ABSTRACT

Title of thesis: BOOTSTRAPPING FREE-SPACE OPTICAL NETWORKS

Fang Liu, Master of Science, 2004

Thesis directed by: Professor Uzi Vishkin

Department of Electrical and Computer Engineering

We consider one challenging problem in establishing a Free Space Optical (FSO) network. In our model, it is assumed that each node is a base station and its number of transceivers is limited. Such a network can be abstracted by a graph where each node represents a base station and each edge represents a link connecting two base stations. The problem is that of forming a connected topology, which is known to be NP-complete because of the transceiver limitation. What makes this problem even more challenging is the need to have a “distributed” solution to form a connected topology, because a node can have knowledge only of its neighbors. We have developed a fully distributed approximation algorithm, which constructs a spanning tree with maximal node degree at most one larger than that in the optimal solution. Due to its distributed nature, this algorithm outperforms serial algorithms.

BOOTSTRAPPING FREE-SPACE OPTICAL NETWORKS

by

Fang Liu

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2004

Advisory Committee:

Professor Uzi Vishkin
Assistant Professor Gang Qu
Dr. Stuart Milner

ACKNOWLEDGEMENTS

The author would like to thank Dr. Uzi Vishkin (Professor, University of Maryland) for giving her valuable guidance and an opportunity to work on an interesting problem. The author wants to thank Dr. Gang Qu (Assistant Professor, University of Maryland) for serving on the thesis committee and helping her by his invaluable suggestions. The author is grateful towards Dr. Stuart Milner (Senior Research Scientist, Civil and Environmental Engineering, University of Maryland), who inspired the author to work on this interesting research problem and also provided valuable insights, suggestions and critique.

The author would like to express her gratitude towards Aniket Desai and Jaime Llorca for valuable discussions about the problem, methodologies as well as the information about working of the optical networks. Aniket rendered his precious time to help the author make her thesis more accurate.

The author would also like to thank Dr. Christopher Davis (Professor, University of Maryland), Tzung-Hsien Ho for giving her useful data about optical networks and Xingzhi Wen who read the original draft and rendered suggestions.

Finally the author wants to express her special gratitude towards Mr. Donald Hirsch, who volunteered to be the editor.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

In the modern communication area, direct, line-of-sight, optical communication by lasers is known as Optical Wireless or Free-Space Optical communication. The main advantages of this technology are a high data rate and security. It has been experimentally verified that a transmitter at a node can point a narrow-width laser beam with very small wavelength towards a receiver by using a technique called as “Pointing Acquisition and Tracking” (PAT). Depending upon the weather and atmospheric obscuration, this beam can go from a few meters to several kilometers [LDVDM-03]. Unlike RF communication, it’s difficult for the third party to monitor the signals, if it does not physically intercept the optical laser beam between the transmitter and the receiver.

We abstracted the all-FSO network consisting of base stations and point-to-point optical links between two base stations by a graph. In our model,

- Initially, there are no point-to-point optical laser links between any nodes in the network.
- Each node can send and receive some simple signals by beacons (omni-directional or directional beacons) or other signaling systems, such as GPS and cameras, to detect the location of its potential neighbors.For example, the beacon can be an omni-directional light source used for signaling purposes, i.e., to say “hello” or some other link state query. Each node in this network could involve a fish-eye lens which is used to detect the beacons within the given distance with which it has line of sight. The purpose of using beacons or GPS system is to build

the potential-link graph, from which the minimum degree spanning tree is derived.

- Initially, each node can only discover those nodes (i.e. location and atmospheric obscuration between them) whose beacon signaling “hello” can be received by this node (by its fish-eye lens).
- The number of transceivers at each node is constrained for mechanical and financial feasibility reasons. These constraints are even more severe if each node represents an aircraft or other mobile object.
- Each transmitter is paired with a receiver, and the transmitter and receiver of each pair point in the same direction, thus making each link bi-directional. FIG. 1.1 is an example of two nodes each with four pairs of transceivers. A bi-directional link between the two nodes in the figure is shown. From now on, the links, as shown in FIG 1.1, are called as transceiver-to-transceiver links.

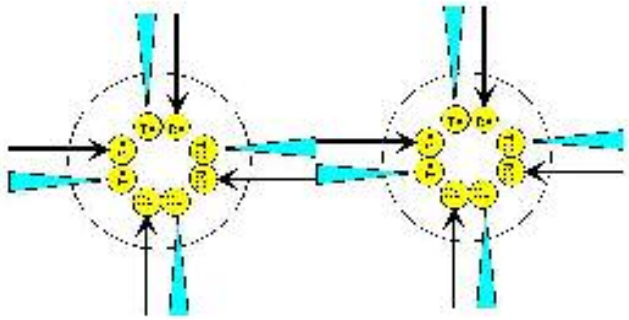


FIG 1.1 TX (TRANSMITTER) AND RX (RECEIVER) PAIR

This thesis investigates efficient algorithms to set up the initial connected optical wireless network for such a model through high-data-rate beams. The efficiency is measured by the time it takes to build a connected high-data-rate laser network in the physical layer. Therefore, these algorithms are designed to ensure "fast connectivity" rather than optimizing other metrics, such as cost of the potential links or average end-to-end traffic in the network. In the real FSO network, to build the initial connected network in a short time is one of the most critical missions. Only after the nodes are connected by high data-rate links, will we be able to optimize the communication quality or reconfigurate the network.

As explained earlier, each node can discover a beacon from other nodes by its fish eye lens (if we use omni-directional beacons) or other point-to-point hardware (if we use directional beacons). If two nodes can discover the beacons of each other, an abstract "beacon-link" exists between them.

The connectivity problem for FSO networks can be transformed into a graph problem, which is known as the Minimum-Degree Spanning-Tree problem[FR-92]. The input is a feasibility graph G = (V, E) with |V|=N vertices and |E| is the set of potential network links. The output graph will be a spanning tree where the node degree represents the number of transceivers used by the node. We define *the degree of the tree* as the largest degree of a node in the spanning tree. Given the above input graph G (V, E), we seek to construct a spanning tree whose maximum degree (i.e., the largest degree of a node in the spanning tree) is the smallest possible. By finding a spanning tree with smallest "degree

of the tree", we increase the likelihood of forming connected networks, when the number of transceivers on each node is constrained. For example, the nodes can be moving aircrafts, which do not have enough space to host many transceivers. We model the physical problem in the following natural way:

Construct a spanning tree given the feasibility graph G = (V, E) with |V|=N vertices whose maximum degree is the smallest among all spanning trees of G. This is an *optimization problem.* An optimization problem is defined as a computational problem in which the object is to find the best of all possible solutions, more formally, to find a solution in the feasible region which has the minimum (or maximum) value of the objective function [GJ-79].

As the Minimum-Degree Spanning Tree problem is NP-Complete, we will describe a bottom-up approach to compute a spanning tree of degree at most $\delta+1$, where δ is the degree of the optimal spanning tree. In the output spanning tree, the tree edges are transceiver-to-transceiver laser beams.

Our algorithm is an *approximation algorithm*. Approximation algorithm is defined as an algorithm, which solves an optimization problem that runs in polynomial time in the length of the input, and outputs a solution that is guaranteed to be close to the optimal solution [GJ-79]. "Close" has some well-defined sense called the approximation guarantee. We will prove that our algorithm can generate a spanning tree with degree at most one larger than the optimal.

In this thesis, Chapter 2 provides the background and related work, including the assumptions and pending issues in our FSO model, and a top-down algorithm for the of the Minimum-Degree Spanning Tree problem. We present a novel bottom-up approach, which is a distributed algorithm, to build the bootstrapping all-free-space-optical network in Chapter 3. The algorithm analysis is presented in Chapter 4. Based on the bottom-up approach that we have designed, we also attempt to parallelize this algorithm in Chapter 5 and check if this parallel algorithm outperforms the distributed algorithm. The simulation model and results are shown in Chapter 6. Chapter 7 points out some future research directions and Chapter 8 concludes this thesis.

CHAPTER 2. BACKGROUND

2.1 Assumptions and Difficulties:

We first present a few assumptions and concepts about the network model consisting of FSO technology:

- Each node in the network has a unique node ID.
- As we mentioned previously, each node in the network can send very simple signals by beacons. In our case, the signals include node ID and some other simple information (i.e., location) requiring a few bits.
- If node A can “see” node B by its beacon, node B can also “see” node A by its beacon. Then, the (bi-directional) “beacon link” between node A and node B is an edge in feasibility graph G.
- Feasibility graph G consists of all the visible beacon links in this network.
- Choosing a pair of nodes in G to form an edge in the spanning tree requires allocating one transceiver in each node to that edge.
- Once an edge of G is picked as a tree edge, two endpoints can communicate with each other, through a high-data-rate communication channel (optical laser beam), as supported by the transceiver-to-transceiver hardware.
- Group: The group consists of such nodes that between any two nodes of the “same group”, a path consisting of high-data-rate links (optical laser beams) exists.

The constraints envisioned on these next generation FSO networks are those of expenses, feasibility and security. The difficulties in our model are:

- Our problem needs to cope with a very limited number of transceivers at each node. It does not allow using a higher number of transceivers to be used even temporarily.
- Nodes belonging to separate groups cannot exchange any information, except using a very low-data-rate beacon signal.
- Due to the above limitation, nodes need to pair up to form tree edges of the spanning tree in a distributed fashion.

2.2 Related Work

A “top-down” approximation algorithm, which gives a solution for the Minimum-Degree Spanning Tree within one-from-the-optimal degree, was presented in [FR-92]. It starts with some arbitrary spanning tree of the feasibility graph G, and then iteratively improves upon it until a tree, whose degree is guaranteed not to exceed $\delta+1$, is produced.

Unfortunately, due to the limited number of transceivers at each node, our model does not allow using a higher number of transceivers in temporary computation stages. The degree of an arbitrary spanning tree can be higher than the specified limitation on the number of transceivers per node. In other words, the hardware in FSO network cannot support the top-down algorithm, which requires the higher degree than the hardware condition can afford. Apparently, the top-down algorithm is physically impossible for constructing an

initial FSO network. Hence, we cannot apply this top-down approach (as it starts with an arbitrary spanning tree without regard to degree limitation).

However, we'll use some concepts and observations from the top-down approach [FR-92]. They will supply us with important supporting proofs when we analyze our algorithm's performance later in this thesis.

