Triangles, Long Paths, and Covered Sets

Dissertation
zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Christian-Albrechts-Universität zu Kiel

vorgelegt von

Christian Schielke

Kiel, 2021

Erster Gutachter: Prof. Dr. Anand Srivastav
Zweiter Gutachter: Prof. Dr. Anusch Taraz

Tag der mündlichen Prüfung: 22.12.2021

Contents

List of Algorithms

Abstract

In chapter 2, we consider a generalization of the well-known Maker-Breaker triangle game for uniform hypergraphs in which Maker tries to build a triangle by choosing one edge in each round and Breaker tries to prevent her from doing so by choosing q edges in each round. The triangle game in simple graphs was first introduced by Chvátal and Erdős (1987) alongside the concept of *biased* games, i.e., Breaker can choose q edges in each round instead of just one. This enabled the research on more complex winning sets than fair games. The natural question is whether there is a so-called threshold bias, i.e., a point q^* where for any bias $q > q^*$ Breaker wins and for any bias $q < q^*$ Maker wins. We give a lower bound for the threshold bias for the Maker-Breaker game played on a 3-uniform hypergraphs and an upper bound for $k \in \mathcal{O}(\sqrt{n})$. While the lower bound can be proved with a generalization of the Maker strategy of Chvátal and Erdős, the main result is the analysis of a new Breaker strategy using potential functions, introduced by Glazik and Srivastav (2019). Both bounds are of the order $\Theta(n^{3/2})$ so they are asymptotically optimal. The constant for the lower bound is $2-o(1)$ and for the upper bound it is $3\sqrt{2}$ thus the threshold bias, if it exists, must be in the interval $[2-o(1)n^{3/2}, 3\sqrt{2}n^{3/2}]$.

In chapter 3, we describe another Maker-Breaker game, namely the P_3-game in which Maker tries to build a path of length 3. First, we show that the methods of chapter 2 are not applicable in this scenario and give an intuition why that might be the case. Then, we give a more simple counting argument to bound the threshold bias.

In chapter 4, we consider the longest path problem which is a classic $\mathcal{NP}$-hard problem that arises in many contexts. Our motivation to investigate this problem in a big-data context was the problem of genome-assembly, where a long path in a graph that is constructed of the reads of a genome potentially represents a long contiguous sequence of the genome. We give a semi-streaming algorithm, i.e., an algorithm that has sequential access to the set of edges of an input graph and limited memory (RAM). Our algorithm delivers results competitive to internal memory algorithms that do not have a restriction on the amount of memory. We compared an implementation of our algorithm experimentally to existing algorithms on various types of graphs with different densities and degree distributions and also experimentally, show the high efficiency of the algorithm, in theory and practice.

In chapter 5, we investigate the b-SETMULTICOVER problem, a classic combinatorial problem which generalizes the set cover problem. Using a simple relaxation of an integer program with subsequent randomized rounding, we show that the expectation of the solution gives a good approximation. Using the bounded differences inequality of C. McDiarmid (1989), we further show that there is a strong concentration around the expectation, thus, we obtain a good approximation with high probability.

Deutsche Zusammenfassung

In Kapitel 2, betrachten wir eine Verallgemeinerung des bekannten Maker-Breaker-Dreiecksspiels für uniforme Hypergraphen, bei dem Maker versucht, ein Dreieck zu bilden, indem sie in jeder Runde eine Kante wählt, und Breaker versucht versucht, sie daran zu hindern, indem er in jeder Runde q Kanten wählt. Das Dreiecksspiel in Graphen wurde erstmals von Chvátal und Erdős (1987) zusammen mit dem Konzept der *biased* Games, d.h., Breaker kann q Kanten in jeder Runde anstelle von nur einer. Dies ermöglichte die Erforschung von komplexeren Gewinnmengen als bei fairen Spielen. Die natürliche Fragestellung ist die nach der existenz eines sogenannten Schwellenwertes, d.h. einem Punkt, an dem für einen größeren Bias Breaker gewinnt und für einen geringeren Bias Maker. Wir geben eine untere Schranke für 3-uniforme Hypergraphen und eine obere Schranke für $k \in \mathcal{O}(\sqrt{n})$.

In Kapitel 3 beschreiben wir das P_3-Spiel, ein weiteres Maker-Breaker-Spiel, in dem Maker versucht, einen Pfad der Länge 3 zu bauen. Zuerst zeigen wir, dass die Methoden aus Kapitel 2 in diesem Szenario nicht anwendbar sind und geben eine Intuition, warum das der Fall sein könnte. Dann geben wir ein einfacheres Abzählargument, um den Threshold Bias abzuschätzen.

In Kapitel 4 betrachten wir das Problem der längsten Pfade, das ein klassisches $\mathcal{NP}$-hartes Problem ist, das in vielen Zusammenhängen auftritt. Unsere Motivation, dieses Problem in einem Big-Data-Kontext zu untersuchen, war das Problem der Genom-Assemblierung, bei der ein Graph, der aus den Reads eines Genoms konstruiert ist, als Eingabe dient. In diesem stellt ein langer Pfad potentiell eine lange zusammenhängende Sequenz des Genoms dar. Wir geben einen Semi-Streaming-Algorithmus an, d.h. einen Algorithmus, der sequentiellen Zugriff auf die Menge der Kanten eines Eingabegraphen hat und begrenzten Speicher hat. Dieser Algorithmus lieferte Ergebnisse, die konkurrenzfähig zu Algorithmen mit internem Speicher sind, die keine Beschränkung auf die Speichermenge haben. Wir verglichen eine Implementierung unseres Algorithmus experimentell mit existierenden Algorithmen auf verschiedenen Typen von Graphen mit unterschiedlichen Dichten und Gradverteilungen und haben experimentell die hohe Effizienz des Algorithmus gezeigt.

In Kapitel 5 untersuchen wir das b-SETMULTICOVER Problem, ein klassisches kombinatorisches Überdeckungsproblem, das das Set-Cover Problem verallgemeinert. Unter Verwendung einer einfachen Relaxation eines ganzzahligen Programms mit anschließender randomisierten Rundung zeigen wir, dass der Erwartungswert der Lösung eine gute Approximation liefert. Unter Verwendung der Ungleichung der begrenzten Differenzen von C. McDiarmid (1989) zeigen wir außerdem, dass dass es eine starke Konzentration um den Erwartungswert herum gibt, so dass wir mit hoher Wahrscheinlichkeit eine gute Annäherung ans Optimum erreichen.

Chapter 1

Introduction

In this chapter we give a short summary of the different areas of research in this thesis, highlighting the techniques used and the results that were achieved.

1.1 The Triangle-Game in Uniform Hypergraphs

The Maker-Breaker triangle game in simple graphs is a classic biased positional game introduced by Chvátal and Erdős [CE78]. In this game, played on the complete graph, Maker attempts to claim all three edges of a triangle and Breaker tries to prevent this. Since this game is biased, i.e., Breaker claims q edges in each round, the focus of research is the so-called threshold bias. This is the point q^* where for any $q > q^*$, Breaker wins and for $q < q^*$ Maker wins. Chvátal and Erdős gave a lower bound of $\sqrt{2}\sqrt{n}$ that has not been improved yet. Their upper bound of $2\sqrt{n}$ has been improved by Balogh and Samotij [BS11], who gave a non-constructive Breaker strategy that uses probabilistic arguments. Recently, in a breakthrough work, the bound has been improved to $\sqrt{\frac{8}{3}}\sqrt{n}$ by Glazik and Srivastav [GS18] who developed a new type of potential function and a two-phase strategy. We give a generalization of this result for 3-uniform hypergraphs. First, we show an asymptotically optimal lower bound for the 3-uniform case of $2n^{3/2}$ using a Chvátal-Erdős-type of argument. We then develop the potential function for the upper bound in the 3-uniform case and give a thorough analysis of the potential's behaviour during the game. We show that if the potential never exceeds $2n$, then Breaker has a winning strategy. We further prove that if Breaker plays according to the potential function, prioritizing edges of high potential, he wins the game for $q > \sqrt{\frac{36}{5}}n^{3/2}$. At the end, we give an outlook for a potential function and an upper bound for general k-uniform hypergraphs where $k \in \mathcal{O}(\sqrt{n})$.

1.2 Path Games

We give an upper bound for the P_3-game where Maker's goal is to build a path of three edges. A trivial upper bound of $2n$ for the threshold bias can be achieved by Breaker by isolating both vertices of Maker's previously claimed edge. We first demonstrate how the techniques from the previous chapter are not useful in this scenario and then give a less sophisticated counting argument that achieves an upper bound of $\frac{\sqrt{3}+1}{2} \approx 1.366$.

1.3 A Streaming Algorithm for the Longest Path Problem

The longest path problem is the problem of finding a directed or undirected path with maximum length within a graph. Our interest in researching this problem in a Big Data context was the application in genome assembly where the genome is read by a sequencing machine and a graph can be constructed from the machine's output. A long contiguous path corresponds to a long sequence of the genome itself. While this is an oversimplification of the genome assembly problem, it might be worth investigating whether or not a path-based genome assembly algorithm would yield good results compared to the convential ones.

The longest path problem is well-known to be $\mathcal{NP}$-hard since it contains the Hamiltonian path problem as a special case. It is even $\mathcal{NP}$-hard to find an n^{δ} approximation for fixed $\delta > 0$. Because of the hardness and inapproximability, the most commonly used algorithms are heuristics that perform well in most cases.

We will first present approximation algorithms for the problem and two RAM heuristics that we compare our own algorithm to. Since the input graphs from the sequencing machine can become too large for a computer's internal memory, we developed an algorithm for the longest path problem that deals with the restrictions of the Semi-Streaming model. In this model, the memory available to the algorithm is restricted to an amount linear in the number of vertices of the input graph and the input is given as a stream in no particular order. The number of passes over the stream must be constant. The existing heuristics are based on BFS and DFS which are not possible in the Semi-Streaming model.

Our approach works in two phases: First, we construct a set of minimum spanning trees in which we limit the degrees of its vertices. This construction retains long paths and is sparse enough to be stored inside the internal memory. We then find a long path inside this graph using an existing heuristic. The algorithm then builds an MST that contains all of the edges of the found path and improves the length of the path by adding edges and removing others. Adding an edge creates a cycle. We show that our algorithm is able to detect which edge to remove from the resulting cycle, such that the length of a longest path in the tree is optimal. We show that this is done in $\mathcal{O}(n)$ time, where the naïve solution would require n^2 steps.

We analyze the results of our algorithm experimentally by comparing it to the heuristic algorithms on various types of graphs. Our algorithm delivers competitive results: with the exception of preferential attachment graphs, we deliver at least 71% of the solution of the best RAM algorithm. The same minimum relative performance of 71% is observed over all graph classes after removing the 10% worst cases. This comparison has strong meaning, since for each instance class there is one algorithm that on average delivers at least 84% of a Hamilton path. In some cases we deliver even better results than any of the RAM algorithms.

1.4 A Randomized Approximation for the Set Multicover Problem in Hypergraphs

We consider the b-MULTICOVER problem in hypergraphs, where a hypergraph $\mathcal{H} = (V, \mathcal{E})$ consisting of a finite set V of vertices, a set of (hyper) edges $\mathcal{E} \subseteq 2^V$, and $b \in \mathbb{N}$ are given as input. The b-MULTICOVER problem is the problem of finding a minimum cardinality set of edges $C \subseteq \mathcal{E}$ such that each vertex $v \in V$ is covered by at least b edges. The special case $b = 1$ is the SETCOVER problem, which is a classical combinatorial problem that is part of Karp's 21 $\mathcal{NP}$-complete problems [Kar72]. Because of the hardness of the special case, the more general version is also $\mathcal{NP}$-hard and thus, we cannot give an exact polynomial time algorithm unless $\mathcal{P} = \mathcal{NP}$. We present an approximation algorithm based on an integer programming formulation of the problem, using both deterministic and randomized rounding along with an additional repair step.

For the analysis of the algorithm, we use the well-known McDiarmid-inequality [McD89] to show that the algorithm achieves a $\left(1 - \frac{(b-1)e^{\frac{\delta+1}{2}}}{32\ell}\right)$-approximation with high probability.

Chapter 2

The Triangle-Game in Uniform Hypergraphs

2.1 Positional Games

Positional games have been an active field of research for several decades. A survey of the topic can be found in the book written by Hefetz et. al. [Hef+14] or the monograph by Beck [Bec08]. We will briefly describe the general setting and important results in the following section.

Positional games are a kind of combinatorial game played by two players on a finite set X and a family of subsets $\mathscr{F} \subseteq 2^X$, usually called *winning sets*. The two players take turns alternately by claiming elements of X that are previously unclaimed. The game ends once one of the players fulfills a winning criterion. If, at the end of the game, no player fulfils a winning criterion, the game is a draw. Positional games are perfect information games and because X is finite and the players take elements in each round, the game ends after a finite number of rounds.

The most famous positional game is Tic-Tac-Toe, where X represents the nine spaces and the set of winning sets $\mathscr{F}$ contains all rows, columns and the diagonals of length three. A player wins if he claimed all three spaces of such a winning set. It is well-known that this game ends in a draw if both players play perfectly. This can be shown by simply constructing a game graph that contains every possible scenario that can occur during the game. For obvious reasons, this enumeration is not feasible for large sets.

In Maker-Breaker games, we call the two players *Maker* and *Breaker*. Maker wins the game if she can claim all elements of a winning set and Breaker wins if he can prevent Maker from doing so until every element of X is claimed by either player.

The following criterion by Erdős and Selfridge [ES73] gives a condition on the winning sets that ensures a Breaker's win.

Theorem 2.1 (Erdős-Selfridge Criterion) *Let $\mathcal{H} = (X, \mathcal{E})$ be a hypergraph. Then,*

$$\sum_{E \in \mathcal{E}} 2^{-|E|} < \frac{1}{2} \implies \text{Breaker has a winning strategy.}$$

Biased Games

So far, we only considered games where each player claimed one element in each round. We call this a *fair* game. Chvátal and Erdős introduced the notion of *biased* positional games in their seminal paper [CE78] where the players claim p and q elements in each turn, respectively, $p, q \geq 1$. We call this a $(p : q)$-biased game. This natural extension of positional games enabled the research on an abundance of subgraph games, e.g. [Bec81; Bec82; Bec85; BŁ00; MS14].

A biased version of Theorem 2.1 was given by Beck [Bec82].

Theorem 2.2 *Let $n, m \in \mathbb{N}$ and $p, q \in [n]$. Let $\mathcal{H} = (X, \mathcal{E})$ be the game hypergraph of a $(p : q)$-Maker-Breaker game with $|X| = n$ and $|\mathcal{E}| = m$.*

(i) If $\sum_{E \in \mathcal{E}} (1+q)^{-\frac{|E|}{p}} < \frac{1}{q+1}$, then Breaker (as second player) has a winning strategy.

(ii) As first player, he has a winning strategy if $\sum_{E \in \mathcal{E}} (1+q)^{-\frac{|E|}{p}} < 1$.

2.2 Previous and Related Work

The triangle game in simple graphs has been studied extensively while to our knowledge there are no known results for the hypergraph triangle game. In the following, we summarize the current state of research on the triangle game played on simple graphs. The Maker-Breaker Triangle Game was introduced along with other biased games by Chvátal and Erdős [CE78]. Their lower bound construction consists of a Maker strategy where she builds a star around a fixed vertex leading to her win for $q < \sqrt{2}\sqrt{n}$. This lower bound has not been improved since and is conjectured to be optimal. For the upper bound given in this paper, Breaker's strategy is to close possible Maker-triangles whenever they appear, because otherwise Maker could win the game in the next round, and to avoid stars of size $q/2$, where q is

Breaker's bias. Their upper bound is $2\sqrt{n}$ and for any $q \geq 2\sqrt{n}$, this is a Breaker winning strategy. The upper bound has been improved slightly by Balogh and Samotij [BS11], who gave a non-constructive Breaker strategy using probabilistic methods leading to an upper bound of $1.935\sqrt{n}$. Recently, the upper bound has been substantially improved to $\sqrt{(8/3+\varepsilon)}\sqrt{n}$, for arbitrarily small $\varepsilon > 0$, by Glazik and Srivastav [GS18], who gave a constructive Breaker strategy that uses a new type of non-monotone potential function and a two-phase strategy.

Bednarska and Łuczak [BŁ00] developed a technique for deriving asymptotically optimal threshold biases for arbitrary Maker-Breaker subgraph games. The asymptotic threshold bias depends on the 2-density $m_2(H)$ of the subgraph H Maker tries to build.

Theorem 2.3 (Bednarska and Łuczak, 2000) *For every fixed graph H with at least 2 edges, let*

$$m_2(H) := \max_{A \subseteq H, |V(A)|>2} \frac{|E(A)|-1}{|V(A)|-2}.$$

We call m_2 the 2-density of H. For the threshold bias q_H of the H-game it holds

$$b_H \in \Theta(n^{1/m_2(H)}).$$

The asymptotic result has been generalized by Kusch et al. [Kus+17] for k-uniform hypergraphs where the threshold bias depends on the k-density of the hypergraph $\mathcal{H}$ that Maker tries to build.

Theorem 2.4 (Kusch et al., 2017) *For every fixed k-uniform hypergraph $\mathcal{H}$ on at least 2 edges, let*

$$m_k(\mathcal{H}) := \max_{A \subseteq \mathcal{H}, |V(A)|>k} \frac{|E(A)|-1}{|V(A)|-k}$$

denote the k-density of the hypergraph. The threshold bias $b_{\mathcal{H}}$ of the $\mathcal{H}$-game satisfies $q_{\mathcal{H}} \in \Theta(n^{1/m_k(\mathcal{H})})$.

Definition 2.5 (K_n^k-Triangle Game) *Let K_n^k be the complete k-uniform hypergraph on which the game is played. We call a triplet of edges $\{e_1, e_2, e_3\}$ a (hypergraph) triangle if $|e_1 \cap e_2| = |e_1 \cap e_3| = |e_2 \cap e_3| = 1$ and $e_1 \cap e_2 \cap e_3 = \emptyset$. The K_n^k-Triangle Game is defined as the $(1:q)$-Maker-Breaker game where Maker wins if she claims all three edges of a triangle and Breaker wins if he can prevent Maker from doing so until the end of the game.*

Depending on k, Theorem 2.4 gives the following asymptotic threshold values $q_\triangle^k$ for the triangle game in k-uniform hypergraphs are shown in Table 2.1. Note that these techniques are non-constructive and do not give concrete constants.

k	2	3	4	5	$\cdots$
$q_\triangle^k$	$\Theta(n^{1/2})$	$\Theta(n^{3/2})$	$\Theta(n^{5/2})$	$\Theta(n^{7/2})$	$\cdots$

Table 2.1: Asymptotic threshold biases for the Triangle game in k-uniform hypergraphs

2.3 Upper and Lower Bounds for the Threshold Bias of the K_n^3-Triangle Game

We begin by adapting the techniques of Chvátal and Erdős to complete 3-uniform hypergraphs on which the game is played in order to find better constants for the threshold bias.

Theorem 2.6 *In the K_n^3 Maker-Breaker hypergraph triangle game, Maker has a winning strategy for $q < (2-o(1))n^{3/2}$.*

Proof. At the beginning, Maker selects an arbitrary vertex v and constructs a star centered in this vertex by choosing edges incident in v that are, except for v, pairwise disjoint. If, at some point in time, there is an edge e, where Maker's choice of e would yield a Maker-triangle, she chooses that edge and as a result, wins the game.

Breaker's defense against Maker's star building strategy has to fulfill two objectives: closing emerging Maker triangles by choosing edges that would allow Maker to close a triangle in her next turn, and preventing the Maker-star in v from getting too big by choosing edges incident in v.

Suppose Breaker, as the first player, can defend against Maker's star building strategy. Then

$$\binom{n-2d-1}{2} + \binom{d}{2} 4(n-5) \leq (d+1)q. \tag{2.3.1}$$

This can be proved as follows. The first summand describes the number of Breaker-edges needed to prevent the Maker-star from exceeding d edges. This can be seen as follows. In the 3-uniform complete hypergraph on which the game is played, each vertex has $\binom{n-1}{2}$ incident edges. From those, we exclude the edges containing the $2d$ vertices that are covered by Maker's d edges and v itself and get the first term. An example is shown in Figure 2.1.

The second term represents the total number of edges needed for Breaker to close every potential Maker-triangle within a Maker-star on d edges. For each pair of Maker's star edges, there are four ways of building a Maker-triangle using one of the $n-5$ remaining vertices not contained in this pair of edges as shown in Figure 2.2. There are $\binom{d}{2}$ pairs of Maker edges in a Maker-star of degree d.

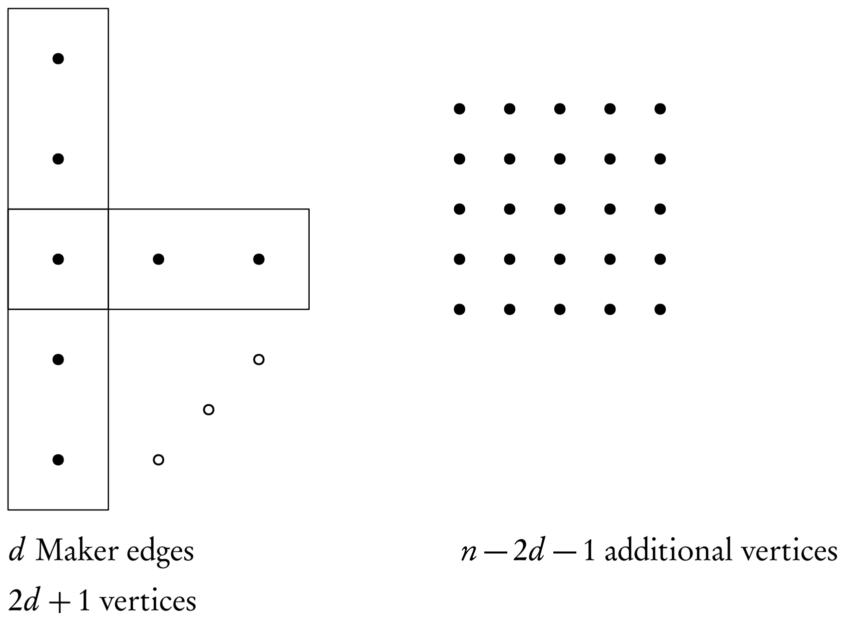


Figure 2.1: Maker builds a star around the vertex v using d edges on $2d-1$ distinct vertices.

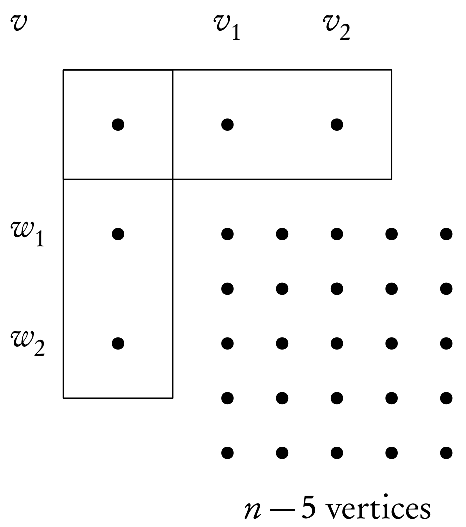


Figure 2.2: Two Maker-edges $\{v, v_1, v_2\}$ and $\{v, w_1, w_2\}$. For each vertex $u \in V \setminus \{v, v_1, v_2, w_1, w_2\}$, Breaker has to claim the edges $\{v_1, u, w_1\}$, $\{v_1, u, w_2\}$, $\{v_2, u, w_1\}$, and $\{v_2, u, w_2\}$.

The right hand side of the inequality gives the total number of edges Breaker picks in $d+1$ rounds. If Breaker wins, the inequality must have a solution for d. Equivalently, if, for some q, no such solution exists, then Maker wins the game by following her strategy described above. We get equality for (2.3.1) solving a quadratic equation for

$$\begin{aligned} d = d^\star &:= \frac{1}{2}\left(\sqrt{\frac{n^2+9n-40}{n-4}} - 2\right) \text{ and} \\ q = q^\star &:= 2\sqrt{n^3+5n^2-76n+160} - 8n + 29. \end{aligned}$$

Note that Maker-Breaker games are bias monotone. We now show that for q smaller than $q^\star$, Breaker cannot defend against Maker's star building strategy. Assume for a moment that $q^\star - 1$ is a sufficient bias for Breaker's win against a star building Maker. The decrease in bias changes only the right hand side of (2.3.1) by an additive term of $-(d+1)$. If (2.3.1) still holds, then d needs to increase. Because d is integral, d increases by at least one. To compute the increase on the left hand side, we consider the summands individually:

Let $\nu := n-2d-1$, then for $n-2(d+1)-1 = \nu-2$ we get

$$\begin{aligned} \binom{n-2(d+1)-1}{2} - \binom{n-2d-1}{2} &= \binom{\nu-2}{2} - \binom{\nu}{2} \\ &= \frac{(\nu-2)(\nu-3)}{2} - \frac{\nu(\nu-1)}{2} \\ &= \frac{\nu^2-3\nu-2\nu+6}{2} - \frac{\nu^2-\nu}{2} \\ &= -2\nu+3 \overset{\text{def}}{=} -2n+4d+5 \end{aligned} \tag{2.3.2}$$

For the second term, we get

$$\begin{aligned} \binom{d+1}{2}4(n-5) - \binom{d}{2}4(n-5) &= \left(\binom{d+1}{2} - \binom{d}{2}\right)4(n-5) \\ &= \left(\frac{d(d+1)}{2} - \frac{d(d-1)}{2}\right)4(n-5) \\ &= \left(\frac{d^2+d-d^2+d}{2}\right)4(n-5) \\ &= 4d(n-5) \end{aligned} \tag{2.3.3}$$

Adding (2.3.2) and (2.3.3) yields $-2n+4d+5+4d(n-5) = 4dn-2n-16d+5$. In order for (2.3.1) to still hold, we would need

$$4dn - 2n - 16d + 5 \leq q - d - 2$$

which does not hold for $n \geq 4$. Thus, the inequality has no solution for $q < q^\star$.

Therefore $q^\star$ is a lower bound for the threshold bias. □

For the upper bound, Breaker needs to prevent Maker from choosing edges between high degree vertices and thereby creating many possible triangles. The next proposition gives an upper bound on the number of Breaker edges q to achieve this goal.

Proposition 2.7 *If the number of Breaker edges q satisfies*

$$4(n-5)(\deg_M(u) + \deg_M(v) + \deg_M(w)) \leq q \text{ for all edges } \{u, v, w\} \in \mathscr{E}, \tag{2.3.4}$$

then Maker cannot build a triangle in the next round.

Proof. To this end, we call $\deg_M(v) \geq \frac{q}{12(n-5)}$ Breaker's *dangerous event* which he tries to avoid for every vertex v. Just as in Figure 2.2, there are $4(n-5)$ ways to build a triangle between two edges that intersect only in v. Maker may claim an edge $\{u, v, w\}$ where for each vertex $x \in \{u, v, w\}$, there are $4(n-5)\deg_M(x)$ ways to close a triangle with $\{u, v, w\} \setminus \{x\}$.

