Distributed Detection of Cliques in Dynamic Networks

Matthias Bonne

Distributed Detection of Cliques in Dynamic Networks

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Matthias Bonne

Submitted to the Senate
of the Technion — Israel Institute of Technology
Elul 5779   Haifa   September 2019

This research was carried out under the supervision of Prof. Keren Censor-Hillel, in the Faculty of Computer Science.

Some results in this thesis have been published as an article by the author and research collaborators in a conference during the course of the author's masters research period, the most up-to-date version of which being:

Contents

List of Figures

Abstract

Real-world computer networks are dynamic in nature – nodes may join or leave the network at any time, and communication links may appear and disappear constantly. To study the capabilities of such networks, we use a dynamic variant of the CONGEST model of computer networks. In this model, a network is represented as an undirected graph, where each vertex is a computation unit, and each edge is a communication link. Since the network is dynamic, the graph may undergo certain changes during the network's operation.

One of the fundamental problems in the CONGEST model – and in other models of distributed computer networks – is the detection of certain types of subgraphs in the network. In particular, the problem of detecting triangles and larger cliques has been extensively studied over the years.

This thesis provides an in-depth study of the problem of clique detection, and other related problems, in distributed dynamic networks. We show that different types of changes in the network have very different effects on the amount of resources required to solve many of these problems, and that some problems are fundamentally harder than others. We also show that, perhaps surprisingly, the amount of resources required to solve most of the problems is similar for triangles and for larger cliques.

2

Abbreviations and Notations

$O(f(n))$	:	A function whose value is at most $c \cdot f(n)$, for some constant c
$\Omega(f(n))$	:	A function whose value is at least $c \cdot f(n)$, for some constant c
$\Theta(f(n))$	:	A function whose value is at least $c \cdot f(n)$ and at most $d \cdot f(n)$, for some constants c and d
$\tilde{O}(f(n))$	:	A function whose value is at most $\log^c n \cdot f(n)$, for some constant c
$\tilde{\Omega}(f(n))$	:	A function whose value is at least $\log^c n \cdot f(n)$, for some constant c
$\tilde{\Theta}(f(n))$	:	A function whose value is at least $\log^c n \cdot f(n)$ and at most $\log^d n \cdot f(n)$, for some constants c and d
K_s	:	A clique on s vertices
Δ	:	The maximum degree of any vertex in a graph
Detect(H)	:	The problem of detection of a given subgraph H
List(H)	:	The problem of listing of a given subgraph H
MemDetect(H)	:	The problem of membership-detection of a given subgraph H
MemList(H)	:	The problem of membership-listing of a given subgraph H

Chapter 1

Introduction

A fundamental problem in many computational settings is to find small subgraphs. In distributed networks it is particularly vital for various reasons, among which is the ability to perform some tasks much faster if, say, triangles do not occur in the underlying network graph (see, eg., [PS15, HRSS17]).

Finding cliques is a *local* task that trivially requires only a single communication round if the message size is unbounded. However, its complexity may dramatically increase when the bandwidth is restricted to the standard $O(\log n)$ bits, for an n-node network. For example, the complexity of detecting 4-cliques is at least $\Omega(n^{1/2})$ [CK18]. For triangles, the complexity is yet a fascinating riddle, where only recently the first non-trivial complexities of $\tilde{O}(n^{2/3})$ and $\tilde{O}(n^{3/4})$ have been given for detection and listing, respectively [IG17], and the current state-of-the-art is the $\tilde{O}(n^{1/2})$-round algorithm of [CPZ19]. For listing, there is an $\tilde{\Omega}(n^{1/3})$ lower bound [IG17, PRS18]. The only non-trivial lower bounds for detection say that a single round is insufficient, as given in [ACKL17] and extended for randomized algorithms in [FGKO18]. In [ACKL17] it was also shown that 1-bit algorithms require $\Omega(\log^* n)$ rounds, improved in [FGKO18] to $\Omega(\log n)$.

In this thesis, we address the question of detecting small cliques in *dynamic* networks of limited bandwidth. We consider a model that captures real-world behavior in networks that undergo changes, such as nodes joining or leaving the network, or communication links between nodes that appear or disappear. Various problems have been studied in many variants of such a setting (see below).

The task of finding cliques is unique, in that it is trivial if the bandwidth is not restricted, and it can be easily guaranteed that at the end of each round all nodes have the correct output. This implies that one does not have to wait for *stabilization* and does not have to assume that the network is *quiet* for any positive number of rounds. If, however, the bandwidth is restricted, the solution may not be as simple, although some problems can still be solved even with very small bandwidth.

As a toy example, consider the case of triangle listing when a new edge is inserted to the graph. The endpoints of the inserted edge simply broadcast a single bit that

indicates this change to all of their neighbors, and hence if a new triangle is created then its third endpoint detects this by receiving two such indications.

Nevertheless, we show that this simplified case is far from reflecting the general complexities of clique problems in such a dynamic setting. For example, the above algorithm does not solve the problem of *membership-listing* of triangles, in which *each* node should list all triangles that contain it. Indeed, we prove that this stronger variant *cannot* be solved with constant bandwidth, and, in fact, every solution must use at least $\Omega(\sqrt{n})$ bits.

Our contributions provide an in-depth study of various detection and listing problems of small cliques in this dynamic setting, as we formally define and summarize next.

Our contributions

For a subgraph H we categorize four types of tasks: Detecting an appearance of H in the network, for which it is sufficient that a single node does so, listing all appearances of H, such that every appearance is listed at least by a single node, and their two *membership* variants, membership-detection and membership-listing, for which each node has to detect whether it is a member of a copy of H, or list all copies of H to which it belongs, respectively.

The model is explicitly defined in Chapter 3. In a nutshell, there can be one topology change in every round, followed by a standard communication round among neighboring nodes, of B bits per message, where B is the bandwidth. An algorithm takes r rounds if the outputs of the nodes are consistent with a correct solution after r communication rounds that follow the last change. In particular, a 1-round algorithm is an algorithm in which the outputs are correct already at the end of the round in which the topology change occurred. Hence, 1-round algorithms are very strong, in the sense that they work even in settings in which no round is free of changes. We note that in what follows, all of our algorithms are deterministic and all of our lower bounds hold also for randomized algorithms.

Triangles

Our upper and lower bounds for triangles ($H = K_3$) are displayed in Table 1.1.

	Node deletions	Edge deletions	Edge insertions	Node insertions
Detection/Listing	0	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Membership detection	0	$\Theta(1)$	$O(\log n)$	$\Theta(n)$
Membership listing	0	$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(n)$

Table 1.1: The bandwidth complexities of 1-round algorithms for dynamic triangle problems.

Most of the complexities in this table are shown to be tight, by designing algorithms

6

and proving matching lower bounds. The one exception is for membership detection with edge insertions, for which we show an algorithm that uses $O(\log n)$ bits of bandwidth, but we do not know whether this is tight. However, we also show a 1-round algorithm for this problem that works with a bandwidth of $O((\Delta \log n)^{1/2})$, where Δ is the maximum degree in the graph, implying that if our logarithmic algorithm is optimal, then a proof for its optimality must exhibit a worst case in which the maximum degree is $\Omega(\log n)$.

A single round is sufficient for solving all clique problems, given enough (linear) bandwidth. Nevertheless, for the sake of comparison, we show that with just one additional communication round, all of the bandwidth complexities in Table 1.1 drop to $\Theta(1)$, apart from membership listing of triangles, whose bandwidth complexity for r-round algorithms is $\Theta(n/r)$.

Larger Cliques

We also study the bandwidth complexities for finding cliques on $s > 3$ nodes. Here, too, a single round is sufficient for all problems, and the goal is to find the bandwidth complexity of each problem. Some of the algorithms and lower bounds that we show for triangles carry over to larger cliques, but others do not. Yet, with additional techniques, we prove that for cliques of constant size, almost all of the 1-round bandwidth complexities are the same as their triangle counterparts.

Combining Types of Changes

In most cases, one can obtain an algorithm that handles several types of changes, by a simple combination of the corresponding algorithms. However, intriguingly, sometimes this is not the case. A prime example is when trying to combine edge insertions and node insertions for triangle detection: a node obtaining 1-bit indications of a change from two neighbors cannot tell whether this is due to an edge inserted between them, or due to an insertion of a node that is a neighbor of both. In some of these cases we provide techniques to overcome these difficulties, and use them to adjust our algorithms to cope with more than one type of change.

Challenges and Techniques

Algorithms

The main challenge for designing algorithms is how to convey enough information about the topology changes that occur, despite non-trivial (in particular, sublinear) bandwidth. Consider, e.g., edge insertions. As described, while listing triangles is trivial with a single indication bit, this fails for membership detection or membership listing.

For membership detection we can still provide a very simple algorithm for which a logarithmic bandwidth suffices, by sending the identity of the new edge, and by *helping* neighbors in the triangle to know that they are such. For membership listing even this

7

is insufficient. To overcome this challenge, we introduce a technique for sending and collecting *digests* of neighborhood information. When all digests from a given neighbor have been collected, one can determine the entire neighborhood of this neighbor. The caveat in using such an approach in a straightforward manner is that a node needs to list its triangles with a newly connected neighbor already at the same round in which they connected, and cannot wait to receive all of its neighbor's digests. By our *specific choice* of a digests, we ensure that a newly-connected neighbor has enough information to give a correct output after a single communication round, despite receiving only a single digest from its latest neighbor.

Lower Bounds

For the lower bounds, our goal is to argue that in order to guarantee that all nodes give a correct output, each node must receive enough information about the rest of the graph. To do this, we identify sequences of topology changes that exhibit a worst-case behavior, in the sense that a node cannot give a correct output if it receives too little information.

One approach for doing this is to look at a particular node x, and define as many sequences as possible such that the correct output of x is different for each sequence. At the same time, we ensure that the number of messages that x receives from the other nodes is as small as possible. These two requirements are conflicting — if x can have many different outputs, it must have many different neighbors, which implies that it receives many messages with information. Still, we are able to find such a family of sequences for each problem, and we wrap-up our constructions by using counting arguments to prove the desired lower bounds. In some cases, e.g., membership-listing under edge insertions, even this is insufficient, and we construct the sequences such that one *critical* edge affects the output of x, but it is *added last*, so that it conveys as little information as possible.

The above can be seen as another step in the spirit of the *fooling views* framework, which was introduced in [ACKL17] for obtaining the first lower bounds for triangle detection under limited bandwidth. After being beautifully extended by [FGKO18], our paper essentially gives another indication of the power of the fooling views framework in proving lower bounds for bandwidth-restricted settings.

On the way to constructing our worst-case graph sequences, we prove combinatorial lemmas that show the existence of graphs with certain desired properties. To make our lower bounds apply also for randomized algorithms, we rely on Yao's lemma and on additional machinery that we develop.

