DIMACS CoSP REU International 2019
Research Experience for Undergraduates

May 28 – August 2, 2019,

Piscataway, NJ, USA and Prague, Czech Republic

Pavel Dvořák (ed.)

DIMACS CoSP REU International 2019

Published by IUUK–ITI series 2020-680
Centre of Excellence – Institute for Theoretical Computer Science,
Faculty of Mathematics and Physics, Charles University
Malostranské náměstí 25, 118 00, Praha 1, Czech Republic

Prague 2020

Preface

The joint REU program between DIMACS and DIMATIA started back in 1999 with the exchange of 5 US and 3 Czech students. Since then, the program has taken place every single year without any interruptions. From 2019 the program is receiving funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 823748. In this age of rapid changes in the research and funding landscape, it takes a lot of effort for any activity to survive for more than 20 years, especially an activity focused on bringing together students from different continents. I think that the success and the durability of the program is a testament to the care and commitment of everyone at Charles University and at Rutgers University to offer this opportunity to our students.

During this period, 127 American REU students have studied in Prague with a similar number of Czech students working at Rutgers. For the majority of the students, this program is their first real step in conducting original research. These first steps can be daunting, but the students greatly benefit by the one-to-one mentoring plan by Rutgers faculty at DIMACS and by Charles University. The role of the (volunteer!) mentors cannot be stressed enough in this process. Students know that they will have the full attention of their mentors for the entire duration of the program and the mentors are committed to helping them grow as scientists. I believe that the key aspect of the program is that everyone is invested in the students' success. We want to make sure that any issue is immediately solved and that all students feel welcome and supported in their work and social environment.

Our students consistently praise this experience and the multitude of benefits they get out of it. Listening to lectures in Charles University, participating in international research workshops, visiting the Art Museum in Prague, and bonding with their Czech peers are always mentioned in the student reports as transformative experiences. Most of them also mention that they acquire a new understanding of how research transcends national or language borders and have a deeper appreciation for the power of ideas and of scientific thinking.

On a personal note, I always know that no matter what problem may arise, our collaborators (and friends!) in Prague will make sure that it is quickly resolved. I owe a big Thank You to Professors Nesetril, Loebl, and Fiala, among others, together with their staff and the Czech students, for taking such good care of our students and of all aspects of the program in Prague. Děkuji!

Lazaros Gallos,
DIMACS REU director

Charles University in Prague and particularly Department of Applied Mathematics (KAM) and Computer Science Institute of Charles University (IÚUK) are very happy to host one of the very few International REU programmes which were established and are in long term supported by the National Science Foundation.

On the Czech side, the programme has been the first time financed by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 823748. We are grateful and proud to receive such major sponsorship of our unique activity.

Since the establishment of DIMACS–DIMATIA REU in 1999, it has been awarded for its accomplishments and educational excellence.

This booklet reports just the activities and the programme that have been conducted in 2019. In particular I thank to Pavel Dvořák, and Jakub Pekárek the Czech mentors of this year, for a very good work both during the programme itself and afterwards.

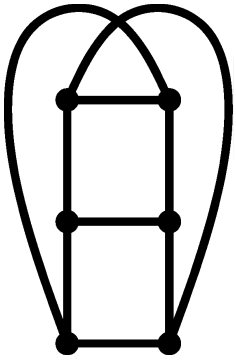
Prague, February 25
Martin Loebl,
The senior organizer of the Czech REU part

Thanks to the EU grant, there were 10 Czech participants (3 graduate and 7 undergraduate students) of the REU 2019 program at DIMACS. The students were divided into 4 groups and worked with 3 mentors Bhargav Narayanan, Periklis Papakonstantinou and Sophie Spirkl on various projects. The topics of projects were part of theoretical computer science and discrete math like graph theory, probability theory and computational complexity. This booklet contains papers which cover the work of Czech students from summer 2019. In time of publishing this booklet, the paper by M. Beliayeu, P. Chmel and J. Petr, supervised by B. Narayanan (Part IV), was already published in Electronic Communications in Probability and results of P. Pelikánová, and A. Šťastná, supervised by S. Spirkl (Part II), were presented at workshop Cycle and Colourings 2019. The booklet also contains one note about a project of two US students who were part of the group which spent a part of the REU program in Prague.

Pavel Dvořák,
Charles University

Funding

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 823748.