If no vertex of such high degree exists, (2.3.4) holds, as a simple calculation shows. Setting $\deg_M(u) = \deg_M(v) = \deg_M(w) = \frac{q}{12(n-5)} - 1$ for an arbitrary edge $\{u, v, w\}$, we get

$$\begin{aligned} 4(n-5)(\deg_M(u) + \deg_M(v) + \deg_M(w)) &= 4(n-5)\left(3 \cdot \left(\frac{q}{12(n-5)} - 1\right)\right) \\ &= 12(n-5)\left(\frac{q}{12(n-5)} - 1\right) < q. \end{aligned}$$

Because Breaker can claim q edges, he can prevent Maker from closing a triangle. □

Theorem 2.8 *In the K_n^3 Maker-Breaker hypergraph triangle game, Breaker has a winning strategy for $q > (3\sqrt{2})n^{3/2}$.*

Proof. Breaker needs to ensure that

$$3 \cdot \frac{\binom{n-1}{2}}{q} < \frac{q}{12(n-5)} \tag{2.3.5}$$

at all times. This can be seen as follows: For each of the three vertices of Maker's previously chosen edge, Breaker claims a portion q of the total edges incident in that vertex such that there are no dangerous events. Now, we can solve (2.3.5) for q:

$$\begin{aligned} & 3 \cdot \frac{\binom{n-1}{2}}{q} < \frac{q}{12(n-5)} \\ \Leftrightarrow \quad & 3 \cdot \binom{n-1}{2} < \frac{q^2}{12(n-5)} \\ \Leftrightarrow \quad & 3 \cdot 12(n-5)\binom{n-1}{2} < q^2 \\ \Leftrightarrow \quad & 18(n-5)(n-2)(n-1) < q^2 \\ \Leftrightarrow \quad & \sqrt{18(n-5)(n-2)(n-1)} < q, \end{aligned}$$

thus for $n > 5$, Breaker has a winning strategy for $q > 3\sqrt{2}n^{3/2}$. □

2.4 A New Breaker Strategy for the K_n^3-Triangle Game

Definition 2.9 (Balance function) *The balance of a vertex reflects how close Maker is to the* dangerous event *of having a vertex v with* $\deg_M(v) \geq \frac{q(1-\delta)}{12(n-5)}$ *for some $\delta > 0$ to be fixed later. We define the balance of v by*

$$\operatorname{bal}(v) := \frac{\binom{n-1}{2} - \deg_B(v)}{q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_M(v)\right) - 4(n-5)\sum_{i=\deg_M(v)}^{q(1-\delta)/12(n-5)} i}.$$

A vertex v has higher balance value if its Maker-degree is closer to the dangerous event that $\deg_M(v) \geq \frac{q}{12(n-5)}$. Let us briefly explain the terms in the balance function. $\binom{n-1}{2}$ is the total number of edges incident in a vertex in the complete 3-uniform hypergraph, so $\binom{n-1}{2} - \deg_B(w)$ is the number of edges that do not belong to Breaker and the balance increases proportionally to this value. $\deg_M(v) = \frac{q(1-\delta)}{12(n-5)}$ is the desastrous event for Breaker, so the balance drastically increases as $\deg_M(v)$ tends to this value. Multiplication by q reflects that Breaker has q edges to compensate the threat. The sum represents an upper bound for additional edges Breaker needed to prevent possible Maker-triangles. For each round

i, Breaker needs to prevent Maker from building triangles using two edges incident in v.

Let $q = (\beta n)^{3/2}$ and let $\mathrm{bal}_0(n,\delta)$ denote the balance of each vertex at the beginning of the game. At this point, we have $\deg_M(v) = \deg_B(v) = 0$ for all vertices v. Thus,

$$\begin{aligned}
\mathrm{bal}_0(n,\delta) &= \frac{\binom{n-1}{2} - \deg_B(v)}{q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_M(v)\right) - 4(n-5)\sum_{i=\deg_M(v)}^{q(1-\delta)/12(n-5)} i} \\
&= \frac{\binom{n-1}{2}}{\frac{q\cdot q(1-\delta)}{12(n-5)} - 4(n-5)\sum_{i=0}^{q(1-\delta)/12(n-5)} i} \\
&= \frac{\binom{n-1}{2}}{\frac{q\cdot q(1-\delta)}{12(n-5)} - 2(n-5)\left(\left(\frac{q(1-\delta)}{12(n-5)}\right)^2 + \frac{q(1-\delta)}{12(n-5)}\right)} \\
&= \frac{\binom{n-1}{2}}{\frac{q\cdot q(1-\delta)}{12(n-5)} - \frac{q^2(1-\delta)^2}{72(n-5)} - \frac{q(1-\delta)}{6}} \\
&= \frac{\binom{n-1}{2}}{q(1-\delta)\left(\frac{q}{12(n-5)} - \frac{q(1-\delta)}{72(n-5)} - \frac{1}{6}\right)} \\
&= \frac{6\cdot\binom{n-1}{2}}{q(1-\delta)\left(\frac{q}{2(n-5)} - \frac{q(1-\delta)}{12(n-5)} - 1\right)} \\
&= \frac{3(n-1)(n-2)\cdot 12(n-5)}{q(1-\delta)(6q - q(1-\delta) - 12(n-5))} \\
&= \frac{36(n-5)(n-2)(n-1)}{q(1-\delta)(q(5+\delta) - 12(n-5))}.
\end{aligned}$$

Proposition 2.10 $\mathrm{bal}_0(n,\delta) < \lim_{n\to\infty}\lim_{\delta\to 0}\mathrm{bal}_0(n,\delta) = \frac{36}{5\beta^3} = 1$ *for* $\beta = \sqrt[3]{\frac{36}{5}}$.

Proof. Setting $\delta = 0$, we get

$$\begin{aligned}
\mathrm{bal}_0(n,0) &= \frac{36(n-5)(n-2)(n-1)}{q(1-0)(q(5+0) - 12(n-5))} \\
&= \frac{36(n-5)(n-2)(n-1)}{5q^2 - q(12(n-5))} \qquad (q = (\beta n)^{3/2}) \\
&= \frac{36(n-5)(n-2)(n-1)}{5\beta^3 n^3 - \beta^{3/2} n^{3/2}(12(n-5))} \\
&= \frac{36\cdot\mathcal{O}(n^3)}{5\beta^3\cdot\mathcal{O}(n^3) - o(n^3)} \xrightarrow{n\to\infty} \frac{36}{5\beta^3}.
\end{aligned}$$

Setting $\frac{36}{5\beta^3} = 1$ gives $\beta^3 = \frac{36}{5}$ and thus $\beta = \sqrt[3]{\frac{36}{5}}$. $\square$

Because for fixed n and δ, $\mathrm{bal}_0(n,\delta)$ is constant, we write bal_0 wherever it is convenient.

Definition 2.11 (Balanced Breaker-degree) *The balanced Breaker-degree of a vertex, denoted by* $\deg^*(v)$, *is the number of Breaker-edges incident in v, so that* $\mathrm{bal}(v) = \mathrm{bal}_0$ *at an arbitrary but fixed round.*

Proposition 2.10 defines balanced Breaker-degree by the following equation:

$$\frac{\binom{n-1}{2} - \deg^*(v)}{q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_M(v)\right) - 4(n-5)\sum_{i=\deg_M(v)}^{q(1-\delta)/12(n-5)} i} \overset{!}{=} \mathrm{bal}_0 .$$

Thus

$$\mathrm{bal}_0\left(q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_M(v)\right) - 4(n-5)\sum_{i=\deg_M(v)}^{\frac{q(1-\delta)}{12(n-5)}} i\right) = \binom{n-1}{2} - \deg^*(v)$$

and

$$\deg^*(v) = \binom{n-1}{2} - \mathrm{bal}_0\left(q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_M(v)\right) - 4(n-5)\sum_{i=\deg_M(v)}^{\frac{q(1-\delta)}{12(n-5)}} i\right). \tag{2.4.1}$$

We define the deficit $d(v)$ as the number of edges Breaker is missing to achieve a balance value of bal_0 at vertex v:

$$d(v) := \deg^*(v) - \deg_B(v).$$

We proceed to the definition of the potential function of the game.

Definition 2.12 (Potential Function) *We assign a potential* $\mathrm{pot}(v)$ *to each vertex as follows:*

$$\mathrm{pot}(v) := \begin{cases} 0 & \textit{if } \deg_M(v) + \deg_B(v) = \binom{n-1}{2} \\ \mu^{d(v)/q} & \textit{otherwise.} \end{cases}$$

where $\mu := 1 + \frac{10\beta^{3/2}\log(n)}{\delta\sqrt{n}}$.

The potential of an edge e is defined as the sum of the potentials of its vertices, $\mathrm{pot}(e) = \sum_{v\in e}\mathrm{pot}(v)$.

The intuitive explanation is the following. High potential means high danger for Breaker to lose the game. If $\deg_M(v)+\deg_B(v) = \binom{n-1}{2}$, then all edges incident in v have been claimed by Maker or Breaker, thus v is not part of the game anymore, and naturally $\mathrm{pot}(v) = 0$. Otherwise, and this is the essential idea of Definition 2.12, the potential may increase exponentially in μ. The total potential of all vertices is defined by $\mathrm{POT}_t := \sum_{v\in V}\mathrm{pot}_t(v)$.

Remark 2.13 *Let v be an arbitrary vertex. At the beginning of the game, it holds $\deg^*(v) = \deg_B(v) = 0$, because v has a balance of bal_0 and Breaker does not need to add any Breaker edges incident in v to reduce its balance. So $d(v) = \deg^*(v) - \deg_B(v) = 0$. Therefore, $\mathrm{pot}_0(v) = \mu^0 = 1$.*

For the overall potential at the beginning of the game, we have

$$\mathrm{POT}_0 := \sum_{v \in V} \mathrm{pot}_0(v) = n.$$

Lemma 2.14 *Let n sufficiently large. Then, for every round t and every vertex v, if $\deg_{M,t}(v) = \frac{q}{12(n-5)} - 1$, then $\mathrm{pot}_t(v) > 2n$.*

Proof. First, we estimate $d_t(v)$ by plugging $\deg_{M,t}(v) = \frac{q}{12(n-5)} - 1$ into the definition of $\deg^*_t(v)$. Note that bal_0 is the constant $\frac{36}{5\beta^3}$ by Proposition 2.10. Now by (2.4.1),

$$\begin{aligned}
\deg^*_t(v) &= \binom{n-1}{2} - \mathrm{bal}_0 \left(q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_{M,t}(v) \right) - 4(n-5) \sum_{i=\deg_{M,t}(v)}^{\frac{q(1-\delta)}{12(n-5)}} i \right) \\
&= \binom{n-1}{2} - \mathrm{bal}_0 \left(q\left(\frac{q(1-\delta)-q}{12(n-5)} + 1 \right) \right) + \mathrm{bal}_0\, 4(n-5) \underbrace{\sum_{\frac{q}{12(n-5)}-1}^{\frac{q(1-\delta)}{12(n-5)}-1} i}_{=0} \\
&= \binom{n-1}{2} - \mathrm{bal}_0 \left(\frac{q^2(1-\delta)-q^2}{12(n-5)} \right) - \mathrm{bal}_0\, q \\
&= \binom{n-1}{2} + \mathrm{bal}_0 \left(\frac{q^2\delta}{12(n-5)} \right) - \mathrm{bal}_0\, q \\
&= \binom{n-1}{2} + \mathrm{bal}_0 \left(\frac{\beta^3 n^3 \delta}{12(n-5)} \right) - \mathrm{bal}_0\, q \qquad\qquad (q = (\beta n)^{3/2}) \\
&= \binom{n-1}{2} + \frac{36}{5\beta^3} \cdot \frac{\beta^3 n^3 \delta}{12(n-5)} - \mathrm{bal}_0\, q \\
&= \binom{n-1}{2} + \frac{3}{5} \cdot \delta \cdot \frac{n^3}{(n-5)} - \mathrm{bal}_0\, q \\
&\geq \binom{n-1}{2} + \frac{3\delta n^2}{5} - \mathrm{bal}_0\, q \\
&\geq \binom{n-1}{2} + \frac{2\delta n^2}{5},
\end{aligned}$$

and the last inequality is true because $\mathrm{bal}_0\, q < \delta n^2/5$ for suffiently large n.

Since trivially, $\deg_{B,t}(v) \leq \binom{n-1}{2}$, we have

$$d_t(v) = \deg^*_t(v) - \deg_{B,t}(v) \geq \binom{n-1}{2} + \frac{2\delta n^2}{5} - \binom{n-1}{2} = \frac{2\delta n^2}{5}.$$

Thus

$$\begin{aligned}
\mathrm{pot}_t(v) = \mu^{d_t(v)/q} &\geq \mu^{\frac{2\delta n^2}{5q}} \\
&= \left(1 + \frac{10\beta^{3/2}\log(n)}{\delta\sqrt{n}}\right)^{\frac{2\delta n^2}{5q}} \\
&= \left(1 + \frac{10\beta^{3/2}\log(n)}{\delta\sqrt{n}}\right)^{\left(1+\frac{\delta\sqrt{n}}{10\beta^{3/2}\log(n)}\right)\left(1+\frac{\delta\sqrt{n}}{10\beta^{3/2}\log(n)}\right)^{-1}\frac{2\delta n^2}{5q}}.
\end{aligned}$$

We use the inequality $f(x) = (1+x)^{1+1/x} > e$ which holds for all $x \in \mathbb{R}_{>0}$ since $\lim_{x\to 0}(1+x)^{1+1/x} = \lim_{x\to\infty}(1+1/x)^x = e$ and the function is monotonically increasing in $\mathbb{R}_{>0}$. So

$$\mathrm{pot}_t(v) \geq \underbrace{\left(\left(1 + \frac{10\beta^{3/2}\log(n)}{\delta\sqrt{n}}\right)^{\left(1+\frac{\delta\sqrt{n}}{10\beta^{3/2}\log(n)}\right)}\right)}_{\geq e}^{\left(1+\frac{\delta\sqrt{n}}{10\beta^{3/2}\log(n)}\right)^{-1}\frac{2\delta n^2}{5q}}.$$

For the exponent, we get with $q = (\beta n)^{3/2}$

$$\begin{aligned}
\frac{2\delta n^2}{5q}\left(1 + \frac{\delta\sqrt{n}}{10\beta^{3/2}\log(n)}\right)^{-1} &= \frac{2\delta n^2}{5q}\left(\frac{\delta\sqrt{n}\mu}{10\beta^{3/2}\log(n)}\right)^{-1} \\
&= \frac{20\,\beta^{3/2}\,\delta\,n^2\,\log(n)}{5\,\beta^{3/2}\,\delta\,n^{3/2}\,n^{1/2}\,\mu} = \frac{4\log(n)}{\mu}.
\end{aligned}$$

Because $\mu < 2$ for sufficiently large n, we get $\frac{4\log(n)}{\mu} > 2\log(n)$ and thus,

$$\mathrm{pot}_t(v) \geq e^{\frac{4\log(n)}{\mu}} > n^2 > 2n.$$

□

2.4.1 Breaker's Strategy

Breaker's strategy has two objectives described in the following:

I. **Closing Paths**

First, Breaker needs to claim every edge that would otherwise complete a Maker-triangle. Let $e_M = \{v_1, v_2, v_3\}$ and $e'_M = \{v'_1, v'_2, v'_3\}$ be Maker-edges with $|e_M \cap e'_M| = 1$, say $v_1 = v'_1$. Then, Breaker has to ensure that every possible edge between vertices of $\{v_2, v_3\}$ and $\{v'_2, v'_3\}$ is claimed because otherwise Maker can choose such an edge that would win her the game. There are four possible ways to choose pairs of vertices with one from each set. Since the hypergraph is 3-uniform, one more vertex is needed to construct an edge. The number of ways to choose that additional vertex is $n - |e_M \cup e'_M| = n - 5$. Overall, Breaker has to claim $4(n-5)$ edges between any pair of Maker-edges. If there are less than $4(n-5)$ such edges left, Breaker claims arbitrary unclaimed edges. This part of Breaker's strategy is mandatory and part of every possible Breaker-strategy.

II. **Free Edges**

If Breaker still has edges left to claim after the first part of his strategy, he continues by prioritizing free edges that have the highest potential. Where *free edges* are those edges that have not been claimed by either player. A high potential reflects a high danger of its vertex to become part of a triangle so it is intuitive for Breaker to claim edges incident in these vertices first. We define f_t as the number of free edges in round t by

$$f_t = q - 4(n-5)(\deg_{M,t-1}(v_1) + \deg_{M,t-1}(v_2) + \deg_{M,t-1}(v_3)), \tag{2.4.2}$$

where $\{v_1, v_2, v_3\}$ is the edge Maker chose in round t. The number of free edges is simply the number of edges Breaker can claim in a round given by the bias q minus the number of edges needed for the first part of Breaker's strategy.

We now show that there is always a minimum number of free edges, if the overall potential remains less than $2n$.

Observation 2.15 *For every round t where $f_t < 12(n-5)$, there is a previous round $t' < t$ in which* $\mathrm{POT}_{t'} > 2n$.

Proof. We show that there exists a vertex v with $\deg_M(v) \geq \frac{q}{12(n-5)}$. Lemma 2.14 then gives us $\mathrm{pot}_{t'}(v) > 2n$ and since the potential is always non-negative, $\mathrm{POT}_{t'} > 2n$. Let t such that $f_t \leq 12(n-5) - 1$ and let $e_M = \{v_1, v_2, v_3\}$ be the edge Maker chooses in that round. We have

$$f_t \overset{(2.4.2)}{=} q - 4(n-5)\big(\deg_{M,t-1}(v_1) + \deg_{M,t-1}(v_2) + \deg_{M,t-1}(v_3)\big).$$

Now, assume for a moment that $\deg_{M,t-1}(v) \leq \frac{q}{12(n-5)} - 1$ for all $v \in e_M$. Then,

$$\begin{aligned} f_t &= q - 4(n-5) \sum_{v \in e_m} \left(\deg_{M,t-1}(v)\right) \\ &\geq q - 4 \cdot 3(n-5) \left(\frac{q}{12(n-5)} - 1 \right) \\ &= 12(n-5), \end{aligned}$$

which contradicts $f_t \leq 12(n-5)-1$. Therefore, $\deg_{M,t-1}(v) \geq \frac{q}{12(n-5)}$ for at least one $v \in \{v_1, v_2, v_3\}$. Since $\deg_M(v)$ increases by at most one in each round, at some point $t' < t$, we have $\deg_{M,t'}(v) = \frac{q}{12(n-5)}$. □

Theorem 2.16 $\mathrm{POT}_s \leq 2n$ *holds in every round* s.

This theorem is the backbone of our work and will be proved at the very end of this section. With Theorem 2.16 we can immediatly show that Breaker wins the game.

Theorem 2.17 *At the end of the game, no vertex has Maker-degree at least* $\frac{q}{12(n-5)}$ *and thus, Breaker wins.*

Proof. Assume for a moment that there is a vertex v with $\deg_M(v) \geq \frac{q}{12(n-5)}$ at the end of the game. Let t denote the round in which Maker claimed her $\frac{q}{12(n-5)}$-th v-edge. Thus, in round $t-1$ we have $\deg_{M,t-1}(v) = \frac{q}{12(n-5)} - 1$. By the previous theorem we get

$$\mathrm{pot}_{t-1}(v) \leq \mathrm{POT}_{t-1} \leq \mu(1+\varepsilon)n < 2n.$$

By the contraposition of Lemma 2.14 we get $\deg_{M,t-1}(v) \neq \frac{q}{12(n-5)} - 1$ which contradicts the assumption. □

Lemma 2.18 *Let* $\deg^{*\prime}(u), \deg'_M(u)$*, and* $d'(u)$ *be the balanced Breaker-degree, Maker-degree, and deficit of a vertex* u *right after a Maker-edge incident in* u *has been chosen and before Breaker's turn. Then*

$$d'(u) - d(u) = \mathrm{bal}_0(q - 4(n-5)\deg_M(u)).$$

Proof. Since by definition $d(u) = \deg^*(u) - \deg_B(u)$ and adding a Maker-edge only affects $\deg^*(u)$, we have $d'(u) - d(u) = \deg^{*\prime}(u) - \deg^*(u)$. By the assumption of the lemma, $\deg'_M(u) = \deg_M(u) + 1$. Now by (2.4.1),

$$\begin{aligned}
&\deg^{*\prime}(u) - \deg^*(u) \\
&= \binom{n-1}{2} - \mathrm{bal}_0\left(q\left(\frac{q(1-\delta)}{12(n-5)} - \deg'_M(u)\right) - 4(n-5)\sum_{i=\deg'_M(u)}^{\frac{q(1-\delta)}{12(n-5)}} i\right) \\
&= \binom{n-1}{2} - \mathrm{bal}_0\left(q\left(\frac{q(1-\delta)}{12(n-5)} - (\deg_M(u)+1)\right) - 4(n-5)\sum_{i=\deg_M(u)+1}^{\frac{q(1-\delta)}{12(n-5)}} i\right) \\
&\quad - \left(\binom{n-1}{2} - \mathrm{bal}_0\left(q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_M(u)\right) - 4(n-5)\sum_{i=\deg_M(u)}^{\frac{q(1-\delta)}{12(n-5)}} i\right)\right) \\
&= \mathrm{bal}_0\left(q\left(\frac{q(1-\delta)}{12(n-5)} - \deg_M(u)\right) - 4(n-5)\sum_{i=\deg_M(u)}^{\frac{q(1-\delta)}{12(n-5)}} i\right) \\
&\quad - \mathrm{bal}_0\left(q\left(\frac{q(1-\delta)}{12(n-5)} - (\deg_M(u)+1)\right) - 4(n-5)\sum_{i=\deg_M(u)+1}^{\frac{q(1-\delta)}{12(n-5)}} i\right) \\
&= \mathrm{bal}_0\left(q - 4(n-5)\left(\sum_{i=\deg_M(u)}^{\frac{q(1-\delta)}{12(n-5)}} i - \sum_{i=\deg_M(u)+1}^{\frac{q(1-\delta)}{12(n-5)}} i\right)\right) \\
&= \mathrm{bal}_0(q - 4(n-5)\deg_M(u)).
\end{aligned}$$

□

Because $\mathrm{bal}_0 < 1$, we have for all $v \in V$

$$\mathrm{bal}_0(q - 4(n-5)\deg_M(v)) \leq q. \tag{2.4.3}$$

Next, we consider the change in the potential caused by one additional Maker-edge and by one additional Breaker-edge, respectively.

Lemma 2.19 *Let e_M and e_B be previously unclaimed Maker- and Breaker-edges, respectively. Let* $\mathrm{pot}(e)$ *denote the potential of e's vertices before e is claimed. Let* $\mathrm{pot}'(e)$ *denote e's potential after Maker's turn.*

(i) Choosing e_M increases the potential of its vertices by at most a factor of μ. The overall potential increases by at most $(\mu-1)\,\mathrm{pot}(e_M)$.

(ii) Choosing e_B decreases the overall potential by at least $(1-\mu^{-1/q})\,\mathrm{pot}(e_B)$.

Proof. *(i)* Let $d'(u)$ be defined as in the previous lemma and let $v \in e_M$. Choosing e_M increases v's Maker-degree by one. The potentials of all vertices not in e_M remain unchanged. By applying Lemma 2.18 to $\mathrm{pot}'(v)$, which denotes v's potential just after Maker adds e_M, we get

$$\begin{aligned}
\mathrm{pot}'(v) = \mu^{d'(v)/q} &= \mu^{d(v)/q} \cdot \frac{\mu^{d'(v)/q}}{\mu^{d(v)/q}} \\
&= \mathrm{pot}(v) \cdot \mu^{(d'(v)-d(v))/q} \\
&= \mathrm{pot}(v) \cdot \mu^{\mathrm{bal}_0(q-4(n-5)\deg_M(v))/q} && \text{(Lemma 2.18)} \\
&< \mathrm{pot}(v) \cdot \mu, && \text{(using (2.4.3))} \qquad (2.4.4)
\end{aligned}$$

For the overall potential, we have with (2.4.4)

$$\begin{aligned}
\mathrm{POT}' &= \sum_{v \in V} \mathrm{pot}'(v) = \sum_{v \notin e_M} \mathrm{pot}'(v) + \sum_{v \in e_M} \mathrm{pot}'(v) \\
&< \sum_{v \notin e_M} \mathrm{pot}(v) + \sum_{v \in e_M} \mathrm{pot}(v) \cdot \mu && \text{(by (2.4.3))} \\
&= \sum_{v \in V} \mathrm{pot}(v) - \sum_{v \in e_M} \mathrm{pot}(v) + \sum_{v \in e_M} \mathrm{pot}(v) \cdot \mu \\
&= \sum_{v \in V} \mathrm{pot}(v) - (\mu - 1) \sum_{v \in e_M} \mathrm{pot}(v) \\
&= \mathrm{POT} + (\mu - 1)\mathrm{pot}(e_M).
\end{aligned}$$

(ii) Let $v \in e_B$. Again, only the potentials of vertices in e_B are affected. Let $\mathrm{pot}''(v)$ and $d''(v)$ denote v's potential and its deficit after Breaker chooses e_B. Since the deficit is reduced by one and thus, $\mathrm{pot}''(v) = \mu^{d''(v)/q} = \mu^{d(v)-1/q}$, we have

$$\frac{\mathrm{pot}''(v)}{\mathrm{pot}(v)} = \frac{\mu^{d(v)-1/q}}{\mu^{d(v)/q}} = \mu^{-1/q}. \qquad (2.4.5)$$

Now, for the overall potential after e_B is added, we get with (2.4.5)

$$\begin{aligned}
\mathrm{POT}'' &= \sum_{v \in V} \mathrm{pot}''(v) \\
&= \sum_{v \notin e_B} \mathrm{pot}''(v) + \sum_{v \in e_B} \mathrm{pot}''(v) \\
&= \sum_{v \notin e_B} \mathrm{pot}(v) + \sum_{v \in e_B} \mathrm{pot}''(v)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{v \notin e_B} \mathrm{pot}(v) + \sum_{v \in e_B} \mathrm{pot}(v) \cdot \mu^{-1/q} \\
&= \sum_{v \in V} \mathrm{pot}(v) - \sum_{v \in e_B} \mathrm{pot}(v) + \sum_{v \in e_B} \mathrm{pot}(v) \cdot \mu^{-1/q} \\
&= \mathrm{POT} - (1 - \mu^{-1/q}) \mathrm{pot}(e_B).
\end{aligned}
$$

□

Each round t begins with a turn by Maker in which she chooses a single edge. For an arbitrary vertex w, let $I_t(w)$ denote the increase in potential caused by adding her edge. Note that only the potentials of vertices in this edge are affected. After Maker claimed her edge, Breaker claims q edges in his turn. We define $D_t(w)$ as the decrease in w's potential caused by the addition of a Breaker-edge. We write

$$
I_t := \sum_{v \in V} I_t(v) \quad \text{and} \quad D_t := \sum_{v \in V} D_t(v)
$$

for the increase/decrease in the overall potential. Head and tail vertices are defined as follows: suppose we are observing the potential of a vertex v_1 and Breaker claims a closing edge $e = \{v_1, v_2, v_3\}$ in part I. of his strategy in some round t, then we call v_1 the edge's *head vertex* and v_2, v_3 its *tail vertices.* Let $D_t^{\top}(w)$, resp. $D_t^{\perp}(w)$ be the decrease in potential, where w is a closing edge's head vertex and when w is one of its tail vertices, respectively. Because e is a closing edge, each of its vertices is by definition either a head vertex or a tail vertex and for each vertex $w \in e$, only one of the values, either $D_t^{\top}(w)$ or $D_t^{\perp}(w)$, is non-zero, depending on the type of the respective vertex.

If, on the other hand, e is a free edge claimed in part II., then for $w \in e$, the decrease of w's potential that Breaker achieves by adding e is denoted by $D_t^{\mathrm{f}}(w)$.

