HIERARCHICAL FULL REVERSAL

An Undergraduate Research Scholars Thesis

by

NAFE ALSAWFTA

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Dylan Shell

May 2017

Major: Computer Science

TABLE OF CONTENTS

ABSTRACT

Hierarchical Full Reversal

Nafe Alsawfta
Department of Computer Science
Texas A&M University

Research Advisor: Dr. Dylan Shell
Department of Computer Science
Texas A&M University

The Full Reversal Algorithm of Gafni and Bertsekas has been traditionally used to solve problems in distributed computing, such as leader election, resource allocation, and routing problems [1]. Full reversal generally works in a decentralized manner, only taking advantage of locality by reorienting edges that are incident on a node and surrounding neighbors, depending on the distributed problem being solved. The fact that Full Reversal looks at edges that are surrounding isn't troublesome; what is that is that it looks at all of these edges, no matter the cost of reversing that edge. This can lead to sub-optimal resolutions that do not minimize the cost of link reversal in a distributed problem. This thesis explores the case where: (1) there are differing costs on edges; (2) these costs are derived naturally from a hierarchical organization of the network. To minimize the cost in link reversals, in such cases, we propose an algorithm, called Hierarchical Full Reversal that takes advantage of information that may arise in neighboring nodes in the form of hierarchical cliques. The algorithm is then analyzed and compared to the traditional Full Reversal Algorithm via cases of routing problems to a leader within a graph. For hierarchical graphs, the algorithm does achieve a reduction. The experiments we conducted over a set of different graph structures show that there can be a reduction in cost, sometimes as much as by 4 8%, but with a reduction of 30% for general examples we tried.

ACKNOWLEDGEMENTS

I would like to thank Dr. Dylan Shell. He has been a great mentor and an instrumental source of guidance and support throughout the course of this research. Without him this research would not have been possible. I would also like to thank my parents for their constant encouragement in all my pursuits.

CHAPTER I

INTRODUCTION

With the rise of the Internet of things (IoT), devices that can access networks and the internet to intercommunicate, the cost of communication across devices is an essential factor to consider. In IoT the ubiquity of small heterogeneous devices (which necessarily include some that are severely resource constrained) demands an effective organization to help save energy costs, especially communication costs; a hierarchy of things, perhaps with rather different forms of local communication, is one way to realize this. For example let us propose a situation where a house hold contains devices that intercommunicate across Bluetooth. A few of these devices also communicate long range to a variety of sensory devices outdoor via Wi-Fi. In situations like these the cost to communicate locally over Bluetooth is much lower than the cost of communicating globally via Wi-Fi. For this reason, in constrained situations, for a variety of potential factors where resources are scarce, communication must be kept at an effective minimum to preserve essential resources. Particularly in situations of routing within such network, the amount of link reversals must be minimized, since link reversals may have variability in costs when communicating globally or locally over Wi-Fi or Bluetooth respectively.

For this paper we focus on the problem of routing in such a network with variability in communication costs, in this case, the cost of link reversals. There has not been much work done considering such constraints with routing algorithms that use link reversal. The earliest form of link reversal was first proposed by Gafni and Bertsekas in 1981[1]. They presented two cases of link reversal, the first Full Reversal (FR) and the second Partial Reversal (PR) [4]. In essence in

their paper, a distributed network is represented by a set of nodes that are contained in a graph with edges connecting the nodes. For each edge a virtual direction is assigned. Full Reversal or Partial Reversal is then run to orient these virtual directions in a way to route to a privileged node. A privileged node, in other words, is a node with all edges oriented towards it, which is referred to as a sink. All edges incident to a sink are incoming. Furthermore, no other node within a graph can be a sink unless it is privileged. This is the basis of Full Reversal, which operates in the following regard; firstly, any node that becomes a sink that is not privileged, has all its incident edges flipped. Secondly, the privileged node, which desires to become a sink, sits idle until all edges orient towards it. This is summarized in the algorithm 1 below [4]. FR will serve as the basis for the algorithm we later propose to reduce the cost in the number of link reversals.