- THEOREM 1.4. [FR-92]. There is a polynomial time approximation algorithm which produces a spanning tree of degree at most (δ + 1).
- The top-down approximation algorithm works iteratively where each iteration is an improvement step as follows.

  DEFINITION 5.1. [FR-92]. Let T be a spanning tree of degree k of a graph G. Let (u, v) be an edge in G, which is not in T. Let p(u) be the degree of node u in T. Suppose w is a vertex in the cycle generated by adding (u, v) to T. If p(u) >= k– 1, we say that u blocks w from (u, v). If neither u nor v blocks w, then (u, v) can be used to reduce the degree of w through a local improvement step. In such a case, we say w benefits from (u, v).
- Concept of "Improvement" [FR-92]: Let T be a spanning tree in graph G. Let p(w) be the degree of node w. Consider an edge (u, v) of G which is not in T. Let C be the unique simple cycle generated when (u, v) is added to T. Suppose there is a vertex w in C with the property that p(w) >= max(p(u), p(v)) +2. We now introduce an "improvement" in T by adding the edge (u, v) and deleting one of the

edges in C incident on w. We call this step an improvement because the maximum of {p(u), p(v), p(w)} has decreased by at least one.

- THEOREM 5.1. [FR-92]. Let T be a spanning tree of degree k of a graph G. Let δ be the degree of a minimum-degree spanning tree. Let S be the set of vertices of degree k. Let B be an arbitrary subset of vertices of degree k – 1 in T. Let S U B be removed from the graph, breaking the tree T into a forest F. Suppose G satisfies the condition that there are no edges between different trees in F. Then k is at most δ+l.

Please refer to [FR-92] for proofs of the above theorems. We will use their conclusions directly, but will not prove them again in this thesis.

The top-down algorithm starts from a spanning tree, i.e., the network has already become a connected graph through high data-rate laser links. Based on this spanning tree, the top-down algorithm makes “improvement” for the tree until its degree is no larger than δ+l. The top-down algorithm is not suitable for constructing connected FSO networks, because it needs an initial connected network and it does not consider about the transceiver limitation. Oppositely, our bottom-up algorithm starts from an empty tree, i.e., there is no initial tree or connected network. Afterwards, the bottom-up algorithm will make the network connected through high data-rate laser links, while building the spanning tree within the degree limit. Thus, the bottom-up algorithm can be applied to build FSO networks.

There has been some previous work in determining FSO topologies using a similar algorithm based on “clustering technique” [DSM-03]. The clustering algorithm is also a bottom-up algorithm. It is based on a position table and a link state table. Each cluster always selects the node with shortest distance from itself in another cluster, forms a link with it and this process is repeated until all the nodes belong to one cluster. This algorithm can solve the network connectivity problem for topology reconfiguration purposes. The differences between the clustering algorithm and the bottom-up spanning tree algorithm in this thesis are:

- The clustering algorithm does not consider the degree limits when two clusters are merging into each other. Our bottom-up spanning tree algorithm solves the network connectivity problem with consideration to the degree limitation.
- In the clustering algorithm, each cluster selects a node in another clusters to connect with, based on the shortest distance between them. In our bottom-up spanning tree algorithm, the group merging will be based on their node ID and group ID, which is arbitrarily assigned.
- In the clustering algorithm, the decision for two nodes to connect with each other should be made jointly, i.e., both nodes know that they have chosen each other to connect with. In our bottom-up spanning tree algorithm, this decision will be made independently, i.e., each node decides to connect with another node without knowing if the other node also choose it to connect with.

In [DM-04] and [LDVDM-03], their authors also describe various algorithms that can be used for minimizing congestion in the network. The critical assumption is that a

centralized node keeps the track of state of the network, for which it needs to be a connected graph. Otherwise it may be extremely inefficient to exchange critical information over low data-rate beacon signals. This mandates getting the topology connected first, and our approach is useful in that sense.

In [LDVDM-03], the authors developed a minimum spanning tree algorithm. Their algorithm is based on a cost matrix and always selects the lowest cost edge within 3 degree. Its major difference from our algorithm is that the [LDVDM-03] algorithm must have at least one central node, which has the overview of the whole network and compute the solution for this network. However, in our algorithm, there is no central node, and each node only needs its local information. Its second difference is that the [LDVDM-93] algorithm cannot find the solution, if the minimum-degree spanning tree of the input graph has a degree higher than 3. Our algorithm can always find a solution for any input graph, because we don't set the higher bound of the degree as 3. Instead, we set the higher bound as $\delta+1$, thus this algorithm can be applied to any input graph.

The algorithm that we have designed can cope with the following physical difficulties: First, each node cannot know other nodes' decision about selecting a tree edge, due to the lack of a laser communication channel between them. If one node chooses another node to connect with, but the other node doesn't know that the first node invites it to shake hands and doesn't point its transceiver to the first node, the attempt to build laser connection between them fails. In our algorithm, separated nodes can pair up to form

edges in T in a distributed fashion. The pairing-up of the two nodes u and v is done through two independent choices. (1) of v among all neighbors of u in G, and (2) of u among all neighbors of v in G. Each node runs the algorithm in its local machine. The algorithm will tell each node that which other node it should try to build a laser communication channel. Afterwards, each node will move its transceiver to its target node. Our goal is to let two nodes automatically point to each other without knowing each other's decisions (or invitations here). This algorithm's strategy can guarantee that at least one laser link can be build in such way in each iteration. (We will explain our strategy in Chapter 3 and the existence guarantee in Chapter 4.) Every tree edge is built without any acknowledge of two endpoints' invitations. Iteratively, the entire resultant spanning tree is built in such a way. Thus, although there is no laser communication channel between separated nodes (this is the physical difficulty), they still can set up bi-directional laser links automatically and independently. Therefore, by using a pure calculation method on each node locally, our approach can cope with the physical difficulty, which is the lack of communication channels between separated nodes. Secondly, because the number of transceivers on each node is limited at all times, our algorithm must be a bottom-up algorithm, such that the degree limit will not be increased until it's absolutely necessary. (This is the reason that we call our model "bootstrapping all-FSO networks".) Please keep in mind, if the input feasibility graph is a sparse graph and its minimum-degree spanning tree has a high degree, the resultant spanning tree in our algorithm cannot have a degree lower than the degree of its minimum-degree spanning tree. (A sparse graph is a graph in which the number of edges is much less than the possible number of edges. A dense grapy is a graph in which the number of edges is

close to the possible number of edges. [NIST]) However, our algorithm can guarantee the resultant spanning has the degree at most $\delta+1$. For a given input feasibility graph, the minimum degree spanning tree has degree δ and we cannot do worse than $\delta+1$. In later sections, we have shown through simulations that for reasonable node spacing in a 2D space (and its appropriately derived potential connectivity graph), we can find a spanning tree with degree 3 in most cases, which is the transceiver limit envisioned for FSO systems.

CHAPTER 3. THE BOTTOM-UP APPROACH

3.1 High Level Description of the Bottom-up Approach

Compared with the top-down algorithm, which starts with an arbitrary spanning tree and then iteratively improves upon it until a degree less than $\delta+1$ tree is produced, our bottom-up algorithm minimizes the spanning tree's degree while we are building it.

Our bottom-up approach builds a spanning tree by steps in which either an edge is replaced by another edge or an edge is added as T is being constructed, until T becomes a spanning tree.

Due to the transceiver limitation, our algorithm will aim at using transceivers as few as possible in each step. Since the purpose of this algorithm is to achieve fast connectivity, the bi-connectivity is not initially necessary. Thus, cycles should always be avoided in the network. (A cycle is a circle in a graph and a kind of bi-connectivity in a network). Obviously, a cycle is redundant for the network's connectivity, although it benefits the communication quality in the network. Once a cycle is broken, the connectivity still remains and each endpoint of the broken edge can save a transceiver, which can be used to connect with other nodes in the future. Since our goal is to use transceivers as few as possible and we are not concerned for communication quality, cycles are not allowed in our algorithm. Moreover, saving transceivers creates more opportunities to add as many tree edges as possible at one time.

Again, we would like to emphasize some assumptions which will be shown in our algorithm.

- We assume that all the nodes are isolated in the beginning. (i.e., the initial T consists of all nodes but no edges.)
- Every node has a distinct integer ID.
- Once some nodes are connected by tree edges, those nodes are in the same group. The nodes in the same group can communicate with each other by the high-data-rate communication channel, rather than the low-data-rate beacon. If two nodes are in the same group, they always remain in the same group. The connections within one group can change, but the connectivity of this group always remains.
- If node u chooses node v as a candidate match to build a tree edge, node u will move its transceiver to point to node v.

Note: Only if both the nodes choose each other as candidate matches, the link between them will become a tree edge. If one node's transceiver points to another one, but the other one chooses a third node as candidate match, the attempt to build a tree edge between the first and the second node fails.

Motivated by the possibility that the algorithm will be applied in real FSO networks in the future, we consider optimizing the actual time taken by this algorithm under some realistic assumptions. The transceiver in some FSO networks usually takes several hundred milliseconds for its one movement. The actual time for transceiver movements is represented by the number of rounds taken by the algorithm to execute, where in each round several transceivers can move concurrently. Thus, to optimize the actual time

performance, the algorithm must consider reducing the number of such transceiver movement rounds. This is the main performing objective of our algorithm.

Our algorithm will use two steps to establish the complete tree. Step I is applied to the initial graph G. By adding edges to T such that no node degree can exceed 2, some groups are created. Step II works iteratively. In each iteration, either the groups merge by not exceeding the current maximum degree k or a decision to increase the degree to k+1 is made, until the spanning tree is complete.

3.2 Procedures of Bottom-up Algorithm

Step 0:

Assign node IDs by random permutation. First, we assign a unique ID to each node. It can be a serial number or IP address for FSO network. Secondly, we give each node an arbitrary integer as its appended ID. Therefore, the format of a node ID is (appended ID, original unique ID). To compare the IDs of two nodes (Node i and Node j), the appended IDs will be compared first. If the value of Node i's appended ID is larger than Node j's, we can say that Node i's ID is larger than Node j's ID. If the value of Node i's appended ID is equal to Node j's ID, we will compare their original unique ID (if the value of Node i's original unique ID is larger than Node j's, we can say Node i's ID is larger than Node j's ID). Through this method, we can guarantee that the node IDs are unique, comparable and distributed as a random permutation.