Chapter 2

Additional Related Work

Dynamic Distributed Algorithms

Dynamic networks have been the subject of much research. A central paradigm is that of *self-stabilization* [Dol00], in which the system undergoes various changes, and after some quiet time needs to converge to a stable state. The model addressed in this paper can be considered as such, but our focus is on algorithms that do not require any quiet rounds (though we also address the gain in bandwidth complexity if the system does allow them, for the sake of comparison). Yet, we assume a single topology change in each round. A single change in each round and enough time to recover is assumed in recent algorithms for maintaining a maximal independent set [CHK16, AOSS19, GK18, AOSS18, DZ18] and matchings [Sol16], and for analyzing amortized complexity [PPS16]. Highly-dynamic networks, in which multiple topology changes can occur in each round are analyzed in [BKM18, CDK$^+$19], and it is an intriguing open question how efficient can subgraph detection be in such a setting. Various other dynamic settings in which the topology changes can be almost arbitrary have been proposed, but the questions that arise in such networks address the flow of information and agreement problems, as they are highly non-structured. A significant paper that relates to our work is [KW13], which differs from our setting in the bandwidth assumption. Clique detection in the latter model is trivial, and indeed their paper addresses other problems.

Distributed Subgraph Detection

In the CONGEST model and in related models, where the nodes synchronously communicate with $O(\log n)$ bits, much research is devoted to subgraph detection, e.g., [FMO$^+$17, GO17, KR17, DKO14, ACKL17, IG17, CPZ19, PRS18].

A related problem is property-testing of subgraphs, where the nodes need to determine, for a given subgraph H, whether the graph is H-free or *far* from being H-free [CFSV19, EFF$^+$17, ELM17, FGO17, FRST16, BP11].

10

Chapter 3

Preliminaries

In the dynamic setting, the network is a sequence of undirected graphs, where each graph represents the state of the network in a particular round. The initial graph, whose topology is known to the nodes, represents the state of the network at some starting point in time, and every other graph in the sequence is either identical to its preceding graph or obtained from it by a single topology change — insertion of a single node or edge, or deletion of a single node or edge. The network is synchronized, and in each round, each node can send to each one of its neighbors a message of B bits, where B is the bandwidth of the network. Each node has a unique ID and knows the IDs of all its neighbors. We denote by n the number of possible IDs. Hence, n is an upper bound on the number of nodes that can participate in the network at all times.

Note that the nodes do not receive any indication when a topology change occurs. A node can deduce that a change has occurred by comparing the list of its neighbors in the current round and in the previous round. This implies that a node u cannot distinguish between the insertion of an edge uv and the insertion of a node v which is connected to u, as the list of neighbors of u is the same in both cases.

An algorithm can be designed to handle edge insertions or deletions or node insertions or deletions, or any combination of these. We say that an algorithm is an r-round algorithm if the outputs of the nodes after r rounds of communication starting from the last topology change are correct. The (deterministic or randomized) r-round bandwidth complexity of a problem is the minimum bandwidth for which an r-round (deterministic or randomized) algorithm exists.

We consider the following problems for a fixed subgraph H. In the problem of detection of H, the nodes give a boolean output, which can be either "yes" or "no", such that the network contains a copy of H if, and only if, at least one node outputs "yes". In the problem of listing of H, each node outputs a set of copies of H in the network, such that every copy is given by at least one node. In the problem of membership detection of H, each node outputs a boolean value that indicates whether or not it is contained in a copy of H. In the problem of membership listing of H, each node outputs the set of all copies of H that contain it. These problems are denoted by $\mathsf{Detect}(H)$, $\mathsf{List}(H)$,

MemDetect(H), and MemList(H), respectively. The following is a simple observation.

Observation 3.0.1. Let r be an integer and let H be a graph. Denote by B_{MemList}, $B_{\mathsf{MemDetect}}$, B_{List} and B_{Detect} the (deterministic or randomized) r-round bandwidth complexities of MemList(H), MemDetect(H), List(H), and Detect(H), respectively, under some type of topology changes. Then: $B_{\mathsf{Detect}} \leq B_{\mathsf{List}} \leq B_{\mathsf{MemList}}$ and $B_{\mathsf{Detect}} \leq B_{\mathsf{MemDetect}} \leq B_{\mathsf{MemList}}$.

Due to this observation, our exposition starts with the more challenging task of membership-listing, then addresses membership-detection, and concludes with listing and detection. Within each case, we first provide algorithms for each type of topology change, and then prove lower bounds.

Chapter 4

Triangle Problems

4.1 Membership listing

Table 4.1 summarizes our results for membership listing of triangles.

	Node deletions	Edge deletions	Edge insertions	Node insertions
$r = 1$	0	$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(n)$
$r \geq 2$	0	$\Theta(1)$	$\Theta(1)$	$\Theta(n/r)$

Table 4.1: Bandwidth complexities of MemList(K_3).

4.1.1 Upper bounds

We start by showing that if only node deletions are allowed, no communication is required.

Theorem 4.1. *The deterministic 1-round bandwidth complexity of* **MemList**(K_3) *under node deletions is* 0.

Proof. The algorithm is very simple: the output of a node u is the set of all triangles uvw that exist in the initial graph, such that v and w are both neighbors of u in the final graph.

Since only node deletions are allowed, every triangle uvw that exists in the final graph must also exist in the initial graph, and all three nodes u, v and w must exist and be connected to each other. Conversely, if a triangle uvw exists in the initial graph and u is still connected to v and w, then the triangle uvw must exist in the final graph. Therefore, the output of u is exactly the set of all triangles that contain u in the final graph.

Next, we show how to handle edge deletions with $O(1)$ bits of bandwidth.

Theorem 4.2. *The deterministic 1-round bandwidth complexity of* **MemList**(K_3) *under edge deletions is* $O(1)$.

Proof. The idea of the algorithm is simple: every node u needs to maintain the list of all triangles that contain it. In order to do this, u needs to be able to determine, for every two of its neighbors v and w, whether or not the edge vw exists.

This can be achieved with only a single bit, as follows. Every node sends to all its neighbors an indication whether or not it lost an edge on the current round. We denote this bit by DELETED. Thus, for every triangle uvw, whenever one of the edges of the triangle is deleted (say, vw), the other node (u in this case) receives DELETED $= 1$ from both v and w. Since the only possible change in the graph is the deletion of an edge, u can deduce that the edge vw was deleted.

The above algorithm does not work as-is if both edge deletions and node deletions are allowed. For example, assume there exists a triangle uvw. When we remove the edge vw, both v and w send DELETED $= 1$ to u, thus u knows that the edge vw was deleted and the triangle uvw no longer exists. However, u receives the exact same input if some other node x, which is connected to both v and w, but not to u, is deleted. Here, too, both v and w lose a neighbor, thus they both send DELETED $= 1$ to u, and u deduces — now incorrectly — that the triangle uvw no longer exists and removes it from its output.

This problem can be fixed, by observing that when the edge vw is deleted, both v and w know whether or not x is connected to u (i.e., whether or not the triangle vxu, or wxu, exists). Therefore, we can have each node v send DELETED $= 1$ to each neighbor u only if the lost neighbor was connected to u; that is, v sends DELETED $= 1$ to u if it lost an edge to some node x such that the triangle vux existed on the previous round. This information allows u to distinguish between the two cases described above and give the correct output. Hence the following corollary.

Corollary 4.3. *The deterministic 1-round bandwidth complexity of MemList(K_3) under node/edge deletions is $O(1)$.*

Handling edge insertions and node insertions is much more complicated. We start by showing an algorithm that handles edge insertions.

Theorem 4.4. *The deterministic 1-round bandwidth complexity of MemList(K_3) under edge insertions is $O(\sqrt{n})$.*

Proof. Let $N_u(r)$ be the set of neighbours of u on round r. Note that $N_u(r)$ can be encoded as an n-bit string, which indicates, for every node x, whether or not x is a neighbour of u on round r.

The algorithm is as follows. When a new edge uv is inserted on round r, both u and v send to all their neighbors the identity of their new neighbor, denoted by NEWID.

In addition, u sends a bitmask of $\lceil \sqrt{n} \rceil$ bits to v, indicating, for every one of the previous $\lceil \sqrt{n} \rceil$ rounds, whether or not one of u's neighbors has sent a NEWID to u. We denote this information by LAST$_u$. Node v sends to u the same information.

14

Finally, u encodes $N_u(r)$ as an n-bit string, denoted $\texttt{ALL}_u(r)$, and starts sending it to v in chunks of $\lceil\sqrt{n}\rceil$ bits per round. This process begins on round r and ends on round $r + \lfloor\sqrt{n}\rfloor$, when the entire string has been sent. During these rounds, u keeps sending $\texttt{NEWID}$ to v, and to all its other neighbors, as described above. Node v does the same. It should be noted that this continuous communication between u and v is not intended for allowing them to detect triangles that appear by the insertion of the edge uv, as these are detected immediately due to previous information. Rather, communicating $\texttt{ALL}$ allows u and v to detect triangles that appear by other topology changes that may occur in subsequent rounds, as we show below.

Overall, $\texttt{NEWID}$ requires $O(\log n)$ bits, while $\texttt{LAST}$ and $\texttt{ALL}$ require $O(\sqrt{n})$ bits. Thus, the required bandwidth is $O(\sqrt{n})$.

We show that at the end of each round, every node u has enough information to determine, for every two of its neighbors v and w, whether or not the edge vw exists.

First, since only edge insertions are considered, if the edge vw exists in the initial graph, it exists throughout. Also, if vw is inserted when at least one of the edges uv or uw already exists, then u receives this information through $\texttt{NEWID}$. The only other case is when vw does not exist in the initial graph, and is inserted when u is not yet connected to either v or w. That is, the initial graph does not contain any of these three edges, and vw is inserted before the other two. W.l.o.g. assume that uv is inserted before uw. Now, let t_v be the round in which uv is inserted, and let t_w be the round in which uw is inserted.

If $t_w - t_v \leq \lceil\sqrt{n}\rceil$, then when uv is inserted, v sends a $\texttt{NEWID}$ to w. Therefore, when uw is inserted, u can determine from $\texttt{LAST}_w$ that in round t_v a neighbor of w has sent a $\texttt{NEWID}$ to w. Since the only edge inserted in that round is uv, u determines that v is a neighbor of w. If $t_w - t_v > \lceil\sqrt{n}\rceil$, then, in round t_w, v has already sent the entire string $\texttt{ALL}_v(t_v)$ to u, which indicates that w is a neighbor of v. Therefore, in all cases, u determines whether or not the edge vw exists, as claimed.

The algorithm given for Theorem 4.4 can be extended to handle edge deletions and node deletions as well, by simply having each node send to all its neighbors an indication whether or not it lost a neighbor on the current round, along with the ID of the lost neighbor (if any). This requires only $O(\log n)$ bits of bandwidth, and one can verify that this extended algorithm allows every node to give the correct output in all cases, as required. Therefore:

Corollary 4.5. *The deterministic 1-round bandwidth complexity of* MemList(K_3) *under node deletions and edge deletions/insertions is $O(\sqrt{n})$.*

Also, if we are promised a quiet round, the problem can be solved with $O(1)$ bits of bandwidth, as the next theorem shows.