Algorithm 1: Full Reversal (FR) [4]

Input: directed graph $\vec{G} = (V, \vec{E})$ with distinguished $D \in V$
1 while $\vec{G}$ has a sink other than D **do**
2 choose a nonempty subset S of sinks in $\vec{G}$, not including D
3 **foreach** $v \in S$ **do** reverse the direction of all links incident on v
4 end

PR differs in the manner of FR in that it attempts to reduce the number of repeated and unnecessary reversals. PR achieves this by allowing each node to maintain a list of links that had been reversed in a previous iteration.

With the development of FR and PR by Gafni and Bertsekas, the building blocks were laid for the developed of link reversal algorithms that tackled distributed computing problems such as leader election, routing, mutual exclusion, and resource allocation [1]. Temporally Ordered Routing Algorithm (TORA) was created by Park and Corson to for routing using link

reversal to tackle asynchronous communication and changing structure in a network [3][4]. For mutual exclusion Demmer and Herlihy leveraged link reversal to provide fair sharing of resources in a distributed network [2][4].

Extensive work has been done in using link reversal for routing, leader election, mutual exclusion, and resource allocation. Link reversal has not, however, been combined with an added action or behavior that attempts to reduce the cost in the number of link reversals in solving a distributed computing problem.

To attempt to reduce the cost in the number of link reversals we propose an algorithm called Hierarchical Full Reversal (HFR). HFR builds off of FR in that it is a routing algorithm that conducts link reversals to form a path to the privileged node that eventually becomes the sink. But our algorithm differs with an added behavior that takes advantage of information that may arise in neighboring nodes within a clique. In the case that the privileged node exists within any neighboring nodes within a clique, a packet is broadcasted within the clique. All nodes within the clique are made aware of the existences of the privileged node. With this added information, nodes can now intelligently flip edges in a direction, particularly edges that are cross level (edges going from one clique to another of differing hierarchy), that reduces the overall number of needed link reversals to resolve the routing problem. This is done by ensuring all cross level edges are incoming to the group, triggering a chain of events that causes a quicker conclusion of link reversals to the privileged node. The specifics of how this is done will be elaborated on in the *Preliminaries and Definitions* section of this paper.

With the above behavior stated, HFR's primary objective is to reduce the number of flips in network of cost variability. Cost variability is in our case is represented by a hierarchy for each clique that exists within the group as seen in figure 1 below. With the added behavior as

explained above, a reduction in the number of flips is achieved when tested against FR, which is elaborated on in the *Experiments* section. With the number of flips reduced and thus the cost of communication, our objective is achieved in a unique manner that can have significant impact on the cost of communication when solving routing problems in the IoT.