Step I:

Every node seeks two other nodes

1. the largest smaller-ID node that has a potential link with it, and
2. the smallest larger-ID node that has a potential link with it

The two transceivers at the node point to these two candidate matches.

If the resulting edges form a spanning tree, the algorithm terminates.

Otherwise, several groups are formed, and the algorithm advances to Step II.

The outcome of step I on an example network is depicted in FIG 3.2-I.

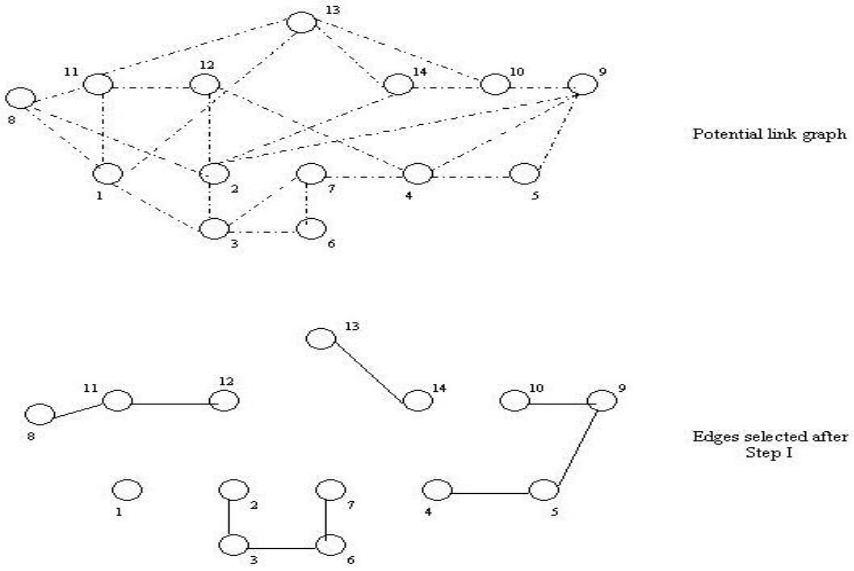


FIG. 3.2-I BOTTOM-UP ALGORITHM, STEP I

In FIG. 3.2-I, in the upper graph (input graph), the dashed lines represent potential links which correspond to the beacons (low data-rate channel). In the upper graph, we can observe that node 3 has potential links with Nodes 1, 2, 6, and 7. Those nodes, which have potential links with node 3, will be split into two sets. One set consists of the nodes,

which have a larger ID than Node 3 (i.e., Node 6 and Node 7); the other one consists of the nodes, which have a smaller ID than Node 3 (i.e., Node 1 and Node 2). In the larger ID set, the one with the smallest ID is Node 6, hence, Node 3 moves one of its transceivers towards Node 6. In the smaller ID set, the one with the largest ID is Node 2, hence Node 3 moves another of its transceiver towards Node 2.

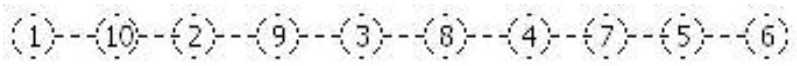
At the same time, all the other nodes also do the same calculation. Nodes 2 and 6 also separately find that Node 3 is the one towards which they must point their transceivers. Consequently, it means they make commitments in a distributed way, and both edges (2, 3) and (3, 6) are added to the tree.

Node 1 cannot build any tree edges with any other nodes in Step I. After Node 1 discovers that Node 3 has the smallest larger-ID (with a potential edge with Node 1), it turns one of its transceivers towards Node 3. However it does not get the same corresponding response from Node 3. Therefore, the attempt to build tree edge (1,3) has failed.

For any input graph, no matter how we assign the node ID to each node, the minimum-degree spanning tree for this graph should be same. Using different methods to assign node ID will get the essentially same minimum-degree spanning tree. Thus, the randomly assigned node ID will not affect the resultant spanning tree in our algorithm, as long as the node ID is consistently.

In Step I, we are trying to build a laser-link path in the network. This is because, for all kinds of network topology, a path is with the smallest degree, which is equal to 2. Let G(V,E) be the graph of potential links. Let E' be the largest possible subset of E where the degree of each node in G'(V,E') is at most 2 and G' is acyclic. Our strategy in Step I is to attempt to cover the E' (or some fractions of E') by laser links. The most ideal situation is that after Step I, we can find the entire E'.

The strategy that we are using in Step I, allows each node can make a decision to build a tree edge independently. As we have mentioned in Step 0, the node IDs are distinct and comparable. There always exist Node i and Node j, such that their IDs are close to each other and they will select each other to connect with. Thus, it also guarantees that at least one tree edge will be added in the tree in Step I. For most input graphs, i.e. the ID are distributed as a random permutation, the likelihood to build bi-direction tree edges concurrently should be very high. Even for an input graph with a pathological permutation of the ID's, at least one tree edge will be added in the tree in Step I. This has been proved in Chapter 4. A pathological permutation is a permutation that is the least suitable for our algorithm. For a 10-node network with a single long path of potential links, the pathological permutation is:



The possibility of all the random permutation is N!, (10*9*8*7…..*1 =) 3628800 in this case. But only one permutation is pathological permutation as it is shown above. This is because any changes in pathological permutation will make it non- pathological, which

means that more than one tree edges can be added in Step I. Therefore, the possibility of the happening of pathological permutation is only 1/(N!). It means, for 3,628,800 10-node input graphs, only 1 case can add only 1 tree edge in Step 1. All other cases have high likelihood to build tree edges. As we have run simulations on a random permutation of IDs in graphs which is a single long path (100 simulations for each size of networks), Step 1 can add 10.87 tree edges in 20-node graphs, 28.02 tree edges in 50-node graphs, and 112.37 edges in 200-node graphs. It shows the likelihood to build tree edges in Step I is higher than 50%. Chapters 4 and 6 will also prove this hypothesis and show the supporting simulation results.

What information exactly is being exchanged in Step I?

In Step I, each node can record the ID and location of other nodes, which have potential links with it, by the beacon signals. The beacon information includes node ID and location. From now on, it's not necessary to send out the ID and location information by beacons any more, since each node can identify other nodes by the direction of the beacons between them and this node.

How does it meet the computation model?

Based on the beacon information exchanged in Step I, each node can simply use the maximum or minimum selection algorithm to get its smallest larger ID node and largest smaller ID node which has potential links with this node.

Step II:

After Step I, several connected components are created, which are called groups. We assign to every group a distinct group ID. Specifically, we make the smallest node ID in the group the group ID.

As we have mentioned previously, the nodes in the same group can exchange information to each other through a high-data rate laser channel. This thesis does not address how each node sends information to other nodes through the high data rate channel. However, it assumes the information, such as each node's local information, can be exchanged, once the laser links are built. For example, we can let each node have an individual connectivity table, which record its neighbors in the spanning tree, and a routing table, which record the routes that this node will take to send message to the other nodes in the same group. Since our thesis does not focus on this issue, we will not address more details about it. However, we assume that the nodes in same group have known each other after Step I.

We allow the degree limit to be *k* in Step II. *k* is an integer. Let *k* be equal to 2 in the beginning.

- Iteratively perform sub-steps A to E. Every group seeks other groups so that they can merge by forming a larger group. In an iteration, each group Z selects two other groups (S and L) for potentially merging with them. The selection of S also comprises the selection of nodes in groups S and Z, which will provide the required transceiver-to-transceiver connection for merging. Similarly, the selection of L also includes the selection of nodes in L and Z, which will provide

the transceiver-to-transceiver connection for merging. Sub-step E determines whether to increase the degree k. Sub-step E can also decide if the algorithm is to be terminated.

A. Improvement within each group: (When the degree limit is 2, this sub-step can be skipped, because it's not meaningful to reduce a node's degree from 2 to 1).

In this sub-step, we want to reduce the number of degree k nodes, so that we have more opportunities to merge groups in later stages without exceeding the current maximum degree k.

Each group performs the "improvement" for degree-k nodes in a way similar to [FR-92]. The details are:

Consider the sub-graph of G, which consists of nodes in the same group and the potential links between them, and the local spanning tree of the group T_{Local}. First, we remove all nodes with degree k and k-1 from the local tree T_{Local}. Mark the remaining connected components as "good". All components consisting of singleton nodes with degree k and k-1 are marked "bad". If there are non-tree edges between "good" components, do the following iteratively: Let (u, v) be a non-tree edge between two "good" components of the same group. We add (u, v) to T_{Local} and create a cycle. If there is a node w of degree k on this cycle, we reduce the number of k degree nodes by one, by adding (u,v) in the tree and deleting one tree edge incident on node w. Otherwise, there must be at least one "bad" node

of degree k-1 on the cycle. At this stage, it will be advantageous not to commit on the node (whose degree can be reduced) of degree k-1. Thus on the current cycle, we mark all “bad” nodes as “good” and join all components along with all degree k-1 nodes into a larger “good” component. In other words, these degree k-1 nodes can help to reduce the degree of other degree k nodes later if it is needed. (There will be two actions for this case: 1. A degree k-1 node reduces its own degree in the cycle first. 2. The degree k-1 node becomes degree k-2 node and it can help a degree k node reduce its degree.) But, before we know whether this degree k-1 node is needed to help other degree k nodes, we will not alter that connection. If there are more non-tree edges between good components, repeat this iteration.

Example:

The following graph , FIG. 3.2-II-1, is a sub-graph of G. All the nodes in this sub-graph are in the same group. Dashed lines are potential links and solid lines are tree edges. Current degree limit is 4.

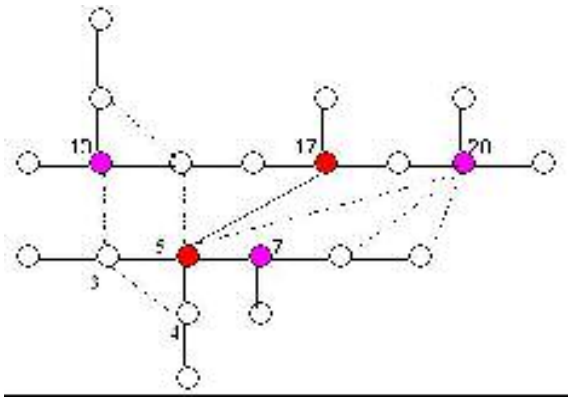


FIG. 3.2-II-1 BEFORE “IMPROVEMENT”

We observe that Nodes 5 and 17 are degree-4 nodes and Nodes 7, 13 and 20 are degree-3 nodes. They will be marked as “bad” and removed from G (along with the incident edges – both solid and dashed) in FIG. 3.2-II-2.