Theorem 4.6. *The deterministic 2-round bandwidth complexity of* MemList(K_3) *under edge insertions is $O(1)$.*

Proof. On every round, for every two neighbors u and v, u sends two bits to v:

1. An indication whether or not u has a new neighbor w (i.e., the edge uw has been inserted on the current round). We denote this information by `NEW`.

2. An indication whether or not any other neighbor of u has sent `NEW` $= 1$ on the previous round. We denote this information by `LAST`.

Consider a triangle uvw, and assume that the edge uv is the last one inserted. When uv is inserted, both u and v send `NEW` $= 1$ to w. Since only one edge can be inserted on a single round, w knows that the edge uv has been inserted, and therefore the triangle uvw exists.

On the next round, since w received `NEW` $= 1$ from v, it sends `LAST` $= 1$ to u, indicating that some other neighbor of w has sent `NEW` $= 1$ on the last round. u knows that the edge uv is the only edge that has been inserted on the last round, hence v must be a neighbor of w. Therefore, u can deduce that the triangle uvw exists. Likewise, w sends `LAST` $= 1$ to v, from which v can deduce that the triangle uvw exists.

It follows that all three nodes can determine that the triangle uvw exists after two rounds of communication, as required.

The above algorithm can be combined with a variant of the algorithm of Corollary 4.3, in order to handle node deletions and edge deletions as well. In that algorithm, every node v sends a bit to every neighbor u, indicating whether v lost an edge on the current round. We denoted this information by `DELETED`. We can do the same thing here; however, we have the same problem that we encountered in Corollary 4.3 — when a node u receives `DELETED` $= 1$ from two neighbors v and w on the same round, it cannot determine whether the edge between v and w was deleted, or some other node x, which was connected to both v and w, was deleted. The triangle uvw exists in the latter case, but not in the former case. Therefore, u must be able to distinguish between these two cases.

In Corollary 4.3 we solved this problem by having every node v send `DELETED` $= 1$ to a neighbor u only if it lost a neighbor which is currently connected to u. This is possible, because the algorithm of Corollary 4.3 guarantees that on every round, v knows the list of all common neighbors of v and u.

In our case there is no such guarantee — u will know whether or not its missing neighbor was connected to v only in the next round. However, by using the extra quiet round, u can easily distuinguish between the two problematic cases described above, by checking whether it received `DELETED` $= 1$ from some other node on the previous round.

Thus, we extend the algorithm of Theorem 4.6 as follows. Whenever a node u loses an edge, it sends `DELETED` $= 1$ to all its neighbors. On the next round, u sends to all its neighbors an indication whether or not it received `DELETED` $= 1$ from any other neighbor on the previous round. One can verify that this information allows every node to give the correct output after 2 rounds of communication. Like the original algorithm, this

16

uses $O(1)$ bits of bandwidth on every round, thus the total bandwidth is still $O(1)$ bits. This implies the following corollary, which is clearly optimal.

Corollary 4.7. *The deterministic 2-round bandwidth complexity of* **MemList**(K_3) *under node deletions and edge deletions/insertions is* $O(1)$.

Node insertions are harder to handle than the other types of changes. If we are promised $r - 1$ quiet rounds, a simple algorithm exists that uses only $O\left(\frac{n}{r}\right)$ bits of bandwidth. As we show in Section 4.1.2, this is tight.

Theorem 4.8. *For every r, the deterministic r-round bandwidth complexity of* **MemList**(K_3) *under any type of change is* $O(n/r)$.

Proof. The algorithm is as follows: on every round, every node u prepares an n-bit message that specifies its current list of neighbors. Then, it breaks this message into B-bit blocks, where $B = \left\lceil \frac{n}{r} \right\rceil$, and sends it to all its neighbors, one block on every round.

Additionally, on every round, u sends to all its neighbors one bit indicating whether or not its list of neighbors has changed on the current round. Whenever the list of neighbors changes, u builds its new list of neighbors, breaks it into blocks, and starts sending it again.

After at most $r - 1$ quiet rounds, all nodes are guaranteed to finish sending their list to all their neighbors, thus every node can determine whether any pair of its neighbors are connected to each other or not, as required.

4.1.2 Lower bounds

The 1-round bandwidth complexities for node and edge deletions are clearly tight. Next, we show that handling edge insertions in 1 round requires at least $\Omega(\sqrt{n})$ bits of bandwidth. By Theorem 4.4, this bound is tight. We first prove the following lemma, which shows that a sufficiently dense bipartite graph includes a large enough complete bipartite subgraph. This has the same spirit as the results of Erdös [Erd64], used in [ACKL17, FGKO18] for bounding the number of single-bit rounds for detecting triangles, but here the sides can be of different sizes.

Lemma 4.1.1. *For every $\varepsilon \in (0,1)$ there exist $\alpha, \beta, \gamma \in (0,1)$, such that for every bipartite graph $G = (L \cup R, E)$ having at least $(1 - \varepsilon)|L||R|$ edges, there are subsets $A \subseteq L$ and $B \subseteq R$, whose sizes are $|A| \geq \alpha \cdot |L|$ and $|B| \geq \beta \cdot \gamma^{|L|} \cdot |R|$, such that $uv \in E$ for every $u \in A$ and $v \in B$ (i.e., the induced subgraph on $A \cup B$ is a complete bipartite graph).*

Proof. Let $\alpha = \frac{1-\varepsilon}{6}$, and let $\mathcal{A}$ be the set of all subsets of L whose size is exactly $\lceil \alpha \cdot |L| \rceil$. For every $A \in \mathcal{A}$, we denote by N_A the set of all vertices in R which are connected to every vertex in A. Consider the sum $S = \sum_{A \in \mathcal{A}} |N_A|$. Let $M = \max\{|N_A| : A \in \mathcal{A}\}$.

17

Then:

$$S \leq \sum_{A \in \mathcal{A}} M = \binom{|L|}{\lceil \alpha \cdot |L| \rceil} \cdot M \tag{4.1}$$

On the other hand, the sum S can be computed by counting, for every $v \in R$, the number of sets $A \in \mathcal{A}$ such that $v \in N_A$:

$$S = \sum_{A \in \mathcal{A}} |N_A| = \sum_{A \in \mathcal{A}} \sum_{v \in N_A} 1 = \sum_{v \in R} \sum_{\substack{A \in \mathcal{A}: \\ v \in N_A}} 1 = \sum_{v \in R} |\{A \in \mathcal{A} : v \in N_A\}|$$

For $p \in (0,1)$, let k_p be the number of vertices in R whose degree is at least $p \cdot |L|$. For every $v \in R$ whose degree is $d(v)$ we have $|\{A \in \mathcal{A} : v \in N_A\}| = \binom{d(v)}{\lceil \alpha \cdot |L| \rceil}$. Therefore we can bound the above sum

$$S = \sum_{v \in R} \binom{d(v)}{\lceil \alpha \cdot |L| \rceil} \geq \sum_{\substack{v \in R: \\ d(v) \geq p \cdot |L|}} \binom{d(v)}{\lceil \alpha \cdot |L| \rceil} \geq \sum_{\substack{v \in R: \\ d(v) \geq p \cdot |L|}} \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil} = k_p \cdot \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil}.$$

Combining this with (4.1) gives $\binom{|L|}{\lceil \alpha \cdot |L| \rceil} \cdot M \geq k_p \cdot \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil}$, which implies:

$$M \geq k_p \cdot \binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil} \Big/ \binom{|L|}{\lceil \alpha \cdot |L| \rceil} \tag{4.2}$$

We can bound k_p as follows:

$$|E| = \sum_{v \in R} d(v) = \sum_{\substack{v \in R: \\ d(v) \geq p \cdot |L|}} d(v) + \sum_{\substack{v \in R: \\ d(v) < p \cdot |L|}} d(v) \leq \sum_{\substack{v \in R: \\ d(v) \geq p \cdot |L|}} |L| + \sum_{\substack{v \in R: \\ d(v) < p \cdot |L|}} p \cdot |L|$$

$$= k_p \cdot |L| + (|R| - k_p) \cdot p \cdot |L| = (1 - p) \cdot k_p \cdot |L| + p \cdot |L| \cdot |R|$$

On the other hand we have $|E| \geq (1-\varepsilon) \cdot |L| \cdot |R|$, and therefore $(1-p) \cdot k_p \cdot |L| + p \cdot |L| \cdot |R| \geq (1 - \varepsilon) \cdot |L| \cdot |R|$, which implies $k_p \geq \frac{1 - \varepsilon - p}{1 - p} \cdot |R|$. Setting $p = \frac{1-\varepsilon}{2}$ gives $k_p \geq \frac{1-\varepsilon}{1+\varepsilon} \cdot |R|$, and substituting this into (4.2) gives:

$$M \geq \frac{1 - \varepsilon}{1 + \varepsilon} \cdot \frac{\binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil}}{\binom{|L|}{\lceil \alpha \cdot |L| \rceil}} \cdot |R|$$

Finally, we bound the binomial fraction on the right-hand size as follows. For simplicity, define $a = |L|, b = \lceil p \cdot |L| \rceil, c = \lceil \alpha \cdot |L| \rceil$. Note that $a \geq b \geq c$. Now:

$$\frac{\binom{b}{c}}{\binom{a}{c}} = \frac{\prod_{i=1}^{c}(i + b - c)}{\prod_{i=1}^{c}(i + a - c)} = \prod_{i=1}^{c} \frac{i + b - c}{i + a - c} = \prod_{i=1}^{c}\left(1 - \frac{a - b}{i + a - c}\right)$$

$$\geq \prod_{i=1}^{c}\left(1 - \frac{a - b}{a - c}\right) = \prod_{i=1}^{c} \frac{b - c}{a - c} = \left(\frac{b - c}{a - c}\right)^{c}$$

18

Therefore:

$$\frac{\binom{\lceil p \cdot |L| \rceil}{\lceil \alpha \cdot |L| \rceil}}{\binom{|L|}{\lceil \alpha \cdot |L| \rceil}} \geq \left(\frac{\lceil p \cdot |L| \rceil - \lceil \alpha \cdot |L| \rceil}{|L| - \lceil \alpha \cdot |L| \rceil} \right)^{\lceil \alpha \cdot |L| \rceil} \qquad (4.3)$$

In order to choose appropriate values for β and γ, we now distinguish between two cases for the graph G. If $|L| \leq \frac{1}{\alpha}$, then $\alpha|L| \leq 1$. Since there must be a vertex in $|L|$ having at least $(1-\varepsilon)|R|$ neighbors in R, the claim holds for every $\beta \leq 1$ and $\gamma \leq 1 - \varepsilon$.