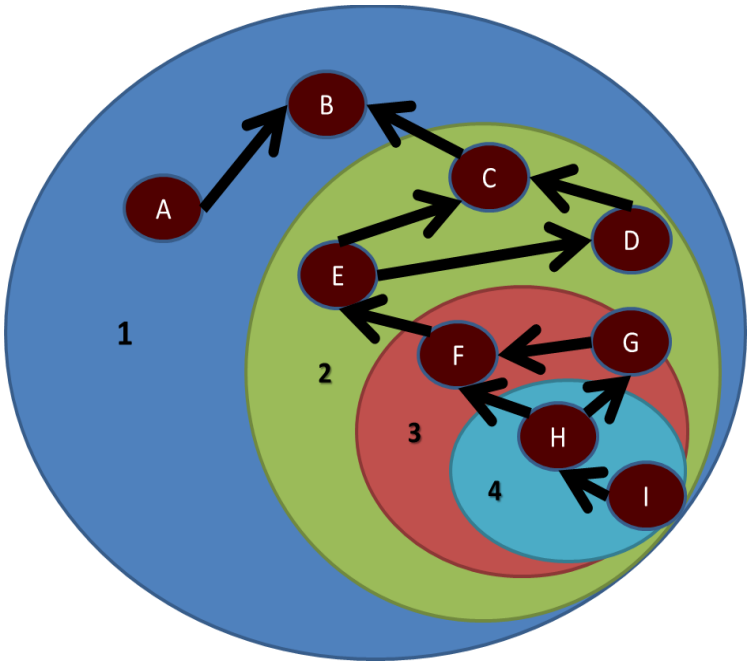

Figure 1 Representation of a hierarchical clique based network

.

CHAPTER II

DEFINITIONS AND PRELIMINARIES

In this section I will formally define key terms related to the system in which HFR operates in. I will then discuss preliminaries that must be understood to establish a frame of reference of the algorithm and how it works.

Definitions

Algorithm Objective

Algorithm has two inputs, a graph G and a partial order. With these two inputs a hierarchy of relations is created. A set of relational operators can then be used to compare hierarchical positions within the graph, which determines if an edge is cross or non-cross level. HFR is then run on each node within the graph. A route will eventually arise to the privileged node.

Graph: Nodes and Edges

We have a graph $G=(V,E)$ where $\forall\ x,y \in V$

Nodes

We have a set of n nodes, where $V=\{v,v_2,...,v_n\}$

There exists a function named $V \rightarrow N$, using our labeling name $(v_i \mapsto i)$.

There is a single node $v_p \in V$, which we call privileged.

$\exists!\ v_s \in V, s \in \forall (v_i,v_j) \in E, v_i \neq v_s$

Edges

$\exists <\bar{e}_0,\bar{e}_1,...,\bar{e}_u>$

Set $\bar{e}_0=(x,-)$, $\bar{e}_1=(-,y)$

and $\bar{e}_i=(-,b)$, $\bar{e}_{i+1}=(b,-)$

$\bar{e} \in E$

More simply put: for each (u,v) *in* $\vec{E}$,$\{u,v\}$ *is in* E

and for each $\{u,v\}$ *in* E, exactly one of (u,v) and (v,u) is in $\vec{E}$

Relational Operator

Once the relational table is created it is quite simple to run operations as such:

$v_a \not\prec v_b$ & $v_b \not\prec v_a$

Neighbors

Neighbors are defined as follows:

v_a & v_b are neighbors $\Leftrightarrow \exists\ T \in \Theta(S_v)$ and $v_a \in T$ & $v_b \in T$. v_a and v_b

exists at the same hierarchical level

Using our relational operators, neighbors can be defined simply as:

$v_a \not\prec v_b$ & $v_b \not\prec v_a$

Non-level Crossing Edge

A non-level crossing Edge is defined as follows:

An edge $e_{ij}=(v_i,v_j)$ is non-level crossing if

$v_i \not\prec v_j \& v_j \not\prec v_i$

Level Crossing Edge

A level crossing Edge is defined as follows:

An edge $e_{ij}=(v_i,v_j)$ is non-level crossing if

$v_i \prec v_j \& v_j \prec v_i$

Mediators

A mediator is a node that is level crossing. By this we mean that the node has neighbors of two different hierarchical levels. In this way the node acts as a mediator between two cliques of different levels. This is defined as follows:

$v_a \not< v_b$ & $v_b \not< v_a$ & ($v_b < v_c \parallel v_c < v_b$) & $v_a \not< v_c$ & $v_c \not< v_a$

Clique

A clique can be defined as a set of nodes that are connected by a set of edges that are non-level crossing. Non-leveling crossing means that all the nodes in the clique have the same hierarchical level. Vertices in a clique must be able to communicate with one and other either directly or through a path. A clique can be defined as follows:

$\Omega_{clique}: U \rightarrow \{c_0, c_1, c_2, ..., c_u\}$

such that if $\Omega_{clique}(v_1) = \Omega_{clique}(v_2)$ then

v_1&v_2 have a path

$e = <e_1, e_2, ..., e_l>$

such that $e_1 = <v_1, v^a>$,

$e_2 = <v^a, v^b>$ ...

such that $e_l = <v^k, v_2>$ and all $<e_1 ... e_l>$ are non-level crossing and if v_1&v_2 are level-intermediary then $\Omega_{clique}(v_1) = \Omega_{clique}(v_2)$

Properties

- Once a clique is collapsed Acyclicity is maintained
- Global Level acyclicity is not necessary

- Edges don't play a role in the definition of neighbors do to the subset structure given
- Neighbors are inferior, superior, or neither (on level).
- Partial Order indicates a group of nodes that are neither inferior or superior (neighbors)
- Partial order can be a set within a set where all nodes are at the same level of hierarchy.

Preliminaries

Hierarchical Structure

A partial order is used to represent the hierarchy and grouping of nodes. For example the following can be a case {A, B, {C, D, E, {F, G, {H, I}}}}. This is represented by figure 1 above. The partial order is then used to determine the type of edges, either cross level or non-cross level. This also allows nodes to determine if they are mediators.

Determining the cost of a flip

The cost of a flip is determined by the following calculation:

Cost = abs(Node1_level – Node2_level) * 10

The greater the difference in cross level communication, the greater the cost of flipping an edge.

Differing nodes

There exist three types of nodes:

- A privileged node: This type of node has been assigned a special insignia identifying it as the leader or a special node.
- A mediator node: This type of node has at least one cross level edge.
- A normal node: A node that is neither privileged nor a mediator.

Edges

Edges of the system represent communication relationships between the various nodes. The edges in the sense of algorithm are unidirectional pointing from one node to the other. From the physical perspective wireless communication can be bidirectional, but for the sake of the algorithm, we create this abstraction to allow for routing and leader election. Stacks of edges with different markings and routings can be developed to allow for multiple pathways of communication.

Packets

Packets are used to communicate within a clique if the privileged exists within the group. The packet is sent using the Broadcast Protocol which is elaborated on in the next section.

Catalysts of System

- A node becomes privileged
- An edge is flipped

The system is adaptive and dynamic and the state remains transient and always changing until a global leader has been assigned. Only then can the system reach a steady state where no flips occur and a path exists to the privileged node.

The Perspectives

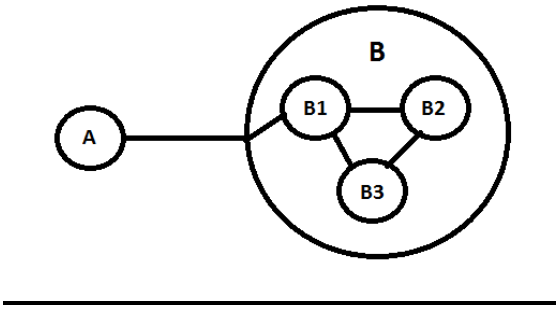

Figure 2 Clique B is shown above, containing nodes B1, B2, and B3. Node A is not a clique but interacts with clique B via node B1.

In our network of nodes and cliques there are a few perspectives that we must define to ultimately understand the behavior and eventual actions of our protocol. In figure 2 I show one example of the many potential states of our network. We can define four different states in the example above and they are as follows:

- Single Node: A
- Clique B:
- Local Nodes of clique B: B1, B2, and B3.
- Mediator Node B1

Regarding perspectives in the example above:

- From the perspective of Node A, I am communicating with Clique B via mediator node B1.
- From the perspective of Clique B, I contain nodes B1, B2, and B3, but act as a single entity when interacting with other nodes in the same level of the hierarchical structure.