Recall that by the definition of the potential function (Definition 2.12), if no unclaimed edges remain in some vertex w, then its potential is set to 0. Because of this special case, we have the additional term $D_t^{0}(w)$ that is non-zero if and only if e was the last unclaimed edge incident in w. The overall decrease of the potential of a vertex w in round t can be written as

$$
D_t(w) = D_t^{\top}(w) + D_t^{\perp}(w) + D_t^{\mathrm{f}}(w) + D_t^{0}(w), \tag{2.4.6}
$$

We can describe the change of the potential between two rounds by the increase and decrease of the potential as follows

$$\begin{aligned}\mathrm{pot}_t(w)-\mathrm{pot}_{t-1}(w) &= I_t(w)-D_t(w)\\ &= I_t(w)-\left(D_t^{\top}(w)+D_t^{\perp}(w)+D_t^{\mathrm{f}}(w)+D_t^{0}(w)\right),\end{aligned} \tag{2.4.7}$$

and for the overall potential

$$\mathrm{POT}_t-\mathrm{POT}_{t-1}=I_t-D_t=I_t-\left(D_t^{\top}+D_t^{\perp}+D_t^{\mathrm{f}}+D_t^{0}\right). \tag{2.4.8}$$

Let $D_t^-:=\min\left\{I_t(w),D_t^{\top}(w)\right\}$ and $D_t^+:=\max\left\{D_t^{\top}(w)-I_t(w),0\right\}$. Then

$$D_t^{\top}(w)=D_t^-(w)+D_t^+(w).$$

From the definition of D_t^-, it follows that

$$D_t^-(v)\le I_t(v). \tag{2.4.9}$$

Lemma 2.20 *Suppose Maker chooses e_M with $v\in e_M$ the head vertex of e_M in round t. Then we have*

(i) $I_t(v)-D_t^-(v)\le(\mu^{\mathrm{bal}_0 f_t/q}-1)\mathrm{pot}_{t-1}(v)$ *and*

(ii) $I_t-D_t^-\le(\mu^{\mathrm{bal}_0 f_t/q}-1)\mathrm{pot}_{t-1}(e_M)$.

Proof. *(i)* We can assume that $D_t^-(v)=D_t^{\top}(v)$, because otherwise $D_t^-(v)=I_t(v)$ and the assertion *(i)* trivially holds. Let $d_t^{(1)}(v),\deg_t^{*(1)},\deg_{B,t}^{(1)}(v),\mathrm{pot}_t^{(1)}(v)$ denote the values of $d_t(v),\deg_t,\deg_{B,t}(v)$, and $\mathrm{pot}_t^{(1)}(v)$ after Breaker executes part I. of his strategy.

Furthermore, since v is the head vertex of e_M, $D_t^{\perp}(v)=0$, because v is not a tail vertex of e_M.

Recall that $D_t^{\top}(v)+D_t^{\perp}(v)$ is the decrease in v of the potential caused by closing edges in part I. of Breaker's strategy and $\mathrm{pot}_t^{(1)}(v)$ describes v's potential at that time as defined in the proof of Lemma 2.20. Maker's turn increases v's potential by $I_t(v)$. Hence, we can describe the change in v's potential caused by Maker's edge and Breaker's first part of his strategy by

$$\mathrm{pot}_t^{(1)}(v)-\mathrm{pot}_{t-1}(v)=I_t(v)-\left(D_t^{\top}(v)+D_t^{\perp}(v)\right)$$

$$\begin{aligned}
&= I_t(v) - \left(D_t^-(v) + \underbrace{D_t^\perp(v)}_{=0 \text{ because } v \text{ is head vertex}} \right) \\
&= I_t(v) - D_t^-(v). && (2.4.10)
\end{aligned}$$

We have

$$\begin{aligned}
d_t^{(1)}(v) - d_{t-1}(v) &= \deg_t^{*(1)}(v) - \deg_{B,t}^{(1)}(v) - \deg_{t-1}^*(v) + \deg_{B,t-1}(v) \\
&= (\deg_t^{*(1)}(v) - \deg_{t-1}^*(v)) - (\deg_{B,t}^{(1)}(v) - \deg_{B,t-1}(v)) && (2.4.11)
\end{aligned}$$

Since $\deg^*(v)$ is not affected by v's Breaker-degree, $\deg_t^{*(1)}(v) = \deg_t^*(v)$ and $\deg_{B,t}^{(1)}(v) = \deg_{B,t}(v)$, with Lemma 2.18 we get

$$\begin{aligned}
\deg_t^{*(1)}(v) - \deg_{t-1}^*(v) &= \deg_t^*(v) - \deg_{t-1}^*(v) \\
&= \mathrm{bal}_0(q - 4(n-5)\deg_{M,t-1}(v)). && \text{(by Lemma 2.18) } (2.4.12)
\end{aligned}$$

The remaining term $\deg_{B,t}^{(1)}(v) - \deg_{B,t-1}(v)$ from (2.4.11) is the number of closing edges incident in v_1 that are claimed by Breaker in round t using the argument depicted in Figure 2.2, we count the number of closing edges between $\{v_1, v_2, v_3\}$ and edges are incident only in v_2. There are $4(n-5)\deg_{M,t-1}(v_2)$ such edges. The analouge argument holds for v_3. Overall, we have

$$\begin{aligned}
&4(n-5)\deg_{M,t-1}(v_2) + 4(n-5)\deg_{M,t-1}(v_3) \\
=&4(n-5)(\deg_{M,t-1}(v_2) - \deg_{M,t-1}(v_3)). && (2.4.13)
\end{aligned}$$

We have

$$\begin{aligned}
d_t^{(1)}(v) - d_{t-1}(v) &= \deg_t^{*(1)}(v) - \deg_{B,t}^{(1)}(v) && \text{(by (2.4.11))} \\
&\quad - \deg_{B,t}^{(1)}(v) - \deg_{B,t-1}(v) \\
&= \mathrm{bal}_0(q - 4(n-5)\deg_{M,t-1}(v)) && \text{(by (2.4.12))} \\
&\quad - \deg_{B,t}^{(1)}(v) - \deg_{B,t-1}(v) \\
&= \mathrm{bal}_0(q - 4(n-5)\deg_{M,t-1}(v)) \\
&\quad - 4(n-5)(\deg_{M,t-1}(v_2) - \deg_{M,t-1}(v_3)) && \text{(by (2.4.13))} \\
&\le \mathrm{bal}_0(q - 4(n-5)\deg_{M,t-1}(v))
\end{aligned}$$

$$\begin{aligned} &-\mathrm{bal}_0\, 4(n-5)(\deg_{M,t-1}(v_2)+\deg_{M,t-1}(v_3)) && (\mathrm{bal}_0<1)\\ &=\mathrm{bal}_0 f_t. && \text{(by (2.4.2))} \quad (2.4.14)\end{aligned}$$

Further we have

$$\begin{aligned} I_t(v)-D_t^-(v) &= \mathrm{pot}_t^{(1)}(v)-\mathrm{pot}_{t-1}(v) && \text{(by (2.4.10))}\\ &= \mu^{d_t^{(1)}(v)/q}-\mathrm{pot}_{t-1}(v)\\ &= \left(\frac{\mu^{d_t^{(1)}(v)/q}}{\mu^{d_{t-1}(v)/q}}-1\right)\mathrm{pot}_{t-1}(v) && (\text{since}\, \mathrm{pot}_{t-1}(v)=\mu^{d_{t-1}(v)/q})\\ &= \left(\mu^{(d_t^{(1)}(v)-d_{t-1}(v))/q}-1\right)\mathrm{pot}_{t-1}(v)\\ &\le (\mu^{\mathrm{bal}_0 f_t/q}-1)\,\mathrm{pot}_{t-1}(v). && \text{(with (2.4.14))}\end{aligned}$$

(ii) By definition, $I_t=\sum_{v\in e_M} I_t(v)$ and $D_t^-=\sum_{v\in e_M} D_t^-(v)$, so

$$\begin{aligned} I_t-D_t^- &= \sum_{v\in e_M} I_t(v)-\sum_{v\in e_M} D_t^-(v)\\ &= \sum_{v\in e_M} I_t(v)-D_t^-(v)\\ &\overset{(i)}{\le} \sum_{v\in e_M}(\mu^{\mathrm{bal}_0 f_t/q}-1)\,\mathrm{pot}_{t-1}(v)\\ &= (\mu^{\mathrm{bal}_0 f_t/q}-1)\,\mathrm{pot}_{t-1}(e_M).\end{aligned}$$

□

2.4.2 Controlling Critical Rounds

Since $\mu=1+o(1)$ and $\mathrm{bal}_0<1$, we get $\mu\,\mathrm{bal}_0<1$ for sufficiently large n. Fix $\eta\in(0,1-\mu\,\mathrm{bal}_0)$. We now split the change in the potential into two parts that will be considered individually.

Definition 2.21 (Critical Round) *For every round t, let*

$$\Delta_t := I_t-D_t^- -(1-\eta)D_t^{\mathrm{f}} \quad \textit{and} \quad r_t := D_t^+ + D_t^{\perp}+\eta D_t^{\mathrm{f}}+D_t^{0}.$$

We call t critical, if $\Delta_t>0$ and non-critical otherwise.

By the previous definition,

$$\Delta_t - r_t = I_t - D_t^- - (1-\eta)D_t^{\mathrm{f}} - \eta D_t^{\mathrm{f}} - D_t^+ - D_t^\perp - D_t^0 = \mathrm{POT}_t - \mathrm{POT}_{t-1}. \tag{2.4.15}$$

Because $r_t \geq 0$, every round t with $\mathrm{POT}_t > \mathrm{POT}_{t-1}$ is a critical one.

Lemma 2.22 *Let t be a critical round where $f_t \geq 12(n-5)$. Let e_M be the edge Maker chooses in this round, and let e be an arbitrary edge that is unclaimed after round t. Then for sufficiently large n,*

$$\mathrm{pot}_t(e) < \frac{\mu\,\mathrm{bal}_0}{(1-\eta)}\mathrm{pot}_{t-1}(e_M).$$

Proof. By Lemma 2.20 *(ii)*,

$$\begin{aligned} I_t - D_t^- &\leq (\mu^{\mathrm{bal}_0 f_t/q} - 1)\mathrm{pot}_{t-1}(e_M) \\ &= \mu^{\mathrm{bal}_0 f_t/q}(1-\mu^{-\mathrm{bal}_0 f_t/q})\mathrm{pot}_{t-1}(e_M) \\ &\leq \mu\,\mathrm{bal}_0 f_t(1-\mu^{-1/q})\mathrm{pot}_{t-1}(e_M) \end{aligned} \tag{2.4.16}$$

The last inequality holds du to the inequality $x(1-\mu^{-1/q}) \geq 1-\mu^{-x/q}\forall_{x>1}$ and the fact that $\mathrm{bal}_0 f_t > 1$. Because t is critical, $\Delta_t > 0$, so

$$\begin{aligned} 0 < \Delta_t &= I_t - D_t^- - (1-\eta)D_t^{\mathrm{f}} \\ &\overset{(2.4.16)}{\leq} \mu\,\mathrm{bal}_0 f_t(1-\mu^{-1/q})\mathrm{pot}_{t-1}(e_M) - (1-\eta)D_t^{\mathrm{f}} \\ \Longrightarrow (1-\eta)D_t^{\mathrm{f}} &< \mu\,\mathrm{bal}_0 f_t(1-\mu^{-1/q})\mathrm{pot}_{t-1}(e_M) \end{aligned} \tag{2.4.17}$$

Consider a previously free edge e_f that was chosen by Breaker. Then $\mathrm{pot}(e_f) \geq \mathrm{pot}(e)$, because otherwise Breaker would have chosen e instead. By Lemma 2.19, we get a decrease in the potential of at least $(1-\mu^{-1/q})\mathrm{pot}(e)$ for every free edge e of which there are f_t. We have

$$D_t^{\mathrm{f}} \geq f_t(1-\mu^{-1/q})\mathrm{pot}_t(e).$$

Thus

$$\mathrm{pot}_t(e) \leq \frac{D_t^{\mathrm{f}}}{f_t(1-\mu^{-1/q})} \overset{(2.4.17)}{<} \frac{\mu\,\mathrm{bal}_0 f_t(1-\mu^{-1/q})}{(1-\eta)f_t(1-\mu^{-1/q})}\mathrm{pot}_{t-1}(e_M) = \frac{\mu\,\mathrm{bal}_0}{(1-\eta)}\mathrm{pot}_{t-1}(e_M).$$

$\square$

2.4.3 Bounding the Increase in the Overall Potential

Although Breaker's strategy leads to a maximal decrease of the potential after Maker added her edge, his strategy does not guarantee that the overall potential decreases in every round. Let t_0 be a round in which the overall potential increases. In the following, we will see that, once the overall potential exceeds n, after few rounds, the overall potential will be at most POT_{t_0} again, ensuring Breaker's win. We follow the pattern of arguments from Glazik and Srivastav [GS18] for the 2-uniform case.

Let $\gamma \in (0,1)$ and $\varepsilon > 0$ be constant parameters with

$$\frac{1-\eta}{(1+\varepsilon)\mu\,\mathrm{bal}_0} > 1. \tag{2.4.18}$$

and let

$$c := \left\lceil \frac{\log(3)-\log(1-\gamma)}{\log(1-\eta)-\log(1+\varepsilon)-\log(\mu\,\mathrm{bal}_0)} \right\rceil. \tag{2.4.19}$$

With (2.4.18), we get $\log(1-\eta)-\log(1+\varepsilon)-\log(\mu\,\mathrm{bal}_0) = \log(\frac{1-\eta}{(1+\varepsilon)\mu\,\mathrm{bal}_0}) > 0$. Because $\gamma \in (0,1)$, we get $1-\log(1-\gamma) > 0$. It follows that $c > 0$. Further, because $1 < \mu < 2$, c is bounded by constants. Now, consider the following points in time:

t_0**:** Let t_0 be the round where the overall potential first exceeds n, i.e., when $\mathrm{POT}_{t_0} > n$ and $\mathrm{POT}_{t_0-1} \leq n$ with $\mathrm{POT}_t < 2n$ for all $t < t_0$. t_0 is a critical round by Definition 2.21. Due to Observation 2.15, there are at least $12(n-5)$ free edges. Let e_{t_0} be the edge Maker chose in round t_0 with $v := \arg\max_{w\in e_{t_0}} \mathrm{pot}_{t-1}(w)$.

t_1**:** Let t_1 be the first round after $t_0 - 1$ at which $\mathrm{pot}_{t_1}(v) \leq (1-\gamma)\mathrm{pot}_{t_0-1}(v)$.

t_2**:** Let t_2 be the first round after $t_0 - 1$ at which $\mathrm{pot}_{t_2}(w) \geq (1+\varepsilon)\mathrm{pot}_s(w)$ for some $w \in V$ and any round s with $t_0 \leq s < t_2$. Note that since t_2 is the first round for which $\mathrm{pot}_{t_2}(w) \geq (1+\varepsilon)\mathrm{pot}_s(w)$, we have $\mathrm{pot}_t(w) \leq (1+\varepsilon)\mathrm{pot}_{t_0}(w)$ for all previous rounds $t < t_2$.

t_3**:** Let t_3 be the c-th critical round after t_0.

If, for $i \in \{1,2,3\}$, the game ends before round t_i, we set $t_i = \infty$. Define $t^* := \min\{t_1, t_2, t_3\}$ where $t^* = \infty$ is possible.

Theorem 2.23 *If the game has not ended before round t^*, then it holds* $\mathrm{POT}_{t^*} \leq \mathrm{POT}_{t_0-1}$.

Theorem 2.23 will be proved by a series of lemmas, Lemma 2.27, Lemma 2.28, and Lemma 2.29. We start with the following observation.

Observation 2.24 *If the game has not ended before turn t, where $t_0 \le t < t_2$, it holds*

$$\mathrm{POT}_t < 2n.$$

Proof. Let v be an arbitrary vertex. By the choice of t_2, for $t < t_2$ it holds that $\mathrm{pot}_t(v) \le (1+\varepsilon)\mathrm{pot}_{t_0}(v)$. For the overall potential we have

$$\mathrm{POT}_t = \sum_{v\in V} \mathrm{pot}_t(v) \le \sum_{v\in V} (1+\varepsilon)\mathrm{pot}_{t_0}(v) = (1+\varepsilon)\mathrm{POT}_{t_0} \tag{2.4.20}$$

Using (2.4.8) and Lemma 2.20 *(ii)*, we get

$$\begin{aligned}
\mathrm{POT}_{t_0} &= \mathrm{POT}_{t_0} - \mathrm{POT}_{t_0-1} + \mathrm{POT}_{t_0-1} \\
&= I_{t_0} - D_{t_0} + \mathrm{POT}_{t_0-1} && \text{(by (2.4.8))} \\
&= I_{t_0} - \left(D_{t_0}^{+} + D_{t_0}^{-} + D_{t_0}^{\perp} + D_{t_0}^{\mathrm{f}} + D_{t_0}^{0}\right) + \mathrm{POT}_{t_0-1} \\
&\le I_{t_0} - D_{t_0}^{-} + \mathrm{POT}_{t_0-1} \\
&\le \mu^{(\mathrm{bal}_0 f_{t_0})/q}\, \mathrm{POT}_{t_0-1} && \text{(by Lemma 2.20(ii))} \\
&\le \mu\, \mathrm{POT}_{t_0-1} && (\mathrm{bal}_0 f_t/q \le 1) \qquad (2.4.21)
\end{aligned}$$

Combining (2.4.20) and (2.4.21), using $\mathrm{POT}_{t_0-1} \le n$ and $\mu = 1 + o(1)$, we get

$$\mathrm{POT}_t \overset{(2.4.20)}{\le} (1+\varepsilon)\mathrm{POT}_{t_0} \overset{(2.4.21)}{\le} \mu(1+\varepsilon)\mathrm{POT}_{t_0-1} = \mu(1+\varepsilon)n < 2n.$$

□

Lemma 2.25 *Let s be a round where $t_0 \le s \le t^*$ and $s < t_2$ and let $\mathrm{crit}(t_0, s) \in [c]$ denote the number of critical rounds between t_0 and s, including both t_0 and s. Then, for every unclaimed edge $e = \{v_1, v_2, v_3\}$, it holds for $v = \arg\max_{u\in e_M} \mathrm{pot}_{t_0-1}(u)$ that*

$$\mathrm{pot}_s(e) < \left(\frac{(1+\varepsilon)\mu\,\mathrm{bal}_0}{1-\eta}\right)^{\mathrm{crit}(t_0,s)} 3\,\mathrm{pot}_{t_0-1}(v) \tag{2.4.22}$$

Proof. We give a proof by induction on $\mathrm{crit}(t_0, s)$, where s is arbitrary as stated in the lemma, but now fixed. Let e_{t_0} be the edge Maker claimed in round t_0 and let $v = \arg\max_{u \in e_{t_0}} \operatorname{pot}_{t_0-1}(u)$.

We start the induction, so let $\operatorname{crit}(t_0, s) = 1$. We have

$$\operatorname{pot}_s(e) = \sum_{w \in e} \operatorname{pot}_s(w) \leq \sum_{w \in e} (1+\varepsilon)\operatorname{pot}_{t_0}(w) = (1+\varepsilon)\operatorname{pot}_{t_0}(e) \tag{2.4.23}$$

using $s \leq t^* \leq t_2$, so $\operatorname{pot}_s(w) \leq (1+\varepsilon)\operatorname{pot}_{t_0}(w)$ by the definition of t_2. Now by Lemma 2.22 and (2.4.23),

$$\operatorname{pot}_s(e) \leq (1+\varepsilon)\operatorname{pot}_{t_0}(e) < (1+\varepsilon)\frac{\mu \operatorname{bal}_0}{1-\eta}\operatorname{pot}_{t_0-1}(e_{t_0}) \leq (1+\varepsilon)\frac{\mu \operatorname{bal}_0}{1-\eta} 3\operatorname{pot}_{t_0-1}(v).$$

For the induction step, assume that claim (2.4.22) is true for any s' where $\operatorname{crit}(t_0, s') = j$, $j \in [c-1]$. Let s be a round where $\operatorname{crit}(t_0, s) = i+1$ and $i+1 \leq c$. Let s' be the last critical round before s or, if s is itself critical, then $s' = s$. Thus, $\operatorname{crit}(t_0, s'-1) = i$ and $\operatorname{crit}(t_0, s) = \operatorname{crit}(t_0, s') = i+1$. Let e_M be the edge Maker claims in round s'. By this choice, after round $s'-1$, the edge e_M is still unclaimed which is required for the application of the induction hypothesis. We have

$$\begin{aligned}
\operatorname{pot}_s(e) &\leq (1+\varepsilon)\operatorname{pot}_{s'}(e) && (s < t_2)\\
&\leq (1+\varepsilon)\frac{\mu \operatorname{bal}_0}{1-\eta}\operatorname{pot}_{s'-1}(e_M) && \text{(Lemma 2.22)}\\
&\leq (1+\varepsilon)\frac{\mu \operatorname{bal}_0}{1-\eta}\left((1+\varepsilon)\frac{\mu \operatorname{bal}_0}{1-\eta}\right)^i \operatorname{pot}_{t_0-1}(e_M) && \text{(induction hypothesis)}\\
&= \left(\frac{(1+\varepsilon)\mu \operatorname{bal}_0}{1-\eta}\right)^{i+1} \operatorname{pot}_{t_0-1}(e_M)\\
&\leq \left(\frac{(1+\varepsilon)\mu \operatorname{bal}_0}{1-\eta}\right)^{i+1} 3\operatorname{pot}_{t_0-1}(v). && \text{(by the choice of } v)
\end{aligned}$$

In Lemma 2.22 (we used above) the assumption $f_t \geq 12(n-5)$ must be fulfilled. To prove this, we use Observation 2.15 and Observation 2.24: The contraposition of Observation 2.15 states that if at any round t, $\operatorname{POT}_t < 2n$, then $f_t \geq 12(n-5)$. The fact that $\operatorname{POT}_t < 2n$ is given by Observation 2.24.

□

Lemma 2.26 *For any $\xi > 0$ and n sufficiently large it holds*

$$\sum_{\substack{s,\ t_0 \leq s \leq t^* \text{ and} \\ s \text{ critical}}} I_s \leq 3c(\mu-1)\operatorname{pot}_{t_0-1}(v) < \xi \operatorname{pot}_{t_0-1}(v).$$

Proof. Let e_{t_0} be the edge Maker claimed in round t_0. Because the overall potential only increases in e_{t_0}'s vertices, by Lemma 2.19 *(i)*, we get

$$\begin{aligned} I_{t_0} &\le (\mu-1)\,\mathrm{pot}_{t_0-1}(e_{t_0}) \\ &\le 3(\mu-1)\,\mathrm{pot}_{t_0-1}(v) \end{aligned}$$

Let s be a critical round with $t_0 < s \le t^*$. Let e_s be Maker's edge claimed in round s, so e_s is claimed after round $s-1$. For round s, we have

$$\begin{aligned} I_s &\le (\mu-1)\,\mathrm{pot}_{s-1}(e_s) && \text{(Lemma 2.19 \textit{(i)})} \\ &< 3(\mu-1)\left(\frac{(1+\varepsilon)\mu\,\mathrm{bal}_0}{1-\eta}\right)^{\mathrm{crit}(t_0,s-1)} \mathrm{pot}_{t_0-1}(v) && \text{(Lemma 2.25 and } e_s \text{ unclaimed in round } s-1) \\ &\le 3(\mu-1)\,\mathrm{pot}_{t_0-1}(v), && \text{(with (2.4.18))} \end{aligned}$$

so

$$I_s \le 3(\mu-1)\,\mathrm{pot}_{t_0-1}(v). \tag{2.4.24}$$

Recall that t_3 is defined as the c-th critial round after t_0. Since $t^* \le t_3$, the number of critical rounds between t_0 and t^* is at most c. Thus, by (2.4.24)

$$\sum_{\substack{s,\ t_0 \le s \le t^* \text{ and} \\ s \text{ critical}}} I_s \le \sum_{\substack{s,\ t_0 \le s \le t^* \text{ and} \\ s \text{ critical}}} 3(\mu-1)\,\mathrm{pot}_{t_0-1}(v) \le 3c(\mu-1)\,\mathrm{pot}_{t_0-1}(v).$$

We have $\mu - 1 = o(1)$ and c is bounded by a constant, therefore for sufficiently large n

$$3c(\mu-1)\,\mathrm{pot}_{t_0-1}(v) < \xi\,\mathrm{pot}_{t_0-1}(v).$$

□

The results of the following three lemmas will imply Theorem 2.23.

Lemma 2.27 *If* $t_1 \le \min\{t_2, t_3\}$, *then* $\mathrm{POT}_{t_0-1} \ge \mathrm{POT}_{t^*}$.

Proof. Let $\xi \in (0, \eta\gamma)$. The assumption on t_1 and the definition of t^*, i.e. $t^* = \min\{t_1, t_2, t_3\}$ implies $t^* = t_1$ and by the definition of t_1, we get $\mathrm{pot}_{t^*}(v) \le (1-\gamma)\,\mathrm{pot}_{t_0-1}(v)$.

Let $R := \sum_{s,\ t_0 \le s \le t^*} r_s$. Then,

$$\begin{aligned}
\mathrm{POT}_{t^*} - \mathrm{POT}_{t_0-1} &= \sum_{s,\ t_0 \le s \le t^*} \mathrm{POT}_s - \mathrm{POT}_{s-1} \\
&= \sum_{s,\ t_0 \le s \le t^*} \Delta_s - r_s && \text{(with (2.4.15))} \\
&= \sum_{\substack{s,\ t_0 \le s \le t^* \text{ and} \\ s \text{ critical}}} \Delta_s + \underbrace{\sum_{\substack{t_0 \le s \le t^* \\ s \text{ non-critical}}} \Delta_s}_{\le 0 \text{ by Definition 2.21}} - R \\
&\le \sum_{\substack{s,\ t_0 \le s \le t^* \text{ and} \\ s \text{ critical}}} \Delta_s - R \\
&= \sum_{\substack{s,\ t_0 \le s \le t^* \text{ and} \\ s \text{ critical}}} I_s - D_s^- - (1-\eta) D_s^{\mathrm{f}} - R && \text{(by Definition 2.21)} \\
&\le \sum_{\substack{s,\ t_0 \le s \le t^* \text{ and} \\ s \text{ critical}}} I_s - R \\
&\le \xi \max_{v \in e_{t_0}} \mathrm{pot}_{t_0-1}(v) - R && \text{(with Lemma 2.26)}
\end{aligned}$$

We now need to show that $R \ge \xi\, \mathrm{pot}_{t_0-1}(v)$. We have

$$\xi\, \mathrm{pot}_{t_0-1}(v) \le \eta\gamma\, \mathrm{pot}_{t_0-1}(v) \le \eta(\mathrm{pot}_{t_0-1}(v) - \mathrm{pot}_{t^*}(v)) \qquad \text{(since } \mathrm{pot}_{t^*}(v) \le (1-\gamma)\mathrm{pot}_{t_0-1}(v))$$

$$\begin{aligned}
&= \eta \left(\sum_{s,\ t_0 \le s \le t^*} D_s(v) - I_s(v) \right) && \text{(by (2.4.7))} \\
&= \eta \left(\sum_{s,\ t_0 \le s \le t^*} D_s^-(v) + D_s^+(v) + D_s^\perp(v) + D_s^{\mathrm{f}}(v) + D_s^0(v) - I_s(v) \right) && \text{(with (2.4.6))} \\
&\le \eta \left(\sum_{s,\ t_0 \le s \le t^*} D_s^+(v) + D_s^\perp(v) + D_s^{\mathrm{f}}(v) + D_s^0(v) \right) && \text{(since } D_s^-(v) \le I_s(v) \text{ by (2.4.9))} \\
&\le \sum_{s,\ t_0 \le s \le t^*} D_s^+(v) + D_s^\perp(v) + \eta D_s^{\mathrm{f}}(v) + D_s^0(v) && (\eta \in (0, 1-\mu\, \mathrm{bal}_0)) \\
&= \sum_{s,\ t_0 \le s \le t^*} r_s = R. && \text{(by Definition 2.21 for } r_s)
\end{aligned}$$

□

Lemma 2.28 *If* $t_2 \le \min\{t_1, t_3\}$, *then* $\mathrm{POT}_{t_0-1} \ge \mathrm{POT}_{t^*}$.