If $|L| > \frac{1}{\alpha}$, we can further develop the right hand side of (4.3) as follows:

$$\left(\frac{\lceil p|L| \rceil - \lceil \alpha|L| \rceil}{|L| - \lceil \alpha|L| \rceil} \right)^{\lceil \alpha|L| \rceil} \geq \left(\frac{p|L| - (\alpha|L| + 1)}{|L| - \alpha|L|} \right)^{\alpha|L|+1} = \left(\frac{p - \alpha}{1 - \alpha} - \frac{1}{(1-\alpha)|L|} \right)^{\alpha|L|+1}$$

$$> \left(\frac{p - \alpha}{1 - \alpha} - \frac{1}{(1-\alpha)\frac{1}{\alpha}} \right)^{\alpha|L|+1} = \left(\frac{p - 2\alpha}{1 - \alpha} \right)^{\alpha|L|+1} = \left(\frac{1 - \varepsilon}{5 + \varepsilon} \right)^{\alpha|L|+1}$$

To sum it all up, we now have $M > \left(\frac{1-\varepsilon}{5+\varepsilon} \right)^{\alpha \cdot |L|+1}$. Recalling the definition of M, this inequality implies that there exists $A \subseteq L$ whose size is exactly $\lceil \alpha \cdot |L| \rceil$, such that $|N_A| > \left(\frac{1-\varepsilon}{5+\varepsilon} \right)^{\alpha \cdot |L|+1}$, i.e., there are more than $\left(\frac{1-\varepsilon}{5+\varepsilon} \right)^{\alpha \cdot |L|+1}$ vertices in R which are connected to every vertex in A. Therefore, the claim holds for every β and γ such that: $\beta \leq \frac{1-\varepsilon}{5+\varepsilon}, \gamma \leq \left(\frac{1-\varepsilon}{5+\varepsilon} \right)^{\alpha} = \left(\frac{1-\varepsilon}{5+\varepsilon} \right)^{\frac{1-\varepsilon}{6}}$. Thus, given $\varepsilon \in (0,1)$, the following values of α, β, γ satisfy the claim for every G:

$$\alpha = \frac{1 - \varepsilon}{6}, \beta = \frac{1 - \varepsilon}{5 + \varepsilon}, \gamma = \min \left\{ 1 - \varepsilon, \left(\frac{1 - \varepsilon}{5 + \varepsilon} \right)^{\frac{1-\varepsilon}{6}} \right\}.$$

Theorem 4.9. *The randomized 1-round bandwidth complexity of* MemList(K_3) *under edge insertions is* $\Omega(\sqrt{n})$.

Proof. Let A be a (randomized) 1-round algorithm that solves MemList(K_3) under edge insertions using bandwidth B with error probability ε. Let t be a parameter to be defined later, and consider a tripartite graph with n nodes as in Figure 4.1.

Let $\mathcal{C}$ be the set of all possible bipartite graphs with vertex sets W and U. Note that $|\mathcal{C}| = 2^{t(n-t-1)}$. For every $C \in \mathcal{C}$ and every $w \in W$, we define a sequence of changes $S_{C,w}$ as follows. We start with no edges. Then, we insert edges between U and W to get the bipartite graph C. During the next t rounds, we connect v to every $u \in U$, one by one. Finally, we insert the edge vw. In the end of $S_{C,w}$, after 1 additional round of communication, node v must output a list of all triangles containing v. By construction, this list is uniquely determined by the set of neighbors of w in U. Thus, node v needs to know the set of all neighbors of w after 1 additional round of communication.

We assume that the output of v is correct with probability at least $1 - \varepsilon$. Therefore, by Yao's lemma, there exists a deterministic algorithm A' that solves the same problem correctly for at least $1 - \varepsilon$ of all inputs. We define a bipartite graph with node sets $\mathcal{C}$
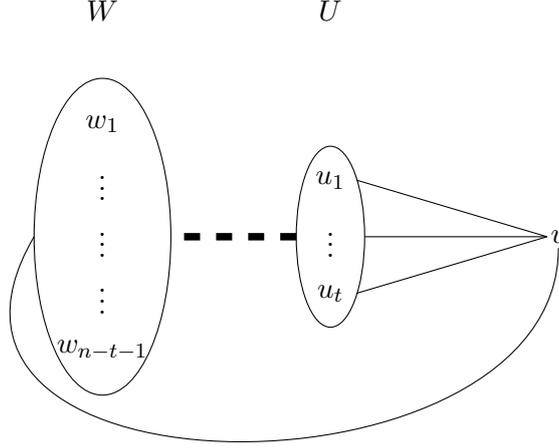
19

Figure 4.1: The lower bound sequence for $\mathsf{MemList}(K_3)$ with edge insertions. The connections between W and U are chosen. Then, v is connected to all of U. The single edge from v to W is added last.

and W, such that the edge $w - C$ exists if and only if the output of v is correct for the sequence $S_{C,w}$.

We have $|W| = n - t - 1$ and $|\mathcal{C}| = 2^{t(n-t-1)}$. Also, by assumption, this graph contains at least $(1 - \varepsilon)$ of all possible edges. Therefore, by Lemma 4.1.1, there exists $W_1 \subseteq W$ of size at least $\alpha \cdot (n - t - 1)$ and $\mathcal{C}_1 \subseteq \mathcal{C}$ of size at least $\beta \cdot \gamma^{n-t-1} \cdot 2^{t(n-t-1)}$, for some $\alpha, \beta, \gamma \in (0, 1)$ that depend only on ε, such that the output of v is correct for the sequence $S_{C,w}$ for every $C \in \mathcal{C}_1$ and every $w \in W_1$.

Now, consider the input of node v during any sequence $S_{C,w}$. During the first stage of building the bipartite graph C, v is isolated and receives no input. Then, it receives a set of messages from the nodes in U, and finally one additional message from w. Note that the messages v receives from the nodes in U depend only on C, and not on w, since the nodes in U cannot know the identity of w until the final round of communication.

Now, on every round, every node in U can send to v any of 2^B possible messages. Altogether, during the entire sequence, the nodes of U send to v a set of $\frac{t(t+3)}{2}$ messages. Hence, the number of possible inputs from the nodes of U to v is $2^{B \cdot \frac{t(t+3)}{2}}$. Therefore, there exists $\mathcal{C}_2 \subseteq \mathcal{C}_1$ of size at least $2^{-B \cdot \frac{t(t+3)}{2}} \cdot |\mathcal{C}_1|$, such that v receives the same input from all nodes in U for every sequence $S_{C,w}$ for $C \in \mathcal{C}_2$ and every $w \in W_1$. Thus, we have:

$$|\mathcal{C}_2| \geq 2^{-B\frac{t(t+3)}{2}}|\mathcal{C}_1| \geq 2^{-B\frac{t(t+3)}{2}}\beta\gamma^{n-t-1}2^{t(n-t-1)} = \beta\gamma^{n-t-1}2^{t(n-t-1)-B\frac{t(t+3)}{2}} \qquad (4.4)$$

On the other hand, we can bound the size of $\mathcal{C}_2$ by considering the number of possible neighbors of w in any $C \in \mathcal{C}_2$, for every $w \in W$. Recall that during the sequence $S_{C,w}$,

20

v receives only one message from w. Also, for every $w \in W_1$, v must determine the set of all neighbors of w in U. Since there are only 2^B possible inputs v can receive from w, every $w \in W_1$ can have at most 2^B possible sets of neighbors in any $C \in \mathcal{C}_2$.

Every $w \in W \setminus W_1$ can have any subset of U as its set of neighbors, hence it can have at most 2^t possible sets of neighbors. Now, since every $C \in \mathcal{C}_2$ is uniquely determined by set of neighbors of every $w \in W$, we have:

$$|\mathcal{C}_2| \leq \left(\prod_{w \in W_1} 2^B \right) \left(\prod_{w \in W \setminus W_1} 2^t \right) = 2^{B \cdot |W_1|} \cdot 2^{t \cdot (|W| - |W_1|)} = 2^{(B-t) \cdot |W_1| + t(n-t-1)} \quad (4.5)$$

Combining (4.4) and (4.5) gives: $\beta \cdot \gamma^{n-t-1} \cdot 2^{t(n-t-1) - B \cdot \frac{t(t+3)}{2}} \leq 2^{(B-t) \cdot |W_1| + t(n-t-1)}$, and setting $t = \lceil \sqrt{n} \rceil$ gives, with some algebraic manipulations, $B \geq \Omega(\sqrt{n})$.

Finally, for node insertions, we show that every r-round algorithm must use at least $\Omega(n/r)$ bits of bandwidth, which is tight by Theorem 4.8.

Theorem 4.10. *For every r, the randomized r-round bandwidth complexity of* MemList(K_3) *under node insertions is $\Omega\left(\frac{n}{r}\right)$.*

Proof. Let A be a (randomized) r-round algorithm that solves MemList(K_3) under node insertions using bandwidth B with error probability ε. We show that $r \cdot B = \Omega(n)$.

Let u be any node, and let $\mathcal{C}$ be the set of all possible graphs on the other $n-1$ nodes. For every $C \in \mathcal{C}$ we define the sequence S_C as follows:

- Start with an empty graph (no nodes and no edges).

- During $n-1$ rounds, insert one node on each round and connect it to the nodes which have been already inserted, according to the edges in C. After $n-1$ rounds, the graph is identical to the graph C.

- On round n insert u and connect to all the other nodes.

After $r-1$ quiet rounds u needs to output the list of all triangles that contain u. Since u is connected to all the other nodes, this implies that u needs to know the graph C. For every $C \in \mathcal{C}$, the output of u is guaranteed to be correct for the sequence S_C with probability at least $(1-\varepsilon)$. Therefore, by Yao's lemma, there exists a deterministic algorithm, A', that guarantees that the output of u is correct for at least $(1-\varepsilon)$ of all sequences. That is, there exists a subset $\mathcal{C}_1 \subseteq \mathcal{C}$, whose size is at least $(1-\varepsilon) \cdot |\mathcal{C}|$, such that A' guarantees the correct output of u for sequences S_C for all $C \in \mathcal{C}_1$.

Now, the number of possible graphs on $n-1$ nodes is $2^{\binom{n-1}{2}}$, hence the size of $\mathcal{C}_1$ is at least $(1-\varepsilon) \cdot 2^{\binom{n-1}{2}}$. Since A' is deterministic, and u needs to distinguish correctly between all possible $C \in \mathcal{C}_1$, this implies that the input u receives from its neighbors must have at least $(1-\varepsilon) \cdot 2^{\binom{n-1}{2}}$ possible values. Every neighbor of u can send any of 2^B messages on every round, thus the number of possible inputs u can receive on a

single round is $2^{B \cdot (n-1)}$. Therefore, during r rounds of communication, the number of possible inputs to u is $2^{r \cdot B \cdot (n-1)}$.

Combining the above we get that $2^{r \cdot B \cdot (n-1)} \geq (1 - \varepsilon) \cdot 2^{\binom{n-1}{2}}$, which can be simplified to $r \cdot B \geq \frac{n}{2} + \frac{\log(1-\varepsilon)}{n-1}$, and it follows that $r \cdot B = \Omega(n)$.

4.2   Membership detection

Table 4.2 summarizes the results of this subsection.

	Node deletions	Edge deletions	Edge insertions	Node insertions
$r = 1$	0	$\Theta(1)$	$O(\log n)$	$\Theta(n)$
$r \geq 2$	0	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

Table 4.2:   Bandwidth complexities of $\mathsf{MemDetect}(K_3)$

4.2.1   Upper bounds

The upper bounds for node deletions and edge deletions follow from Observation 3.0.1. The following shows that edge insertions can be handled with $O(\log n)$ bits of bandwidth.