Project acronym: CoSP

Project title: Combinatorial Structures and Processes

Grant agreement no.: 823748

Website: kam.mff.cuni.cz/rise

The DIMACS REU program is supported by the NSF grant CCF-1852215.

Participants of REU 2019 from Charles University, Faculty of Mathematics and Physics

- **Pavel Dvořák**
  - 4th year of the doctoral program of Computer Science
- **Jakub Pekárek**
  - 2nd year of the doctoral program of Computer Science
- **Mikhail Beliayeu**
  - 2nd year of the bachelor program of Computer Science
- **Andrej Dedík**
  - 3rd year of the bachelor program of Computer Science
- **Petr Chmel**
  - 2nd year of the bachelor program of Computer Science
- **Lukáš Ondráček**
  - 1st year of the doctoral program of Computer Science
- **Petra Pelikánová**
  - 1st year of the master program of Computer Science
- **Jan Petr**
  - 2nd year of the bachelor program of Mathematics
- **Jan Soukup**
  - 1st year of the master program of Computer Science
- **Aneta Šťastná**
  - 1st year of the master program of Computer Science

Contents

A Graph Game from Extremal Combinatorics

Jan Soukup, Andrej Dedík

Abstract

A game $\mathbb{G}_i(n,k,H)$ is played by two players on initially empty graph with n vertices. In every round *player A* selects a stable set of size k and *Player B* adds some new edges to this stable set. *Player A* is trying to force an appearance of an induced sub-graph isomorphic to H while *Player B*'s effort is to not allow this to happen.

Chudnovsky et al. [3] showed that for every k and H, *Player A* have winning strategy for every game $\mathbb{G}_i(n,k,H)$ with sufficiently large n. We look in the asymptotic of the minimal n for which *Player A* has a winning strategy. We prove that for certain well-known families of graphs, we can get polynomial bounds (with respect to k and size of H). In particular, the growth of n is at most polynomial for paths, trees and cycles.

1 Introduction

The graphs in this paper are finite, without loops and multiple edges between vertices. We denote the vertex set of a graph G by $V(G)$ and the edge set of a graph G by $E(G)$. The graph $G[X]$ is a graph induced by a vertex set X. $\omega(G)$ is the size of the the largest clique in G and $\alpha(G)$ is the size of the maximal stable set (also known as the maximal independent set). The eccentricity of a vertex v in a graph G, denoted by $ex_G(v)$, is a maximum distance from v to any vertex of G. The diameter of G, denoted by $diam(G)$, is a maximum eccentricity of vertices of G.

We say that G *contains (induced)* H if some (induced) sub-graph of G is isomorphic to H. If G does not contain (induced) H we say that G is (*weakly*) *H-free*.

Definition 1.1. Let H be a graph, k, n be integers and $k \geq 2$. A game $\mathbb{G}_i(n, k, H)$ is game played by two players on an initially empty graph G with n vertices by two players *Player A* and *Player B*. The goal of *Player A* is to force G to contain induced H. The goal of *Player B* is to not allow this to happen. The game is played in rounds. Each round proceeds as follows:

- *Player A* selects a stable set $S \subseteq V(G)$ of size k. If there is no stable set of size k and G does not contain induced H, then *Player B* wins.
- *Player B* adds at least one edge into the stable set S. If G contains induced H after addition of new edge(s), then *Player A* wins.

For convenience we also define a game $\mathbb{G}(n, k, H)$ the same way as $\mathbb{G}_i(n, k, H)$, with only difference that *Player A* is trying to force an appearance of H as a sub-graph (not as an induced sub-graph) in G.

Definition 1.2. For a graph H we say that (induced) H is (n, k)-*forcible* if there is a strategy for *Player A* to always win game $\mathbb{G}(n, k, H)$ ($\mathbb{G}_i(n, k, H)$). Alternatively we say that *Player A* can (n, k)-*force* (induced) H.

Chudnovsky et al. [3] proved the following.

Theorem 1.3. For every graph H and every $k \geq 2$ there exists $n \geq k$ such that induced H is (n, k)-forcible.

As a consequence of this, we know that for each H and $k \geq 2$ there exists minimal integer m such that induced H is (m, k)-forcible. Call this number $m(H, k)$. One can deduce the upper bound for $m(H, k)$ from the proof of Theorem 1.3, which would bound $m(H, k)$ by double exponential function in $|E(H)|$ and k.