Proof. Let $\xi > 0$ with $\xi \leq \eta(1-\gamma)(1-(1+\varepsilon)^{-1/\mathrm{bal}_0})$. By the choice of t_2, we have $t^* = t_2$. Thus, there is a round s_0, where $t_0 \leq s_0 < t^*$ and a vertex w with $\mathrm{pot}_t(w) \geq (1+\varepsilon)\mathrm{pot}_{s_0}(w)$. We have $t^* < t_1$ and thus, $\mathrm{pot}_{t_0-1}(v) > 0$. As in the proof of the previous Lemma, we need to show that $R \geq \xi\, \mathrm{pot}_{t_0-1}(v)$. To this end, we first show that for t, $s_0 \leq t \leq t^*$,

$$\mathrm{pot}_t(w) \leq \mathrm{pot}_{s_0}(w) \prod_{s_0 < s \leq t} \mu^{\mathrm{bal}_0 f_s/q}. \tag{2.4.25}$$

Proof of (2.4.25) *via induction over* t. For $t = s_0$ the claim is

$$\mathrm{pot}_{s_0}(w) \leq \mathrm{pot}_{s_0}(w) \underbrace{\prod_{s,\ s_0 \leq s \leq t} \mu^{\mathrm{bal}_0 f_s/q}}_{\geq 1 \text{ since } \mu \geq 1},$$

so (2.4.25) is true for $t = s_0$. Now let t such that $t-1 \geq s_0$. We have

$$\begin{aligned}
\mathrm{pot}_t(w) - \mathrm{pot}_{t-1}(w) &= I_t(w) - D_t(w) && \text{(by (2.4.7))}\\
&= I_t(w) - \left(D_t^+ + D_t^- + D_t^\perp(w) + D_t^{\mathrm{f}}(w) + D_t^0(w)\right) && \text{(by (2.4.6))}\\
&\leq I_t(w) - D_t^-(w)\\
&\leq \mathrm{pot}_{t-1}(w)\left(\mu^{\mathrm{bal}_0 f_t/q} - 1\right), && \text{(by Lemma 2.20 \textit{(i)})}
\end{aligned}$$

thus $\mathrm{pot}_t(w) \leq \mathrm{pot}_{t-1}(w)\mu^{\mathrm{bal}_0 f_t/q}$.

Using the induction hypothesis for $\mathrm{pot}_{t-1}(w)$, we get from the last inequality

$$\mathrm{pot}_t(w) \leq \left(\mathrm{pot}_{s_0}(w) \prod_{s_0 < s \leq t-1} \mu^{\mathrm{bal}_0 f_s/q}\right)\mu^{\mathrm{bal}_0 f_t/q} = \mathrm{pot}_{s_0}(w) \prod_{s_0 < s \leq t} \mu^{\mathrm{bal}_0 f_s/q}.$$

This concludes the proof of (2.4.25). Further, since $s_0 \leq t^* = t_2$, we get by the definition of t_2 and (2.4.25)

$$(1+\varepsilon)\mathrm{pot}_{s_0}(w) \leq \mathrm{pot}_{t_2}(w) \leq \mathrm{pot}_{t^*}(w) \leq \mathrm{pot}_{s_0}(w) \prod_{s_0 < s \leq t^*} \mu^{\mathrm{bal}_0 f_s/q}.$$

Hence

$$(1+\varepsilon) \leq \prod_{s_0 < s \leq t^*} \mu^{\mathrm{bal}_0 f_t/q} = \mu^{\mathrm{bal}_0 \sum_{s_0 < s \leq t^*} f_s/q}.$$

Taking the logarithms on both sides of the last inequality delivers

$$\sum_{s_0<s\leq t^*} f_s \geq \frac{q\log(1+\varepsilon)}{\mathrm{bal}_0\log(\mu)} =: x. \tag{2.4.26}$$

So, x is the minimum number of free edges claimed by Breaker between rounds s_0 and t^*. By the choice of t^*, we have $t^* < t_1$. Thus, for any s between t_0 and t^*, $\mathrm{pot}_s(v) \geq (1-\gamma)\mathrm{pot}_{t_0-1}(v)$ and thereby $\mathrm{pot}_s(e) \geq (1-\gamma)\mathrm{pot}_{t_0-1}(v)$ for every unclaimed edge e that is incident in v. Every free edge claimed by Breaker has at least this potential. We can estimate D_s^{f}, the decrease in the potential caused by free edges, for some round s by the minimum number of free edges times the minimum potential of a free edge times the minimum reduction of the potential relative to the previous potential given by Lemma 2.19 *(ii)*. Formally,

$$\begin{aligned} D_s^{\mathrm{f}} &\geq f_s(1-\gamma)\mathrm{pot}_{t_0-1}(v)\left(1-\mu^{-1/q}\right) && \text{(by Lemma 2.19 \textit{(ii)})} \\ &\geq x(1-\gamma)\mathrm{pot}_{t_0-1}(v)\left(1-\mu^{-1/q}\right). && \text{(by (2.4.26))} \end{aligned} \tag{2.4.27}$$

$$\begin{aligned} R = \sum_{s,\ t_0\leq s\leq t^*} r_s &= \sum_{s,\ t_0\leq s\leq t^*} D_s^{+} + D_s^{\perp} + \eta D_s^{\mathrm{f}} + D_s^{0} && \text{(by Definition 2.21)} \\ &\geq \eta \sum_{s,\ t_0\leq s\leq t^*} D_s^{\mathrm{f}} \\ &\geq \eta x(1-\gamma)\mathrm{pot}_{t_0-1}(v)\left(1-\mu^{-1/q}\right) && \text{(by (2.4.27))} \\ &\geq \eta(1-\gamma)\mathrm{pot}_{t_0-1}(v)\left(1-\mu^{-x/q}\right) && \text{(using } x(1-\mu^{-1/q}) \geq 1-\mu^{-x/q}\forall_{x\in\mathbb{R}_{\geq 1}}) \\ &= \eta(1-\gamma)\mathrm{pot}_{t_0-1}(v)\left(1-\mu^{-\frac{q\log(1+\varepsilon)}{q\,\mathrm{bal}_0\log(\mu)}}\right) \\ &= \eta(1-\gamma)\mathrm{pot}_{t_0-1}(v)\left(1-(1+\varepsilon)^{-1/\mathrm{bal}_0}\right) && \text{(with } \mu = e^{\log(\mu)}) \\ &\geq \xi\,\mathrm{pot}_{t_0-1}(v). && \text{(by the choice of } \xi) \end{aligned}$$

□

Lemma 2.29 $t_3 \geq \min\{t_1, t_2\}$.

Proof. Assume for a moment that $t_3 < \min\{t_1, t_2\}$. Then $t_3 = t^*$, i.e., t^* must be the c-th critical round after $t_0 - 1$. By Lemma 2.25, for every unclaimed edge e after t^*, it holds

$$\mathrm{pot}_{t^*}(e) < \left(\frac{(1+\varepsilon)\mu\,\mathrm{bal}_0}{1-\eta}\right)^c 3\,\mathrm{pot}_{t_0-1}(v). \tag{2.4.28}$$

Using the fact $x^{1/\log(1/x)} = 1/e$ for all $x \in \mathbb{R}_{>0}$ with $x = \frac{(1+\varepsilon)\mu\,\mathrm{bal}_0}{1-\eta}$ and the value for c defined in (2.4.19), we can write $c = (\log(3) - \log(1-\gamma))/\log(1/x)$. The right side of (2.4.28) can be simplified as follows:

$$\begin{aligned}\left(\frac{(1+\varepsilon)\mu\,\mathrm{bal}_0}{1-\eta}\right)^c 3\,\mathrm{pot}_{t_0-1}(v) &= x^{\frac{\log(3)-\log(1-\gamma)}{\log(1/x)}} 3\,\mathrm{pot}_{t_0-1}(v)\\ &= \left(x^{\left(\frac{1}{\log(1/x)}\right)}\right)^{\log(3)-\log(1-\gamma)} 3\,\mathrm{pot}_{t_0-1}(v)\\ &= \left(\frac{1}{e}\right)^{\log(3)-\log(1-\gamma)} 3\,\mathrm{pot}_{t_0-1}(v)\\ &= \frac{(1-\gamma)}{3} 3\,\mathrm{pot}_{t_0-1}(v)\\ &= (1-\gamma)\,\mathrm{pot}_{t_0-1}(v).\end{aligned}$$

Thus, $\mathrm{pot}_{t^*}(e) < (1-\gamma)\,\mathrm{pot}_{t_0-1}(v)$.

Because $t^* < t_1$, it holds $\mathrm{pot}_{t^*}(v) \geq (1-\gamma)\,\mathrm{pot}_{t_0-1}(v)$ and thus after round t^* every unclaimed edge incident in v has potential at least $(1-\gamma)\,\mathrm{pot}_{t_0-1}(v)$. Hence, there can be no unclaimed edges incident in v and therefore $\mathrm{pot}_s(v) = 0$ in some round s, where $t_0 \leq s \leq t^*$, but $t_1 \leq s \leq t^* = t_3$ which contradicts our assumption. □

2.4.4 Proof of the Main Result

In this section we prove that Breaker's strategy works correctly and is indeed a winning strategy.

We now show Theorem 2.16 that for every round s and sufficiently large n, it holds $\mathrm{POT}_s \leq \mu(1+\varepsilon)n < 2n$.

Proof of Theorem 2.16. Let s be a round where $\mathrm{POT}_s > n$ and let t_0 be such that $t_0 \leq s$ is maximal with $\mathrm{POT}_{t_0-1} \leq n$. Because the overall potential at the beginning of the game is at most n, such a t_0 always exists. Define t^* as in subsection 2.4.3. If $s = t^*$, then we can apply Theorem 2.23 and get $\mathrm{POT}_s = \mathrm{POT}_{t^*} \leq \mathrm{POT}_{t_0-1} \leq n$.

If $s < t^*$ and because $t^* = \min\{t_1, t_2, t_3\}$, it follows that $s < t_2$. Thus, by (2.4.20) we get

$$\mathrm{POT}_s \leq (1+\varepsilon)\mathrm{POT}_{t_0}. \tag{2.4.29}$$

By (2.4.21) we have

$$\mathrm{POT}_{t_0} \leq \mu\,\mathrm{POT}_{t_0-1}. \tag{2.4.30}$$

Combining (2.4.29) and (2.4.30) gives, using $\mu = 1 + o(1)$ and $\varepsilon < 1$,

$$\mathrm{POT}_s \leq (1+\varepsilon)\mathrm{POT}_{t_0} \leq \mu(1+\varepsilon)\mathrm{POT}_{t_0-1} \leq \mu(1+\varepsilon)n < 2n.$$

□

By Theorem 2.17, Breaker wins the game.

2.5 Extension to k-Uniform Hypergraphs

In the following, we will describe the techniques from section 2.4 for k-uniform hypergraphs where $k \ll \sqrt{n}$. Consider two edges $e_1 = \{v_1, \ldots, v_k\}$ and $e_2 = \{v_1', \ldots, v_k'\}$ that intersect in exactly one vertex, say $v = v_1 = v_1'$. We now determine the number of possible triangles that contain both e_1 and e_2. An edge closing a triangle contains exactly one vertex from e_1 and one from e_2 other than v. So there are $(k-1)^2$ ways to pick such a pair. To form a k-uniform edge, there are still $k-2$ vertices missing which can be chosen from the remaining vertices $V \setminus (e_1 \cup e_2)$ of which there are $n - 2k + 1$. Just as in the 3-uniform case, Breaker wants to prevent Maker from connecting high-degree vertices. The event in which Breaker may be unable to prevent Maker from creating too many possible triangles by adding an edge e_M, i.e.,

$$q < (k-1)^2 \binom{n-2k+1}{k-2} \sum_{v \in e_M} \deg_M(v). \tag{2.5.1}$$

Now enables us to define the *dangerous event* which Breaker aims to avoid:

$$\deg_M(v) \leq \frac{q}{k(k-1)^2\binom{n-2k+1}{k-2}} \tag{2.5.2}$$

If Breaker can prevent Maker from achieving (2.5.2) for all $v \in V$ then (2.5.1) does not hold and Breaker wins the game. The balance function is now define analogous to the 3-uniform case with the total number of edges incident in v now at $\binom{n-1}{k-1}$.

$$\frac{\binom{n-1}{k-1} - \deg_B(v)}{q\left(\frac{q(1-\delta)}{k(k-1)^2\binom{n-2k+1}{k-2}} - \deg_M(v)\right) - (k-1)^2\binom{n-2k+1}{k-2}\sum_{i=\deg_M(v)}^{q(1-\delta)/\left(k(k-1)^2\binom{n-2k+1}{k-2}\right)} i}.$$

By setting $\deg_M(v) = \deg_B(v) = 0$, we obtain

$$\lim_{\substack{n\to\infty \\ \delta\to 0}} \mathrm{bal}_0(n,\delta) = 1 \text{ for } \beta^{(2k-3)/2} = \sqrt{2}\sqrt{\frac{k^2 - 2k^3 + k^4}{(2k-1)(k-1)!(k-2)!}}$$

which allows us to compute the leading factors for the upper bound depending on k.

k	2	3	4	5	$\cdots$
	$1.6330n^{1/2}$	$2.6833n^{3/2}$	$1.852n^{5/2}$	$0.7857n^{7/2}$	$\cdots$

Table 2.2: Upper Bounds for the Triangle game in k-uniform hypergraphs

Chapter 3

Path Games

3.1 Path-Maker-Breaker and Walker-Breaker Games

In a Path-Maker-Breaker game, Maker attempts to claim edges such that the Maker-graph contains a path of fixed length, where the length depends on the concrete game. The game is played on either the complete graph or random graphs. Most research on Path-Maker-Breaker games has been dedicated to Hamiltonian Paths, i.e., paths that visit every vertex of the graph on which the game is played exactly once, see e.g. [CE78; Bec82; Hef+14]. Hefetz et al. [Hef+09] analyzed the fair game in $\mathcal{G}(n,p)$ graphs. They showed that a threshold of $p = \frac{\log n}{n}$ exists. This means that for any larger p, the game is almost surely a Maker's win. A classic result by Bollobás [Bol84] states that this asymptotical threshold also holds for the emergence of Hamiltonian paths in random graphs. Stojaković and Trkulja [ST19] showed that for $n > 8$ the game is a Maker's win in the fair variant.

Walker-Breaker Games were first introduced by Espig et. al [Esp+14]. These games are an alteration of Maker-Breaker Games with additional conditions on how Maker can choose her edges. In this game, Maker is called *Walker*. Walker starts at an arbitrary vertex at the beginning of the game and moves along one edge that does not belong to Breaker in each round with the restriction that the edge has to be incident to the new vertex of the edge chosen in the previous round. Walker wins the game, if she claimed all edges of a winning set and Breaker wins if, at the end of the game, there is no winning set belonging to Maker or if there is no free incident edge available for Maker. Path-Walker-Breaker games present an additional restriction on the set of edges that Walker can choose from. In this game, Walker can only walk along previously unclaimed edges. The natural question in this scenario is how many vertices Walker can visit.

3.2 The P_3-Game

We consider the $(1:q)$-Maker-Breaker subgraph game with a path of length 3 as the subgraph played on the complete graph K_n. Maker chooses one edges in each round and Breaker chooses q edges. q is the bias of the game.

In this section, we give an upper bound for the P_3-game. Asymptotically, the threshold bias can be determined using Theorem 2.3. Let P_ℓ be a path of length ℓ, then

$$m_2(P_\ell) = \max_{A \subseteq P_\ell, |V(A)|>2} \frac{|E(A)|-1}{|V(A)|-2} = \frac{|V(A)|-1-1}{|V(A)|-2} = 1$$

The 2-density of a path is 1, independent of its length ℓ. Hence, the threshold bias is $\Theta(n^{1/1}) = \Theta(n)$. We first show that the methods from section 2.4 cannot be applied to this type of game. Instead, we give a more prosaic counting argument.

A trivial upper bound of $2n$ for the threshold bias can be achieved by Breaker by isolating both vertices of Maker's previously claimed edge. For the lower bound, it is easy to see that if $q \leq n-2$, Breaker cannot isolate any vertices and thus loses the game after three rounds.

3.2.1 Potential Function Based Approach

The notation in this section largely follows the one from chapter 2. For an edge e, we let $\mathrm{d}_M(e)$ denote the number of vertices of e that have incident Maker-edges other than e itself. Thus, $d_M \in \{0,1,2\}$ for all edges. Again, we first construct a *balance function* that later leads to the potential function. The *dangerous event*, i.e., the event where Breaker might not be able to prevent Maker from winning, is now defined as $d_M(e) = 2$ for some edge e. In case Maker can claim an edge e where $d_M(e) = 2$, she wins the game. If $\mathrm{d}_M(e) < 2$ for all $e \in E$, no Maker P_3 exists and thus, Breaker wins the game.

We define the balance function for edges instead of vertices due to the definition of the dangerous event that is also defined on edges.

Definition 3.1 (Balance function) *Let $q = \beta n$. The balance of an edge* $\mathrm{bal} : E \longrightarrow \mathbb{R}$ *reflects how close Maker is to the* dangerous event *of having an edge $\{v,w\}$ for which $\mathrm{d}_M(\{v,w\}) \geq 2$ with $1 > \delta > 0$. Let us* *define*

$$\mathrm{bal}(\{v,w\}) := \frac{2n - \deg_B(v) - \deg_B(w)}{q\,(2(1-\delta) - \mathrm{d}_M(\{v,w\})) - n\left(1+\frac{1}{2\beta}\right)}. \tag{3.2.1}$$

The numerator describes the number of edges incident in v and w that are available for Maker, i.e., not belonging to Breaker. In the denominator, we multiply the number of edges q that Breaker can claim in one round by the term expressing the distance of Maker to the dangerous event for Breaker. The additive term in the denominator is an upper bound for the number of edges that Breaker needs to claim in order to prevent Maker from connecting two disjoint edges. From Breaker's point of view, this can be achieved by isolating one of Maker's edge's vertices in each round and then claiming an edge that connects each one of Maker's existing edges to the one she chose in that round.

At the beginning of the game, for every edge e, we have $d_M(e) = 0$ and thus the balance at the beginning of the game $\mathrm{bal}_0(e)$ can be computed with (3.2.1) and $\deg_B(u) = \deg_M(u)$ for all $u \in E$. Since $\mathrm{bal}_0(e)$ has the same value for every edge e, we can write bal_0.

$$\begin{aligned}\mathrm{bal}_0 = \mathrm{bal}_0(\delta) &= \frac{2n}{\beta n\,(2(1-\delta)) - n\left(1+\frac{1}{2\beta}\right)} \\ &= \frac{2}{2\beta(1-\delta) - 1 - \frac{1}{2\beta}} \\ &< \frac{2}{2\beta - 1 - \frac{1}{2\beta}}\end{aligned}$$

Solving $\frac{2}{2\beta-1-\frac{1}{2\beta}} = 1$ for β using a quadratic equation gives $\beta = \frac{3+\sqrt{13}}{4} \approx 1.65139$.

Let $\{v,w\}$ be an arbitrary edge. We now define the balanced breaker degree, i.e., the Breaker-degree Breaker would need to achieve $\mathrm{bal}(\{v,w\}) = \mathrm{bal}_0$ as the function $\deg^* : V \longrightarrow \mathbb{N}$ by the following equation

$$\begin{aligned} &\frac{2n - \deg^*(v) - \deg^*(w)}{q\,(2(1-\delta) - \mathrm{d}_M(\{v,w\})) - n\left(1+\frac{1}{2\beta}\right)} \overset{!}{=} \mathrm{bal}_0 \\ \Leftrightarrow\quad & 2n - \deg^*(v) - \deg^*(w) = \mathrm{bal}_0\left(q\,(2(1-\delta) - \mathrm{d}_M(\{v,w\})) - n\left(1+\frac{1}{2\beta}\right)\right) \\ \Leftrightarrow\quad & \deg^*(v) + \deg^*(w) = 2n - \mathrm{bal}_0\left(q\,(2(1-\delta) - \mathrm{d}_M(\{v,w\})) - n\left(1+\frac{1}{2\beta}\right)\right) \end{aligned} \tag{3.2.2}$$

The deficit d, $d : V \longrightarrow \mathbb{N}$ is now defined as the number of edges Breaker is short of reaching $\mathrm{bal} = \mathrm{bal}_0$, so

$$d(v) := \deg^*(v) - \deg_B(v).$$

For the potential function argument in Glazik [Gla19], it is crucial for the potential to exceed a certain value, i.e. $2n$ in the dangerous event for Breaker. In that event, Breaker is not guaranteed to be able to prevent Maker from winning the game. In our case, this is the case for $d_M(\{v,w\}) = 2$ for some edge $\{v,w\}$. We set $d_M(\{v,w\}) = 2$ and from (3.2.2), we get

$$\begin{aligned}
\deg^*(v) + \deg^*(w) &= 2n - \mathrm{bal}_0\left(q(2(1-\delta)-2) - n\left(1 + \frac{1}{2\beta}\right)\right) \\
&= 2n + \mathrm{bal}_0\left(2q\delta + n\left(1 + \frac{1}{2\beta}\right)\right) \\
&\geq 2n + \mathrm{bal}_0(2q\delta + n)
\end{aligned}$$

and thus,

$$\begin{aligned}
d(v) + d(w) &= \deg^*(v) - \deg_B(v) + \deg^*(w) - \deg_B(w) \\
&\geq 2n + \mathrm{bal}_0(2q\delta + n) - \deg_B(v) - \deg_B(w) \\
&\geq 2n + \mathrm{bal}_0(2q\delta + n) - 2n \\
&= \mathrm{bal}_0(2q\delta + n)
\end{aligned}$$

Because $q = \beta n$, we get

$$\mu^{(d(v)+d(w))/q} \geq \mu^{\mathcal{O}(1)}$$

This expression exceeds $2n$, if $\mu = \omega(1)$, for example

$$\mu = \left(1 + \frac{\beta \log(n)}{\delta}\right),$$

But the estimations of the potential in Section 3.3 of Glazik's dissertation rely on the fact that $\mu = 1 + o(1)$. Thus, these methods appear not to be applicable in this type of game as no function is compatible to the two conditions that on the one hand, the potential exeeds $2n$ at the dangerous event for Breaker and that on the other hand, $\mu = 1 + o(1)$. It seems that the potential function method as displayed by Glazik works for games, where Maker can increase the potential of an edge (or of a vertex) over a non-constant number of rounds, such that Breaker can prioritize those edges whose potential is highest, thereby decrasing the potential, albeit not monotonously, but over a number of rounds. In the P_3-game, on the other hand, Maker can increase the potential of an edge only twice, because $\mathrm{d}_M(e) \in \{0,1,2\}$ for all edges e and thus, if Maker claims an edge $e = \{v, w\}$ that increases $\mathrm{d}_M(e)$ to one, Breaker has to claim all edges incident in one vertex of e, say v. So he must decrease e's potential to zero, because otherwise Maker could claim another edge incident in v and thereby construct a P_3.

However, this reasoning does not exclude the existence of another potential function strategy distinct from the work of Glazik.

3.2.2 Counting Argument

We now give a less sophisticated argument to estimate an upper bound for the threshold bias of the P_3 game along with a winning strategy for Breaker. First, we compute the maximum number of rounds played. Let q be the bias. Then, the game lasts for at most $T := \lceil \binom{n}{2}/(q+1) \rceil$ rounds because there are $\binom{n}{2}$ edges in the complete graph on which the game is played and in each round, $q+1$ edges are claimed—q by Breaker and 1 by Maker. After T rounds, every edge is claimed. As in the previous section, Breaker's strategy consists of three steps. Let $e_{M,t} = \{v, w\}$ be the edge Maker claimed in round t Breaker's strategy is as follows:

1. Isolate the vertex of $e_{M,t}$ that has the lower Maker-degree, i.e., claim every edge incident in that vertex. If both vertices have Maker-degree 1, pick an arbitrary vertex. In this step, Breaker claims at most $n-2$ edges.

2. For all previously claimed Maker-edges $e_{M,1}, \dots, e_{M,t-1}$ that are disjoint from $e_{M,t}$, claim the edge that connects the two edges $e_{M,t}$ and $e_{M,t'}$ where $t' < t$. Because for each Maker-edge, Breaker isolated one vertex, there can be at most one free edge connecting two Maker edges $e_{M,t}$ and $e_{M,t'}$ for each $t' < t$. If Maker picked an edge that is connected to some other Maker-edge, Maker has constructed a P_2, whose endpoints are blocked due to Breaker's first step of his strategy.

3. If Breaker has not used up all his edges for the previous two steps, he claims arbitrary edges.

Theorem 3.2 *Maker cannot build a P_3 if Breaker plays according to the above strategy.*

Proof. To see that the Maker-graph consists of isolated edges and stars, consider two Maker-edges $e_{M,s}$ and $e_{M,t}$, claimed in round s and t, respectively. Let $s < t$. If $e_{M,s} \cap e_{M,t} = \varnothing$, then by part 1 of Breaker's strategy, both edges have a vertex each, where all remaining incident edges have been claimed by Breaker. In round t, the last remaining edge connecting $e_{M,s}$ and $e_{M,t}$ is claimed by Breaker in part 2 of his strategy. If $e_{M,s} = \{v, w\}$ and $e_{M,t} = \{v, u\}$ intersect in a vertex v, then, in round s, Breaker claimed all unclaimed edges incident in w in part 1 of his strategy and in round t, he claims all unclaimed edges incident in w. Thus, Maker cannot build a P_3 using $e_{M,s}$ and $e_{M,t}$. $\square$

Theorem 3.3 *For $q > (\frac{1+\sqrt{3}}{2} + o(1))n$, the strategy stated above is a winning strategy for Breaker for the P_3-game.*

Proof. As in any Maker-Breaker game, the number of rounds is $T := \lceil \frac{\binom{n}{2}}{q+1} \rceil$ unless Maker wins the game in an earlier round. In the t-th round, Breaker needs to claim $n-2$ edges in the first part of his strategy and $t-1$ edges in the second part. Thus, the number of edges needed by Breaker increases each round. If Breaker can ensure that he can play according to this strategy until all the edges are claimed, he wins the game.

Consider the last round T. If $(n-2)+(T-1) < q$, i.e. the number of edges that Breaker needs to claim in the last round is less than q, then Breaker wins. We have

$$\begin{aligned}
& (n-2)+(T-1) < q \\
\Leftrightarrow\quad & (n-2)+\left\lceil \frac{\binom{n}{2}}{q+1} \right\rceil - 1 < q \\
\Rightarrow\quad & (n-3)+\frac{\binom{n}{2}}{q+1} < q \\
\Rightarrow\quad & (q+1)(n-3)+\binom{n}{2} < q(q+1) \\
\Rightarrow\quad & \binom{n}{2} < q(q+1)-(q+1)(n-3) \\
\Rightarrow\quad & \binom{n}{2} < (q+1)(q-(n-3)).
\end{aligned}$$

Solving this quadratic inequality for q gives $q > \frac{\sqrt{3n^2-6n+4}+n-4}{2} \xrightarrow{n\to\infty} \frac{\sqrt{3}+1}{2}n$. $\square$

Chapter 4

A Streaming Algorithm for the Longest Path Problem

The longest path problem is the problem of finding a directed or undirected path with maximum length within a graph. The problem has applications e.g. in genome assembly where a graph is built from the output of a sequencing machine. A long contiguous path corresponds to a long sequence of the genome itself. While this is an oversimplification of the genome assembly problem, it might be worth investigating whether or not a path-based genome assembly algorithm would yield good results compared to the convential ones. Other applications include project planning where one searches for a critical path within the activity graph, i.e. the sequence of activities that determine the longest overall duration.

Unlike the shortest path problem, which is solvable in polynomial time, the longest path problem is $\mathcal{NP}$-hard and even finding an n^{δ} approximation for fixed $\delta > 0$ is $\mathcal{NP}$-hard. Because of the hardness and inapproximability, the most commonly used algorithms are heuristics that perform well in most cases.

