cited at p. 6]

[18] W. Whiteley T. Eren, D. K. Goldenberg and Y. R. Yang. Rigidity, computation, and randomization in network localization. In *In Proceedings of IEEE INFOCOM 04, Hong Kong*, pages 2673–2684, 2004. [cited at p. 7, 9]

[19] W. C. Maness Y. Richard Y. A. Young A. S. Morse A. Savvides D. K. Goldenberg, A. Krishnamurthy and B. D. O. Anderson. Network localization in partially localizable networks. In *In Proceedings of IEEE INFOCOM 05*, pages 313–326, 2005. [cited at p. 8]

[20] R. Connelly. Generic global rigidity. *Discrete Comput. Geom.*, 33(4):549–563, 2005. [cited at p. 8]

[21] A. R. Berg and T. Jordán. A proof of connelly's conjecture on 3-connected circuits of the rigidity matroid. *J. Comb. Theory Ser. B*, 88(1):77–97, 2003. [cited at p. 8]

[22] T. Reports, B. Jackson, and T. Jordn. Connected rigidity matroids and unique realizations of graphs, 2003. [cited at p. 8]

[23] D. J. Jacobs and B. Hendrickson. An algorithm for two-dimensional rigidity percolation: the pebble game. *J. Comput. Phys.*, 137(2):346–365, 1997. [cited at p. 8]

[24] K. Mehlhorn and S. Naher. Leda: A platform for combinatorial and geometric computing. *Communications of the ACM*, 38, 1999. [cited at p. 19]

Curriculum Vitae