The game was initially introduced as a tool in a work analyzing weakly H-free graphs. Namely,Chudnovsky et al. [3] strengthen in several different ways the following theorem by Erdő et al. [8]

Theorem 1.4. For every graph H there exists $\varepsilon > 0$ such that for every weakly H-free graph G with $n > 1$ vertices, there are disjoint $A, B \subseteq V(G)$ such that $|A|, |B| \geq \varepsilon n^{\varepsilon}$ and either every vertex of A is adjacent to every vertex of B, or every vertex of A is non-adjacent to any vertex of B.

This problem itself can be used as a step in partial proof of Erdős-Hajnal Conjecture [6]. The best general bound for this conjecture to date was proved by Erdős and Hajnal [7], namely:

Theorem 1.5. For every graph H, there exists $\varepsilon > 0$ such that for every H-free graph G with $n > 0$ vertices, some clique or stable set of G has cardinality at least $2^{\varepsilon\sqrt{\log n}}$.

This result was not improved in general case despite much attention. For partial results see e.g. [9, 4, 1, 11]. We refer to the survey of Chudnovsky [2] for more details. Another result of this type is due to Prömel and Rödl [12].

Theorem 1.6. For each C there is $c > 0$ such that every graph on n vertices contains every graph on at most $c \log n$ vertices as an induced sub-graph or has a clique or independent set of size at least $C \log n$.

For an exhaustive survey of the whole Ramsey theory we refer to survey by Conlon et al. [5].

Even though we investigate the game independently, it is closely connected to other Ramsey type results. Special case, already noted by Chudnovsky et al. [3], is that $m(K_t, k) = R(t, k)$, where $R(t, k)$ is a Ramsey number. The upper bound follows from the fact that *Player A* can force a graph with small stable set. The lower bound follows from the fact that there exist a graph G on $R(t, k)$ vertices without big stable set and big clique. Therefore, player B can keep the game graph isomorphic to some sub-graphs of G during the whole game.

We believe this game can help in understanding of other Ramsey related questions.

Main results of our paper are regarding about paths, trees and cycles. In order to get polynomial bounds for $m(k, H)$, we first derive bounds effective for small k, namely that $m(P_n, k) \leq k^{\lceil \log n \rceil}$ and $m(P_n, k) \leq 2^k n$ and nice bound for trees with maximal degree d giving us $m(T, 3) \leq n + 1 + (d + 1)(n - 2)$. Next results show that the bounds are polynomial for paths and trees in general case, $O((max\{n, k\})^4)$ and $O((max\{n, k\})^{12})$ respectively. We use this results to get polynomial bounds for cycles in the last section.

2 Bounds for small k

This section presents basic results for paths and trees. These results are effective as long as k is "small", more general theorems are in the next section.

By P_n we denote a path on n vertices.

Theorem 2.1. For each path P_n, $n \in \mathbb{N}$, $m(P_n, k) \leq k^{\lceil \log n \rceil}$.

Proof. Let us prove one auxiliary result first:

1. *Let* $x \in \mathbb{N}$. *Then* $m(P_{2^x}, k) \leq k^x$.

We prove this by induction on x. For $x = 0$ it is trivial. By induction, *Player A* can (k^{x-1}, k)-force $P_{2^{x-1}}$. Thus he can (k^x, k)-force k independent copies of $P_{2^{x-1}}$. The endpoints of these paths are independent, hence *Player A* can select a stable set of size k consisting of the endpoints of these k paths. Two of these path will be connected by new edge. So the forced graph contains an induced $P_{2 \cdot 2^{x-1}}$, proving (1).

Following this result *Player A* can $(k^{\lceil \log n \rceil}, k)$-force path on $2^{\lceil \log n \rceil}$ vertices, which contains an induced path on n vertices. □

While this result cuts down the exponential part of the original result significantly, it is still not polynomial. The next result becomes linear for constant k.

Theorem 2.2. For each path P_n on n vertices, $n \in \mathbb{N}$, $m(P_n, k) \leq 2^{k+1} n$.

Proof. With sufficient amount of vertices *Player A* is able to keep k components of the game graph G such that their diameters are large compared to the number of vertices (we will state this precisely later).