In section 4.2 through section 4.4 we will present approximation algorithms for the problem, in section 4.5 and section 4.6 we show two RAM heuristics, and in section 4.10 we present our own approach for a heuristic within the semi-streaming model, i.e. the memory used by the algorithm is restricted linearly in the number of vertices. Section 4.15 shows experimental results of our algorithm compared to various heuristical algorithms.

4.1 Notation, Hardness, and Polynomial Algorithms for Special Graph Classes

We let $G = (V,E)$ denote an undirected simple graph with vertex set V and edge set $E \subseteq \binom{V}{2}$. m denotes the number of edges $|E|$ and n the number of vertices $|V|$. A path $P = (v_0, v_1, \ldots, v_k)$ is defined as a sequence of vertices such that $\{v_{i-1}, v_i\} \in E$ for all $i \in [k]$ and $v_i \neq v_j$ for all $i \neq j$. The length of P is defined as the number of edges the path traverses, i.e. a path $P = (v_0, v_1, \ldots, v_k)$ has length $\ell(P) = |E[P]| = k$ where $E[P]$ is the set of edges traversed by P. A path of length k will also be called a k-path. A k^+-path is a path of length at least k.

Problem 4.1 (k-PATH) **Input** *Undirected graph* $G = (V,E)$

Question *Does G contain a path of length k?*

The k-path problem is the problem of determining whether the input graph contains a path of length k. This problem is closely related to the HAMILTONIAN PATH problem as we will show in the following section. The question that naturally arises from this problem is: *What is the longest path contained in G?*

Problem 4.2 (LONGEST PATH) **Input** *Undirected graph* $G = (V,E)$

Output *Path* $P = (v_0, v_1, \ldots, v_k)$ *where* $k \leftarrow \max$

In later sections of this thesis we also consider the k-CYCLE and LONGEST CYCLE problems, which are defined analogously. For $k \geq 3$, a k-cycle consists of a k-path $(v_0, \ldots, v_k)$ and an edge $\{v_0, v_k\}$.

The focus of this work is on algorithms for the LONGEST PATH problem but k-path and k-cycle occur as subproblems.

Another version of the longest path problem is the s-t-LONGEST PROBLEM, which is the problem of finding a longest path between two fixed vertices s and t. Research about heuristics for this version has been conducted by Scholvin [Sch99].

Lemma 4.3 *The k-PATH problem (Problem 4.1) is $\mathcal{NP}$-complete.*

Proof. We first show that k-PATH $\in \mathcal{NP}$. Therefore we construct a polynomial verifier V. Given any YES-instance I of the k-path problem, there has to be a witness W such that, given (I, W) the verifier returns YES. If I is a NO-instance, the verifier returns NO for all possible W.

The witness is an ordered set of vertices $P = (v_0, \ldots, v_k)$ given by an oracle.

The verifier V then has to check the following:

1. Is $E[P] \subseteq E$?
2. Is P acyclic?

The first step can be done in $\mathcal{O}(m)$ time by checking if all pairs of adjacent vertices from P are contained in E. Thereby it is also ensured that P is connected but not that P does not contain any cycles. To this end, the second step needs $\mathcal{O}(n)$ time for testing if all vertices of V occur in P at most once.

To show that $k-$PATH is $\mathcal{NP}$-hard there is a fairly obvious reduction from the HAMILTONIAN PATH problem. This is the problem of determining whether the input graph contains a path that passes every vertex, i.e. the HAMILTONIAN PATH problem is a special case of $k-$PATH where $k = n - 1$. HAMILTONIAN PATH is one of Karp's 21 $\mathcal{NP}$-complete problems [Kar72]. □

According to Feder, Motwani, and Subi the problem is notorious for the difficulty of understanding its approximation hardness [FMS02] because of the wide gap between upper and lower bounds.

There is an approximation algorithm presented by Björklund and Husfeldt [BH03] with an approximation ratio of $\mathcal{O}(n(\log\log n/\log n)^2)$, which we will introduce in section 4.4.

For certain classes of graphs, there are polynomial-time algorithms that can solve the problem exactly. These classes include grid graphs [KBA12], and trees [Bul+02] the latter result will become useful as we construct our streaming algorithm in chapter 4. In Algorithm 4.1 we show how to find long paths in trees efficiently.

In this work we only consider undirected graphs but there are also polynomial time algorithms for certain directed graphs such as DAGs [SW11, pp. 661–666].

We can find a longest path within a tree by Algorithm 4.1. This algorithm was first presented by Dijkstra in 1960. A proof of correctness can be found in [Bul+02]. It consists of two depth first searches, the first of which starting from an arbitrary vertex of a tree, say r, determines the farthest vertex s and the second DFS starts from s and finds the vertex t whose distance to s is maximal. The (unique) $s-t$-path is a longest path within the tree. This algorithm has a running time of $\mathcal{O}(n)$. The choice of the path itself may vary depending on the choice of r if there are multiple paths of maximal length, but the result is certainly optimal.

Algorithm 4.1: Longest Path in trees

```
Data: Tree $T$
Result: Longest path $P$ in $T$
choose random $r \in V[T]$;
$s := \mathrm{DFS}(T, r)$;
$t := \mathrm{DFS}(T, s)$;
construct path $P$ connecting $s$ and $t$;
return $P$;
Procedure DFS(T,r)
    $S := (r)$;
    $L := S$;
    Mark $r$ as visited;
    while $S = (S_1, \dots, S_i)$, $i > 0$ do
        $v \leftarrow v_i$;
        if there is an unvisited $y \in N(v)$ then
            $S = S\&y$;
            Mark $y$ as visited;
        else
            if $|L| < |S|$ then
                $L \leftarrow S$;
            $S \leftarrow (S_1, \dots, S_{i-1})$;
    return last vertex on $L$.
```

4.2 Color Coding

In this section we show that the k-path problem is fixed-parameter tractable by giving an algorithm, whose running time is polynomial in n for fixed k.

Definition 4.4 ($\mathscr{FPT}$) *A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$ over a finite alphabet Σ. Such a problem is called fixed-parameter tractable, if there is a function f and an algorithm that for each $(x,k) \in \Sigma^* \times \mathbb{N}$ decides whether $(x,k) \in L$ in running time $f(k) \cdot \mathrm{poly}(|x|)$.*

Note that the running time may be exponential in k or worse. Obviously $\mathscr{P} \subseteq \mathscr{FPT}$ since for $L \in \mathscr{P}$ the function f can be chosen constant resulting in a polynomial running time.

To show that the k-PATH problem is fixed-parameter tractable we give an algorithm presented by Alon, Yuster, and Zwick [AYZ95].

4.2.1 Randomized Algorithm

We show two slightly different approaches of the color coding technique for solving the k-path problem. First we show a straightforward way, which is improved in the following section.

Simple Version

The color coding technique described in [AYZ95] is a randomized algorithm for the k-path problem. In the first step the vertices of the input graph are colored randomly with $k+1$ colors such that the color of each vertex is chosen uniformly at random from $\{0,\dots,k\}$ independently of the other vertices. The resulting coloring does not have to be a valid coloring, i.e. $c(v) \neq c(w)$ f.a. $\{v,w\} \in E$ may be violated.

The simplest way of finding a k-path through color coding is to check whether a path $P=(v_0,\dots,v_k)$ in G exists with $c(v_i)=i$ f.a. $i \in \{0,\dots,k\}$. Let A be the event that a fixed k-path in G is colored $0,\dots,k$.

$$\Pr[A] = \prod_{i\in\{0,\dots,k\}} \Pr[c(v_i)=i] = \prod_{i\in\{0,\dots,k\}} \frac{1}{k+1} = (k+1)^{-(k+1)} \qquad (4.2.1)$$

Such a path can then be found by removing edges whose vertices are in nonadjacent color classes. In the resulting graph is a $k+1$-partite graph in which a k-path with coloring $0,\dots,k$ can be found by DFS starting at all vertices v where $c(v)=0$.

Let $p=\Pr[A]$, so $p=(k+1)^{-(k+1)}$.

Since

$$(1-p)^{\frac{1}{p}} < \left(e^{-p}\right)^{\frac{1}{p}} = e^{-1} \qquad (4.2.2)$$

the algorithm finds a k-path with probability at least $1-e^{-1}$ after $\frac{1}{p}=(k+1)^{(k+1)}$ repetitions. The probability of not finding a k-path in a graph that contains one can be made arbitrarily small by repeated execution.

Advanced Version

A more sophisticated version of color coding uses *colorful paths*. A colorful path is a path whose vertices are colored pairwise distinct. Let B be the event that at least one k-path is colored using all $k+1$ colors. There are $(k+1)!$ ways to color a k-path colorfully.

Under the assumption that there exists a k-path,

$$\Pr[B] \geq \frac{k+1!}{(k+1)^{k+1}} \underset{\text{Stirling}}{\geq} \frac{\sqrt{2\pi(k+1)}\left(\frac{k+1}{e}\right)^{k+1}}{(k+1)^{k+1}} = \frac{\sqrt{2\pi(k+1)}}{e^{k+1}} \geq e^{-(k+1)} \tag{4.2.3}$$

Using the argument from (4.2.2), if the algorithm is repeated e^{k+1} times, a k-path that is contained in G gets a colorful coloring with probability at least $1-e^{-1}$.

Comparing (4.2.1) and (4.2.3), the probability of coloring a k-path colorfully is much higher than coloring it $0,\dots,k$. Finding such a randomly colored colorful path however is a lot harder than finding a long path with colors in ascending order. We use a dynamic programming algorithm described in Algorithm 4.2.

Algorithm 4.2: Find colorful k-path

```
   Data: Graph $G=(V,E)$ with coloring $c : V \longrightarrow \{0,\dots,k\}$
   Result: Colorful $k$-path $p$ if existent, $\emptyset$ otherwise
 1 foreach $w \in V$ and $C \subseteq \{0,\dots,k\}$ do
 2     $P(w,C)=0$
 3     $P(w,\{c(w)\})=1$
 4 for $i=1\dots k$ do
 5     foreach pair $(w,C) : P(w,C)=1 \wedge |C|=i$ do
 6         if $u \in N(w)$ exists with $c(u) \notin C$ then
 7             $P(u, C \cup \{c(u)\})=1$
 8 if $u$ exists with $P(u,\{0,\dots,k\})=1$ then
 9     $p \leftarrow (u)$;
10     $C_p \leftarrow \{c(u)\}$;
11     while $|p| < k+1$ do
12         find $v \in N(u) \setminus p$ with $P\left(v,\{0,\dots,k\} \setminus C_p\right)=1$;
13         $p \leftarrow p \& v$;
14         $C_p \leftarrow C_p \cup \{c(v)\}$;
15         $u \leftarrow v$;
16     return $p$;
17 else
18     return $\emptyset$
```

First, the algorithm computes the $0/1$ function P on $V \times \{0,\dots,k\}$ that is defined by the rule that $P(w,C)=1$ if and only if there is a path of length $|C|$ ending in w and using exactly the colors in C. If there is u such that $P(u,\{0,\dots,k\})$, then u can be reached by a colorful path. The actual path p is then constructed in the following way: if u is the current vertex, we look in u's neighborhood for a vertex that has not been visited by p and that can be reached by a path using only the remaining colors (i.e., colors not on p yet).

We made a modification to the random coloring to find only k-paths containing a vertex v. This is achieved by coloring the vertices $V \setminus \{v\}$ like before using only colors $[k]$ and coloring v with $c(v) = 0$. Every colorful k-path now must contain v because it is the only vertex of its color.

4.2.2 Derandomization

A naïve way of derandomizing the previous algorithm is considering each subset $V' \subseteq V$ of size $k+1$ and coloring its vertices pairwise distinctly and then search for a colorful path as in Algorithm 4.2 in to $G[V]$. This would yield a running time of $\mathcal{O}(\binom{n}{k+1}) = \mathcal{O}(2^n)$. This derandomization is impractical due to the exponential dependency on n.

This section describes a derandomization of the color coding algorithm presented in section 4.2.1 whose running time is exponential in the length k instead of the number of vertices n.

Definition 4.5 *A family $\mathcal{H}$ of functions $\{1,\ldots,n\} \longrightarrow \{1,\ldots,\ell\}$ is a ℓ-perfect family of hash functions if for every $S \subseteq \{1,\ldots,n\}$ with $|S| = \ell$, there is an $h \in \mathcal{H}$ such that $h(x) \neq h(y)$ for any $x, y \in S, x \neq y$.*

If there is a k-path P in the graph, then by using each hash function $h \in \mathcal{H}$ as coloring, we can guarantee that at least once the $k+1$ vertices on P will be mapped injectively into $\{0,\ldots,k\}$, so the path is colored colorfully.

The construction of this family used by Alon et. al. [AYZ95] is based on the construction from Schmidt and Siegel [SS90]. Its size is $2^{(\mathcal{O}(k))} \log n$ and can be constructed in polynomial time relative to its size. Another construction for k-perfect hash families is presented in [Che+07]. Their construction's size is $\mathcal{O}\left(6.4^k \cdot n\right)$.

Overall, the running time of the derandomized algorithm is $\mathcal{O}\left(12.8^k \cdot n\right)$ and thus a $\mathcal{FPT}$ algorithm for the k-path problem exists.

For implementation it might be impracitcal to deradomize the algorithm because large constant factors within the exponent are involved [DF13]. By repeated execution of the randomized algorithm often enough, the probability of error can be made arbitrarily small.

4.3 Gabow and Nie's Algorithm

The algorithm given by Gabow and Nie [GN08] is an approximation algorithm for the longest cycle problem in directed and undirected graphs. A long cycle describes a cycle whose length is at least some integer k that ist part of the input. It has a worst case running time of $2^{\mathcal{O}(k)} n \log n$ improving Bodlaender's bound of $\mathcal{O}\left(2^k k!\, n\right)$ [Bod93]. This enables us to find a cycle of logarithmic length in polynomial time.

The algorithm first performs a depth first search starting at a random vertex $r \in V$ labeling each vertex with its respective distance to r. After the DFS is performed, the algorithm searches for long fundamental cycles containing a specified vertex v within the DFS tree. Such a long cycle is found if the distances of two vertices in the DFS tree is at least k and these vertices are connected by an edge $e = \{x, y\}$. This is done by finding the lowest common ancestor $\text{lca}(x, y)$ of x and y and then linking $P[\text{lca}(x, y), x]$ with $\{x, y\}$ and $P[y, \text{lca}(x, y)]$. If no such cycle is found then the algorithm calls the color coding algorithm (Algorithm 4.2) on the input graph. This algorithm finds every k-path containing v. At least one k-path $P = (v_0, \dots, v_k)$ can then be extended into a k^+ cycle by performing a DFS to find a path from v_0 to v_k that contains no vertices of P other than v_0 and v_k.

Algorithm 4.3: Gabow and Nie's Algorithm

```
   Data: Graph $G = (V, E)$, vertex $v$, desired length $k$
   Result: $k^+$ cycle containing $v$ if found, $\emptyset$ otherwise
 1 choose random start vertex $r \in V$;
 2 $T :=$ DFS tree rooted at $r$;
 3 foreach non tree edge $\{x, y\} = e \in E$ do
 4     if $|d(x) - d(y)| \geq k$ then
 5         $T' \leftarrow T + e$;
 6         Find $k^+$ cycle $C$ in $T'$;
 7         if $v \in C$ then
 8             return $C$;
 9 Search for $k^+$ cycle $C$ containing $v$ with Color Coding (Algorithm 4.2) and DFS;
10 if long cycle $C$ found then
11     return $C$;
12 else
13     return $\emptyset$;
```

4.4 Björklund and Husfeldt's Algorithm

The algorithm presented by Björklund and Husfeldt [BH03] is a polynomial-time algorithm that finds paths of length $\Omega((\log L/\log\log L)^2)$ where L denotes the length of a longest path within the input graph. The input graph is decomposed into a tree consisting of long cycles and paths connecting those cycles. In their original paper Björklund and Husfeldt used Bodlaender's algorithm [Bod93] to find long cycles. This algorithm has a running time of $\mathcal{O}\left(2^k k!n\right)$ to find a cycle of length k. In a note at the end of their paper, the authors remarked that their algorithm's bound can be improved to $\Omega((\log^2 L/\log\log L))$ by using the algorithm by Gabow and Nie, described in the previous section, to find long cycles.

4.4.1 The Algorithm

The algorithm starts by constructing a cactus graph that contains cycles and paths. It starts at some randomly chosen vertex v and searches for a cycle containing this vertex using the algorithm by Gabow and Nie described in section 4.3. If such a cycle C is found, it is inserted into the block graph along with an edge $\{v,w\}$ where $w \in N(C)$ and the algorithm is recursively called on $G - C$ to find a cycle through w. If no cycle is found, the algorithm adds an edge $\{w,w'\}$, where $w' \in N(w)$ to the block graph, and searches for a cycle through w'. The lengths of these cycles are given as parameters to the algorithm by Gabow and Nie whose running time depends exponentially on the length.

Algorithm 4.4: Björklund and Husfeldt's Algorithm

```
Data: Graph $G = (V,E)$, $k \in \mathbb{N}$
Result: Block Graph $B$ with $V[B] = V[G]$
pick $v \in V$ uniformly at random
while $|V[B]| < n$ do
    search for $k^+$-cycle $C$ containing $v$ in $V[G] \setminus V[B]$;
    if $C \neq \emptyset$ then
        $V[B] \leftarrow V[B] \cup V[C]$;
        $E[B] \leftarrow E[B] \cup E[C]$;
        choose $\{u,v\}$ randomly where $u \in C, v \in N(C) \setminus V[B]$;
        $E[B] \leftarrow E[B] \cup \{\{u,v\}\}$;
    else
        $V[B] \leftarrow V[B] \cup \{v\}$;
        choose $u$ randomly from $N(v) \setminus V[B]$;
        $E[B] \leftarrow E[B] \cup \{\{u,v\}\}$;
        $v \leftarrow u$;
return B;
```

We now transform B into a weighted tree T_k by replacing paths between vertices that lie on the same cycle by a weighted edge. Let v, w be vertices on some cycle C whose length is $\ell \geq k$. The weight that is attributed to the v, w-edge is the maximum of $d(v, w)$ and $\ell - d(v, w)$ where $d(\cdot, \cdot)$ is the distance between two vertices.

The longest path within the tree T_k can then be found using Algorithm 4.1.

4.4.2 Approximation Ratio

Lemma 4.6 (Björklund and Husfeldt [BH03]) *If a connected graph contains a path of length r, then every vertex is an endpoint of a path of length at least $\frac{1}{2}r$.*

The lower bound is given by the following lemma:

Lemma 4.7 (Björklund and Husfeldt [BH03]) *If G contains a path of length $r \geq 2^8$ starting in v, then T_k for*

$$k = \left\lceil \frac{2 \log r}{\log \log r} \right\rceil$$

contains a weighted path of length at least $\frac{1}{8}k^2 - \frac{1}{4}k - 1$.

Using Lemma 4.6 and inserting this k into the expression $\frac{1}{8}k^2 - \frac{1}{4}k - 1$ leads to a lower bound of $\Omega((\log L / \log \log L)^2)$ and an approximation ratio of

$$\mathcal{O}\left(L(\log \log L / \log L)^2\right) \subseteq \mathcal{O}\left(n(\log \log n / \log n)^2\right).$$

Where the approximation ratio is defined by the optimum divided by the length of the path found by the algorithm. By using Gabow and Nie's algorithm instead of Bodleaner's algorithm, the lower bound can be improved to $\Omega\left(\log L^2 / \log \log L\right)$.

Note that this approximation ratio depends on the length of a longest path within the graph compared to the color coding algorithm in section 4.2, whose approximation ratio depended on the number of vertices in the input graph. In a graph whose longest path has length $\mathcal{O}(\log n)$ the color coding algorithm would find an optimal path whereas the algorithm by Björklund and Husfeld would only guarantee an $\Omega\left(\log^2(\text{opt}) / \log \log(\text{opt})\right)$ approximation. In graphs with a longest path of length $\Omega(n)$ however, the algorithm by Björklund and Husfeld guarantees a superlogarithmic approximation.

4.5 Pongrácz's Algorithm

The below algorithm is based on an idea by Lajos L. Pongrácz presented in [Pon12]. The heuristic has been slightly modified in order to produce better results at the cost of an increase of runtime by factor n. The algorithm works in two stages, *create* and *search*.

In the first stage each vertex is being labeled with its respective distance to a root vertex, which is chosen randomly from the graph's vertex set. The algorithm therefor uses a Breadth First Search as shown in Algorithm 4.5.

Algorithm 4.5: create

```
Input: Undirected Graph $G = (V, E)$, root vertex $r \in V$
Output: Labeling $d_r : V \to \mathbb{N}$, $d_r(v) = d(r, v)$
$Q \leftarrow (r)$;
$X \leftarrow \emptyset$;
$d_r(r) \leftarrow 0$;
while $Q = (Q_1, \ldots, Q_k)$ with $k > 0$ do
    $v \leftarrow Q_1$; $Q \leftarrow (Q_2, \ldots, Q_k)$;
    $X \leftarrow X \cup \{v\}$;
    foreach $u \in N(v) \setminus X$ do
        $d_r(u) \leftarrow d_r(v) + 1$;
        $Q \leftarrow Q \& u$
return $d_r$
```

After labeling each vertex, the algorithm enters its second stage, which constructs a path by choosing locally optimal neighbors, i.e. incident vertices whose distance is maximal among the neighbors of the previously chosen vertex. If there is more than one neighbor with maximum distance, the algorithm picks one of them uniformly at random. If there are no unvisited neighbors, the length of the currently constructed path is compared with the longest path found so far and the latter is updated if necessary. Then the algorithm backtracks and continues, until all vertices have been visited.

Two modifications have been made to the second pass of the original algorithm in [Pon12] in order to improve its results:

1. The labeling step (4.5) is performed starting at each vertex (and then the second step is performed for each of the labelings) instead of choosing only one labeling with a random start vertex. This increases the running time of the overall algorithm by a factor of n, but also increases the number of constructed paths by the same number and can thereby yield longer paths. A similar approach can be used to start the second stage from each of the vertices and thus increasing the running

time by another factor n. This, however, has not shown itself to be useful.

2. In the original algorithm, there was no rule given on how to choose which vertex gets added in the search stage (Algorithm 4.6) if there is more than one neighbor with maximum distance to the root. In a naïve implementation the algorithm would choose the same vertex solely depending on the output of the create stage. This would produce the same results each time the entire algorithm is executed. Because of this, we implemented a straightforward tie breaker rule which chooses one of the neighbors with maximum distance uniformly at random to produce varying results in each iteration and thus increase the number of paths that can be constructed by the algorithm. This makes it possible to find longer paths by executing the algorithm multiple times.

Algorithm 4.6: search

```
Input: Undirected Graph $G = (V, E)$, labeling $d_r : V \to \mathbb{N}$
Output: Path $L \subseteq G$
choose $v \in V$ uniformly at random;
$L \leftarrow ()$;
$S \leftarrow (v)$;
$X \leftarrow \{v\}$;
while $S = (S_1, \dots, S_k)$ with $k > 0$ do
    if $Y \leftarrow N(S_k) \setminus X \neq \emptyset$ then
        $m \leftarrow \max_{u \in Y} d(u)$;
        pick $u$ uniformly at random from $\{u \in Y : d(u) = m\}$;
        $S \leftarrow S \& u$;
        $X \leftarrow X \cup \{u\}$;
    else
        if $|S| > |L|$ then
            $L \leftarrow S$;
        $S \leftarrow (S_1, \dots, S_{k-1})$;
return $L$;
```

Algorithm 4.7: Pongrácz

```
$L \leftarrow ()$;
foreach $r \in V$ do
    $d_r \leftarrow$ create$(G, r)$;
    $P \leftarrow$ search$(G, d_r)$;
    if $|P| > |L|$ then
        $L \leftarrow P$;
return $L$;
```

4.6 Pohl-Warnsdorf's Algorithm

Pohl-Warnsdorf's algorithm is an extension of Warnsdorf's rule which will be described in the first section of this section.

4.6.1 Warnsdorf's Rule

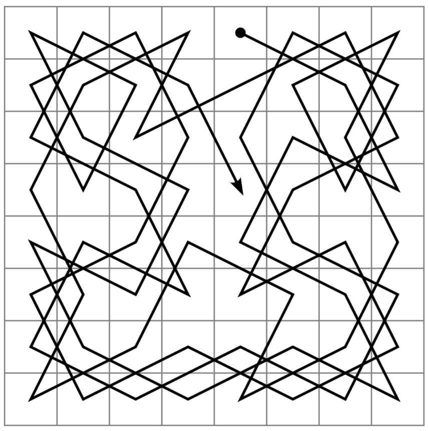


Figure 4.1: Knight's tour on a chess board

Warnsdorf's rule is a heuristic that finds kight's tours on a chessboard, i.e. a sequence of moves according to the rules of chess that visits each square exactly once. The algorithm also works for chessboard dimensions other than 8×8 [Con+94]. The chess board can be regarded as a graph in which each square is a vertex and two vertices are connected if there is a move possible between those vertices. A tour constructed by Warnsdorf's rule is a Hamiltonian path. Although the Hamiltonian path problem is $\mathcal{NP}$-hard, this algorithm constructs such a path in linear time. The algorithm constructs the Hamiltonian path by starting at an arbitrary vertex and then extending the path recursively by adding the neighbor that has the fewest unvisited neighbors.

4.6.2 Pohl's Extension

In [Poh67; PS04] an extension of Warnsdorf's rule is described. The extension gives a rule on how to extend the path if there is more than one possible vertex to be added according to Warnsdorf's rule. This tiebreak function is given in Algorithm 4.9. The tie break is achieved by considering the next layer, i.e. the neighbor's neighbors of the current vertex. The path is then extended with the vertex whose neighbor has fewest unvisited neighbors.

Algorithm 4.8: Warnsdorf's rule

```
Data: Graph $G = (V, E)$
Result: Path $P$ in $G$
foreach vertex $v \in V[G]$ do
    $P' \leftarrow (v)$;
    $G' \leftarrow G - \{v\}$;
    while $P' = (p_1, \ldots, p_i)$ with $i > 0$ do
        $v \leftarrow p_1$;
        $X \leftarrow \left\{ x \in N_G(v) \mid \forall_{x' \in N_G(v)} : \deg_{G'}(x') \geq \deg_{G'}(x) \right\}$;
        switch $|X|$ do
            case $|X| = 1$ do
                $v \leftarrow x,\ x \in X$;
                $P' \leftarrow P' \& v$;
                $G' \leftarrow G' - \{v\}$;
            case $|X| > 1$ do
                pick $v$ randomly from $\{v' \in X \mid v' = \arg\min_{x \in X} \text{tiebreak}(x)\}$;
                $P' \leftarrow P' \& v$;
                $G' \leftarrow G' - \{v\}$;
            case $X = \emptyset$ do
                if $|P'| > |P|$ then
                    $P \leftarrow P'$;
                $P' \leftarrow (p_1, \ldots, p_{i-1})$;
return $P$;
```

Algorithm 4.9: Tiebreak

```
Data: Graphs $G, G'$, vertex $v \in V[G]$
Result: integer tiebreak$(v)$
tiebreak$(v) \leftarrow \min_{w \in N_G(v)} \deg_{G'}(w)$;
return tiebreak$(v)$;
```

4.7 Streaming Algorithms and the Semi-Streaming Model

In the streaming model, the data is given as a stream that has no particular order. This stream can represent a tape or hard disk with fast sequential access and very slow random access. The machine that processes this data has a limited amount of memory, typically logarithmic in the input size. This makes the model a useful tool in a big data context. Sometimes the processing time per item is also constrained. This restriction can be useful in a context where the data stream cannot be halted, e.g., in a scenario where data is recieved from a satellite. Usually, it is desired to have a one-pass algorithm. Where one pass corresponds to reading the entire data stream once. However, sometimes it is admissable to have a small constant number of passes but the goal is always to keep the number of passes to a minimum.