Theorem 4.11. *The deterministic 1-round bandwidth complexity of $\mathsf{MemDetect}(K_3)$ under edge insertions is $O(\log n)$.*

Proof. The algorithm works as follows. On every round, every node sends to all its neighbors an indication whether or not it got a new neighbor on the current round, along with the ID of the new neighbor (if any). We denote this information by `NEWID`. Note that with just this information, for every pair of adjacent edges $u - v - w$, at least one of u and w know that these two edges exist (specifically, the first one connected to v knows that the other one is also connected to v).

Additionally, every node u sends to every neighbor v one bit, indicating whether u knows that v is part of some triangle. We denote this bit by `ACCEPT`.

Suppose the edge uv is inserted and creates at least one triangle uvw. As explained above, in this case at least one of u and v knows that the triangle exists, therefore it sends `ACCEPT` = 1 to the two other nodes. Thus all three nodes know that they are part of a triangle. It follows that on each round all nodes can determine whether they are in a triangle.

We can extend the algorithm of Theorem 4.11 to handle node/edge deletions as well. Now, in addition to `NEWID`, every node sends to all its neighbors an indication whether or not it has lost a neighbor on the current round, along with the ID of the lost neighbor (if any). Also, every node u sends every neighbor v one additional bit, indicating whether u received `NEWID` = v from any other node on the last round. We denote this bit by `LAST`.

22

Now, suppose that on round t the edge uv is inserted and creates a triangle uvw. As in Theorem 4.11, after one round of communication all three nodes can determine that they are part of a triangle. Moreover, both u and v send the appropriate value of NEWID to all their neighbors. Then, on round $t+1$, every common neighbor of u and v sends to both of them LAST $= 1$. From this information, u and v can deduce the list of all of their common neighbors.

It follows that on every round, every node knows the exact set of triangles that contained it on the previous round. This allows every node to handle node/edge deletions appropriately, and to determine when all triangles that contained it are gone.

Corollary 4.12. *The deterministic 1-round bandwidth complexity of MemDetect(K_3) under node deletions and edge deletions/insertions is $O(\log n)$.*

If we only consider sequences of graphs with bounded degree Δ, and only allow edge insertions, the complexity of Theorem 4.11 can be improved to $O(\sqrt{\Delta \log n})$, using an algorithm similar to that of Theorem 4.4, but with $O(\sqrt{\Delta \log n})$ bits instead of $O(\sqrt{n})$, as follows. On every round, every node sends to all its neighbors an indication whether or not it has a new neighbor. We denote this information by NEW. Additionally, whenever a new edge uv is inserted, the following happens:

- u sends to v an indication whether or not it knows about a triangle that contains v.

- For every round i within the last $\left\lceil \sqrt{\Delta \log n} \right\rceil$ rounds, u sends to v an indication whether or not it has had a new neighbor on round i.

- For every round i within the last $\left\lceil \sqrt{\Delta \log n} \right\rceil$ rounds, u sends to v an indication whether or not any of its neighbors has sent NEW $= 1$ on round i.

- u computes the list of IDs of all its current neighbors, and starts sending it to v, $\left\lceil \sqrt{\Delta \log n} \right\rceil$ bits on every round. Since u has at most Δ neighbors, after $O(\sqrt{\Delta \log n})$ rounds v has the complete list. Note that by the time this process completes, the list may not be up-to-date.

All of this requires a $O(\sqrt{\Delta \log n})$ bandwidth, and it can be shown, similarly to Theorem 4.4, that this allows every node to give the correct output in all cases.

For a quiet round we have an upper bound of $O(1)$ bits from Observation 3.0.1, Theorem 4.6, and Corollary 4.7.

Finally, for node insertions, we have a trivial upper bound of $O(n)$ bits. For a quiet round, the next theorem shows an upper bound of $O(1)$ bits.

Theorem 4.13. *The deterministic 2-round bandwidth complexity of MemDetect(K_3) under node insertions is $O(1)$.*

Proof. On every round, every node u sends to every neighbor v two bits:

23

- One bit, denoted by NEW, indicates whether or not u has a new neighbor which has been inserted on the current round.

- One bit, denoted by LAST, indicates whether or not any neighbor of u other than v has sent u NEW $= 1$ on the last round.

Consider a triangle uvw and assume that w is the last node inserted. When w is inserted, u and v send NEW $= 1$ to each other. Since only one node can be inserted on a single round, both u and v know that w is a common neighbor, hence the triangle uvw exists.

On the next round, both u and v send LAST $= 1$ to w. This implies that some neighbor of u (resp. v) has sent NEW $= 1$ on the last round, i.e., some neighbor of u (resp. v) has had a new neighbor inserted on the last round. Since only one node can be inserted on a single round, w can determine that the new neighbor must be w itself, that is, some neighbor of u (resp. v) is now also a neighbor of w. Hence w can determine that it is part of at least one triangle, and output 1 as required.

The above algorithm can be executed in parallel with the algorithm of Corollary 4.7 to handle all four types of changes. However, we now have a problem combining node insertions and edge insertions, just as we had a problem in Corollary 4.7 when we combined edge deletions and node deletions. The problem is that when a node v has a new neighbor w, it does not know whether the edge vw was inserted on the current round, or w is a new node which was inserted on the current round and may be connected to other nodes as well. For example, assume u has two neighbors, v and w, and the edge vw does not exist. Then, either the edge vw is inserted, or a new node x is inserted and connected to both v and w. In both cases, v and w send NEW $= 1$ to u, and u cannot distinguish between the former case (in which it should output 1), and the latter case (in which it should output 0).

To solve this problem, we can have every new node indicate to all its neighbors that it is new. Thus, whenever a node u has a new neighbor x, after one round of communication u can determine whether x is a new node or not and send this information to all its neighbors. This allows all nodes to distinguish between the two problematic cases we described, and give the correct output in all cases.

Corollary 4.14. *The deterministic 2-round bandwidth complexity of MemDetect(K_3) under node/edge deletions/insertions is $O(1)$.*

4.2.2 Lower bounds

For node deletions and edge deletions, we have trivial constant lower bounds, which are tight, as shown above. For edge insertions, we have shown a general algorithm that uses $O(\log n)$ bits, and also an algorithm that uses $O(\sqrt{\Delta \log n})$ bits for graphs with bounded degree Δ. The latter algorithm implies that showing a lower bound of B bits

24

for the bandwidth complexity of this problem would require looking at sequences of graphs with degree at least $\Omega(\frac{B^2}{\log n})$. In particular, in order to show that the algorithm of Theorem 4.11 is optimal, one has to consider sequences of graphs with degree at least $\log n$. Other than that, it remains an open question whether or not the algorithm of Theorem 4.11 can be improved.

Open Question 1. What is the 1-round bandwidth complexity of MemDetect(K_3) under edge insertions?

Finally, for node insertions, the following theorem shows a lower bound of $\Omega(n)$ bits.

Theorem 4.15. *The randomized* 1-*round bandwidth complexity of MemDetect(K_3) under node insertions is* $\Omega(n)$.

Proof. Let A be a (randomized) 1-round algorithm that solves MemDetect(K_3) under node insertions using bandwidth B with error probability ε. Fix $x \in V$, let $U = V \setminus \{x\}$, and let $\mathcal{C}$ be the set of all possible graphs on the nodes of U. For every $C \in \mathcal{C}$ and every $u, v \in U$, define the sequence $S_{C,u,v}$ as follows:

- Start with an empty graph (no nodes and no edges).

- During $n - 1$ rounds, insert one node of U on each round and connect it to the nodes which have been already inserted, according to the edges in C. After $n - 1$ rounds, the graph is identical to the graph C.

- On round n insert x and connect it to u and v.

Then, x should output 1 iff the edge uv exists in C. By our assumption, for every C and every $u, v \in U$, the output is correct with probability at least $1 - \varepsilon$. Note that during the final round, x receives only one message from each of u and v. Also, during this final round, u and v do not know the identity of the other neighbor of x. Hence, the message received from u depends only on the identity of u and the graph C, and not on the identity of v, and the message received from v depends only on the identity of v and the graph C.

Now, consider the following experiment for a given graph $C \in \mathcal{C}$. First, we run the sequence $S_{C,u,v}$ for some $u, v \in U$, and stop just before the last round, in which x is connected to u and v. Then, every node $w \in U$ generates a message to be sent to x, as if it has been connected to x on the final round. For every $w \in U$, let m_w be the message generated by w (note that m_w is a random variable).

Note again, that the messages generated by the nodes of U depend only on the graph C, and not on the other nodes that may have been connected to x on the last round. Therefore, for every $u, v \in U$, given the messages generated by u and v, x should be able to determine, with probability at least $1 - \varepsilon$, whether or not the edge uv exists in C.

Next, for every pair of nodes $u, v \in U$, let I_{uv} be the output of x given the two messages m_u and m_v. Note that for nodes u, v, w, the variables I_{uv} and I_{uw} are not necessarily independent. Now, let C' be the graph on the nodes of U, in which the edge uv exists if and only if $I_{uv} = 1$. We consider C' to be the result of the experiment.

Let p_C denote the probability that at the end of the experiment we have $C' = C$. Since, for every $u, v \in U$, the value of I_{uv} corresponds to the edge uv in the graph C with probability at least $1 - \varepsilon$, we have $p_C \geq (1 - \varepsilon)^{\binom{n-1}{2}}$. Summing the above for every $C \in \mathcal{C}$ we get:

$$\sum_C p_C \geq |\mathcal{C}| \cdot (1 - \varepsilon)^{\binom{n-1}{2}} = (2 - 2\varepsilon)^{\binom{n-1}{2}} \tag{4.6}$$

On the other hand, consider the set of messages generated by the nodes of U at the end of the first stage of the experiment. For every possible set of messages M generated by the nodes of U during the first stage of the experiment, denote by $\phi_C(M)$ the probability for generating exactly the messages of M. Also, for every $C' \in \mathcal{C}$, let $\Psi_M(C')$ denote the probability for the result of the experiment to be equal to C', given the set of generated messages M. We have:

$$\sum_C p_C = \sum_C \sum_M \phi_C(M) \cdot \Psi_M(C) \leq \sum_C \sum_M \Psi_M(C) = \sum_M \sum_C \Psi_M(C) = \sum_M 1 = |\mathcal{M}|$$

where $\mathcal{M}$ is the set of all possible values of M. Since every message has exactly 2^B possible values, the number of possible sets of $(n-1)$ messages is $|\mathcal{M}| = 2^{B(n-1)}$, and hence $\sum_C p_C \leq 2^{B(n-1)}$. Combining this with (4.6) gives $2^{B(n-1)} \geq (2 - 2\varepsilon)^{\binom{n-1}{2}}$. After some simplifications we get the desired bound:

$$B \geq \log(2 - 2\varepsilon) \cdot \frac{n-2}{2} = \Omega(n)$$

4.3   Listing and detection

Table 4.3 summarizes the results of this subsection.