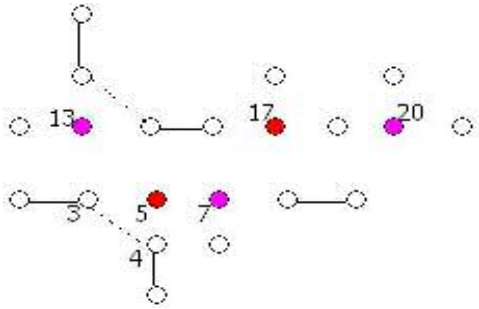


FIG. 3.2-II-2 REMOVE “BAD” NODES

Add the non-tree edges between “good” components to check if there is a cycle. After adding edges between Nodes 3 and 4, a cycle including a degree-4 node, which is Node 5, is generated (FIG.3.2-II-3).

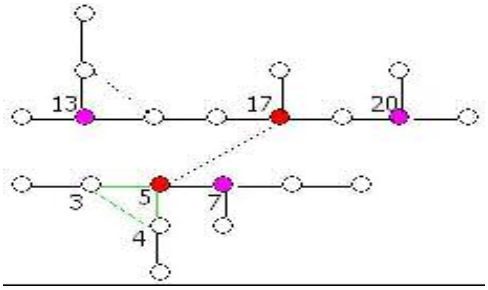


FIG. 3.2-II-3 FIND CYCLE

The previous tree edge between Nodes 4 and 5 can be removed from the tree and replaced by an edge between Nodes 3 and 4.

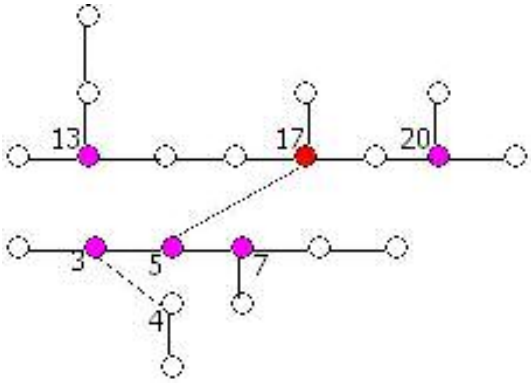


FIG. 3.2-II-4 AFTER "IMPROVEMENT"

After sub-step A, the number of degree 4 nodes drops from 2 to 1.

What information exactly is being exchanged in sub-step A?

The information exchanged among the nodes in the same group in sub-step A includes their IDs, other group members' ID which have the laser links with them, other group members' ID which have the potential links with them.

How does it meet the computation model?

Based on the exchanged information, each node can have an overview of the whole group. The overview is a sub-graph of G. Each node will search the sub-graph and apply sub-step A's algorithm. Once a cycle, which includes a degree-k node, is found by adding a non-tree edge in sub-step A, the degree-k node will decide which laser link incident to it will be replaced by that non-tree edge. It will inform the other endpoint that the laser link between them should be disconnected.

Remember that the sub-step A starts with the new-formed groups from the last step. Because every endpoint of the new tree edges added in the last step before sub-step A can have an overview of its whole group, the sub-step A's algorithm can be applied to every endpoint of the new added tree edges concurrently.

B. Mark "candidates"

In this sub-step, every node, whose degree is less than k or can be reduced to a degree which is less than k (in its local spanning tree), will be marked as a "candidate". A candidate uses a special beacon to inform all the other nodes who can "see" it. If a candidate can see one or more candidates from other groups, we call it as a "real candidate".

In sub-step A, we have reduced the number of degree k nodes by adding edges between "good" components. A "good" component consists of nodes with degree at most k-2. In sub-step A, the degree of some degree-k nodes was not reduced. In this step, an attempt is made to reduce the degree of those nodes by adding non-tree edges incident to the degree k-1 nodes. If one degree k node can reduce its degree by increasing one or two degree (k-1) nodes' degree, this degree k node also can be a candidate to merge with other groups. We don't make any improvement for these degree k node candidates in this sub-step. This will be done in later sub-steps if necessary.

Algorithm to mark candidate:

If we remove all nodes with degree k from the local tree T, some connected components are left. If there are non-tree edges incident to degree k-1 nodes from different components in the same group, do the following iteratively: Let (u, v) be a non-tree edge incident to a degree k-1 node from a different component in the same group. Add (u, v) to T and generate a cycle. If there are some nodes with degree k in this cycle, it means that it is possible for these k degree nodes to reduce their degree by one. Thus we mark these degree k nodes as “reducible”. Let “reducible” nodes on this cycle record (u,v) as “reducing edge” for later use.

If there are more non-tree edges incident to degree k-1 nodes from different components, repeat the iteration. When there are no non-tree edges incident to degree k-1 nodes from different components, the following nodes are marked as “candidates”:

- nodes with a degree less than k which have potential links with other groups
- “reducible” degree k nodes, which have potential links with other groups

All candidates send a beacon by using a low-data-rate channel. If a candidate can see one or more candidates from other groups, it is marked as a “real candidate”.

Following is an example that illustrates the marking of “candidates” and “real candidates”:

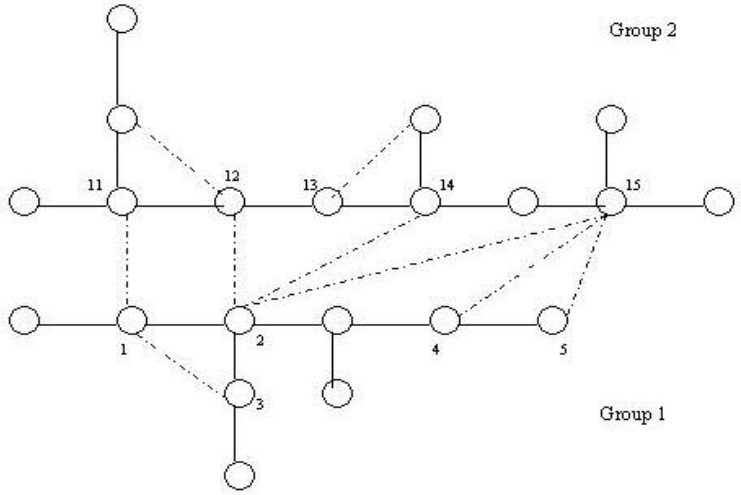


FIG 3.2-III. EXAMPLE: MARKING CANDIDATES AND REAL CANDIDATES

In the above graph, when *k* is 3, Nodes 1, 2, 4 and 5 in Group I and Nodes 11, 12 and 14 in Group II are marked as “candidates”. After all the candidates are marked, Nodes 1 and 2 in Group I and Nodes 11, 12 and 14 in Group II are marked as “real candidates” because they have potential links with other “candidates”.

What information exactly is being exchanged in sub-step B?

In sub-step B, a node will send out a simply beacon signal, if it’s a “candidate”. The nodes between different groups only exchange the “candidate” information by beacon signals. Once they realize that themselves are “real candidates”, these “real candidates” will send out

send its ID with a “real candidate” tag to the other nodes in the same group by high data-rate laser links.

How does it meet the computation model?

Based on the “candidate” beacon, each “candidate” can realize whether it’s a “real candidate”. “Real candidate” will send its ID with a “real candidate” tag to other nodes in the same group by laser links. The “real candidate” information will be used in the next sub-step. Because each degree-k node in sub-step B has already has an overview of its whole group (based on the exchanged information in sub-step A), they can concurrently and locally search their sub-graph to check whether they are “reducible”. Afterwards, all the nodes also can concurrently check whether themselves are “candidates” or “real candidates” by the beacon information.

For simplification, from now on, only a “real candidate” will be referred to as a “candidate”.

C. Group selection:

Every group seeks two other groups with

1. the smallest larger group ID which has a candidate-to-candidate link:

   Let us denote this group as Group L.

2. the largest-smaller group ID which has a candidate-to-candidate link:
   Let us denote this group as Group S

Each group can be abstracted by a node, and Sub-step C uses the same algorithm as sub-step A.

What information exactly is being exchanged in sub-step C?

The beacon sent out by a “candidate” includes its group ID and “I’m candidate” signal. In sub-step C, each “candidate” should inform other “candidates” belonging to the same group that which other groups have candidate-to-candidate links with it. Afterwards, the “candidates” will compare those group IDs and get the smallest larger ID group and the largest smaller ID group. The selected groups’ ID will be informed to each “candidate”. All these information’s should be transmitted by high data-rate laser link.

How does it meet the computation model?

The sub-step C’s algorithm can be applied to each group concurrently.

D. Node selection:

After group selection, each group assigns two nodes to take the task of merging with two other selected groups.

Each group (call them Group Z collectively) does the following:

1. Assume the largest node ID candidate in Group Z, which has a candidate-to-candidate link with Group L, is Node A. Assume Node C is the smallest node in Group L, which has a candidate-to-candidate link with Node A. Then, Node A points to Node C. This step is composed of two sub-steps: (i) if Node A's degree is k, its degree is reduced to k-1. Node A can check its record of "reducing edges". Add the recorded edge in local tree and delete one edge incident to A; (ii) Node A points to Node C.
2. Assume the largest node ID candidate in Group S, which has a candidate link with Group Z, is Node R. Assume Node B is the smallest candidate in Group Z, which has a candidate link with Node R. Node B points to Node R. (If Node B's degree is k, it reduces its degree to k-1 and points to Node R.)
3. In the same group, if Nodes A and B are the same node and its degree is greater than k-2, Node B yields to Node A. If both Nodes A and B have degree k, both of which cannot be reduced within Group Z, Node B yields to Node A.