The study of graph problems in streaming models started around the beginning of the 21st century, see [BKS02; Fei+05] for early works. The idea of using a linear amount of memory is due to Muthukrishnan [Mut05]. Up until then, the 'streaming' term was associated with sub-linear memory, which is not enough for many graph problems [Fei+05]. To emphasize the difference, the streaming model with linear RAM (that we use) is also referred to as the *semi-streaming model* in the literature.

Since then, many kinds of graph problems have been addressed, such as shortest paths, spanning trees, connectivity, cuts, matching, and vertex cover. Several lower bounds are known. Most importantly for us, Feigenbaum et al. [Fei+08] proved that any BFS algorithm computing the first k layers with probability at least $\frac{2}{3}$, requires more than $\frac{k}{2}$ passes if staying within $\mathcal{O}(n \cdot \operatorname{poly}\log(n))$ memory (see Guruswami & Onak [GO14] for improved lower bounds). This constitutes a substantial hurdle when transferring existing algorithms into the streaming model. To the best of our knowledge, longest paths have not been addressed before in a streaming model.

It must be emphasized that streaming techniques also make sense when the graph is of size $c \cdot n \cdot \log(n)$ if a streaming algorithm can guarantee to stay within $c' \cdot n \cdot \log(n)$ for $c' < c$. Therefore, we give a memory guarantee for our algorithm using concrete constants.

4.8 Previous and Related Work

Algorithms for the longest path problem have been studied extensively in the RAM model. We start by listing algorithms with proven guarantees. Bodlaender [Bod93] and Monien [Mon85] gave algorithms that find a path of length k (if it exists) in $\mathcal{O}(2^k k! n)$ and $\mathcal{O}(k! nm)$ time, respectively. Alon et al. [AYZ95] introduced the method of *color coding* and based on that gave an algorithm running in expected time $2^{\mathcal{O}(k)}n$. There is a recent randomized algorithm by Björklund et al. [Bjö+10] that given k, finds a path of length k (if it exists) in $\mathcal{O}(1.66^k \cdot \mathrm{poly} n)$ time (see also [Kou08; Wil09; Bjö14]). Those works show that the problem is fixed-parameter tractable: a path of length k can be found (if it exists) in polynomial time, for fixed k. The particular dependence of the running time on k (factorial or exponential) determines up to which k we stay polynomial and thus determines the length guarantee for a polynomial-time approximation algorithm.

In Hamiltonian graphs, a path of length $\Omega(\frac{\log(n)}{\log\log(n)}^2)$ can be found with the algorithm by Vishwanathan [Vis04]; and Feder et al. gave further results for sparse Hamiltonian graphs [FMS02]. Björklund and Husfeldt [BH03] gave an algorithm that finds a path of length $\Omega(\frac{(\log(\mathrm{OPT}))^2}{\log\log(\mathrm{OPT})})$, where OPT is the length of a longest path. It works by a decomposition of the graph into paths and cycles. Their technique subsequently was extended by Gabow [Gab07] and Gabow and Nie [GN08] yielding guarantees for the length of the path of $\exp(\Omega\sqrt{\frac{\log(\mathrm{OPT})}{\log\log(\mathrm{OPT})}})$ and $\exp(\Omega\sqrt{\log(\mathrm{OPT})})$, respectively. Apart from that, the field is dominated by heuristics, such as (Pohl-) Warnsdorf [Poh67; PS04] and Pongrácz [Pon12].

The Björklund-Husfeldt algorithm uses color coding as an important subroutine. We implemented and tested a simple algorithm based on color coding, which gave inferior results and more importantly took very long time to complete, substantially longer than (Pohl-) Warnsdorf, Pongrácz, or our algorithm. Thus we refrained from further implementing the Björklund-Husfeldt algorithm. (The original description in [BH03] uses Bodlaender's algorithm [Bod93], which has an even higher running time than color coding.) The Gabow-Nie algorithm [GN08] does not use color coding, but at the time of writing was only available as a short conference version, making it difficult to implement.

Several non-approximability results have been shown by Karger et al. [KMR97]: a constant-factor approximation is $\mathcal{NP}$-hard; and for any $\varepsilon > 0$, the LPP cannot be approximated with a ratio of $2^{\mathcal{O}(\log^{1-\varepsilon}(n))}$, unless $\mathcal{NP} \subseteq \mathrm{DTIME}(2^{\mathcal{O}(\log^{1/\varepsilon}(n))})$, that is, such an approximation is quasi-$\mathcal{NP}$-hard. Bazgan et al. showed that the same holds even when restricting to cubic Hamiltonian graphs [BST99].

The LPP is also interesting in directed graphs. For any $\varepsilon > 0$, it is $\mathcal{NP}$-hard to approximate in directed graphs within $n^{1-\varepsilon}$ [BHK04]. The best approximation guarantee in the directed case (unless restricting to special classes of graphs) is still the color coding algorithm that also works in the undirected case [AYZ95]. For special graph classes, there exist exact polynomial-time algorithms, e.g., for (undirected) trees (given by Dijkstra around 1960, see [Bul+02] for a proof), for directed acyclic graphs [SW11, pp. 661-666], for grid graphs [KBA12], and for cactus graphs [UU07; Mar+12].

4.9 Previous Algorithms

Trees. An algorithm for longest paths in trees was presented by Dijkstra around 1960; a proof of correctness can be found in [Bul+02]. It consists of two invocations of DFS, the first starting at an arbitrarily chosen vertex (e.g., chosen uniformly at random), and the second starting at a vertex that is in the final layer constructed by the first DFS.

Warnsdorf and Pohl-Warnsdorf. Warnsdorf's rule was originally presented in 1823 and is a DFS that always picks a neighbor with a minimum number of unvisited neighbors. In case there are multiple such neighbors to choose from, Pohl gave a refinement [Poh67; PS04]: we restrict to those neighbors which themselves have a minimum-degree neighbor. Each vertex is used once as the starting point of the DFS, and the best path found is returned. This gives a total runtime of $\mathcal{O}(nm)$.

Pongrácz. This algorithm was announced in 2012 [Pon12] and to the best of our knowledge has not been thoroughly studied since. We give a technically slightly modified description here. Given a start vertex r, using BFS we compute for each vertex v its distance to r. Then, starting at a randomly chosen v, we conduct a DFS that always picks an unvisited neighbor with maximum distance to r. Each vertex is used once as the start r, and the longest path found is returned. In the original version, for each r, also *each* v is tried (and not just one chosen randomly). In order to stay within $\mathcal{O}(nm)$, we decided to enumerate only one of the two possibilities: either r or v. In preliminary experiments, we found the choice given here (enumerate all r, pick one v randomly) to be superior. We leave a thorough study of the different variants of Pongrácz's algorithm for future work.

4.10 Description of Our Streaming Algorithm

Our algorithm works in two phases: (1) spanning tree construction, (2) spanning tree diameter improvement. Phase (1) is characterized by a parameter $\tau \in \mathbb{N}$ and a sequence $D = (1, \dots, q_1)$ of degree limits, where $q_1 \geq 2$ and $D_{q_1} = \infty$. For each $i \in [\tau] = \{1, \dots, \tau\}$, a tree T_i is constructed. We start with the empty graph $T_i = (V, \emptyset)$ and then add edges to T_i over a number of q_1 passes. In each pass $p \in [q_1]$, we add an edge to T_i iff that does not create any cycle and it does not increase the maximum degree in T_i beyond D_p. Since $D_{q_1} = \infty$, we arrive at a spanning tree eventually (recall that we assume all our input graphs to be connected). The motivation for the degree limit is to favor path-like structures over clusters of edges. As an extreme example, consider a complete graph. Without degree restriction, it is possible that a spanning tree is constructed that is a star; whereas with a degree restriction of 2, we find

Algorithm 4.10: Streaming Phase (1): Spanning Tree Construction

```
Input: connected graph $G = (V,E)$ as a stream of edges, parameter $\tau$,
       degree limit sequence $D = (\{D_1,\dots,D_{q_1})$
Output: spanning tree of $G$
1 foreach $i = 1,\dots,\tau$ do
2     $T_i := (V,\emptyset)$;
3     SpanningTree($T_i$);
4 $U := (V, \bigcup_{i=1}^{\tau} E(T_i))$;
5 find a long path $P$ in $U$ using Warnsdorf's algorithm;
6 $T := (V,E(P))$;
7 SpanningTree($T$);
8 return $T$;
```

a Hamilton path during the first pass.

In order to not just create the same tree τ times, in the first pass, we pick a number $r \in [m]$ uniformly at random (where $\{e_1,\dots,e_m\}$ is the stream of edges) and ignore any edges with an index smaller than r. Due to this offset for the first pass, it makes sense (but is not necessary) to use the same degree limit for the second pass. We will test $D = (2,\infty)$ and $D = (2,2,3,\infty)$ in experiments. By standard techniques (keeping track of the connected components), this algorithm can be implemented with a per-edge processing time of $\mathcal{O}(n)$: we can decide in $\mathcal{O}(1)$ if the current edge is to be inserted and if so, it takes $\mathcal{O}(n)$ to update connectivity information.

When all trees $T_1,\dots,T_\tau$ have been constructed, we build $U := (V,\bigcup_{i=1}^{\tau} E(T_i))$ by uniting them. This graph will in general contain cycles, but it has no more than τn edges. Since we construct U from trees, it is guaranteed to be connected and to span all the vertices of the input graph. In U, a long path P is constructed with a RAM algorithm; we use the Warnsdorf algorithm for this task. The final step of the first phase is to isolate P and then to build a spanning tree T around it using the same technique as for the trees $\{T_1,\dots,T_\tau\}$. Since we may assume that the constructions of $\{T_1,\dots,T_\tau\}$ are fed from the same passes, we thus have $2q_1$ passes for the first phase. We summarize phase (1) in Algorithm 4.10, which uses procedure SpanningTree, also given below. For a set X, we write $x :=_{\text{unif}} X$ to express that x is drawn uniformly at random from X.

When phase (1) is concluded, we determine a longest path P in the spanning tree T using the Dijkstra algorithm (section 4.9). In phase (2), we try to modify this tree in order that it admits longer paths than P. A number of additional passes is conducted. In order to save time, we developed a criterion based on which we only consider a fraction of the edges during those passes. We explored the two options: (i) consider each edge independently with probability $\frac{n}{m+1}$ (resulting in only $\mathcal{O}(n)$ edges being considered);

Procedure SpanningTree(T)

```
Input: forest $T$ on $V$, possibly empty
Output: spanning tree on $V$
1 $r :=_{\text{unif}} [m]$;
2 fast-forward the stream to position $r$;
3 for $p = 1, \dots, q_1$ do
4     while not at the end of the stream do
5         get next edge $vw$ from the stream;
6         if $T + vw$ is cycle-free and $\max\{\deg_T(v), \deg_T(w)\} < D_p$ then $T := T + vw$;
7         if $|T| = n - 1$ then break;
8     rewind the stream to its beginning;
```

or (ii) skip an edge if both endpoints are on the so-far longest path P. After preliminary experiments, we decided for option (ii) due to better solution quality at a moderate runtime expense, however we already give results for one variant of our algorithm using option (i).

For each edge e that is considered and that is not in T, we temporarily add e to T, creating a fundamental cycle C in $T' := T + e$. We want to go back to a tree. To this end, we have to remove an edge from C. This edge is chosen so that among all possibilities, the resulting tree has maximum diameter.

It should not be assumed that an edge with both endpoints on P could not yield an improvement. Intuitively, relative to P it acts like a shortcut, but examples can be found where adding such an edge (and subsequently removing one edge from the fundamental cycle) improves the diameter of the tree. Still, criterion (ii) has shown to be effective in practice.

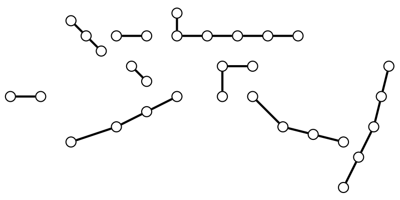
Phase (2) terminates after a preset number of passes q_2. We summarize phase (2) in Algorithm 4.11, where for any graph H, we denote $\ell(H)$ the length of a longest path in H.

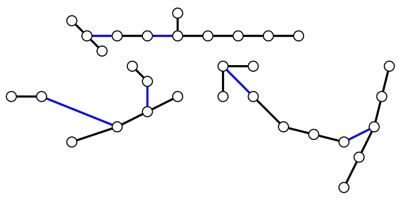
An example run of the Algorithm is shown in Figure 4.2.

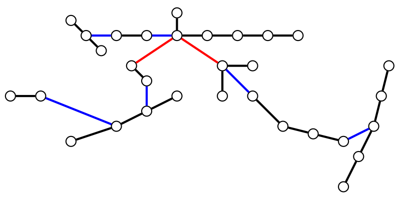
4.11 Properties of Our Streaming Algorithm

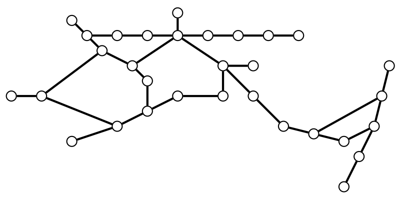
If the cycle C is of length $\Omega(n)$, then a naive implementation requires $\Omega(n^2)$ to find an edge e' to remove (temporarily remove each edge on the cycle and invoke the Dijkstra algorithm). However, we have:

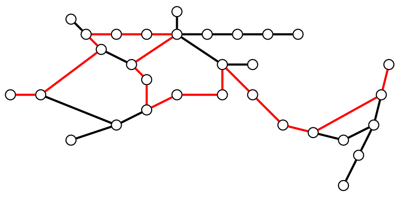
Theorem 4.8 *Phase (2) can be implemented with per-edge processing time $\mathcal{O}(n)$.*

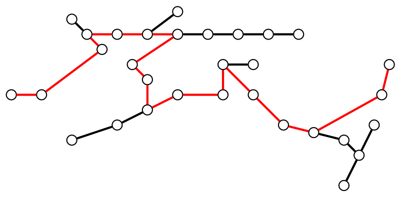

(a) Degree Limit $D = 2$

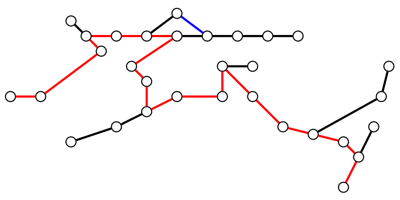

(b) Degree Limit $D = 3$

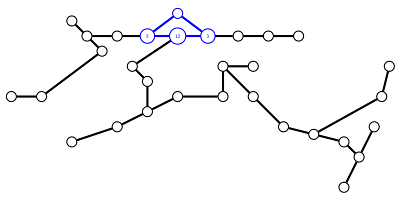

(c) Degrees Unlimited

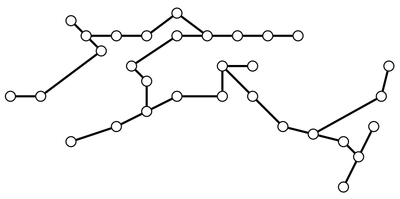

(d) Union of Trees


(e) Path Found by Warnsdorf's Algorithm


(f) New Spanning Tree Built around Path


(g) Added Edge from Stream


(h) Depths of Trees Extending from Fundamental Cycle


(i) Remove Edge from the Cycle

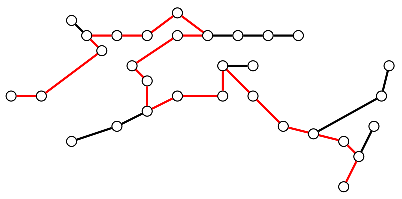

(j) New Longest Path

Figure 4.2: Example run of the Algorithm's Steps.

Algorithm 4.11: Streaming Phase (2): Improvement

```
Input: connected graph $G$ as a stream of edges, spanning tree $T$, pass limit $q_2$
Output: a (long) path in $G$
1  compute longest path $P$ in $T$ with Dijkstra algorithm;
2  for $q_2$ times do
3      rewind the stream to its beginning;
4      while not at the end of the stream do
5          get next edge $e = \{v, w\}$ from stream;
6          if $v \in V(P)$ and $w \in V(P)$ then discard and continue with next iteration;
7          $T' := T + e$;
8          compute fundamental cycle $C$ in $T'$;
9          $\ell^* := \max_{f \in E(C) \setminus \{e\}} \ell(T' - f)$;
10         if $\ell^* > \ell(P)$ then
11             pick any $e'$ from the set $\{f \in E(C) \setminus \{e\} : \ell(T' - f) = \ell^*\}$;
12             $T := T' - e'$;
13             update $P$ with longest path in $T$;
14 return $P$;
```

Proof. An $\mathcal{O}(n)$ bound is clear for all lines of Algorithm 4.11, except line 9 and line 11. Denote

$$\ell' := \max_{f \in E(C) \setminus \{e\}} \max\{\ell(P) : P \text{ is path in } T' - f \text{ and } e \in E(P)\}$$

and let $R' \subseteq E(C) \setminus \{e\}$ be the set of edges where this maximum is attained. Then the following implications hold: $\ell' \leq \ell(P) \implies \ell^* \leq \ell(P)$ and $\ell' > \ell(P) \implies \ell' = \ell^*$. This is because if a longest path in $T' - f$ is supposed to be longer than P, it must use e (since otherwise it would be a path in T). Hence it suffices to determine ℓ', and if $\ell' > \ell(P)$, to find an element of R'.

Denote $C = (v_1, \ldots, v_k)$ the fundamental cycle for some $k \in \mathbb{N}$ written so that $e = v_1 v_k$. When computing ℓ', we can restrict to paths in T' of the form

$$(\ldots, v_s, v_{s-1}, \ldots, v_1, v_k, v_{k-1}, \ldots, v_t, \ldots) \tag{4.11.1}$$

for $1 \leq s < t \leq k$, where v_s is the first and v_t is the last common vertex, respectively, of the path and C. For each i, let T_i be the connected component of v_i in $T - E(C)$, i.e., T_i is the part of T that is reachable from v_i without using the edges of C. Denote $\ell(T_i)$ the length of a longest path in T_i that starts at v_i and denote $c_i := \ell(T_i) + i - 1$ and $a_i := \ell(T_i) + k - i$. Then a longest path entering C at v_s and leaving it at v_t, as in (4.11.1), has length exactly $c_s + a_t$. Hence we have to determine a pair (s, t) such that $c_s + a_t$ is maximum (this maximum value is ℓ'); we call such a pair an *optimal pair*. If the so

```
compute $c_1,\dots,c_{k-1}$ and $\{a_2,\dots,a_k\}$ using DFS;
$M := 0$; $L := 0$;
for $i = 1,\dots,k-1$ do
    if $c_i > M$ then
        $M := c_i$;
        $s := i$;
    if $M + a_{i+1} > L$ then
        $L := M + a_{i+1}$;
        $t := i + 1$;
return $(s,t)$;
```

determined value ℓ' is not greater than $\ell(P)$, then nothing further has to be done (the edge e cannot give an improvement). Otherwise, having constructed our optimal pair (s,t), we pick an arbitrary edge (e.g., uniformly at random) from $\{v_i v_{i+1} : s \le i < t\}$, which are the edges between v_s and v_t on C. We show that the following algorithm computes the value ℓ' and an optimal pair in $\mathcal{O}(n)$.

The total of computations in line 1 can be done by DFS in $\mathcal{O}(n)$, and the loop in $\mathcal{O}(k) \le \mathcal{O}(n)$. We prove that the final (s,t) is optimal. For fixed t, the best possible length $c_s + c_t$ is obtained if t is combined with an $s < t$ where $c_s \ge c_j$ for all $j < t$. In the algorithm, for each t (when $t = i+1$ in the loop) we combine a_t with the maximum $\max_{j<t} c_j$ (stored in the variable M). Thus, when the algorithm terminates, $L = \ell'$ and $c_s + c_t = \ell'$.

□

Corollary 4.9 *Our streaming algorithm (with the two phases as in Algorithm 4.10 and Algorithm 4.11) can be implemented with a per-edge processing time of $\mathcal{O}(n)$.*

We turn to the memory requirement. Denote b the amount of RAM required to store one vertex or one pointer (e.g., $b = 32$bit or $b = 64$bit) and call $n \cdot b$ one *unit*.

Theorem 4.10 *Our streaming algorithm (with the two phases as in Algorithm 4.10 and Algorithm 4.11) conducts at most $2q_1 + q_2$ passes. Moreover, the algorithm can be implemented such that the RAM requirement is at most $(\max\{4\tau, 2\tau + 4\} \cdot n + c) \cdot b$ with a constant c.*

Proof. The construction of each of the initial trees $T_1,\dots,T_\tau$ can be fed from the same passes, so we obtain those τ trees within at most q_1 passes. After isolating the path P, we need at most q_1 more passes to get back to a spanning tree. A bound of q_2 for phase (2) is obvious. We turn to the memory requirement.

Phase (1). All the adjacency lists of one tree together require 2 units, plus 1 unit for an array of pointers to each of the lists. We need 1 additional unit per tree to store connectivity information during tree construction. This amounts to 4τ units for the main data structures at any time so far, plus a few extra bits required for bookkeeping (loop variables, etc) that are covered by the constant c. The graph U can be stored in $2\tau+1$ units (adjacency lists plus pointer array). For the Warnsdorf algorithm, we need 1 unit to store DFS information (e.g., store the DFS tree as a predecessor relation) and 1 unit for the best path so far. To determine the next vertex to visit (according to Warnsdorf's rule), we need $\Delta b \leq nb$ bits, where Δ is the maximum degree in the graph. This amounts to $2\tau+4$ units for the main data structures for the Warnsdorf algorithm on U. The rest of phase (1) is clearly covered by this as well.

Phase (2). We need 3 units to store the tree, 1 unit to store the longest path so far, 1 unit for the fundamental cycle, and 1 unit for DFS. This amounts to 6 units during this phase plus bookkeeping, which is covered by the stated bound. □

Remark 4.11 *On a graph with average degree d, the Warnsdorf, Pohl-Warnsdorf, and Pongrácz algorithms each requires at least $(d+3)\cdot nb$ bits of memory.*

Proof. Since those algorithms perform special variants of DFS (and Pongrácz also BFS), we cannot restrict them to sequential access and thus we have to load the instance into RAM as adjacency lists.[1] Hence, $d+1$ units are required to store the graph. Two more units must be allotted to store DFS information and the longest path found so far, in the case of Pongrácz need one more unit for the distance information. □

Corollary 4.12 *Not counting the additive constant c from Theorem 4.10, the RAM algorithms require $\frac{d+3}{\max\{4\tau,2\tau+4\}}$ times more RAM than our streaming algorithm, on a graph with average degree d. For $\tau=2$, this ratio is $\frac{d+3}{8}$.*

4.12 Test Instances

Connected Random. We denote this model by $\mathcal{G}^*(n,p)$. A graph is constructed by starting with a random tree on n vertices (via a randomly chosen Prüfer sequence) and then adding further edges as in $\mathcal{G}(n,p)$. The average degree in such a graph is slightly larger than np due to the $n-1$ initial tree edges.

[1]This is unless we invoke external-memory techniques, which is unexplored for the LPP at this time.

Chains. Parameters for a chain graph are n, p, and k, with n being a multiple of k. We create k graphs $G_1, \dots, G_k$, the *clusters*, according to $\mathscr{G}^*(n, p)$, each on n/k vertices. Then we insert an edge $v_i w_i$ with randomly chosen $v_i \in V(G_i)$ and $w_i \in V(G_{i+1})$ for each $1 \le i < k$, making sure that $w_i \neq v_{i+1}$. Such graphs pose a particular challenge to DFS-based LPP algorithms, since if the DFS visits the connecting point to the next cluster (w_i or v_i) too early, it will eventually miss out on a large number of vertices in the current cluster.

Preferential Attachment and Small World. Preferential attachment graphs are created as per the Barabási-Albert model [BA99]: parameters are $n, n_0, d \in \mathbb{N}$, where n is total the number of vertices, n_0 is the size of the initial tree, and in each step the new vertex is connected by d new edges. This model guarantees connectedness. Small world graphs are created as per the Watts-Strogatz model [WS98], with a small modification. Parameters are $n, d \in \mathbb{N}$, with d even, and $0 \le \beta \le 1$. We start with a ring lattice where each vertex is connected to each $d/2$ vertices on either side, then each edge $\{v, w\}$ with v and w not being next to each other on the ring is replaced with a random edge $\{v, u\}$ with probability β (the *rewiring probability*). Our modification (not to rewire certain edges) guarantees that the result is Hamiltonian (and in particular connected).

These two models were chosen since they yield very different degree distributions: for preferential attachment, we have a power-law and there exist a few *hubs*, i.e., vertices with high degree. In the small world model on the other hand, vertices tend to have similar degree.

Hyperbolic Geometric. Hyperbolic geometric graphs are a very interesting new class of graphs, for which efficient generators were given by von Looz et al. [Loo+15]. The graphs are constructed in hyperbolic space of constant negative curvature. Vertices correspond to points that are randomly inserted into this space, and an edge between two vertices is inserted if the corresponding points are within a certain distance from each other. This model has been shown to exhibit many features of complex real-world networks. We refer to [Kri+10; Loo+15] for details. Parameters are number of vertices n, average degree d, and the exponent γ of the power-law degree distribution. We use the generator implementation from [Loo+15]. Connectedness is ensured by initializing the graph with a random tree.

4.13 Experimental Setup

All algorithms were implemented using `C++14` and each graph stream is realized as a `std::vector` of pairs of 32 bit integers. We keep those vectors in RAM for the sake of faster running times and hence more experiments conducted — but it is guaranteed that we access those vectors only sequentially and all other data structures are $\mathcal{O}(n)$. Our implementation also allows to process graphs stored in a file on disk, without copying the contents of the file into RAM (it is accessed via a `std::ifstream`). Using the *Valgrind* tool *Massif*,[2] we verified that RAM consumption of our algorithm is indeed independent of the number of edges. For each instance, the stream of edges is randomized once and the order does not change between passes or between the invocations of the algorithms. Each implementation concludes immediately when a Hamilton path is found.

For each random graph model under consideration, we test three settings: $n = 16\,000$ and nominal average degree $d = 14$ (*sparse*); $n = 16\,000$ and nominal average degree $d = \sqrt[3]{n}$ (*dense*); and $n = 100\,000$ and nominal average degree $d = 10$ (*large*). (Note that chain and hyperbolic graphs will have a slightly larger average degree than the given d due to the additional tree that is used to guarantee connectedness.) The *dense* graphs have $\Omega(n^{4/3})$ edges and are thus beyond the theoretical RAM capacity of the semi-streaming model. More on the practical side, note that by Corollary 4.12 (not counting the small additive constant), even for average degree $d = 14$, the RAM algorithms require more than two times more memory than ours when configured with $\tau \leq 2$. Due to lack of space we skip the details for *sparse* and *dense* small world graphs, and we only use a selection of algorithms for the *large* graphs.

We run Warnsdorf, Pohl-Warnsdorf, Pongrácz, the simple randomized DFS, and different variants of our algorithm on 100 randomly generated instances for each parameter set (only 50 instances for *large* graphs in order to save time) and record the length of the path that is found and the running time. Variants of our algorithm are denoted in the form $\tau/q_1/q_2$, where τ is the number of trees in the beginning, q_1 is the maximum number of passes used to construct a spanning tree using degree limiting, and q_2 is the number of improvement passes. In order to save time, for fixed τ and q_1, we obtain results for $\tau/q_1/0$ up to $\tau/q_1/q_2$ by running $\tau/q_1/q_2$ and recording intermediate results.