- From the perspective of nodes B1, B2, and B3, we are operating at a local level within clique B, we only care about local nodes at the same hierarchical level unless I am a mediator node.
- From the perspective of node B1, I am a mediator node, I communicate with the outside world because I am connected to a node of a higher hierarchical level.
- The arrangement of nodes can be thought of in the perspective of a company hierarchy, where the inner workings of a group, a clique in our case, are masked and encapsulated, and where points of contact are termed mediators or heads of local groups in dealing with the outside world.

CHAPTER III

PSEUDOCODE AND PROOF OF CORRECTNESS

An Overview of the Algorithm

HFR in its entirety is run on each node for an infinite amount of time unless determined by the simulation. In our case the algorithm is run until a path to the privileged node exists and there are no link reversals remaining. HFR consists of three distinct behaviors that are run based off of the state of the node.

HFR

1. The node is privileged
    a. The Broadcast packet indicates that the privileged node is within the group if not sent previously
    b. If also a mediator, then flip all incident cross level outgoing edges
2. The node is a mediator
    a. Check if packet has been received
        i. If so process packet and rebroadcast to neighbors
    b. If Privileged node is within the group then
        i. Flip all incident cross level outgoing edges
        ii. If still a sink after flip then
        iii. Flip all incident non-cross level outgoing edges
    c. If privileged node not amongst us then
        i. If sink, flip all edges
3. The node is neither a mediator nor privileged
    a. If sink, flip all edges

Broadcast

The broadcast protocol sends packets to all neighboring nodes, by queuing them into their queues for processing. Each node maintains a list of process packets. If the packet received is a duplicate then the packet is not process or broadcasted to neighbors. Packets

are also destined only for people in the same level of hierarchy of the originator node of the packet.

Process Packet

Checks if packet is destined for the same hierarchy as the node that is processing the packet. If that is true the packet is then read to see if it indicates if the privileged node is amongst the group. If this is true, the node takes note. This piece of information is then fed into the HFR algorithm which changes the behavior of the node if it is a mediator.

Pseudocode

Algorithm 2: Hierarchical Full Reversal (HFR)

Input: directed graph $\vec{G} = (V, \vec{E})$ with distinguished $D \in V$ and Poset P of the hierarchical structure of G

```
1 foreach v ∈ V do
2       if v is privileged
3               if i_am_privilegd packet sent to neighboring nodes is false
4                       broadcast privileged_amongst_us is true
5               if v is mediator
6                       flip all cross_level_out_going_edges
7       else if v is mediator
8               Receive and Process Packet
9               if privileged_amongst_us is true
10                      flip all cross_level_out_going_edges
11                      if v is sink
12                              flip all non_cross_level_edges
13              else
14                      if v is sink
15                              flip all edges
16      else
17              Receive and Process Packet
18              if v is sink
19                      flip all edges
20 end
```

Algorithm 3: Broadcast

Input: Packet *P*
1 for all $\vec{e}$ incident on *v* **do**
2 **enqueue** new Packet *P** in neighbor *v** on $\vec{e}$
3 end

Algorithm 4: Receive and Process Packet

Input: Queue *Q*
1 if *Q* has Packet *p* **then**
2 **if** *p* **!previously_receieved then**
3 **record** *p received*
4 **if** *p.hierarchy = v.hierarchy* **then**
5 **if** *p.privileged_amongst_us* **then**
6 *v.privileged_amongst_us* = true
7 **broadcast** *p*
8 **pop** *p* **from** *Q*

Proof of Correctness

Algorithm 2: Hierarchical Full Reversal (HFR)

Lemma 1 *HFR Terminates*

Proof Suppose HFR never ends. Let S be a set of sinks chosen from the list of *V*, nodes of the *G*. *V* is finite and has two nodes, node A and Node B. Node A is privileged, but Node B is a sink. There exists and edge going from Node A to Node B. HFR is called, and the edge is flipped towards Node A. Node A is privileged and a sink. HFR terminates, yielding a contradiction.