Example:

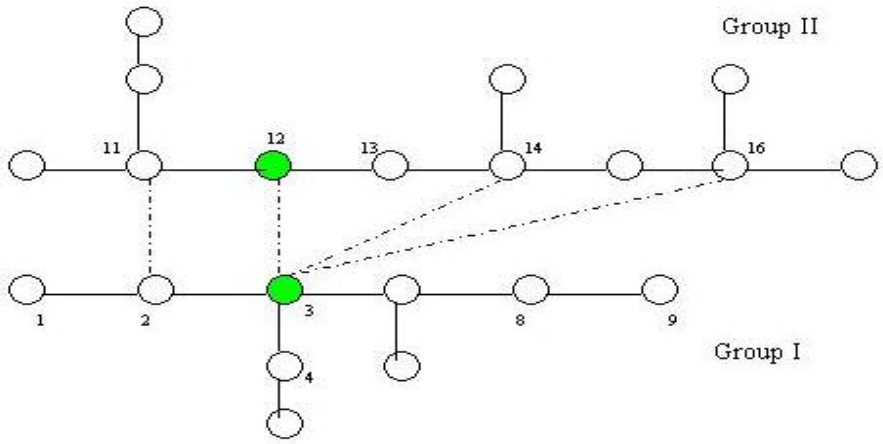


FIG 3.2-IV. EXAMPLE: NODE SELECTION

In the above graph, there are two separate groups which have committed to merge within each other. Assume current degree limit (k) is 4. In Group I, Node 3 has the largest node ID among all candidates. Hence, Node 3 has the task of building a tree edge with Group II. Node 3 finds that Node 12 is the smallest ID node that has a candidate-to-candidate link with it. Node 3 moves its transceiver to Node 12. Meanwhile, in Group II, all the candidates (Nodes 11, 12, 14 and 16) exchange information and Nodes 12, 14 and 16 are detected to have links with Node 3, the largest ID node, which has links with Group II. Since Node 12 has the smallest ID among Nodes 12, 14 and 16, it takes over the job of connecting with Group I. Consequentially, it moves one of its transceivers to Node 3. The commitment is made and a new tree edge (3, 12) is added to T.

What information exactly is being exchanged in sub-step D?

In the sub-step D, the “candidates” in Group Z will exchange the information, including their own ID, the largest ID in Group S that it has candidate-to-candidate link with, and the smallest ID in Group L that it has candidate-to-candidate link with. They will send out this information to other “candidates” in the same group.

How does it meet the computation model?

Before the “candidates” exchange information in sub-step D, they can use the information that it got from sub-step B and C to calculate which node has the largest ID in Group S that this node has candidate-to-candidate link with, and which node has the smallest ID in Group L that this has candidate-to-candidate link with. After they exchange information in sub-step D, according to this information, each “candidate” can follow the Sub-step D’s algorithm to calculate locally and check whether itself is the Node A or Node B in its group. Once it realizes that itself is Node A or Node B, consequently it can calculate locally to find its Node C in Group L or Node R in Group S.

The Sub-step D’s algorithm can be concurrently applied to each group, as well as other sub-steps.

E. When groups are merged, their group ID needs to be updated. If it is feasible to merge more groups within degree limit k, go back to sub-step A and run the next iteration to seek other groups to merge with. Otherwise,

increase k by one. Then, repeat all the procedures in Step II until all the nodes are in the same group.

What information exactly is being exchanged in sub-step E?

In the sub-step E, the information exchanged among the new-merged larger groups includes the previous group IDs, which were used before the groups merged into each other.

How does it meet the computation model?

Based on their previous group ID information, within each new-merged group, the sub-step E's algorithm can be applied concurrently and locally to compare their previous group IDs and only pick one previous group ID as the new-merged group's group ID. It can be the largest or smallest previous group ID within this new-merged group.

In one iteration from the sub-step A to E, each group (Group Z) can determine its unique Node A and Node B. Node A will determine its unique Node C in Group L to connect with, and Node C will determine its unique Node R in Group S to connect with. Concurrently, Group L (selected by Group Z) also treats itself as "Group Z" and chooses Group Z as its "Group S". The node treated as Node C by Group Z is determined as "Node B" by Group L. According to our algorithm, this "Node B" will also determine Node A in Group Z as its "Node R". Thus, the tree edge between Node A and Node C

can be formed with the distributed control. This outcome by the distributed control is exactly what we desired.

If the high data-rate laser link is formed successfully, the two endpoints expect to receive a message from each other which includes their group ID, before the algorithm enters the next iteration. If the message reaches each endpoint through the laser link, the endpoints realize that the laser link between them is connected. If a node does not get the message from the other node that this node wants to connect with, this node will know that the other node's group didn't select this node's group to connect with and the laser link between them is not formed. Thus, this node is able to use its transceiver to connect with other groups later.

The bottom-up minimum-degree spanning algorithm assigns a unique node ID to each node in a random permutation. It begins to look for a degree-2 connectivity solution for the network. If the first try fails, it performs five sub-steps iteratively until the spanning tree is built. First, each group tries to reduce the number of its highest degree nodes. Secondly, each group finds all its nodes which are available to connect with other groups. Thirdly, each group chooses other two groups to merge into. Fourthly, each group determines its two nodes to connect with the two groups that it selected. Fifthly, after group merging, each new merged group needs to update its group ID and the unconnected network needs to decide whether to increase the degree limit. Once the spanning tree is formed, this algorithm terminates.

CHAPTER 4. ALGORITHM ANALYSIS

4.1 Approximation Guarantee

In this section, first we will prove the “existence guarantee”. By existence guarantee, we mean that a solution (spanning tree T) exists if the input connectivity graph G is connected. Then we prove the “approximation guarantee”. By this, we mean that we can derive an exact expression to indicate the worst case deviation from the optimal solution of our approximation algorithm.

The bottom-up algorithm guarantees to form a spanning tree as long as the input graph G is a connected graph. The resultant spanning tree has a degree within one-from-the-optimal degree.

Theorem 1:

Given a connected graph G, the proposed bottom-up algorithm guarantees a spanning tree solution.

Proof for Theorem 1:

Each group only adds tree edges which connect it to other groups, and only one potential edge between two groups will be added to the tree. This guarantees that no redundant edge exists in the tree.

The algorithm terminates when all nodes are added to the tree. This guarantees that the tree is a “spanning tree”.

We can also prove that the degree limit will not be increased incorrectly in our algorithm. This is because, at each Step as described by iterations A-to-E, at least one new group will be merged into an existing group, if the degree limit (k) does not increase. Thus the degree limit is always increased efficiently.

Take the following case as an example:

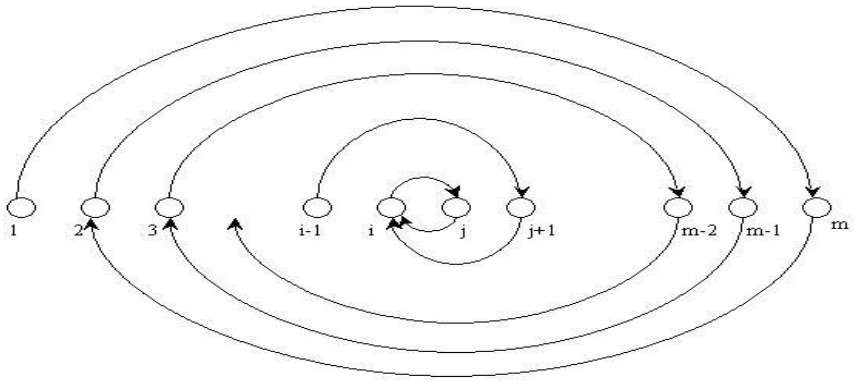


FIG. 4.1-I NON-DEADLOCK BOTTOM-UP ALGRITHM

In the above graph, it can be seen that m separate groups exist at the current iteration. The worst case is:

After correct computation, the group with group ID 1 chooses the group with group ID m (where $m > j > i > 1$) as its larger candidate match group. If group m doesn’t choose group 1 as its smaller candidate match, it must have chosen another group which has a

group ID greater than 1 (since that group will be “closer” to group m than group 1). Assume group m chooses group 2, while, group 2 chooses group m-1 as its larger candidate match group. The worst case is that all the attempts of group merging fail recursively, as we show in the figure.

However, if G is connected, two groups i and j exist such that i chooses j and j chooses i. We can observe that FIG. 4.1-I is a contractive spiral. At the end of the spiral, two groups (group i and group j) will choose each other as candidate match. This is because these two groups have a potential link between them and they are “closest” to each other respectively. It makes a new group merging. In the next iteration, this new combined group will connect with another group along the reverse direction of the contractive spiral. After several iterations, all the groups in the spiral will be connected. No deadlock will happen.

For the whole network, the worst case is that it can consist of several sub-networks, as in the above case. However, it will not stall or incorrectly increase the degree limit.

Theorem 2:

Let the degree of the final tree generated by the bottom-up approach be D. Then D is at most (δ+1), where δ is the degree of an optimal minimum-degree spanning tree.

Proof for Theorem 2:

In the last sub-step E of the algorithm, we increase the degree limit from D-1 to D.

Recall that, each potential link between the two D-1 degree groups has at least one degree D-1 node. Suppose we combine two groups by one extra edge of G, one of the following cases would apply:

1. One node whose degree is (D-1) connects with another node whose degree is (D-1).
2. One node whose degree is (D-1) connects with another node whose degree is less than (D-1)

Case 1:

The degree of at least one of the endpoints (call it Node X) cannot be reduced by our algorithm within its D-1 degree group. If the degree of another endpoint (call it Node Y) can be reduced, it is impossible to reduce the degree of both the endpoints. If both degrees can be reduced, both of them should be “real candidates” and they can connect without increasing the degree limit of D-1.

Case 2:

Denote the node of degree (D-1) Node X. The degree of Node X cannot be reduced within its D-1 degree group.

For case 1, after group merging, if there exists a Node Y, we can perform sub-step A of Step II to make a degree improvement for Node Y.

Now, for both cases 1 and 2, we get a tree with one or more degree D nodes (Node X) whose degree cannot be reduced. After sub-step A of the last iteration in Step II, we put all the degree D nodes in set S and put all the degree D-1 nodes, which are marked as bad, in set B. It can be seen that after removing nodes in S and B, all the remaining connected components are marked as good. We know that when sub step A ends, there are no non-tree edges in G between good components.

For the final tree with degree D, S consists of all the degree D nodes while B is the subset of degree D-1 nodes. We observe that after removing sets S and B, there are some separated sub-trees among which no potential links exist. This condition satisfies Theorem 5.1 of [FR-92]. According to that theorem, the degree of this spanning tree <= $\delta+1$.

4.2 Traditional computation time complexity

Theorem 3:

There is a O(N*E) bottom-up algorithm for the minimum degree spanning tree problem, which produces a spanning tree whose degree is at most (δ + 1) degree.