Node deletions	Edge deletions	Edge insertions	Node insertions
0	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

Table 4.3: Bandwidth complexities of $\mathsf{List}(K_3)$ and $\mathsf{Detect}(K_3)$

4.3.1   Upper bounds

The upper bounds for node deletions and edge deletions follow from Observation 3.0.1 and the results of the last subsection.

26

For edge insertions, the algorithm of Theorem 4.6, which solves MemList(K_3) in 2 rounds, guarantees that on every round, every triangle is known to at least one of its nodes. Hence, it can be used to solve List(K_3) in 1 round.

Corollary 4.16. *The deterministic 1-round bandwidth complexity of List(K_3) under edge insertions is $O(1)$.*

The same is true for the algorithm of Corollary 4.7, where we also handle node/edge deletions. Therefore:

Corollary 4.17. *The deterministic 1-round bandwidth complexity of List(K_3) under node deletions and edge deletions/insertions is $O(1)$.*

Considering node insertions, the problem can be solved with $O(1)$ bits of bandwidth.

Theorem 4.18. *The deterministic 1-round bandwidth complexity of List(K_3) under node insertions is $O(1)$.*

Proof. The algorithm is as follows: on every round, every node sends to each one of its neighbors an indication whether or not it got a new neighbor on the current round. Thus, whenever a node w is inserted and creates a triangle uvw, u receives this indication from v (and vice versa). Since u also got connected to w on the current round, and since only one node can be inserted on every round, u can deduce that the triangle uvw exists.

Note that v also receives the same indication from u, and can deduce that the triangle uvw exists. Thus, this algorithm guarantees that every triangle is known to at least two of its nodes.

In order to handle node deletions and edge deletions, we can combine the above algorithm with the technique used in Corollary 4.7 to distinguish between the deletion of an edge and the deletion of a node. This technique ensures that when either a noe or an edge is delete on round t, all incident nodes can determine, on round $t + 1$, whether an edge or a node was deleted. This, combined with the guarantee that every triangle is known to at least two of its nodes, ensures that on every round at least one node can determine whether the triangle still exists, even if a node or an edge was deleted on the current round. Thus, listing all triangles is possible on every round, as required.

Corollary 4.19. *The deterministic 1-round bandwidth complexity of List(K_3) under node/edge deletions/insertions is $O(1)$.*

By Observation 3.0.1, all upper bounds shown here for List(K_3) also hold for Detect(K_3).

4.3.2 Lower bounds

It is not clear whether it is possible to handle both node insertions and edge insertions with $O(1)$ bits of bandwidth. The algorithm of Corollary 4.12 can be used to handle all four types of changes with $O(\log n)$ bits. Other than that, the exact bandwidth complexity of this combination is an open question.

Open Question 2. What is the 1-round bandwidth complexity of $\mathsf{List}(K_3)$ under node/edge insertions?

Chapter 5

Larger Cliques

Throughout this chapter, let $s \geq 4$ be some constant, and denote by K_s a clique on s nodes.

5.1 Membership listing

Table 5.1 summarizes the results of this subsection.

	Node deletions	Edge deletions	Edge insertions	Node insertions
$r = 1$	0	$\Theta(1)$	$\Theta(\sqrt{n})$	$\Theta(n)$
$r \geq 2$	0	$\Theta(1)$	$\Theta(1)$	$\Theta(n/r)$

Table 5.1: Bandwidth complexities of $\mathsf{MemList}(K_s)$

5.1.1 Upper bounds

For every node u, the set of all cliques that contain u is completely determined by the set of all triangles that contain u. Therefore, every algorithm that solves $\mathsf{MemList}(K_3)$ can be used to solve $\mathsf{MemList}(K_s)$ under the same set of changes. All upper bounds follow thus from the upper bounds for $\mathsf{MemList}(K_3)$ given in Section 4.1.1.

5.1.2 Lower bounds

For node deletions and edge deletions we have the trivial lower bounds of 0 and $\Omega(1)$, respectively, both of which are tight.

The lower bound for edge insertions is shown next.

Theorem 5.1. *The randomized 1-round bandwidth complexity of $\mathsf{MemList}(K_s)$ under edge insertions is $\Omega(\sqrt{n})$.*

Proof. The proof is almost identical to the proof of Theorem 4.9, with only one difference: instead of starting with an empty tripartite graph, we add a clique on $s - 3$ nodes,
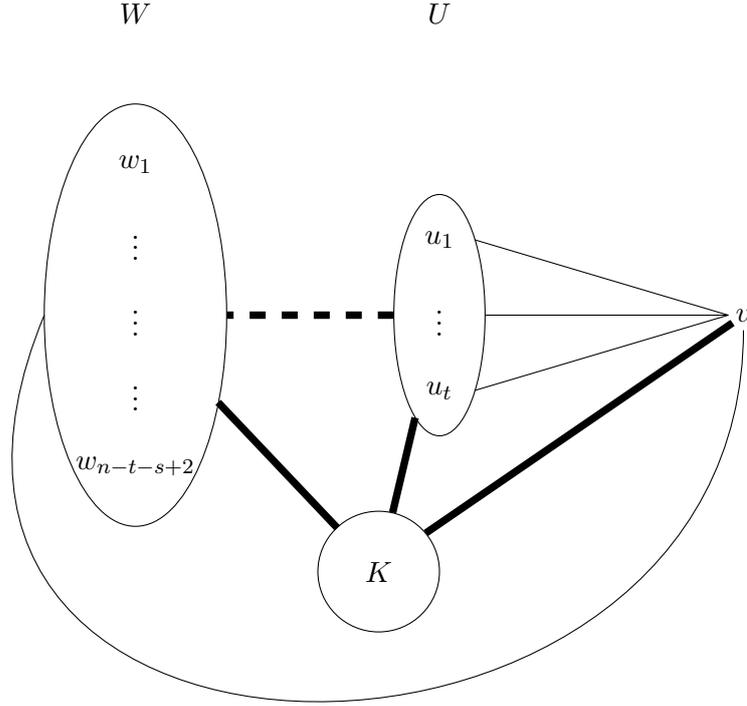
29

Figure 5.1:   The lower bound sequence for $\mathsf{MemList}(K_3)$ with edge insertions.

which are connected to every node in the graph except v. The initial graph now looks like this:

with edges between every node in K and every other node except v.

As in Theorem 4.9, for every bipartite graph C between the nodes of U and W and every node $w \in W$, we define a sequence of changes $S_{C,w}$ as follows. We start with the initial graph defined above. Then, we insert edges between U and W to get the bipartite graph C. Then, during the next $t + s$ rounds, we connect v to every node in $U \cup K$, one by one. Finally, we insert the edge vw. Note that now, since v is connected to all nodes in U and K, the set of instances of K_s that contain v is uniquely determined by the set of neighbors of w in U. Therefore, after a single round of communication, v should be able to determine the set of all neighbors of w in U.

The proof proceeds as in Theorem 4.9. The size of W is now $n - t - s + 2$, and the number of messages that v receives from the nodes of U and K is now $\frac{(t+s)(t+s+3)}{2}$. We end up with the following inequality, where $|W_1| \geq \alpha|W|$ for some constants $\alpha, \beta, \gamma \in (0,1)$:

$$\beta \cdot \gamma^{n-t-s+2} \cdot 2^{t(n-t-s+2)-B \cdot \frac{(t+s)(t+s+3)}{2}} \leq 2^{(B-t) \cdot |W_1|+t(n-t-s+2)}$$

30

Setting $t = \lceil \sqrt{n} \rceil$ gives, as in Theorem 4.9, $B \geq \Omega(\sqrt{n})$.

For $r \geq 2$, since edge insertions require communication, we have a trivial lower bound of $\Omega(1)$ bits.

For node insertions, we again use a similar construction to the one we used for triangles to show a lower bound of $\Omega(n/r)$ bits.

Theorem 5.2. *For every r, the randomized r-round bandwidth complexity of $\mathsf{MemList}(K_s)$ under node insertions is $\Omega\left(\frac{n}{r}\right)$.*

Proof. The proof is almost identical to Theorem 4.10, with the only difference being that we limit the possible graphs that are constructed on the first $n - 1$ rounds to a certain family of graphs.

Fix $x \in V$, and fix a set of $s - 3$ other nodes $K \subseteq V \setminus \{x\}$. Let $U = V \setminus (K \cup \{x\})$ be the set of all other nodes, and let $U = L \cup R$ be a partition of U into two subsets of equal size (if $|U|$ is odd, let $|L| = |R| + 1$). Let $\mathcal{C}$ be the set of all possible bipartite graphs on the sets L and R. For every $C \in \mathcal{C}$ we define a sequence of changes S_C as follows:

- Start with an empty graph (no nodes and no edges).

- During $n - (s - 2)$ rounds, insert one node of U on each round and connect it to the nodes which have been already inserted, according to the edges in C. After $n - (s - 2)$ rounds, the graph is identical to the graph C.

- During the next $s - 3$ rounds, insert one node of K on each round and connect it to all other nodes in the graph. After this stage, the graph consists of the graph C, a clique K, and all possible edges between the nodes of C and the nodes of K.

- On round n insert x and connect it to all other nodes in the graph.

After one round of communication, x should be able to determine the set of all isntances of K_s that contain it. Note that, for $C, C' \in \mathcal{C}$, if an edge uv exists in C but not in C' then the set $\{x, u, v\} \cup K$ is a clique in the final graph of S_C, but not in the final graph of $S_{C'}$. Therefore, the correct output of x is different for every two distinct $C, C' \in \mathcal{C}$.

The proof proceeds as in Theorem 4.10. The number of possible sequences is now $2^{|L| \cdot |R|} = 2^{\lfloor \frac{n-s+2}{2} \rfloor \cdot \lceil \frac{n-s+2}{2} \rceil}$, and we end up with the following inequality:

$$2^{r \cdot B \cdot (n-1)} \geq (1 - \varepsilon) \cdot 2^{\lfloor \frac{n-s+2}{2} \rfloor \cdot \lceil \frac{n-s+2}{2} \rceil}$$

After some simplifications we obtain $r \cdot B = \Omega(n)$, as claimed.

5.2 Membership detection

Table 5.2 summarizes the results of this subsection.

31

	Node deletions	Edge deletions	Edge insertions	Node insertions
$r = 1$	0	$\Theta(1)$	$O(\sqrt{n})$	$\Theta(n)$
$r \geq 2$	0	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

Table 5.2: Bandwidth complexities of $\mathsf{MemDetect}(K_s)$

5.2.1 Upper bounds

All upper bounds follow from Observation 3.0.1 and the results of the last subsection, except for the upper bound of $O(1)$ in 2 rounds under node insertions, which can be solved by the same algorithm used in Theorem 4.13 and its corollary for triangles.

5.2.2 Lower bounds

The lower bounds for node deletions and edge deletions are trivial.

As for edge insertions, it is not clear whether the upper bound of $O(\sqrt{n})$ is tight, and in general, what is the exact bandwidth complexity of the problem.