Algorithm 3: Broadcast

Lemma 2 *All nodes within the clique process the packet sent by the originator*

Proof Suppose Node A calls broadcast on a packet *p*. Node A has two neighbors, Node B and Node C at the same hierarchy level. Node B and Node C have neighbors that are not at the same hierarchy level. Node A enqueues packet *p* into Node B and Node C. Node B and Node C receive and process the packet. Node B and Node C broadcast packet *p* to their neighbors on other hierarchical levels, lets call them Nodes X. Nodes X receive packet *p* but the packet is not for their hierarchical level. The packet is not processed and is dequeued, yielding a contradiction.

CHAPTER IV

EXPERIMENT

Environment

All tests were run on Ubuntu 14.04 on a dual Intel Core i3 M370 processor. State of the graph while HFR and HR were simulated was captured using GraphViz.

Assessing Performance

To assess performance two metrics were used. The first the number of flips was recorded to route to the privileged node in the graph. Once complete the average over 100 simulations of the test case was recorded. Concurrently cost variability for edge flipping was also assessed and summed for each simulation. This was then also averaged. Cost variability is assigned by the following function: abs(Node1_level – Node2_level) * 10.

Test Cases

To test the performance of HFR versus FR 2 different graph structures were used. The first is a balanced tree and the second a linear tree. Two sample sizes for each structure were then selected; a sample size of 9 and a sample size of 18. For the linear tree (see figure 3), 4 test cases were created as follows:

1. Middle: The privileged node of the graph was selected in the middle of the graph, with about an equal number of nodes on either side of the node.
2. Completely Reversed: The privileged node was selected to be the node at the top of the linear graph. All edges were then set to outgoing, create the worst possible case for a routing algorithm.

3. Opposites: A privileged node was again selected in the middle of the graph, but this time instead of edged being directed randomly, both sides of the node has edges directed to outgoing.
4. Random: A random list of numbers was generated to assign direction to each edge