Proof for Theorem 3:

The time complexity of the approximate bottom-up algorithm can be derived as follows:

Step I:

Each node selects its maximum or minimum ID candidate match independently of others (i.e., in parallel). The maximum or minimum selection algorithm is of O(N), where N is the number of nodes in the network. Therefore, the time complexity of Step I is O(N).

Step II:

The number of sub-steps A, B, C and D round is bounded by O(N).

Sub-step A, Improvement:

To reduce the degree of one endpoint of the newly added tree edge, we check the non-tree edges. It can be done by a standard graph-searching algorithm (i.e., DFS), which takes O(N+E), where E is the number of edges.

Sub-step B, Mark Candidates:

To check if each node is “candidate”, it can be done in parallel by a standard graph-searching algorithm which takes O(N+E).

Sub-step C, Group Selection:

Each group selects its candidate match group in parallel. This sub-step is similar to step I. It takes O(N) to run a maximum or minimum selection algorithm.

Sub-step D, Node Selection:

Node selection only needs to run a maximum or minimum selection algorithm in parallel which takes O(N).

Sub-step E, Make Decision:

A decision whether or not to increase the degree limit takes O(1).

Therefore, time complexity of Step II is bounded by (O(N)*(2*O(N+E) +O(N)+O(1)) = O(N*E).

$$\text{Step I: } O(N) + \text{Step II: } O(N*E) = O(N*E)$$

Therefore, the overall time complexity is bounded by O(N*E). This is polynomial time complexity. Compared with the top-down algorithm, we have an explicit time complexity, while the top-down algorithm only has a simulated non-explicit time complexity which the authors of [FR-92] believe should be O(N*E) [FR-94]. We thus prove that our algorithm's computation time complexity is no worse than this authors' believed bound.

4.3 Round Complexity

4.3.1 Estimates of the actual time taken by the algorithm under some realistic assumptions

At the beginning of this thesis, we have described that this algorithm pertains to the FSO network. In the real world, the actual time taken by the algorithm to execute (as a function of nodes) is more germane (and more interesting) than the formula of the computational time complexity. The CPU speed of a PC can be several GHz, while the movement of a transceiver (for aligning purposes) can take several hundred milliseconds.

Thus it is easy to estimate that the time taken by the transceivers' movement is much more dominant than the computation time.

Hence we can assume that the computation time is negligible. The time taken by the whole process is dominated by the time of transceivers' movements, which can be represented by the number of rounds taken by the algorithm to execute (each round represents movement of several transceivers in conjunction).

Estimated Actual Time = Rounds * One Transceiver's Movement Time

Since the movement time of one transceiver is constant, the actual time will be dominated by the number of rounds taken by the algorithm to execute. In each round, a number of transceivers move independently of each other to establish tree edges.

4.3.2 Round Complexity of the Bottom-up Approach

In every iteration, the bottom-up algorithm adds at least one edge to the tree. Hence, in the worst case, it takes N-1 iterations to build the tree, where N is the number of nodes. Assume the current degree limit is k. Once an edge is added to the tree, one or both the endpoints of this tree edge become degree k nodes.

Let us begin from the next iteration, after this edge is added to the tree.

Sub-step A:

According to the prior analysis, at least one of the endpoints (Node X and Node Y) cannot reduce its degree by "improvement".

This is because the previous two groups, to which Nodes X and Y belong, only have potential links with degree k-1 or k endpoints. Thus we know that it is impossible to make improvement by adding the above potential links as tree edges. On the other hand, all the potential links within the local group cannot reduce the degree of both the Nodes X and Y. It follows since if the improvement for both of them is feasible within these two groups separately, they should have been connected in the previous steps when the degree limit was k-1. Therefore, at least one of the Nodes X and Y's degrees cannot be reduced. Now, we consider the reducible node in the pair of Node X and Node Y. To reduce its degree to k-1, another non-tree edge in its previous group is added, and one edge incident to the node of which we want to reduce the degree (one of the Nodes X or Y) is deleted. Afterward, the new merged group is an improved group *(**Lemma** 1: In sub-step A, after the endpoints of the new edges are improved, the new connected groups are improved as well. To preserve the continuity of the chapter, the proof for this lemma will be given at the end of this chapter).* If more than one edge is added in one iteration, the improvement can be done in parallel. Thus, in sub-step A, we need only one round of transceiver movement.

Sub-step B, C:

None of the sub steps B (which is called "marking candidate") and C (which is called "group selection") involve transceiver movement. Thus they do not involve "actual time" and can be omitted from the analysis.

Sub-step D:

In this sub-step (which is called “Node selection”) one or more edges are added to the tree. Since the transceivers can be moved in parallel, the number of transceiver movement rounds is still equal to one.

Sub-step E:

This sub step (which is called “ the decision-making” sub-step) is a pure computation sub step and does not involve any transceiver movement.

Finally, the whole process needs 2(N-1) rounds of transceiver movement, because there are (N-1) sub-step A-to-E iterations and at most 2 transceiver movement rounds in each iteration. The total number of transceiver movement rounds is 2(N-1).

Therefore, the round complexity is O(N) and the estimated actual time is O(N)*Constant, where the Constant is the actual time of one transceiver’s moving.

4.3.3 Comparison

From above analysis, we can conclude that our algorithm is a serial algorithm, because in the worst case, only one tree edge could be added in one iteration consisting of sub-steps A-to-E. The serial algorithm is designed for a computation environment where computers execute one instruction of an algorithm at a time. However, our algorithm is not a pure serial algorithm in FSO networks, because each node in the network can execute the algorithm concurrently and independently. In general, it can add more than one tree edge at a time. Therefore, we classify our algorithm as distributed algorithm.

For all the other pure serial algorithms, it takes at least N-1 steps to build a spanning tree, because that is the nature of the serial algorithm. As our algorithm uses several concepts from the top-down algorithm, we'd like to make a comparison with it.

The top-down algorithm in [FR-92] generates a random spanning tree first. The worst case is that the spanning tree is a star tree, i.e., one node has degree N-1 and all the others have degree 1. It takes N-1 steps to build a spanning tree, so it has to take N-1 rounds of transceiver movement.

The top-down algorithm will perform the "improvement" afterwards. The worst case is that the node with degree N-1 transfers half of its degrees to some other node. This takes (N-1)/2 rounds of transceiver movements. Now the tree has two nodes with degree (N-1)/2. Again, these two nodes separately transfer their degree to some other two nodes by making an "improvement". This also takes (N-1)/4*2 = (N 1)/2 rounds of transceiver movements. Assume the resultant tree has degree k generated by the top-down algorithm for this input graph. If the current highest degree nodes always transfer their degree to some other nodes separately, it will take log(N-1)/k iterations for such a degree transfer until the resultant tree has (N-1)/k nodes whose degree is k. In each iteration of degree transfer, the current "highest-degree" nodes will transfer half of their existing degree to other nodes, so there are (N-1)/2 rounds of transceiver movements.

Finally, in the worst case, the whole procedure takes [log[(N-1)/k]]* (N-1)/2 + (N-1) rounds of transceiver movement. The round complexity of the top-down algorithm is O(N*log(N-1)/k)

If [log[(N-1)/k]]* (N-1)/2 + (N-1) > 2(N-1), then (N-1) > 4k. Hence, when (N-1) > 4k, the bottom-up algorithm takes less rounds of transceiver movement than the top-down algorithm, where N is the number of nodes and k is the degree of spanning tree which can be achieved by the top-down or the bottom-up algorithm.

Proof of Lemma 1:

We will use the induction method to prove Lemma 1:

1st:

Degree limit = 2. In this case, several “improved” groups exist each with a degree 2 and no new group merging is possible without increasing the degree limit. Each group is “improved” because its degree cannot be further reduced from the existing degree 2.

2nd:

Degree limit = k.

Assume: when degree limit = k and no new merging happens, all the groups are “improved”.

This means all the potential links between different groups are incident on nodes with degree k.

Degree limit = k+1.

Now the degree limit is increased from k to k+1. After two groups are merged into each other by adding one new tree edge, at least one endpoint of this new tree edge becomes a degree k+1 node and the degree of this endpoint cannot be reduced further by " improvement". However it can be possible to reduce another endpoint's degree if its degree is k+1 and if it is reducible. After this endpoint's degree is reduced to k, the whole new group is an "improved" group, i.e. we cannot reduce this local spanning tree's degree any more.

Therefore, beginning from the degree limit 2, the bottom-up algorithm can generate an improved tree by reducing the degree of only one endpoint of the newly added tree edge.

CHAPTER 5. PARALLEL BOTTOM-UP ALGORITHM

5.1 Parallel model

The bottom-up algorithm described in Chapter 4 is reminiscent of a serial algorithm because in the worst case, only one tree edge could be added in one iteration consisting of sub-steps A-to-E. However in concept, a more powerful implementation should be able to add as many tree edges as possible in parallel within the degree limit in a given iteration. Hence, we call our serial algorithm, which adds as many tree edges as possible at a time, a distributed algorithm. On the other hand, a parallel algorithm should be able to add more than one edge in one iteration.

To compare our distributed algorithm with the parallel algorithm, we have designed a parallel algorithm, which is called the clustered algorithm, as described in the following steps:

Assume the total number of nodes to be N.

1. Set degree limit as 2. Regard the connected nodes or singleton nodes as one component.

   Let X be the number of components. At the beginning, X=N.
2. Split the X components into X/Y blocks. Each block has Y components
3. Every node has Block ID, Group ID, and Node ID.

   Inside each block, execute the bottom-up algorithm that we have described before.

4. If no new group merging happens within a block, this block sends out a “stall” beacon. If two or more blocks are stalled, every two nearby blocks will be combined into one block. Continue to seek and perform new group merging and combinations.
5. Repeat step 3 and 4 until all the nodes are within one block and the block is “stalled”.
6. Increase degree limit by one
7. Update the value of X and go back to step 2, until the spanning tree is complete.

5.2 Comparison

Due to the varying nature of the real input graphs, we cannot tell (using worst case analysis) if the clustered algorithm (that we have described above) has a better actual-time performance than the distributed algorithm. On the one hand, the clustered algorithm can add at least one tree edge in each block separately, but it has fewer chances to make connection between nodes within same block. On the other hand, the distributed algorithm has more chances to make a connection between the nodes because the whole graph is one big block. However the worst case for it is that it might add only one tree edge in one iteration for the whole graph. Since the worst case seldom occurs for most input graphs, we propose using simulations for determining the advantage of one algorithm over the other. We must also bear in mind that our clustered model may not be the best parallel model in terms of performance.