Note that the algorithm of Theorem 4.11, which solves $\mathsf{MemDetect}(K_3)$ with $O(\log n)$ bits of bandwidth, does not work for larger cliques. To understand why, consider an almost-clique on s nodes, with one missing edge uv. That is, inserting the edge uv would create a clique on s nodes.

For triangles, this is simply a node which is connected to both u and v. In this case, we can guarantee that at least one of the nodes u and v knows that this construct exists. Thus, when the edge uv is inserted, either u or v can indicate to the two other nodes that they are part of a triangle. For larger cliques, we have more than one common neighbor and more than two edges, and it is not clear how to provide the same guarantee. Therefore, the bandwidth complexity of this problem remains an open question.

Open Question 3. What is the 1-round bandwidth complexity of $\mathsf{MemDetect}(K_s)$ under edge insertions?

For node insertions, we show that at least $\Omega(n)$ bits of bandwidth are required.

Theorem 5.3. *The randomized* 1-*round bandwidth complexity of* $\mathsf{MemList}(K_s)$ *under node insertions is* $\Omega(n)$.

Proof. Our construction is a generalization of the construction used in Theorem 4.15 for triangles.

Let A be a (randomized) 1-round algorithm that solves $\mathsf{MemDetect}(K_3)$ under node insertions using bandwidth B with error probability ε. Fix $x \in V$, and fix a set of $s - 3$ other nodes $K \subseteq V \setminus \{x\}$. Let $U = V \setminus (K \cup \{x\})$, and let $U = L \cup R$ be a partition of U into two subsets of equal size (if $|U|$ is odd, let $|L| = |R| + 1$). Let $\mathcal{C}$ be the set of all possible bipartite graphs on the setst L and R. For every $C \in \mathcal{C}$ and every $u \in L, v \in R$, define the sequence $S_{C,u,v}$ as follows:

32

- Start with an empty graph (no nodes and no edges).

- During $n - (s - 2)$ rounds, insert one node of U on each round and connect it to the nodes which have been already inserted, according to the edges in C. After $n - (s - 2)$ rounds, the graph is identical to the graph C.

- During the next $s - 3$ rounds, insert one node of K on each round and connect it to all other nodes in the graph. After this stage, the graph consists of the graph C, a clique K, and all possible edges between the nodes of C and the nodes of K.

- On round n insert x and connect it to u, v, and all nodes of K.

In the final graph, x is connected to exactly $s - 1$ nodes, all of which are connected to each other except maybe u and v. Thus, x is part of a clique of size s if and only if the edge uv exists. Therefore, after one round, x should output 1 if the edge uv exists in C, or 0 otherwise. By our assumption, for every C and every $u, v \in U$, the output is correct with probability at least $1 - \varepsilon$.

During the final round, x receives only one message from u, one message from v, and $s - 3$ messages from the nodes of K. As in Theorem 4.15, when x receives these messages from u, v, and the nodes of K, these nodes do not know the identity of u and v. Therefore, the message received from u depends only on the identity of u and the graph C, and not on the identity of v. Likewise, the message received from v depends only on the identity of v and the graph C, and the $s - 3$ messages received from the nodes of K depend only on the graph C.

Now, consider the following experiment for a given graph $C \in \mathcal{C}$. First, we run the sequence $S_{C,u,v}$ for some $u, v \in U$, and stop just before the last round, in which x is connected to u and v. Then, every node $w \in U \cup K$ generates a message to be sent to x, as if it has been connected to x on the final round. For every $w \in U$, let m_w be the message generated by w, and let m_K be the set of messages generated by the nodes of K (note that m_w and m_K are random variables).

Note again, that the messages generated by the nodes of U depend only on the graph C, and not on the other nodes that may have been connected to x on the last round. Therefore, for every $u \in L, v \in R$, given the messages generated by u, v, and all nodes of K, x should be able to determine, with probability at least $1 - \varepsilon$, whether or not the edge uv exists in C.

Next, for every pair of nodes $u \in L, v \in R$, let I_{uv} be the output of x given the messages m_u, m_v, and m_K. Now, let C' be the graph on the nodes of U, in which the edge uv exists if and only if $I_{uv} = 1$. We consider C' to be the result of the experiment.

We now compute the probability that the x has the correct output for all pairs $u \in L, v \in R$. For $C \in \mathcal{C}$, let p_C denote the probability that at the end of the experiment we have $C' = C$. For every $u \in L, v \in R$, the value of I_{uv} corresponds to the edge uv in

the graph C with probability at least $1 - \varepsilon$. Thus we have:

$$p_C \geq (1 - \varepsilon)^{|L| \cdot |R|}$$

Summing the above for every $C \in \mathcal{C}$ we get:

$$\sum_C p_C \geq |\mathcal{C}| \cdot (1 - \varepsilon)^{|L| \cdot |R|} = (2 - 2\varepsilon)^{|L| \cdot |R|} \tag{5.1}$$

On the other hand, consider the set of messages generated by the nodes of $U \cup K$ at the end of the first stage of the experiment. For every possible set of messages M generated by the nodes of $U \cup K$ during the first stage of the experiment, denote by $\phi_C(M)$ the probability for generating exactly the messages of M. Also, for every $C' \in \mathcal{C}$, let $\Psi_M(C')$ denote the probability for the result of the experiment to be equal to C', given the set of generated messages M. We have:

$$\sum_C p_C = \sum_C \sum_M \phi_C(M) \cdot \Psi_M(C) \leq \sum_C \sum_M \Psi_M(C) = \sum_M \sum_C \Psi_M(C)$$
$$= \sum_M 1 = |\mathcal{M}|$$

where $\mathcal{M}$ is the set of all possible values of M. Since every message has exactly 2^B possible values, the number of possible sets of $(n - 1)$ messages is:

$$|\mathcal{M}| = 2^{B(n-1)}$$

Hence:

$$\sum_C p_C \leq 2^{B(n-1)}$$

Combining this with (4.6) gives:

$$2^{B(n-1)} \geq (2 - 2\varepsilon)^{|L| \cdot |R|}$$

Both L and R contain $\Theta(n)$ nodes. Therefore, after some simplifications we get the desired bound:

$$B \geq \log(2 - 2\varepsilon) \cdot \frac{|L| \cdot |R|}{n - 1} = \Omega(n)$$

5.3 Listing and detection

Table 5.3 summarizes the results of this subsection.

Node deletions	Edge deletions	Edge insertions	Node insertions
0	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

Table 5.3: Bandwidth complexities of $\mathsf{List}(K_s)$ and $\mathsf{Detect}(K_s)$

5.3.1 Upper bounds

The upper bounds for node deletions and edge deletions follow from Observation 3.0.1 and the results of the last subsection.

For edge insertions, the algorithm used in Corollary 4.16 for triangle listing does not work for larger cliques. However, we can use the algorithm of Theorem 4.6, which solves membership-listing of triangles in 2 rounds, to solve listing of larger cliques in 1 round.

Theorem 5.4. *The deterministic* 1*-round bandwidth complexity of* $\mathit{MemDetect}(K_s)$ *under edge insertions is* $O(1)$.

Proof. As said, we use the exact same algorithm of Theorem 4.6. This algorithm allows every node u to determine, on round t, the list of all triangles that contained u on round $t-1$. Hence, each node can determine the set of all cliques — of any size — that contained it on the previous round.

We show that every clique of size s is known to at least one node. For this, we recall that by the algorithm of Theorem 4.6, on every round, every node sends to all its neighbors an indication whether or not it got a new neighbor on the current round. This information is denoted by `NEW`.

Now, assume that the insertion of an edge uv on round t creates a clique K of size s, and let x be a node in K other than u and v. We show that on round t, x can determine that K is a clique. On round $t-1$, all nodes of K are connected by an edge except u and v. In particular, on round $t-1$, $K \setminus \{u\}$ and $K \setminus \{v\}$ are both cliques that contain x. Hence, as stated above, after round t, x knows that these two cliques exist. Moreover, on round t, both u and v send `NEW` $= 1$ to x, from which x can deduce that the edge uv has just been inserted. Hence, x can determine on round t that the nodes of K are all connected to each other, as required.

Combining this with node/edge deletions can be done similarly with the algorithm of Corollary 4.7.

Corollary 5.5. *The deterministic* 1*-round bandwidth complexity of* $\mathit{MemDetect}(K_s)$ *under node deletions and edge deletions/insertions is* $O(1)$.

For node insertions, we can use the same algorithm used for triangles in Theorem 4.18.

Theorem 5.6. *The deterministic* 1*-round bandwidth complexity of* $\mathsf{List}(K_s)$ *under node insertions is* $O(1)$.

35

Proof. We use the same algorithm as in Theorem 4.18. By this algorithm, on every round, each node sends to all its neighbors an indication whether or not it has a new neighbor on the current round. We show that this guarantees that every clique on s nodes is known to at least one node.

Consider a clique K with s nodes, and assume that u is the first node of K that has been inserted (if some of the nodes of K exist in the initial graph, let u be any one of them). We show that u can determine that K is a clique 1 round after the last node of K is inserted.

By induction, we show that for every non-empty $L \subseteq K$, u can determine that L is a clique 1 round after the last node of L is inserted. The claim then follows for $L = K$.

The claim is trivial for $L = 1$, since it contains only u. It is also trivial if L is the set of nodes of K that exist in the initial graph. Now, assume that a new node $v \in K$ is inserted, along with edges to all nodes of L. When this happens, u is connected to all nodes of L, therefore it receives from each of them an indication that they are connected to the new node. Thus, after a single round of communication, u can determine that the set $L \cup \{v\}$ is a clique, as claimed.

By Observation 3.0.1, all upper bounds shown here for $\mathsf{List}(K_s)$ also hold for $\mathsf{Detect}(K_s)$.

We note that all results of this subsection hold for any value of s, even $s = n - O(1)$.

5.3.2 Lower bounds

The lower bounds for each type of change are either 0 or $\Omega(1)$, which is trivial.

The algorithm of Theorem 5.6 also works in combination with either node deletions or edge deletions, by simply having each node send an indication whether it lost a neighbor on the current round. However, unlike the case of triangles, this does not work if we allow both node deletions and edge deletions. This is because this algorithm guarantees that every clique is known to at least two of its nodes. For triangles, this suffices to guarantee that when either a node or an edge is deleted, at least one of these two nodes know whether the triangle still exists. For larger cliques, this is not the case — for example, consider a clique on $s \geq 4$ nodes, and let u, v, x, y be four nodes in that clique. Assume that only u and v know that the clique exists. Now, either the edge between x and y is deleted, or some other node is deleted which is connected to both x and y, but not to either u or v. In both cases x and y can send an indication that they lost a neighbor, but until the next round, u and v cannot distinguish between the two cases, and therefore cannot determine whether the clique still exists.

This can be fixed by having every node send, together with an indication whether or not it has a new neighbor or a missing neighbor, the ID of the new or missing neighbor. However, this requires $O(\log n)$ bits of bandwidth, which is significantly more than the $O(1)$ bits required to handle each type of change alone. Thus, the exact bandwidth complexity of this combination remains an open question.