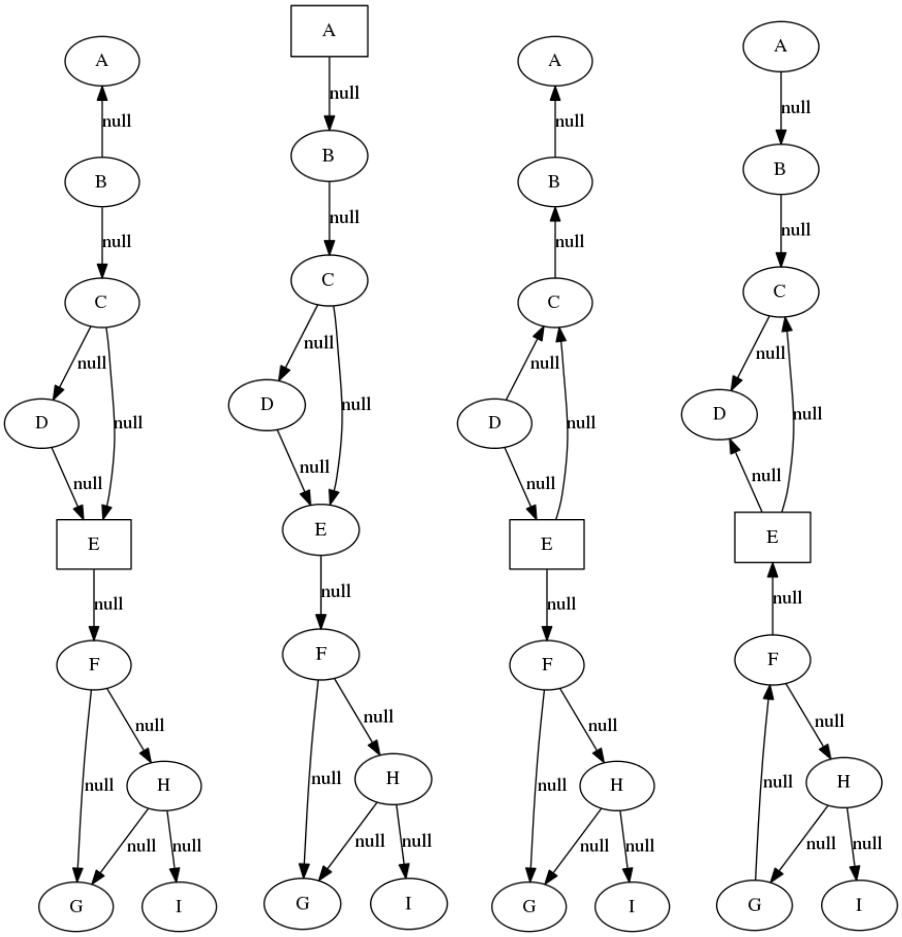

Figure 3 from left to right: middle, completely reversed, opposites, and random for a linear graph of size 9

For the balanced tree (see figure 4), 3 cases were created; middle, completely reversed, and random.

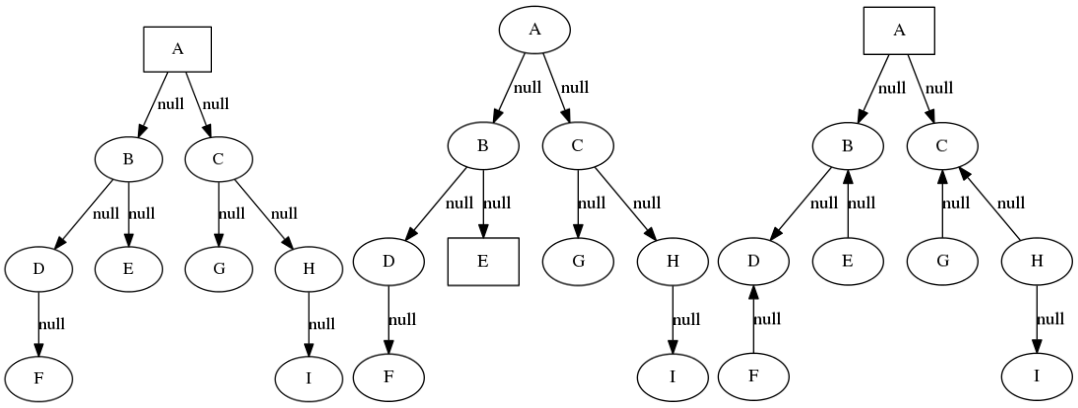

Figure 4 from left to right: completely reversed, middle, and random for a balanced graph of size 9

The same test cases were developed for both linear and balanced trees with a graph size of 18. Simulations were then run on each test case for both HFR and FR, and results were then compared.

Simulation

Nodes were awoken 700 times randomly and then called a wake function that ran the HFR or FR algorithm, depending on the case. Each case was then simulated 100 times, capturing the average flip cost and the average number of flips to resolve the routing problem. 28 test cases were run.

Results

Table 1 Experimental Results

test_case FR vs HFR	# of nodes	type of tree	performance gain in the cost of flips	performance gain in the # of flips
case1_middle	9	linear	42.07%	39.98%
case2_completely_reversed	9	linear	42.96%	45.35%
case3_opposites	9	linear	48.57%	45.89%
case4_random	9	linear	58.28%	32.44%
case1_middle	18	linear	27.45%	28.99%
case2_completely_reversed	18	linear	49.42%	48.31%
case3_opposites	18	linear	35.28%	38.94%
Case3_random	18	linear	28.51%	26.18%
case1_middle	9	balanced tree	1.38%	5.25%
case2_completely_reversed	9	balanced tree	5.97%	10.38%
case4_random	9	balanced tree	36.39%	40.93%
case1_middle	18	balanced tree	8.73%	8.30%
case2_completely_reversed	18	balanced tree	38.60%	33.95%
Case3_random	18	balanced tree	42.95%	38.71%