Solution quality is analyzed in terms of *relative solution quality*. For an instance I and algorithm A, denote $\ell(A,I)$ the length of the path delivered by A on I. Then we define $\rho(A,I) := \frac{\ell(A,I)}{\max_{A'} \ell(A',I)} \in [0,1]$, where A' runs over all algorithms under investigation. That is the result of A divided by the best result on any of the algorithms. Clearly, one algorithm per instance will always have relative solution

[2] `http://valgrind.org/docs/manual/ms-manual.html`

quality 100%.

4.14 Data and Discussion

Tables with detailed experimental data can be found in section 4.15. The column labeled "ℓ" gives statistics (mean value μ and standard deviation σ) for the lengths of the paths found and is intended as a general orientation in which range our solutions are located. The column labeled "wins" counts how many times this algorithm delivered the best solution, i.e., how many times it achieved relative solution quality $\rho = 100\%$. Detailed statistics are given for the relative performance in the following columns: mean value, standard deviation, minimum, 5th and 10th percentile, and median. We use percentile notation everywhere: P_0 for the minimum, P_5 and P_{10} for the 5th and 10th percentile, and P_{50} for the median. In the final two columns, we give the running time in seconds. The algorithm marked with a star ($2/4/3^*$) uses the randomized criterion for skipping edges in the improvement phase, whereas all other variants of our algorithm use the path criterion as stated in Algorithm 4.11. In the following, we distill the data from the tables into several observations and conclusions.

- The fact that the simple randomized DFS algorithm (denoted "DFS" in the tables) delivers clearly inferior results in many cases is an indication that at least those instances are not "too easy".
- Warnsdorf and Pohl-Warnsdorf are generally best, except for chain graphs. For many of the instances, they find a Hamilton path, and then they are very fast, sometimes below one second. Note that this advantage could easily be removed by making the graphs non-Hamiltonian, e.g., by connecting two additional vertices as leafs to the same vertex.
- Warnsdorf and Pohl-Warnsdorf are close to each other in terms of solution quality, but unsurprisingly the former is faster.
- In terms of the average path length $\mu(\ell)$, for each set of parameters there is one algorithm that delivers at least $0.84 \cdot n$, i.e., 84% of a Hamilton path. It follows that a good relative performance also means a good absolute performance.
- Our strongest variant, 2/4/3, with the exception of preferential attachment graphs, always delivers a relative solution quality of at least 71%. For preferential attachment, we record a minimum of 49% in Table 4.3. In terms of the 5th percentile, i.e., after removing the 5% worst cases, and omitting preferential attachment graphs, our minimum relative solution quality is 83%. In terms of the 10th percentile and including preferential attachment graphs, we still have at least 71%. In

terms of mean and median, we have at least 83%.

- Regarding running time, we compare our variant 2/4/3 with Warnsdorf, which is the fastest RAM algorithm, not counting the simple randomized DFS. Clearly, we cannot compete in cases where Warnsdorf finds a Hamilton path within a second, but as remarked before, this advantage of Warnsdorf could easily be removed by making the graph non-Hamiltonian. Apart from those cases, in the *sparse* and *dense* sets, the biggest difference is for *sparse* hyperbolic graphs, where Warnsdorf only needs about 56% of our running time on average. For *dense* chains, we are faster than Warnsdorf. For the *large* set, our variant 2/4/1 has similar running times as Warnsdorf, while delivering at least 71% in terms of P_{10}, and when excluding preferential attachment graphs it delivers 82% in terms of P_5. More than one improvement pass here only gives incremental gain, so in order to save time on large graphs, the variant 2/4/1 is recommended over 2/4/2 or 2/4/3.

- Using $\tau = 2$ has a clear advantage over $\tau = 1$, in particular compare 1/2/0 with 2/2/0 in terms of ℓ in Table 4.1 and Table 4.2.

- The degree-limiting technique yields substantial improvements. For $q_1 = 2$ (i.e., for variants of the form $\tau/2/q_2$), we use the sequence $D = (2, \infty)$, i.e., in the first pass we limit the degree to 2 and in the second pass we have no limit. In the configuration with $q_1 = 4$ we use $D = (2, 2, 3, \infty)$. Comparing for example 1/2/0 with 1/4/0 with respect to ℓ in Table 4.2 for preferential attachment and hyperbolic graphs, we see that 1/2/0 delivers roughly $50 - 60\%$ length on average compared to 1/4/0. Comparing 2/2/3 with 2/4/3 in particular with respect to P_0, P_5, and P_{10} for preferential attachment graphs in Table 4.1, we see that $q_1 = 4$ brings an improvement even on top of the improvement gained by using $\tau = 2$ and by the improvement phase.

- The improvement phase (phase (2)) can bring further improvements, in particular with respect to P_0. This is seen for example by comparing 2/4/0 with 2/4/3 for preferential attachment and hyperbolic graphs in Table 4.1.

- Comparing the runtimes of 2/2/3 and 2/4/3 over all tables, we find that consistently the former is slower, while delivering inferior solutions. The same goes for 1/4/3 and 2/4/3; here the difference in running time is very high for preferential attachment graphs. This shows that a lack of effort in phase (1) can make phase (2) substantially slower. An explanation is that more improvement steps have to be carried out.

- Comparing 2/4/3 with 2/4/3*, we find the former being consistently better in terms of solution quality, but requiring up to roughly 30% more time.

- Our biggest advantage (using 2/4/3) over the other algorithms is for chain graphs.

In particular, we conclude from those observations that none of the three features (namely using multiple trees, degree-limiting, and improvement) should be missed. The combination of all those features makes our algorithm competitive.

4.15 Tables of Experimental Data

On the following pages please find tables of results for the experiments as discussed in section 4.14. By μ we denote the mean value and by σ the standard deviation. By P_i we denote the ith percentile, in particular P_0 is the minimum and P_{50} is the median. By ℓ we denote the path length and by ρ the relative performance. Running times t (last two columns) are in seconds. For further explanations, please see section 4.14.

Table 4.1: *Sparse Set:* $n = 16\,000$ and $d = 14$

		ℓ			ρ in %						t	
graph class	algo	μ	σ	wins	μ	σ	P_0	P_5	P_{10}	P_{50}	μ	σ
	1/2/0	3 032	645	0	20	4	11	12	13	21	26	2
	1/2/3	13 435	606	0	89	4	79	80	82	91	185	34
	1/4/0	3 947	222	0	26	1	22	24	24	26	23	0
	1/4/3	14 150	55	0	94	1	93	93	93	94	141	4
chain	2/2/0	10 985	1 336	0	73	9	17	58	64	75	46	4
$k = 125$	2/2/3	14 522	373	7	96	2	91	92	93	97	129	18
$l = 128$	2/4/0	11 683	617	0	77	4	63	68	73	78	48	4
$p = 0{,}11$ $n = 16\,000$	2/4/3	15 062	122	93	100	0	98	100	100	100	103	3
$m \approx 127\,044$	2/4/3*	14 346	283	0	95	2	86	90	94	96	92	4
	Pon	14 518	52	0	96	1	95	95	95	96	110	4
	War	10 598	304	0	70	2	66	67	68	70	86	2
	PW	10 539	306	0	70	2	65	67	68	70	131	3
	DFS	9 255	165	0	61	1	59	60	60	61	38	1
	1/2/0	464	137	0	3	1	1	2	2	3	35	6
	1/2/3	6 653	1 212	0	42	8	27	27	28	43	437	23
	1/4/0	748	105	0	5	1	3	4	4	5	29	0
	1/4/3	8 281	122	0	52	1	50	50	51	52	394	8
	2/2/0	12 712	2 350	0	79	15	15	43	58	85	47	3
pref. attach. $n_0 = 7$	2/2/3	13 000	1 859	0	81	12	36	53	65	86	169	71
$d = 14$	2/4/0	13 743	1 825	0	86	11	18	66	76	90	46	4
$n = 16\,000$	2/4/3	14 060	1 100	0	88	7	55	73	79	91	133	44
$m = 111\,957$	2/4/3*	13 817	1 265	0	86	8	51	69	75	90	104	3
	Pon	13 565	28	0	85	0	84	84	85	85	113	4
	War	16 000	0	100	100	0	100	100	100	100	0	0
	PW	16 000	0	100	100	0	100	100	100	100	0	0
	DFS	12 385	27	0	77	0	77	77	77	77	41	1
	1/2/0	586	185	0	4	1	1	2	2	4	33	22
	1/2/3	9 791	826	0	61	5	49	50	54	62	337	38
	1/4/0	968	144	0	6	1	4	5	5	6	25	0
	1/4/3	10 987	145	0	69	1	67	67	67	69	290	8
hyperbolic	2/2/0	12 822	1 808	0	80	11	37	47	62	84	46	4
$d = 14$	2/2/3	14 099	1 013	0	88	6	67	71	77	90	130	41
$\gamma = 3$	2/4/0	13 732	941	0	86	6	56	72	78	88	46	5
$n = 16\,000$	2/4/3	14 646	509	0	92	3	77	84	86	93	108	19
$m \approx 128\,185$	2/4/3*	14 303	587	0	89	4	69	82	85	91	97	3
	Pon	14 373	106	0	90	1	87	89	89	90	117	5
	War	15 997	5	92	100	0	100	100	100	100	61	38
	PW	15 998	4	94	100	0	100	100	100	100	83	52
	DFS	12 908	22	0	81	0	80	80	81	81	40	1

Table 4.2: *Dense Set:* $n = 16\,000$ and $d = \sqrt[3]{n}$

		ℓ			ρ in %						t	
graph class	algo	μ	σ	wins	μ	σ	P_0	P_5	P_{10}	P_{50}	μ	σ
	1/2/0	3 749	860	0	24	6	11	13	15	26	25	2
	1/2/3	14 633	438	0	94	3	86	87	89	95	200	52
	1/4/0	4 666	314	0	30	2	24	27	27	30	23	1
	1/4/3	15 079	34	0	97	0	97	97	97	97	144	4
chain	2/2/0	11 646	755	0	75	5	57	62	70	77	49	5
$k = 125$	2/2/3	15 248	251	12	98	2	92	95	96	99	139	26
$l = 128$	2/4/0	11 864	867	0	77	5	40	67	69	78	50	6
$p = 0{,}21$ $n = 16\,000$	2/4/3	15 489	67	88	100	0	99	100	100	100	112	5
$m \approx 222\,351$	2/4/3*	14 715	135	0	95	1	91	93	94	95	99	5
	Pon	15 034	39	0	97	0	96	96	97	97	165	10
	War	10 420	313	0	67	2	63	64	65	67	126	5
	PW	10 380	315	0	67	2	62	64	65	67	208	6
	DFS	9 348	142	0	60	1	58	59	59	60	52	3
	1/2/0	639	176	0	4	1	2	2	2	4	32	5
	1/2/3	8 783	1 238	0	55	8	34	39	43	56	721	67
	1/4/0	1 037	151	0	6	1	4	5	5	6	27	1
	1/4/3	10 576	113	0	66	1	64	65	65	66	604	16
	2/2/0	14 248	1 587	0	89	10	26	72	83	92	50	3
pref. attach. $n_0 = 13$	2/2/3	14 534	969	0	91	6	59	80	85	93	182	80
$d = 26$	2/4/0	14 878	720	0	93	5	65	84	91	94	51	4
$n = 16\,000$	2/4/3	15 102	422	0	94	3	80	88	93	95	139	33
$m = 207\,843$	2/4/3*	15 004	450	0	94	3	73	88	91	95	111	4
	Pon	14 754	18	0	92	0	92	92	92	92	165	13
	War	16 000	0	100	100	0	100	100	100	100	0	0
	PW	16 000	0	100	100	0	100	100	100	100	0	0
	DFS	13 923	14	0	87	0	87	87	87	87	55	4
	1/2/0	737	242	0	5	2	1	2	3	5	29	5
	1/2/3	11 818	852	0	74	5	60	62	67	75	458	59
	1/4/0	1 304	188	0	8	1	6	6	7	8	25	1
	1/4/3	12 885	122	0	81	1	78	79	80	81	369	13
hyperbolic	2/2/0	13 867	1 090	0	87	7	48	69	80	89	49	4
$d = 26$	2/2/3	14 998	480	0	94	3	81	87	90	95	139	38
$\gamma = 3$	2/4/0	14 299	554	0	89	3	64	84	89	90	50	4
$n = 16\,000$	2/4/3	15 237	210	0	95	1	88	93	95	96	119	16
$m \approx 224\,369$	2/4/3*	14 727	423	0	92	3	77	87	89	93	101	4
	Pon	14 971	77	0	94	0	91	93	93	94	169	10
	War	16 000	0	100	100	0	100	100	100	100	0	0
	PW	16 000	0	100	100	0	100	100	100	100	0	0
	DFS	13 769	19	0	86	0	86	86	86	86	55	3

Table 4.3: *Large Set:* $n = 100\,000$ and $d = 10$

		ℓ			ρ in %						t	
graph class	algo	μ	σ	wins	μ	σ	P_0	P_5	P_{10}	P_{50}	μ	σ
chain $k = 1\,000$ $l = 100$ $p = 0{,}01$ $n = 100\,000$ $m \approx 599\,468$	2/4/0	69 738	8 809	0	82	8	35	69	72	84	2 345	387
	2/4/1	84 335	3 661	0	99	0	98	99	99	99	3 997	395
	2/4/2	84 884	3 511	0	100	0	100	100	100	100	5 334	603
	2/4/3	84 909	3 500	50	100	0	100	100	100	100	6 591	833
	War	68 212	2 071	0	80	4	76	77	77	79	3 587	192
pref. attach. $n_0 = 5$ $d = 10$ $n = 100\,000$ $m = 499\,979$	2/4/0	80 190	8 064	0	82	8	49	58	71	86	2 612	778
	2/4/1	80 380	7 681	0	82	8	51	59	71	86	4 385	1 308
	2/4/2	80 510	7 459	0	82	8	53	60	71	86	5 974	1 957
	2/4/3	80 606	7 322	0	83	7	54	60	71	86	7 465	2 660
	War	97 685	52	50	100	0	100	100	100	100	4 110	588
hyperbolic $d = 10$ $\gamma = 3$ $n = 100\,000$ $m \approx 599\,680$	2/4/0	84 202	4 582	0	85	5	60	76	82	87	2 641	358
	2/4/1	88 147	3 772	0	89	4	68	82	86	91	3 978	591
	2/4/2	88 426	3 519	0	90	4	70	82	87	91	5 015	858
	2/4/3	88 494	3 400	0	90	3	71	83	87	91	5 963	1 076
	War	98 710	84	50	100	0	100	100	100	100	3 910	294
small world $d = 10$ $\beta = 0{,}3$ $n = 100\,000$ $m = 500\,000$	2/4/0	86 928	4 924	0	89	5	68	80	83	91	2 441	401
	2/4/1	90 810	4 038	0	93	4	76	86	89	95	3 457	531
	2/4/2	91 171	3 758	0	94	4	78	86	89	95	4 275	837
	2/4/3	91 253	3 590	0	94	4	79	87	89	95	5 072	1 148
	War	97 212	44	50	100	0	100	100	100	100	3 357	223

Chapter 5

A Randomized Approximation for the Set Multicover Problem in Hypergraphs

5.1 Introduction

We consider the b-MULTICOVER problem in hypergraphs, where a hypergraph $\mathcal{H} = (V, \mathcal{E})$ consisting of a finite set V of vertices, a set of (hyper) edges $\mathcal{E} \subseteq 2^V$, and $b \in \mathbb{N}$ are given as input. The b-MULTICOVER problem is the problem of finding a minimum cardinality set of edges $C \subseteq \mathcal{E}$ such that each vertex $v \in V$ is covered by at least b edges. The special case $b = 1$ is the SETCOVER problem, which is a classical combinatorial problem that is part of Karp's 21 $\mathcal{NP}$-complete problems [Kar72]. Because of the hardness of the special case, the more general version is also $\mathcal{NP}$-hard and thus, we cannot give an exact polynomial time algorithm unless $\mathcal{P} = \mathcal{NP}$. Instead, we present an approximation algorithm based on an integer programming formulation of the problem, using both deterministic and randomized rounding along with an additional repair step.

Hypergraph	Approximation ratio	
—	$H(\ell)$	[Vaz13, pp. 112–116]
bounded ℓ	$H(\ell)-\frac{1}{6}$	[FK05]
—	δ	[HH86; PSW93]
—	$\left(1-\left(\frac{c}{n}\right)^{\frac{1}{\delta}}\right)\cdot\delta$ for const. $c>0$.	[PSW97]
$\ell\in\mathcal{O}\left(\max\left\{(nb)^{\frac{1}{5}},n^{\frac{1}{4}}\right\}\right)$	$\left(1-\frac{11(\Delta-b)}{72\ell}\right)\cdot\delta$	[EMS16]
$b\geq 2$ and $\delta\geq 3$	$\max\left\{\frac{148}{149}\delta,\left(1-\frac{(b-1)e^{\frac{\delta}{4}}}{94\ell}\right)\delta\right\}$	[Gor+21]
$b\geq 2$, $\delta\geq 3$, and $\ell\in\mathcal{O}\left(n^{\frac{1}{2}}\right)$	$\max\left\{\frac{19}{24}\delta,\left(1-\frac{(b-1)e^{\frac{\delta+1}{2}}}{32\ell}\right)\delta\right\}$	(this thesis)

Table 5.1: Fundamental results and approximations for SET MULTICOVER problem

5.2 Definitions and Preliminaries

Let $\mathcal{H}=(V,\mathcal{E})$ be a hypergraph, V and $\mathcal{E}$ is the set of vertices and hyperedges, respectively. We denote the number of vertices of $\mathcal{H}$ by $n:=|V|$. For every vertex $v\in V$ we define the vertex degree of v as $d(v):=|\{E\in\mathcal{E}\mid v\in E\}|$ and $\Gamma(v):=\{E\in\mathcal{E}\mid v\in E\}$ the set of edges incident in v. The maximum vertex degree is $\Delta:=\max_{v\in V}d(v)$. Let ℓ denote the maximum cardinality of a hyperedge from $\mathcal{E}$. It is convenient to order the vertices and edges, i.e., $V=\{v_1,\dots,v_n\}$ and $\mathcal{E}=\{E_1,\dots,E_m\}$, and to identify the vertices and edges by their indices.

Let us now give a formal definition of the multicover problem.

Problem 5.1 (SET MULTICOVER) *Let $\mathcal{H}=(V,\mathcal{E})$ be a hypergraph and $(b_1,\dots,b_n)\in\mathbb{N}^n_{\geq 2}$. We call $C\subseteq\mathcal{E}$ a set multicover if every vertex $i\in V$ is contained in at least b_i hyperedges of C.* SET MULTICOVER *is the problem of finding a set multicover with minimum cardinality.*

For the concentration of a random variable around its mean we will use the famous bounded differences inequality due to C. McDiardmid in the analysis of our algorithm:

Theorem 5.2 ([McD89]) *Let $X=(X_1,X_2,\dots,X_n)$ be a family of independent random variables with X_k taking values in a set A_k for each k. Suppose that the real-valued function f defined on $A_1\times\cdots\times A_n$ satisfies $|f(x)-f(x')|\leq c_k$ for every pair of vectors x and x' that differ only in the k-th coordinate. Then for any $t>0$,*

$$\Pr[f(X)\geq\mathbb{E}[f(X)]+t]\leq\exp\left(\frac{-2t^2}{\sum_{k=1}^{n}c_k^2}\right).$$

5.3 The Randomized Rounding Algorithm

Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph with maximum vertex degree Δ and maximum edge size ℓ. An integer (linear) programming formulation of SET MULTICOVER problem is the following:

$$\text{ILP}(\mathbf{b}): \quad \begin{array}{lll} \text{minimize} & \sum_{j=1}^{m} x_j, & \\ \text{subject to} & \sum_{j=1}^{m} a_{ij} x_j \geq b_i & \text{for all } i \in [n], \\ & x_j \in \{0,1\} & \text{for all } j \in [m], \end{array}$$

where $A = (a_{ij})_{i\in[n], j\in[m]} \in \{0,1\}^{n\times m}$ is the vertex-edge incidence matrix of $\mathcal{H}$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ is the positive integer vector that was given as part of the input. We define $b := \min_{i\in[n]} b_i$ and we set $\delta := \Delta - b + 1$. The linear programming relaxation LP($\mathbf{b}$) of ILP($\mathbf{b}$) is given by allowing $x_j \in [0,1]$ for all $j \in [m]$. Let Opt, resp. Opt* be the value of an optimal solution to ILP($\mathbf{b}$), resp. LP($\mathbf{b}$). Let $(x_1^*, \ldots, x_m^*)$ be the optimal solution for LP($\mathbf{b}$). So $\text{Opt}^* = \sum_{j=1}^{m} x_j^*$ and $\text{Opt}^* \leq \text{Opt}$.

Lemma 5.3 ([PSW93]) *Let b_i, d, Δ, $n \in \mathbb{N}$ with $2 \leq b_i \leq d-1 \leq \Delta - 1$, $i \in [n]$ and let $x_j \in [0,1]$, $j \in [d]$ such that $\sum_{j=1}^{d} x_j \geq b_i$. Then, at least b_i of the x_j satisfy $x_j \geq \frac{1}{\delta}$.*

In the next lemma we show that the $b_i - 1$ largest values of the LP solution are at least $\frac{2}{\delta+1}$. This will later ensure that the rounded LP solution delivers a feasible $b-1$ set cover.

Lemma 5.4 *As in the previous lemma, let b_i, d, Δ, $n \in \mathbb{N}$ with $2 \leq b_i \leq d-1 \leq \Delta-1$, $i \in [n]$ and let $x_j \in [0,1]$, $j \in [d]$ such that $\sum_{j=1}^{d} x_j \geq b_i$. Then, at least $b_i - 1$ of the x_j values satisfy $x_j \geq \frac{2}{\delta+1}$ and there exists one additional x_j, which fulfills $x_j \geq \frac{1}{\delta}$.*

Proof. Without loss of generality, let $x_1 \geq x_2 \cdots \geq x_{b_i} \geq \cdots \geq x_d$. We have

$$b_i - 2 \geq \sum_{j=1}^{b_i-2} x_j \quad \text{and} \quad (d - b_i + 2)x_{b_i-1} \geq \sum_{j=b_i-1}^{d} x_j. \tag{5.3.1}$$

Thus,

$$\begin{aligned} b_i - 2 + (\Delta - b + 2)x_{b_i-1} &\overset{b\leqslant b_i}{\geq} b_i - 2 + (\Delta - b_i + 2)x_{b_i-1} \\ &\overset{\Delta\geqslant d}{\geq} b_i - 2 + (d - b_i + 2)x_{b_i-1} \\ &\overset{(5.3.1)}{\geq} \sum_{j=1}^{b_i-2} x_j + \sum_{j=b_i-1}^{d} x_j \end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=1}^{d} x_j \\
&\geq b_i. && \text{(assumption of the lemma)}
\end{aligned}
$$

So

$$
\begin{aligned}
& & b_i - 2 + (\Delta - b_i + 2)x_{b_i-1} &\geq b_i \\
&\Leftrightarrow & (\Delta - b_i + 2)x_{b_i-1} &\geq 2 \\
&\Leftrightarrow & (\delta + 1)x_{b_i-1} &\geq 2 && \text{(definition of } \delta\text{)} \\
&\Leftrightarrow & x_{b_i-1} &\geq \frac{2}{\delta+1}.
\end{aligned}
$$

Recall that the indices are chosen in such a way that the values are descending. Thus, we have $x_j \geq x_{b_i-1} \geq \frac{2}{\delta+1}$ for all $j \in [b_i - 1]$ and Lemma 5.3 gives $x_{b_i} \geq \frac{1}{\delta}$.

□

In this section we present an algorithm using randomized rounding and afterwards repairing the solution to meet feasibility.

Algorithm 5.1: SET b-MULTICOVER

```
Input  : Hypergraph $\mathcal{H} = (V, \mathcal{E})$ with maximum degree $\Delta$ and maximum hyperedge size $\ell$.
         Let $b_i \in \mathbb{N}_{\geq 2}$ for $i \in [n]$, $b := \min_{i \in [n]} b_i$, and $\delta = \Delta - b + 1$.
Output: set MultiCover $C$
1  $C \leftarrow \emptyset$, $\lambda \leftarrow \frac{\delta+1}{2}$, and $\alpha \leftarrow \frac{(b-1)\delta e^{\frac{\delta+1}{2}}}{16\ell}$
2  Obtain an optimal solution $x^* \in [0,1]^m$ by solving the LP($\mathbf{b}$) relaxation
3  $C_1 \leftarrow \{E_j \in \mathcal{E} \mid x_j^* \geq \frac{1}{\lambda}\}$, $C_2 \leftarrow \{E_j \in \mathcal{E} \mid \frac{1}{\lambda} > x_j^* \geq \frac{1}{\delta}\}$, and $C_3 \leftarrow \{E_j \in \mathcal{E} \mid 0 < x_j^* < \frac{1}{\lambda}\}$
4  $C \leftarrow C_1$
5  if $|C_1| \geq \alpha \cdot \mathrm{Opt}^*$ then
6      return $C \leftarrow C_1 \cup C_2$
7  else
8      foreach $E_j \in C_3$ do
9          add $E_j$ to $C$ with probability $\lambda x_j^*$
10     Repair the cover $C$ (if necessary) as follows: Include arbitrary edges from $C_3$, incident in vertices
         $i \in [n]$ not covered by $b_i$ edges, to $C$ until all vertices are fully covered.
11     return $C$
```

We now give a brief explanation of the algorithm SET b-MULTICOVER, (Algorithm 5.1). We start with an empty set C, which will be extended to a feasible set multicover. First, we solve the LP-relaxation LP($\mathbf{b}$), this can be done in polynomial time using polynomial-time LP solvers. Note that the execution time of the subsequent steps is dominated by the first one, therefore Algorithm 5.1 is polynomial. Let $\alpha = \frac{(b-1)\delta e^{\frac{\delta+1}{2}}}{16\ell}$. The choice of the actual set cover depends on the following two cases.

If $|C_1| \geq \alpha \cdot \mathrm{Opt}^*$: We choose all edges of the two sets C_1 and C_2 as set cover C and terminate. Recall that by Lemma 5.4, $C = C_1 \cup C_2$ is a feasible set multicover.

If $|C_1| < \alpha \cdot \mathrm{Opt}^*$: We use LP-based randomized rounding, every edge of C_3 is independently added to the cover with probability $\frac{\delta+1}{2}x_j^*$. In order to guarantee feasibility, we eventually proceed with a step of repairing in which additional edges are added.

5.4 Analysis of the Algorithm

First, we consider the case $|C_1| \geq \alpha \cdot \mathrm{Opt}^*$.

Theorem 5.5 *Let $\mathcal{H}$ be a hypergraph with maximum vertex degree Δ and maximum edge size ℓ. Let $\alpha = \frac{(b-1)\delta e^{\frac{\delta+1}{2}}}{16\ell}$ as defined in Algorithm 5.1. If $|C_1| \geq \alpha \cdot \mathrm{Opt}^*$, the algorithm achieves an approximation factor of $(1 - \frac{(b-1)e^{\frac{\delta+1}{2}}}{32\ell})\delta$ with respect to* Opt^*.

Proof. With the definition of the sets C_1 and C_2, we have

$$\begin{aligned}
\delta \cdot \mathrm{Opt}^* = \sum_{j=1}^{m} \delta x_j^* &\geq \sum_{j, E_j \in C_1} \delta x_j^* + \sum_{j, E_j \in C_2} \delta x_j^* \\
&\geq \frac{\delta}{\lambda}|C_1| + |C_2| && \text{(using the definion of } C_1, C_2\text{)} \\
&= \frac{2\delta}{\delta+1}|C_1| + |C_2| && \text{(since } \lambda = \frac{\delta+1}{2}\text{)} \\
&= \frac{2\delta}{\delta+1}|C_1| + (|C| - |C_1|) && \text{(since } C = C_1 \cup C_2\text{)} \\
&= \frac{\delta-1}{\delta+1}|C_1| + |C| \\
&\overset{\delta \geqslant 3}{\geq} \frac{1}{2}|C_1| + |C| \\
&\geq \frac{1}{2}\alpha \cdot \mathrm{Opt}^* + |C|. && \text{(assumption of the theorem)}
\end{aligned}$$

Hence, by the choice of α, we get

$$|C| \leq \delta \cdot \mathrm{Opt}^* - \frac{\alpha}{2}\mathrm{Opt}^* = \left(1 - \frac{(b-1)e^{\frac{\delta+1}{2}}}{32\ell}\right)\delta \cdot \mathrm{Opt}^*. \qquad \square$$

Next, we consider the case $|C_1| < \alpha \cdot \mathrm{Opt}^*$.