CHAPTER 6. SIMULATION AND RESULTS

6.1 Simulation model

We have designed our input graph as close to the real optical wireless network as possible. The main concerns for designing a sample of an input graph are the distribution of the nodes in 2-D space, and the potential links that exist between them (as an example, the nodes that lie far from each other typically won't have a potential edge).

First, we set a grid with 100*100 pixels. Afterwards, we place a certain number of nodes in this grid according to some random distribution. Thus each node has a unique coordinate in 2-D space. The strength of the potential link between two nodes depends upon the power received by each node. The formula for the received power at a node is:

$$P_R = P_T e^{-\alpha L} 2A/(\pi\theta^2 L^2)$$

where A is the area of the receiver aperture, P_T is the transmitter power, θ is the beam divergence half angle, L is the distance between the transmitter and the receiver, and α is an attenuation constant, which factors in atmospheric obscuration [LDVDM-03]. As a reasonable assumption, we set the probability of existence of the link between two nodes as proportional to (1/L), where L is the distance between the two nodes in the grid.

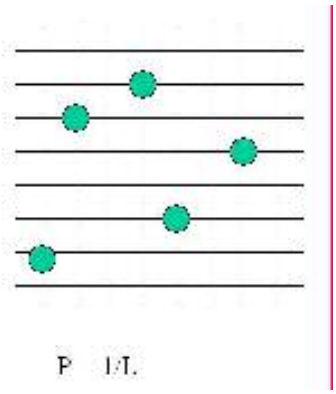


FIG 6.1.1 Input Graph in grid

The simulation environment is implemented in C++ on UNIX platform.

6.2 Simulation Results

We generate 100 connected sample-graphs for each nodal case (varying from 10 to 200) of input connectivity graph G for each algorithm (distributed algorithm, best pure serial algorithm, clustered algorithm). As we have mentioned previously, an important measurement of the actual time to build the connected network is the actual time of transceivers' movements, which can be represented by the number of rounds taken by the algorithm to execute (each round represents movement of several transceivers in conjunction). The estimated actual-time performance can be presented by the number of transceiver movement (also called change) rounds. For any other pure serial algorithm, the best results are (N-1) transceiver change rounds, because they add only one tree edge at a time. This is due to the nature of serial algorithms. Given this fact, the simulation results for our distributed bottom-up algorithm are seen to be very positive (TABLE 6.2-I).

Number of Nodes	Distributed Algorithm Avg. transceiver change rounds	Best Pure serial Algorithm Avg. transceiver change rounds
10	4.28	9
20	6.43	19
50	11.15	49
100	18.53	99
200	31.70	199

TABLE 6.2-I DISTRIBUTED BOTTOM-UP ALGORITHM RESULTS

From the above results, we make the plots by EXCEL to illuminate the advantages of our distributed algorithm (FIG. 6.2-I). The speed-up formula is given by:

$$\text{Speed-up} = \frac{\text{(Transceiver's movement rounds in the pure serial algorithm)}}{\text{(Transceiver's movement rounds in the distributed algorithm)}}$$

The speed-up is abstracted as TABLE 6.2-II and FIG. 6.2-II.

Number of Nodes	Speed-up
10	2. 10
20	2. 95
50	4. 39
100	5. 34
200	6. 28

TABLE 6.2-II DISTRIBUTED BOTTOM-UP ALGORITHM SPEED-UP

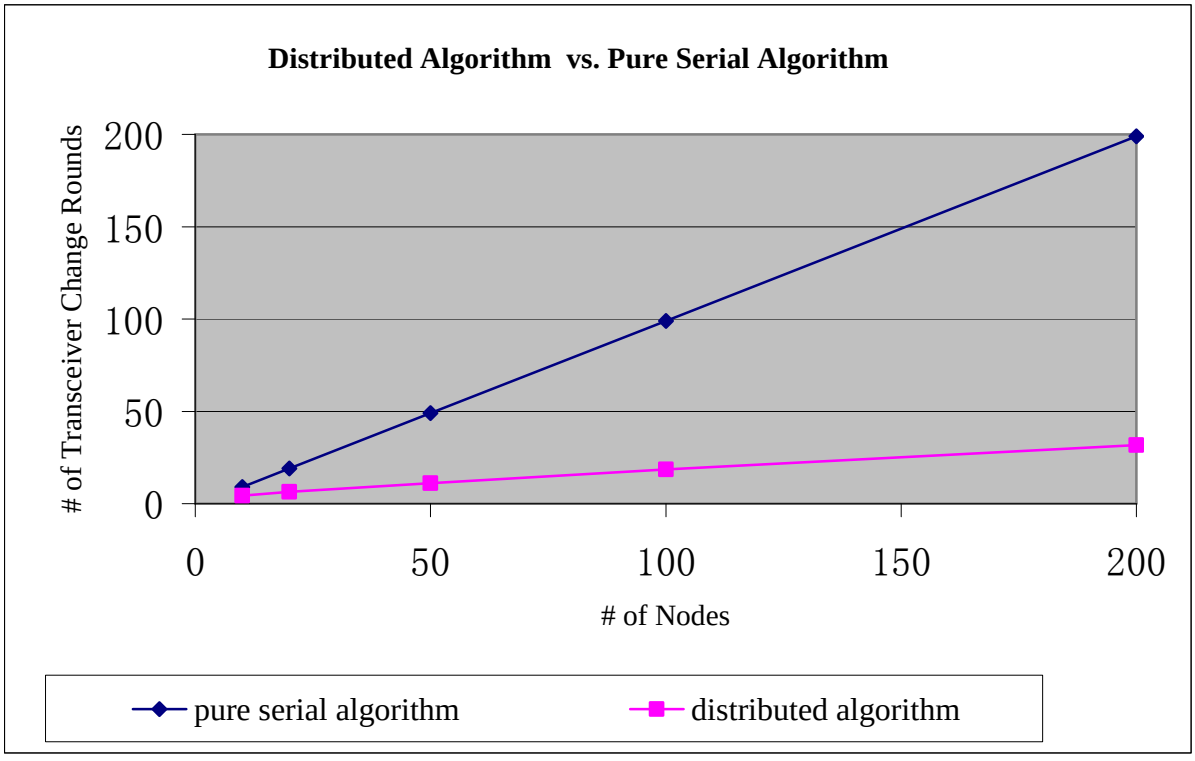


FIG. 6.2-I DISTRIBUTED ALGORITHM RESULTS VS. PURE SERIAL ALGORITHM RESULTS

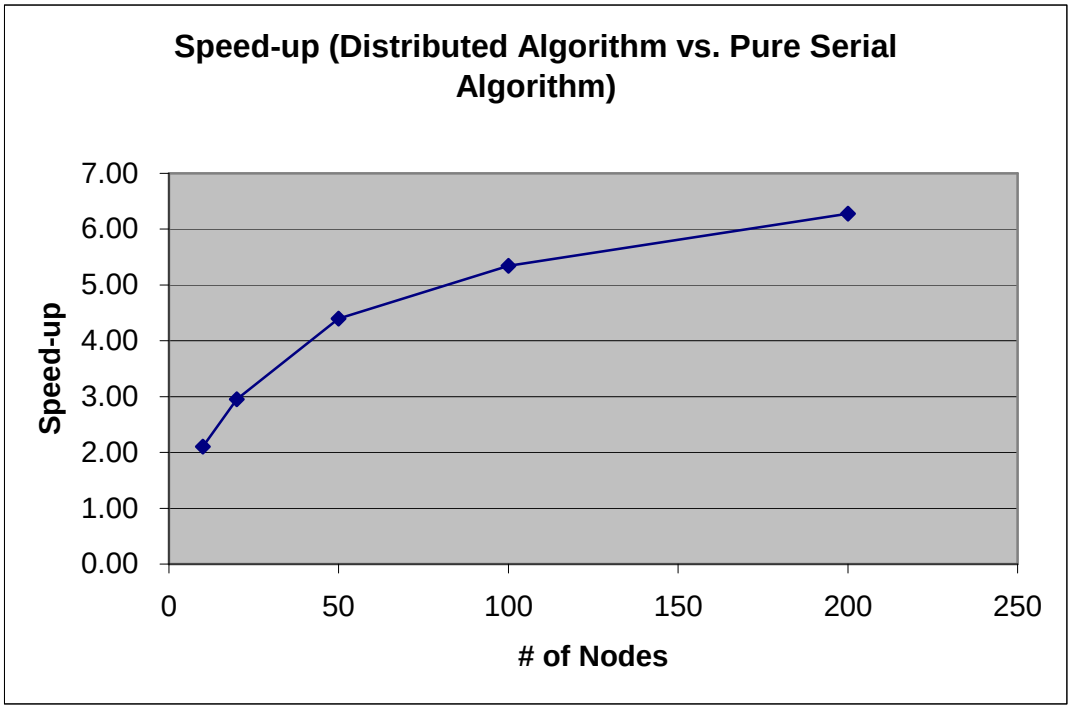


FIG. 6.2-II SPEED-UP (DISTRIBUTED ALGORITHM VS. PURE SERIAL ALGORITHM)

From FIG. 6.2-I, we can see, for a 200-node network, transceivers need to move in only 31 rounds in our algorithm, but need 199 rounds for the best pure serial algorithm. That means in general, the number of transceivers' movement rounds in our algorithm is expected to be much less than N. In FIG. 6.2-II, the larger the size of network is, the larger the speed-up value is. For a 100-node network, the speed-up value is above 5. Although the speed-up is not linear to the size of network (as the size is increasing, the slope of the speed-up line becomes smaller), we still can expect that for any network

larger than 200 nodes, the speed-up value is larger than 6. It means that comparing with any pure serial algorithm, our algorithm can save at least 6 times of the actual time of the transceivers' movement.

As we have anticipated, the distributed algorithm is found to have better actual-time performance than the clustered algorithm (TABLE. 6.2-III and FIG. 6.2-III).