Discussion

As seen in table 1 above, average performance gains for the number of flips was 5.25% in the worst case to 48.3% in the best case for the various tests cases. In worst case situations where edges in a graph are all oriented away from the privileged node, HFR achieved gains of 46% on average for a linear graph versus 22% for a balanced graph. HFR gained higher performance with a linear graph over a balanced tree, but regardless performance still improved. Overall average performance gain for all test cases for a linear graph was a 38% reduction in the number of flips whereas for a balanced graph a 22.9% reduction in the number of flips.

In regards to performance gains in the cost of flips, average performance was quite the same when compared to the gains in the number of flips. This is not surprising. HFR in its current state is designed to reduce to the number of needed link reversals. Cost for each flip is assigned depending on hierarchical levels across an edge. It makes intuitive sense that the fewer

the number of flips, the lower cost of conducting the flips, since the algorithm is conducting less link reversals. You might also expect gains in the performance in the cost of flipping to be higher than the gains in the performance of the number of flips. This is expected, as seen in many of the cases in Table 1 above, but not always the case. In some graph structures, more link reversals with lower variable costs may be needed to resolve the routing problem. This can reduce the performance in the reduction in the cost of flipping, weighing done performance.

CHAPTER V

APPLICATIONS

With an on average reduction of link reversals by 22% to 38%, depending the structure of the graph, HFR has shown the potential of reducing the cost of communication within a network. Application of HFR can be beneficial in network with variability of communication costs, such as devices connected in an IoT. HFR can also be used in situations were resources and energy are scarce. Decentralized and autonomous robots could potentially use HFR to coordinate leader election in a cost effective way due to the variability of communication methods. The list goes on and on. Link reversal algorithms, like HFR and FR, have great potential in creating efficiency and optimality in solving distributed computing problems.

CHAPTER VI

CONCLUSION

HFR is a link reversal algorithm that leverages insight and information from surrounding neighbors within a hierarchical clique to reduce the number of link reversals necessary to solve a routing problem. By leveraging mediator nodes and awareness of the privileged node within a group a potential average reduction of 22% to 38% in the number of link reversals can be achieved. With this reduction the cost of communication is also reduced. With the rise of IoT on the horizon an emphasis on communication cost savings is needed. HFR shines light on the potential for huge communication cost saving, with only minor tweaking of FR. To further continue this research and to solidify the findings in efficiency, larger sample sizes must be tested to better form an understanding of average cost saving of link reversal on a longer spectrum. Furthermore there is potential to leverage the awareness of hierarchical levels to preference lower cost operations to achieve routing goals. This should also be explored. By potentially combining the additional mediator behavior as implemented in HFR and a preference for lower cost communications by leveraging hierarchical level preferences, further gains in a reduction of costs may be seen. Lastly, one other case that is worth exploring is leveraging Partial Reversal as a basis for a hierarchical routing, instead of Full Reversal as we have done. A researcher may potentially find additional gains in reduction of costs when compared to HFR.

REFERENCES

[1] Gafni, E., and D. Bertsekas. "Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology." IEEE Transactions on Communications 29, no. 1 (1981): 11-18. doi:10.1109/tcom.1981.1094876.

[2] Demmer, Michael J., and Maurice P. Herlihy. "The arrow distributed directory protocol." Lecture Notes in Computer Science Distributed Computing, 1998, 119-33. doi:10.1007/bfb0056478.

[3] Park, V.D., and M.S. Corson. "A highly adaptive distributed routing algorithm for mobile wireless networks." Proceedings of INFOCOM '97, April 7, 1997. doi:10.1109/infcom.1997.631180.

[4] Welch, Jennifer L., and Jennifer E. Walter. Link Reversal Algorithms. S.L.: Morgan and Claypool, 2012.