36

Open Question 4. What is the 1-round bandwidth complexity of $\mathsf{List}(K_s)$ under node insertions and node/edge deletions?

Also, as is the case for triangles, it is an open question whether it is possible to combine edge insertions and node insertions with $O(1)$ bits of bandwidth.

Open Question 5. What is the 1-round bandwidth complexity of $\mathsf{List}(K_s)$ under node/edge insertions?

38

Chapter 6

Conclusion

In this thesis we studied the problem of detecting cliques, and other related problems, in dynamic networks of limited bandwidth. We analyzed the amount of bandwidth required to solve each problem, when the network can undergo different types of changes. For almost every problem and every set of changes that the network can undergo, we found the best possible algorithm (in terms of bandwidth requirements), and showed that it is optimal. To prove optimality of our algorithms, we showed that every algorithm that solves the same problem necessarily requires at least the same amount of bandwidth (up to a constant factor). In most cases, we showed that this is true even for randomized algorithms, which are allowed to give an incorrect output with some probability. We introduced several new techniques, both for designing distributed algorithms and for showing lower bounds for distributed problems, which may be useful in other areas of distributed computing.

40

Bibliography

[ACKL17]   Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Christoph Lenzen. Fooling views: A new lower bound technique for distributed computations under congestion. *CoRR*, abs/1711.01623, 2017.

[AOSS18]   Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 815–826, 2018.

[AOSS19]   Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936, 2019.

[BKM18]   Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Local distributed algorithms in highly dynamic networks. *CoRR*, abs/1802.10199, 2018.

[BP11]   Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011.

[CDK+19]   Keren Censor-Hillel, Neta Dafni, Victor I. Kolobov, Ami Paz, and Gregory Schwartzman. Fast and simple deterministic algorithms for highly-dynamic networks. *CoRR*, abs/1901.04008, 2019.

[CFSV19]   Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *Distributed Computing*, 32(1):41–57, 2019.

[CHK16]   Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226, 2016.

[CK18]   Artur Czumaj and Christian Konrad. Detecting cliques in CONGEST networks. In *32nd International Symposium on Distributed Computing,*

DISC 2018, New Orleans, LA, USA, October 15-19, 2018, pages 16:1–16:15, 2018.

[CPZ19]   Yi-Jun Chang, Seth Pettie, and Hengjie Zhang. Distributed triangle detection via expander decomposition. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 821–840, 2019.

[DKO14]   Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 367–376, 2014.

[Dol00]   Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[DZ18]   Yuhao Du and Hengjie Zhang. Improved algorithms for fully dynamic maximal independent set. *CoRR*, abs/1804.08908, 2018.

[EFF+17]   Guy Even, Orr Fischer, Pierre Fraigniaud, Tzlil Gonen, Reut Levi, Moti Medina, Pedro Montealegre, Dennis Olivetti, Rotem Oshman, Ivan Rapaport, and Ioan Todinca. Three notes on distributed property testing. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 15:1–15:30, 2017.

[ELM17]   Guy Even, Reut Levi, and Moti Medina. Faster and simpler distributed algorithms for testing and correcting graph properties in the congest-model. *CoRR*, abs/1705.04898, 2017.

[Erd64]   P. Erdös. On extremal problems of graphs and generalized graphs. *Israel Journal of Mathematics*, 2(3):183–190, Sep 1964.

[FGKO18]   Orr Fischer, Tzlil Gonen, Fabian Kuhn, and Rotem Oshman. Possibilities and impossibilities for distributed subgraph detection. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 153–162, 2018.

[FGO17]   Orr Fischer, Tzlil Gonen, and Rotem Oshman. Distributed property testing for subgraph-freeness revisited. *CoRR*, abs/1705.04033, 2017.

[FMO+17]   Pierre Fraigniaud, Pedro Montealegre, Dennis Olivetti, Ivan Rapaport, and Ioan Todinca. Distributed subgraph detection. *CoRR*, abs/1706.03996, 2017.

[FRST16]   Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. In *Distributed Computing - 30th International Symposium, DISC 2016, Paris, France, September 27-29, 2016. Proceedings*, pages 342–356, 2016.

[GK18]   Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018.

[GO17]   Tzlil Gonen and Rotem Oshman. Lower bounds for subgraph detection in the CONGEST model. In *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, pages 6:1–6:16, 2017.

[HRSS17] Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. *Electr. J. Comb.*, 24(4):P4.21, 2017.

[IG17]   Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 381–389, 2017.

[KR17]   Janne H. Korhonen and Joel Rybicki. Deterministic subgraph detection in broadcast CONGEST. In *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, pages 4:1–4:16, 2017.

[KW13]   Michael König and Roger Wattenhofer. On local fixing. In *OPODIS*, volume 8304 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2013.

[PPS16]   Merav Parter, David Peleg, and Shay Solomon. Local-on-average distributed tasks. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 220–239, 2016.

[PRS18]   Gopal Pandurangan, Peter Robinson, and Michele Scquizzato. On the distributed complexity of large-scale graph computations. In *SPAA*, pages 405–414. ACM, 2018.

[PS15]   Seth Pettie and Hsin-Hao Su. Distributed coloring algorithms for triangle-free graphs. *Inf. Comput.*, 243:263–280, 2015.

43

[Sol16]   Shay Solomon. Fully dynamic maximal matching in constant up-
            date time. In *Proceedings of the IEEE 57th Annual Symposium on
            Foundations of Computer Science (FOCS)*, pages 325–334, 2016.

44

בהינתן קבוצת השינויים האפשריים שרשת יכולה לעבור בכל סיבוב, נאמר כי אלגוריתם פותר בעיה מסוימת ב־r סיבובים אם הפלט של המחשבים ברשת הוא נכון לאחר r סיבובי תקשורת שמבוצעים אחרי השינוי הטופולוגי האחרון. סיבוכיות רוחב־הפס של בעיה נתונה ב־r סיבובים היא הערך המינימלי של B שעבורו קיים אלגוריתם שפותר אותה ב־r סיבובים, כפי שהוגדר לעיל.

הבעיות שבהן נדון מוגדרות כדלקמן. יהא H גרף כלשהו. בבעיה של זיהוי H, כל צומת פולט ערך בוליאני – "כן" או "לא" – כך שהרשת מכילה עותק של H אם ורק אם לפחות אחד המחשבים פולט כן". בבעייה של רישום H, כל מחשב פולט רשימה של עותקים של H שקיימים ברשת, כך שכל עותק מוחזר על־ידי מחשב אחד לפחות. בבעייה של זיהוי־עצמי של H, כל מחשב מחזיר ערך בוליאני המציין האם הוא מוכל בעותק כלשהו של H ברשת. בבעייה של רישום־עצמי של H, כל מחשב מחזיר את קבוצת כל העותקים של H שמכילים אותו.

תקציר

רשתות מחשבים בעולם האמיתי הן דינמיות מטבען – מחשבים יכולים להצטרף לרשת או לעזוב אותה בכל זמן נתון, וקווי תקשורת יכולים להופיע ולהיעלם דרך קבע. כדי לחקור את היכולות של רשתות מסוג זה, נשתמש בגרסה דינמית של מודל ה־CONGEST של רשתות מחשבים. במודל זה, רשת מחשבים מיוצגת על־ידי גרף לא מכוון, שבו כל צומת מייצג מחשב וכל קשת מייצגת קו תקשורת בין שני מחשבים. כיוון שהרשת דינמית, גרף זה עשוי להשתנות במהלך פעילות הרשת.

אחת הבעיות המרכזיות במודל ה־CONGEST – ובמודלים אחרים של רשתות מחשבים מבוזרות – היא בעיית הזיהוי של תת־גרפים מסוגים מסויימים. בפרט, הבעייה של זיהוי משולשים וקליקאות נחקרה רבות לאורך השנים.

בתזה זו מוצג מחקר מעמיק של בעיית זיהוי קליקאות ובעיות דומות ברשתות מחשבים מבוזרות. אנו נראה כי סוגים שונים של שינויים אפשריים ברשת משפיעים באופן שונה מאוד על כמות המשאבים הדרושה לפתרון בעיות רבות, וכן נראה כי בעיות מסויימות קשות באופן מהותי מבעיות אחרות. כמו־כן נוכיח כי ברוב המקרים, כמות המשאבים הדרושה לפתרון של בעיות של זיהוי משולשים היא זהה או דומה לכמות הדרושה לפתרון אותן בעיות עבור קליקאות גדולות יותר.

במודל שבו נשתמש, הרשת הינה סינכרונית, כלומר, התקשורת בין המחשבים מתבצעת בסיבובים מתוזמנים, שבהם כל מחשב יכול לשלוח לכל אחד משכניו הודעה אחת באורך B ביטים, כאשר B הוא רוחב הפס של הרשת. לכל מחשב ברשת יש מזהה ייחודי, וכל מחשב יודע את המזהה שלו ושל כל שכניו. אנו נסמן ב־n את מספר המזהים האפשריים; אי־לכך, n הוא חסם עליון של מספר המחשבים המשתתפים ברשת בכל זמן נתון.

הרשת יכולה לעבור שינוי טופולוגי יחיד בכל סיבוב, ולאחר־מכן כל מחשב ברשת – או צומת בגרף – מתקשר עם שכניו על־ידי שליחת הודעה לכל שכן. אלגוריתם שפועל ב־r סיבובים מבטיח כי עבור כל סדרה של שינויים שהרשת עוברת, הפלט של הצמתים יהיה נכון לאחר r סיבובי תקשורת שבמהלכם הרשת לא משתנה. לדוגמה, אלגוריתם שפועל בסיבוב אחד מבטיח כי בכל סיבוב, לאחר שכל הצמתים שלחו הודעות לשכניהם באותו סיבוב, הצמתים יכולים להחזיר פלט תקין.

באופן פורמלי יותר, רשת דינמית מוגדרת כסדרה של גרפים לא מכוונים, כאשר כל גרף מייצג את מצב הרשת בסיבוב מסויים. הגרף הראשון בסדרה ידוע לכל המחשבים המשתתפים ברשת והוא מייצג את מצב הרשת בנקודת זמן התחלתית כלשהי. כל גרף אחר בסדרה הוא זהה לקודמו או מתקבל ממנו על־ידי ביצוע שינוי טופולוגי יחיד – הוספה או הסרה של צומת אחד, או הוספה או הסרה של קשת אחת.

i

המחקר בוצע בהנחייתה של פרופסור קרן צנזור-הלל, בפקולטה למדעי המחשב.

חלק מן התוצאות בחיבור זה פורסמו כמאמר מאת המחבר ושותפיו למחקר בכנס במהלך תקופת מחקר המאסטר של המחבר, אשר גרסתו העדכנית ביותר הינה:

Matthias Bonne and Keren Censor-Hillel. Distributed detection of cliques in dynamic networks. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 132:1–132:15, 2019.

זיהוי מבוזר של קליקאות ברשתות דינמיות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים במדעי המחשב

מתיאס בונה

זיהוי מבוזר של קליקאות ברשתות דינמיות

מתיאס בונה