	Clustered Algorithm (Block Size = 2)	**Clustered Algorithm (Block Size = 16)**
# of Nodes	**Avg. transceiver change rounds**	**Avg. transceiver change rounds**
10	4. 72	4. 28
20	8. 12	6. 43
50	15. 46	12. 79
100	26. 15	20. 29
200	46. 13	35. 71

TABLE 6.2-III CLUSTERED ALGORITHM SIMULATION RESULTS

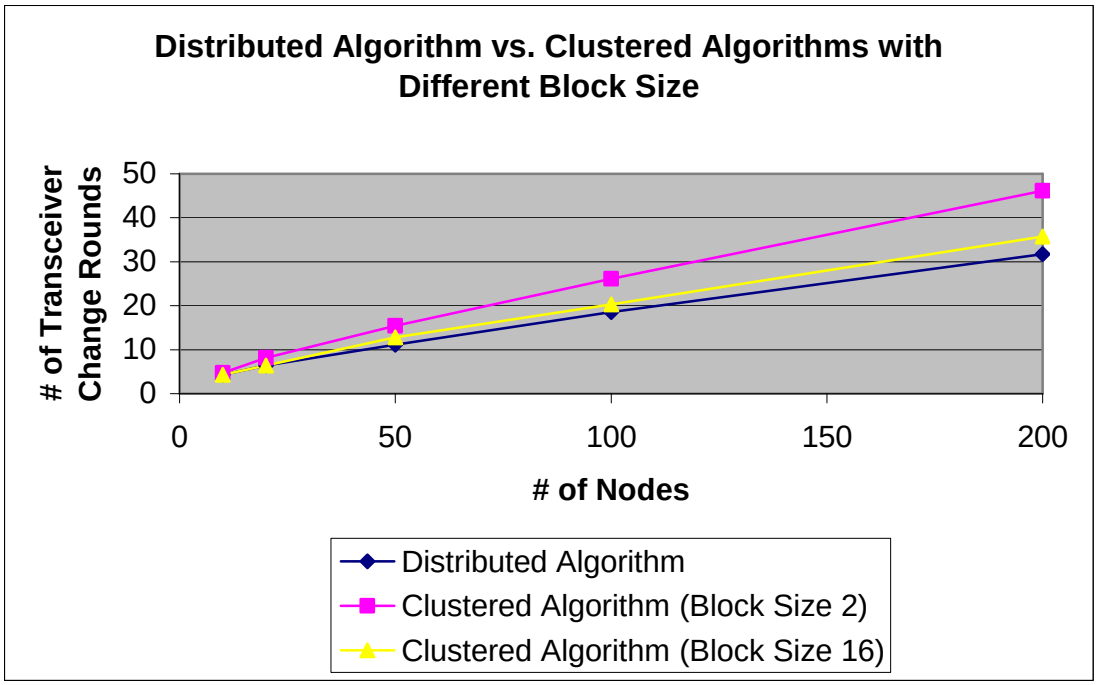


FIG 6.2-III DISTRIBUTED ALGORITHM VS. CLUSTERED ALGORITHM PERFORMANCE

From FIG. 6.2-III, we can see that, for a 200-node network, the distributed algorithm takes 31 transceivers movement rounds, while the first clustered algorithm, which splits the network into 200/2 (=100) blocks, takes 43 transceivers movement rounds, and the second clustered algorithm, which splits the network into 200/16 (=13) blocks, takes 35 transceivers movement rounds. Both clustered algorithms have worse performance than the distributed algorithm. There are two reasons that can explain this phenomenon. First, it is unknown whether the clustered algorithm, which we have used for comparison purposes, is the best available. Second, the distributed model's actual-time performance is unlikely to be exceeded by other parallel models. The conclusion is that the second explanation is more convincing. In other words, parallelizing our distributed algorithm cannot guarantee any more performance improvement. Observing from the simulation results, we find that the larger the block size is, the more the performance of the clustered algorithm is closer to the distributed one.

CHAPTER 7. FUTURE WORK

The bottom-up Minimum-Degree Spanning Tree algorithm is a good solution for the physical model of FSO networks. We think our future work can be split into two separate research directions. First, strategies need to be investigated to improve the actual-time performance of our distributed algorithm even more. We realize that several problems remain in our algorithm, such as synchronization. They can affect the actual-time performance somewhat. A possibly better parallel method (than the one that we present in Chapter 5) could be of interest. Secondly, other applications should be sought where our bottom-up model is more effective than the existing top-down model (or variants).

7.1 Future work with regard to Optical Wireless Network

As we have compared the clustered bottom-up algorithm with the distributed bottom-up algorithm in Chapter 6, the clustered bottom-up algorithm that we have designed, does not have a better actual-time performance than the distributed algorithm for most of the practical cases we observed. One thing that needs to be investigated is if a better real-parallel algorithm exists. A variant of this fundamental question is if the distributed algorithm we have designed serves as a lower bound to the performance of the best real-parallel algorithm. A starting point for this research would be to design more examples of real-parallel algorithms and test our algorithm against them.

Another interesting research topic is related to synchronization. At the end of each Sub-step, all the nodes should realize it’s the point to enter next sub-step. Especially, when the

algorithm reaches the sub-step E in step II and cannot make any new group merging, all the nodes should simultaneously know of the deadlock and realize that they should increase the degree limit by 1. Currently, we don't have a complete solution for the synchronization problem. However, we can assume that each node can send out a special beacon ("non-stall") when its group merges with other groups. If there is "non-stall" beacon detected by any node in the network, this node also sends out "non-stall" signal. After waiting for a certain period of time, if a node does not detect "non-stall" signal, it knows that the whole network has no new group merging. Then, it will increase the degree limit. How long the nodes should wait for "non-stall" information is one important parameter needed to be determined. This will also affect the actual-time performance. It's a very important task in the future.

Our future work will also involve communication complexity. Currently, the transceiver movement time is the largest bottleneck for the actual time complexity; thus, we assume that we can ignore the communication complexity in our previous algorithm analysis. Since we didn't address many details about how the nodes exchange information through the high data-rate communication channel, we cannot measure the communication complexity in our algorithm. Since the data rate of laser communication is above several hundred Mbps, we expect that the actual time for communication is optimistic. However, we need to do much research on this issue.

7.2 Future work with regard to the Bottom-up Algorithm's applications

The bottom-up minimum-degree spanning tree algorithm can be applied not only to the optical wireless network model, but also to other diverse areas, including networks other than the optical networks, VLSI design and the vehicle routing problem [RMRRH-01].

As we have explained in the introduction part, in real communication networks, the nodes are usually subject to a degree constraint. Exchanges or switches can only be physically connected to a limited number of other switches. In addition, to build a maximum reliability network, imposing the degree constraint can mitigate the damage that could be caused due to the failure of a single node/switch. Thus approximation algorithms for the problem of degree-constrained minimum-damage-cost spanning tree are similar to the one we have developed in this thesis. The cost reflects the vulnerability of the network to single point failures and the amount of load at a given point in the network is the maximum degree of any node in the network. Minimizing this cost gives rise to the minimum-degree network design problem.

For VLSI design, when wiring among pins in the backplane requires that no more than a fixed number of wires can be wrapped around any pin on the wiring panel, the minimum degree spanning tree algorithm will play a key role.

The Vehicle Routing Problem is to find an optimal route for one or more vehicles through a graph [NIST]. For example, the vehicles may be delivery trucks. Each delivery center can be abstracted by the node and each truck at the delivery station/node (which

goes to another delivery station/node) can be abstracted by the edge. Thus the number of trucks at a delivery station represents the degree of the node in the graph. Since the capacity of each delivery station is limited and yet at the same time, all the stations combined want to span the entire geographic area, the problem becomes reminiscent of minimizing the maximum degree of the graph. This means the individual delivery station can hold just enough trucks and yet all the stations combined, can make delivery from anywhere to anywhere. Again, the minimum-degree spanning tree algorithm will make a significant contribution here.

Thus our future work will aim at the following two directions: modifying the distributed algorithm in different ways to obtain better actual-time performance for the optical wireless networks and extending the existing minimum degree spanning tree problem to other diverse fields.

CHAPTER 8. CONCLUSIONS

The bottom-up minimum spanning tree algorithm is a novel algorithm for optical wireless networks. It can be used to set up the initial network connectivity in a distributed fashion as well as optimize the degree of the tree. Compared with other approximation algorithms germane to this problem, such as the top-down approach, our algorithm is found to have a superior performance in terms of actual time. For our algorithm, we have provided the proof for the approximation guarantee.

The dual advantage of our algorithm is that it can not only deal with the physical layer limitations in optical wireless networks, but also have a much better actual-time performance. Our algorithm only increases the degree limit of the tree when it's absolutely necessary; therefore, it presents the best guarantee among all the other available algorithms to preserve the transceiver limitation at each node. Meanwhile, it takes a lesser number of steps for the transceiver movement rounds, which is superior to other serial algorithms in terms of the actual-time performance.

REFERENCES

- [FR-92] "Approximating the minimum degree spanning tree to within one from the optimal degree", Martin Furer and Balaji Raghavachari, Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms, Orlando, Florida, United States, 1992, Pages: 317 – 324
- [FR-94] "Approximating the minimum-degree Steiner tree to within one from optimal", Martin Fürer and Balaji Raghavachari, Journal of Algorithms, United States, November 1994, 17(3): 409-423
- [LDVDM-03] "Reconfigurable Optical Sensor Networks", J. Llorca, A. Desai, U. Vishkin, C. Davis and S. Milner, Proceedings of SPIE, Remote Sensing (2003), Barcelona, Spain, 2003, Vol. 5237, p. 136-146
- [RMRRH-01] "Approximation algorithms for degree-constrained minimum-cost network-design problems". R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III. Algorithmica, 31:58--78, 2001.
- [GJ-79] "Computers and Intractability: A Guide to the Theory of NP-Completeness", Michael R. Garey, David S. Johnson, W. H. Freeman & Co. New York, NY, USA, 1979
- [NIST] Dictionary of Algorithms and Data Structures, NIST, http://www.nist.gov/dads/
- [DM-04] "Autonomous Reconfiguration in Free Space Optical Networks", Aniket Desai and Stuart D. Milner, submitted to IEEE-JSAC October 2004 edition.

- [DSM-03] “Flexible Optical Wireless Links and Networks”, Christopher C. Davis, Igor I. Smolyaninov, and Stuart D. Milner, IEEE Communications Magazine, March 2003, page 51-57.