Let $X_1, \dots, X_m$ be $\{0,1\}$-random variables defined as follows:

$$X_j = \begin{cases} 1 & \text{if the edge } E_j \text{ was added to the cover before repairing} \\ 0 & \text{otherwise.} \end{cases}$$

Note that the $X_1, \dots, X_m$ are independent random variables for a given $x^* \in [0,1]^m$. For all $i \in [n]$ we define the $\{0,1\}$- random variables Y_i as follows:

$$Y_i = \begin{cases} 1 & \text{if the vertex } v_i \text{ is fully covered before repairing} \\ 0 & \text{otherwise.} \end{cases}$$

We denote by $X := \sum_{j=1}^{m} X_j$ and $Y := \sum_{i=1}^{n} Y_i$ the size of the cover and the number of vertices fully covered before the step of repairing, respectively. At this step by Lemma 5.4, at most one more edge for each vertex is needed to be fully covered. The cover denoted by C obtained by Algorithm 5.1 is bounded by

$$|C| \leq X + n - Y. \tag{5.4.2}$$

Our goal in the next lemma is to estimate the expectation of the random variable X and the expectation and variance of the random variable Y for the proof of Theorem 5.7. This is a restriction of Lemma 4 in [EMS16] to the last case in Algorithm 5.1.

Lemma 5.6 *Let ℓ and Δ be the maximum size of an edge, resp. the maximum vertex degree, not necessarily constants. Let $\alpha > 0$ and $\lambda = \frac{\delta+1}{2}$ as in Algorithm 5.1. In case $|C_1| < \alpha \cdot \mathrm{Opt}^*$, we have*

(i) $\mathbb{E}[Y] \geq (1 - e^{-\lambda})n$.

(ii) $\mathbb{E}[X] \leq \lambda \mathrm{Opt}^*$.

(iii) $\frac{(b-1)n}{\alpha \ell} < \mathrm{Opt}^*$.

(iv) $\mathbb{E}[|C|] \leq \left(\frac{\lambda}{\delta} + \frac{1}{16}\right)\delta \cdot \mathrm{Opt}^*$

Proof. *(i)* Let $i \in [n]$. If $|C_1 \cap \Gamma(v_i)| \geq b_i$, then the vertex v_i is fully covered and $\Pr[Y_i = 0] = 0$. Otherwise, we get $|C_1 \cap \Gamma(v_i)| = b_i - 1$ and $\sum_{E_j \in (\Gamma(v_i) \cap C_3)} x_j^* \geq 1$ by Lemma 5.4. Recall that each $E_j \in C_3$ is chosen indepentently with probability λx_j^*. Therefore the event Y_i, i.e., that the vertex

v_i is not fully covered is true if and only if none of the incident E_j are added. We have

$$\begin{aligned}
\Pr[Y_i = 0] &= \prod_{E_j \in (\Gamma(v_i) \cap C_3)} (1 - \lambda x_j^*) \\
&\leq \prod_{E_j \in (\Gamma(v_i) \cap C_3)} e^{-\lambda x_j^*} && (1 + x \leqslant e^x \text{ f.a. } x \in \mathbb{R} \text{ with } x = -\lambda x_j^*) \\
&= e^{-\lambda \sum_{E_j \in (\Gamma(v_i) \cap C_3)} x_j^*} \\
&\leq e^{-\lambda}. && (\text{using } \textstyle\sum_{E_j \in (\Gamma(v_i) \cap C_3)} x_j^* \geq 1) \qquad (5.4.3)
\end{aligned}$$

Thus,

$$\begin{aligned}
\mathbb{E}[Y] &= \sum_{i=1}^{n} \Pr[Y_i = 1] = \sum_{i=1}^{n} (1 - \Pr[Y_i = 0]) \\
&\overset{(5.4.3)}{\geq} \sum_{i=1}^{n} (1 - e^{-\lambda}) \\
&= (1 - e^{-\lambda}) n.
\end{aligned}$$

(ii) Using the LP relaxation and the definition of the sets C_1 and C_3, we have $X_j = 1$ for all $E_j \in C_1$ and $\mathbb{E}[X_j] = \Pr[X_j = 1] = \lambda x_j^*$ for all $E_j \in C_3$. We get

$$\begin{aligned}
\mathbb{E}[X] = \mathbb{E}\left[\sum_{j=1}^{m} X_j\right] &= |C_1| + \sum_{E_j \in C_3} \lambda x_j^* \\
&\leq \sum_{E_j \in C_1} \lambda x_j^* + \sum_{E_j \in C_3} \lambda x_j^* && (\text{using } \lambda x_j^* \geq 1 \text{ f.a. } E_j \in C_1) \\
&\leq \lambda \sum_{E_j \in \mathscr{E}} x_j^* \\
&= \lambda \cdot \mathrm{Opt}^*.
\end{aligned}$$

(iii) Consider $\tilde{\mathscr{H}}$, the hypergraph induced by C_1. In $\tilde{\mathscr{H}}$, we have

$$\sum_{i \in V} d(i) = \sum_{E_j \in C_1} |E_j|,$$

which holds for any hypergraph. Since the minimum vertex degree in $\tilde{\mathscr{H}}$ is $b - 1$ (with $b =$

$\min_{i\in[n]} b_i$ as defined earlier), we have

$$(b-1)n \le \sum_{i\in V} d(i) = \sum_{E\in C_1} |E_j| \le \ell |C_1|.$$

Therefore

$$\frac{(b-1)n}{\ell} \le |C_1|.$$

With $|C_1| < \alpha \cdot \mathrm{Opt}^*$, we obtain

$$\frac{(b-1)n}{\alpha\ell} < \mathrm{Opt}^*.$$

(iv) By *(iii)*, we have $n < \frac{\alpha\ell}{b-1}\mathrm{Opt}^*$. By plugging in the definition of α, we get $ne^{-\lambda} < \frac{\delta}{16}\mathrm{Opt}^*$. Now we can bound the expectation of $|C|$:

$$\begin{aligned}\mathbb{E}[|C|] &\overset{(5.4.2)}{\le} \mathbb{E}[X] + n - \mathbb{E}[Y] \\ &\overset{(ii),(i)}{\le} \lambda\mathrm{Opt}^* + ne^{-\lambda} \\ &\le \left(\frac{\lambda}{\delta} + \frac{1}{16}\right)\delta \cdot \mathrm{Opt}^*.\end{aligned}$$

□

Theorem 5.7 *Let $\mathcal{H}$ be a hypergraph with maximum vertex degree Δ and maximum edge size ℓ where $\ell \le (\Delta^{\frac{3}{2}} e^{\lambda})^{-1}\sqrt{n}$. Let $\alpha = \frac{(b-1)\delta e^{\frac{\delta+1}{2}}}{16\ell}$ as in Algorithm 5.1. In case $|C_1| < \alpha \cdot \mathrm{Opt}^*$, the algorithm returns a set multicover C with*

$$|C| < \frac{19}{24}\delta \cdot \mathrm{Opt}$$

with probability at least $1-\exp(-2) \approx 0.86$ in polynomial time.

We will use Theorem 5.2 and the following lemma to estimate the cardinality of the cover at the end of the algorithm.

Lemma 5.8 *Let $D = \ell\Delta$. Then* $\Pr\left[|C| \ge \mathbb{E}[|C|] + D\sqrt{m}\right] \le \exp(-2)$.

Proof. For $j \in [m]$, let $\hat{X}_j = 1$ if and only if the edge E_j is contained in the partial set b-multicover after repairing. Consider the function $f(X_1,\ldots,X_m) := \sum_{j=1}^m \hat{X}_j$. Then we have for any two vectors $x = (x_1,\ldots,x_k,\ldots,x_m)$ and $x' = (x_1,\ldots,x'_k,\ldots,x_m)$ that only differ in the k-th coordinate

$$|f(x_1,\ldots,x_k,\ldots,x_m) - f(x_1,\ldots,x'_k,\ldots,x_m)| \le 1 \le D$$

for all $k \in [m]$. Thus, we can use the bounded differences inequality (Theorem 5.2) with $c_k := D$ for all $k \in [m]$ and $t = D\sqrt{m}$ and have

$$\Pr\left[|C| \geq \mathbb{E}[|C|] + D\sqrt{m}\right] \leq \exp\left(\frac{-2\left(D\sqrt{m}\right)^2}{\sum_{k=1}^{m} D^2}\right) = \exp(-2). \tag{5.4.4}$$

□

Proof of Theorem 5.7. First, we prove that $D\sqrt{m} \leq \frac{\delta}{16}\mathrm{Opt}^*$. For the hypergraph $\mathcal{H}$, we have $m \leq \Delta n$. By rearranging Lemma 5.6 *(iii)*, we obtain $n < \frac{\alpha \ell}{b-1}\mathrm{Opt}^*$, then we plug in the definition of α and get

$$n < \frac{e^{\lambda}\delta}{16}\mathrm{Opt}^*. \tag{5.4.5}$$

Therefore

$$\begin{aligned} D\sqrt{m} &\overset{m \leqslant \Delta n}{\leq} \ell\Delta\sqrt{\Delta}\sqrt{n} \\ &= \Delta^{\frac{3}{2}}\ell\sqrt{n} \\ &\overset{\text{Choice of } \ell}{\leq} \Delta^{\frac{3}{2}}\left(\Delta^{\frac{3}{2}}e^{\lambda}\right)^{-1}\sqrt{n}\sqrt{n} \\ &= e^{-\lambda}n \\ &\overset{(5.4.5)}{\leq} \frac{\delta}{16}\mathrm{Opt}^*. \end{aligned} \tag{5.4.6}$$

So we have

$$\begin{aligned} \Pr\left[|C| > \left(\frac{\lambda}{\delta} + \frac{1}{8}\right)\delta\mathrm{Opt}^*\right] &= \Pr\left[|C| > \left(\frac{\lambda}{\delta} + \frac{1}{16}\right)\delta\mathrm{Opt}^* + \frac{1}{16}\delta \cdot \mathrm{Opt}^*\right] \\ &\overset{\text{(iv)},(5.4.6)}{\leq} \Pr\left[|C| > \mathbb{E}[|C|] + D\sqrt{m}\right] \\ &\leq \exp(-2). && \text{(by (5.4.4))} \end{aligned} \tag{5.4.7}$$

For $\delta \geq 3$, $\frac{\delta+1}{2\delta} + \frac{1}{8} \geq \frac{19}{24}$, so

$$\begin{aligned} \Pr\left[|C| \leq \frac{19}{24}\delta \cdot \mathrm{Opt}^*\right] &\overset{\delta \geqslant 3}{\geq} \Pr\left[|C| \leq \left(\frac{\delta+1}{2\delta} + \frac{1}{8}\right)\delta \cdot \mathrm{Opt}^*\right] \\ &= \Pr\left[|C| \leq \left(\frac{\lambda}{\delta} + \frac{1}{8}\right)\delta \cdot \mathrm{Opt}^*\right] \\ &\overset{(5.4.7)}{\geq} (1 - \exp(-2)) \approx 0.86. \end{aligned}$$

□

Bibliography

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. "Color-Coding". In: *Journal of the ACM* 42.4 (1995), pp. 844–856. DOI: 10.1145/210332.210337.

[BA99] Albert-László Barabási and Réka Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.5439 (1999), pp. 509–512. DOI: 10.1126/science.286.5439.509.

[Bar01] Reuven Bar-Yehuda. "Using Homogeneous Weights for Approximating the Partial Cover Problem". In: *Journal of Algorithms* 39.2 (2001), pp. 137–144. DOI: 10.1006/jagm.2000.1150.

[Bec08] József Beck. *Combinatorial games: tic-tac-toe theory*. Cambridge University Press, 2008.

[Bec81] József Beck. "Van der Waerden and Ramsey Type Games". In: *Combinatorica* 1.2 (1981), pp. 103–116. DOI: 10.1007/BF02579267.

[Bec82] József Beck. "Remarks on Positional Games. I". In: *Acta Mathematica Hungarica* 40.1-2 (1982), pp. 65–71. DOI: 10.1007/BF01897304.

[Bec85] József Beck. "Random Graphs and Positional Games on the Complete Graph". In: *Random Graphs '83*. Ed. by Michał Karoński and Andrzej Ruciński. Vol. 118. North-Holland Mathematics Studies. North-Holland, 1985, pp. 7–13. DOI: 10.1016/S0304-0208(08)73609-0.

[BH03] Andreas Björklund and Thore Husfeldt. "Finding a Path of Superlogarithmic Length". In: *SIAM Journal on Computing* 32.6 (2003), pp. 1395–1402. DOI: 10.1137/S0097539702416761.

[BHK04] Andreas Björklund, Thore Husfeldt, and Sanjeev Khanna. "Approximating Longest Directed Paths and Cycles". In: *Proceedings of the 31st International Colloquium on Automata, Languages and Programming, Turku, Finland, July 2004 (ICALP 2004)*. 2004, pp. 222–233. DOI: 10.1007/978-3-540-27836-8_21.

[Bjö+10] Andreas Björklund et al. *Narrow Sieves for Parameterized Paths and Packings*. 2010. arXiv: `1007.1161`.

[Bjö14] Andreas Björklund. "Determinant Sums for Undirected Hamiltonicity". In: *SIAM Journal on Computing* 43.1 (2014). Conference version at FOCS 2010., pp. 280–299. DOI: `10.1137/110839229`.

[BKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. "Reductions in Streaming Algorithms, with an Application to Counting Triangles in Graphs". In: *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, USA, January 2002 (SODA 2002)*. 2002, pp. 623–632. DOI: `10.5555/545381.545464`.

[BŁ00] Małgorzata Bednarska and Tomasz Łuczak. "Biased Positional Games for which Random Strategies are Nearly Optimal". In: *Combinatorica* 20.4 (2000), pp. 477–488. DOI: `10.1007/s004930070002`.

[Bod93] Hans L. Bodlaender. "On Linear Time Minor Tests with Depth-First Search". In: *Journal of Algorithms* 14 (1993). Conference version at WADS 1989., pp. 1–23. DOI: `10.1006/jagm.1993.1001`.

[Bol84] Béla Bollobás. "The evolution of sparse graphs". In: *Graph theory and combinatorics* (1984), pp. 35–57.

[BS11] József Balogh and Wojciech Samotij. "On the Chvátal-Erdős Triangle Game". In: *Electronic Journal of Combinatorics* 18.1 (2011), P72. DOI: `10.37236/559`.

[BST99] Cristina Bazgan, Miklos Santha, and Zsolt Tuza. "On the Approximation of Finding A(nother) Hamiltonian Cycle in Cubic Hamiltonian Graphs". In: *Journal of Algorithms* 31.1 (1999). Conference version at STACS 1998., pp. 249–268. DOI: `10.1006/jagm.1998.0998`.

[Bul+02] R.W. Bulterman et al. "On Computing a Longest Path in a Tree". In: *Information Processing Letters* 81.2 (2002), pp. 93–96. DOI: `10.1016/S0020-0190(01)00198-3`.

[CE78] Václav Chvátal and Paul Erdős. "Biased Positional Games". In: *Algorithmic Aspects of Combinatorics*. Elsevier, 1978, pp. 221–229. DOI: `10.1016/s0167-5060(08)70335-2`.

[Che+07] Jianer Chen et al. "Improved Algorithms for Path, Matching, and Packing Problems". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 298–307. DOI: `10.5555/1283383.1283415`.

[Con+94] Axel Conrad et al. “Solution of the Knight’s Hamiltonian Path Problem on Chessboards”. In: *Discrete Applied Mathematics* 50.2 (1994), pp. 125–134. URL: `http://www.sciencedirect.com/science/article/pii/0166218X9200170Q`.

[CT16] Dennis Clemens and Tuan Tran. “Creating Cycles in Walker-Breaker Games”. In: *Discrete Mathematics* 339.8 (2016), pp. 2113–2126. DOI: `10.1016/j.disc.2016.03.007`.

[DF13] Rod Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer Publishing Company, Incorporated, 2013. DOI: `10.1007/978-1-4471-5559-1`.

[EMS16] Mourad El Ouali, Peter Munstermann, and Anand Srivastav. “Randomized Approximation for the Set Multicover Problem in Hypergraphs”. In: *Algorithmica* 74.2 (2016), pp. 574–588. DOI: `10.1007/s00453-014-9962-9`.

[ES73] Paul Erdős and John L. Selfridge. “On a Combinatorial Game”. In: *Journal of Combinatorial Theory, Series A* 14.3 (1973), pp. 298–301. DOI: `10.1016/0097-3165(73)90005-8`.

[Esp+14] Lisa Espig et al. *Walker-Breaker Games*. 2014. arXiv: `1401.5538`.

[Fei+05] Joan Feigenbaum et al. “On Graph Problems in a Semi-Streaming Model”. In: *Theoretical Computer Science* 348 (2005). Conference version at ICALP 2004., pp. 207–216. DOI: `10.1016/j.tcs.2005.09.013`.

[Fei+08] Joan Feigenbaum et al. “Graph Distances in the Data-Stream Model”. In: *SIAM Journal on Computing* 38 (2008), pp. 1709–1727. DOI: `10.1137/070683155`.

[FK05] Toshihiro Fujito and Hidekazu Kurahashi. “A Better-Than-Greedy Algorithm for k-Set Multicover”. In: *International Workshop on Approximation and Online Algorithms*. Springer. 2005, pp. 176–189. DOI: `10.1007/11671411_14`.

[FMS02] Tomás Feder, Rajeev Motwani, and Carlos Subi. “Approximating the Longest Cycle Problem in Sparse Graphs”. In: *SIAM Journal on Computing* 31.5 (2002), pp. 1596–1607. DOI: `10.1137/S0097539701395486`.

[Gab07] Harold N. Gabow. “Finding Paths and Cycles of Superpolylogarithmic Length”. In: *SIAM Journal on Computing* 36.6 (2007), pp. 1648–1671. DOI: `10.1137/S0097539704445366`.

[Gla19] Christian Glazik. “Positional and Detection Games”. PhD thesis. Christian-Albrechts-Universität zu Kiel, 2019. URL: `https://macau.uni-kiel.de/receive/macau_mods_00000127`.

[GN08] Harold N. Gabow and Shuxin Nie. “Finding Long Paths, Cycles and Circuits”. In: *Proceedings of the 19th International Symposium on Algorithms and Computation, Gold Coast, Australia, December 2008 (ISAAC 2008)*. 2008, pp. 752–753. DOI: `10.1007/978-3-540-92182-0_66`.

[GO14] Venkatesan Guruswami and Krzysztof Onak. "Superlinear Lower Bounds for Multipass Graph Processing". In: *Electronic Colloquium on Computational Complexity* (2014). Conference version at CCC 2013. DOI: 10.1109/CCC.2013.37.

[Gor+21] Abbass Gorgi et al. "Approximation Algorithm for the Multicovering Problem". In: *Journal of Combinatorial Optimization* (2021), pp. 1–18. DOI: 10.1007/s10878-020-00688-9.

[GS18] Christian Glazik and Anand Srivastav. *A new Bound for the Maker-Breaker Triangle Game*. 2018. arXiv: 1812.01382 [math.CO].

[Hef+09] Dan Hefetz et al. "A Sharp Threshold for the Hamilton Cycle Maker-Breaker Game". In: *Random Structures & Algorithms* 34 (Jan. 2009), pp. 112–122. DOI: 10.1002/rsa.20252.

[Hef+14] Dan Hefetz et al. *Positional Games*. Birkhäuser Basel, 2014. DOI: 10.1007/978-3-0348-0825-5.

[HH86] Nicholas G Hall and Dorit S Hochbaum. "A Fast Approximation Algorithm for the Multicovering Problem". In: *Discrete Applied Mathematics* 15.1 (1986), pp. 35–40. DOI: 10.1016/0166-218X(86)90016-8.

[Kar72] Richard M. Karp. "Reducibility among Combinatorial Problems". English. In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. The IBM Research Symposia Series. Springer US, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[KBA12] Fatemeh Keshavarz-Kohjerdia, Alireza Bagherib, and Asghar Asgharian-Sardroudb. "A linear-time algorithm for the Longest Path Problem in Rectangular Grid Graphs". In: *Discrete Applied Mathematics* 160.3 (2012), pp. 210–217. DOI: 10.1016/j.dam.2011.08.010.

[KMR97] David Karger, Rajeev Motwani, and G.D.S. Ramkumar. "On Approximating the Longest Path in a Graph". In: *Algorithmica* 18 (1997), pp. 82–98. DOI: 10.1007/BF02523689.

[Kou08] Ioannis Koutis. "Faster Algebraic Algorithms for Path and Packing Problems". In: *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Reykjavik, Iceland, July 2008 (ICALP 2008)*. 2008, pp. 575–586. DOI: 10.1007/978-3-540-70575-8_47.

[Kri+10] Dmitri Krioukov et al. "Hyperbolic Geometry of Complex Networks". In: *Physical Review E* 82 (2010). DOI: 10.1103/PhysRevE.82.036106.

[Kri14] Michael Krivelevich. "Positional Games". In: *Proceedings of the International Congress of Mathematicians (ICM 2014)* 4 (2014), pp. 355–379. URL: https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM2014.4/ICM2014.4.pdf.

[KSS16] Lasse Kliemann, Christian Schielke, and Anand Srivastav. “A Streaming Algorithm for the Undirected Longest Path Problem”. In: *24th Annual European Symposium on Algorithms (ESA 2016)*. 2016, 56:1–56:17. DOI: `10.4230/LIPIcs.ESA.2016.56`.

[Kus+17] Christopher Kusch et al. “Random Strategies are Nearly Optimal for Generalized van der Waerden Games”. In: *Electronic Notes in Discrete Mathematics* 61 (2017), pp. 789–795. DOI: `10.1016/j.endm.2017.07.037`.

[Kus+19] Christopher Kusch et al. “On the Optimality of the Uniform Random Strategy”. In: *Random Structures & Algorithms* 55.2 (2019), pp. 371–401. DOI: `10.1002/rsa.20829`.

[Loo+15] Moritz von Looz et al. *Fast generation of complex networks with underlying hyperbolic geometry*. 2015. arXiv: `1501.03545`.

[Mar+12] Minko Markov et al. “A Linear Time Algorithm for Computing Longest Paths in Cactus Graphs”. In: *Serdica Journal of Computing* 6.3 (2012). URL: `http://serdica-comp.math.bas.bg/index.php/serdicajcomputing/article/view/158`.

[McD89] Colin McDiarmid. “On the Method of Bounded Differences”. In: *Surveys in combinatorics* 141.1 (1989), pp. 148–188. DOI: `10.1017/CBO9781107359949.008`.

[Mon85] Burkhard Monien. “How to Find Long Paths Efficiently”. In: *Annals of Discrete Mathematics* 25 (1985), pp. 239–254. URL: `https://digital.ub.uni-paderborn.de/hs/content/titleinfo/42079`.

[MS14] Tobias Müller and Miloš Stojaković. “A Threshold for the Maker-Breaker Clique Game”. In: *Random Structures & Algorithms* 45.2 (2014), pp. 318–341. DOI: `10.1002/rsa.20489`.

[Mut05] Muthu Muthukrishnan. “Data Streams: Algorithms and Applications”. In: *Foundations and Trends in Theoretical Computer Science* 1.2 (2005), 67 pages. Preliminary version available since 2003.

[Poh67] Ira Pohl. “A Method for Finding Hamilton Paths and Knight’s Tours”. In: *Communications of the ACM* 10.7 (1967), pp. 446–449. DOI: `10.1145/363427.363463`.

[Pon12] Lajos L. Pongrácz. *A Greedy Approximation Algorithm for the Longest Path Problem in Undirected Graphs*. 2012. arXiv: `1209.2503v2`.

[PS04] Ira Pohl and Larry Stockmeyer. “Pohl-Warnsdorf — Revisited”. In: *Proceedings of the International Conference on Intelligent Systems and Control, Honolulu, Hawaii, USA, August 2004 (ISC 2004)*. 2004. URL: `https://users.soe.ucsc.edu/~pohl/Papers/Pohl_Stockmeyer_full.pdf`.

[PSW93] David Peleg, Gideon Schechtman, and Avishai Wool. "Approximating Bounded 0-1 Integer Linear Programs". In: *The 2nd Israel Symposium on Theory and Computing Systems*. IEEE Computer Society. 1993, pp. 69–70. DOI: 10.1109/ISTCS.1993.253482.

[PSW97] David Peleg, Gideon Schechtman, and Avishai Wool. "Randomized Approximation of Bounded Multicovering Problems". In: *Algorithmica* 18.1 (1997), pp. 44–66. DOI: 10.1007/BF02523687.

[Sch15] Christian Schielke. "An Experimental Study of RAM and Streaming Algorithms for the Longest Path Problem". MA thesis. Kiel University, 2015.

[Sch99] John K. Scholvin. "Approximating the Longest Path Problem with Heuristics: A Survey". MA thesis. University of Illinois at Chicago, 1999.

[SS05] Miloš Stojaković and Tibor Szabó. "Positional Games on Random Graphs". In: *Random Structures & Algorithms* 26.1–2 (2005), pp. 204–223. DOI: 10.1002/rsa.20059.

[SS90] Jeanette P. Schmidt and Alan Siegel. "The Spatial Complexity of Oblivious k-probe Hash Functions". In: *SIAM J. Comput.* 19.5 (1990), pp. 775–786. DOI: 10.1137/0219054.

[ST19] Miloš Stojaković and Nikola Trkulja. "Hamiltonian Maker–Breaker Games on Small Graphs". In: *Experimental Mathematics* (2019), pp. 1–10. DOI: 10.1080/10586458.2019.1586599.

[SW11] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-Wesley Professional, 2011. ISBN: 9780321573513.

[UU07] Ryuhei Uehara and Yushi Uno. "On Computing Longest Paths in Small Graph Classes". In: *International Journal of Foundations of Computer Science* 18.5 (2007). DOI: 10.1142/S0129054107005054.

[Vaz13] Vijay V. Vazirani. *Approximation Algorithms*. Springer Science & Business Media, 2013. DOI: 10.1007/978-3-662-04565-7.

[Vis04] Sundar Vishwanathan. "An Approximation Algorithm for Finding Long Paths in Hamiltonian Graphs". In: *Journal of Algorithms* 50.2 (2004). Conference version at SODA 2000., pp. 246–256. DOI: 10.1016/S0196-6774(03)00093-2.

[Wil09] Ryan Williams. "Finding Paths of Length k in $O^*(2^k)$ Time". In: *Information Processing Letters* 109 (2009), pp. 315–318. DOI: 10.1016/j.ipl.2008.11.004.

[WS98] Duncan J. Watts and Steven H. Strogatz. "Collective Dynamics of 'Small-World' Networks". In: *Nature* 393 (1998), pp. 440–442. DOI: 10.1038/30918.

Erklärung

Hiermit erkläre ich,

- dass die Abhandlung – abgesehen von der Beratung durch den Betreuer – nach Inhalt und Form die eigene Arbeit ist,
- dass die Arbeit weder ganz noch zum Teil schon einer anderen Stelle im Rahmen eines Prüfungsverfahrens vorgelegen hat, veröffentlicht worden ist oder zur Veröffentlichung eingereicht wurde,
- dass die Arbeit unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden ist,
- dass mir kein akademischer Grad entzogen wurde.