Let $\mathcal{G}^s = \{G_1^s, G_1^s, ..., G_k^s\}$ be a set of these k components of G after s-th step of the game (initially each consists of single vertex). In every step *Player A* selects stable set $S = \bigcup_{i=1}^{k} v_i$ where $v_i \in G_i^s$ such that $\forall w \in G_i^s : ex_{G_i}(v_i) \geq ex_{G_i}(w)$ (and so $ex_{G_i}(v_i) = diam(G_i)$). *Player B* then connects some of these components into new ones. Then *Player A* will form $\mathcal{G}^{s+1}$ as follows:

- Each maximal set of components $\left\{G_{i_1}^s, G_{i_2}^s, \ldots, G_{i_h}^s\right\}$, with indices ordered as $i_1 < \cdots < i_h$, connected by *Player B* into a single component becomes the component $G_{i_h}^{s+1}$. Additionally, for each such set and each $j \in \{i_1, ..., i_{h-1}\}$ assign singletons to G_j^{s+1} (except for the last step). See Figure 1.

Now we show that for each s and each component G_i^s the following statement holds:

$$V(G_i^s) \leq 2^{i-1} \left(diam(G_i^s) + 1\right) \tag{1}$$

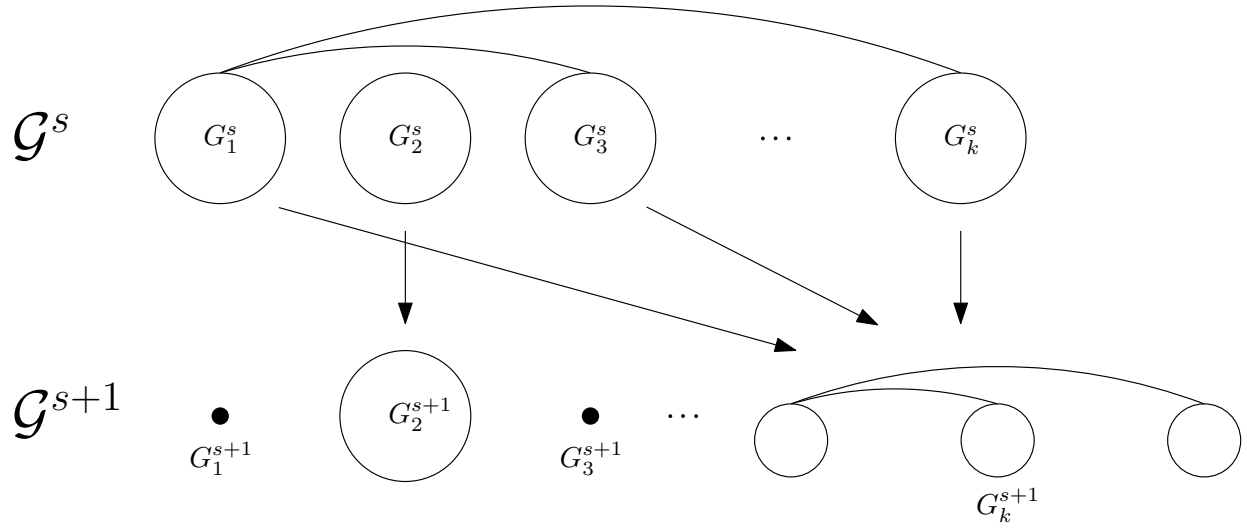


Figure 1: Rearranging $\mathcal{G}^s$ into $\mathcal{G}^{s+1}$

We proceed by induction. Suppose the statement holds for s (clearly statement holds for $s = 0$ since all components are singletons). For G_i^{s+1} formed by a singleton it clearly holds. Otherwise G_i^{s+1} is not a singleton, so component $G_i^{s+1} = G_{a_h}^{s+1}$ was formed with *Player B* joining components $G_{a_1}^s, ..., G_{a_h}^s$ where $h \geq 2$ and a_h is the greatest index from among them. Since *Player A* selected vertices with maximum eccentricity from the components, then clearly

$$diam(G_{a_h}^{s+1}) \geq diam(G_{a_h}^s) + \max_{j \in [h-1]} \{diam(G_{a_j}^s))\} + 1.$$

Using this and the induction hypothesis, we see that

$$\begin{aligned}
V(G_{a_h}^{s+1}) &= \sum_{i=1}^{h} V(G_{a_i}^s) \leq \sum_{i=1}^{h} 2^{a_i - 1}(diam(G_{a_i}^s) + 1) \\
&\leq 2^{a_h - 1}\left(diam(G_{a_h}^s) + 1\right) + \left(\sum_{i=1}^{h-1} 2^{a_i - 1}\right) \max_{j \in [h-1]} \{diam(G_{a_j}^s) + 1\} \\
&\leq 2^{a_h - 1}\left(diam(G_{a_h}^s) + 1\right) + 2^{a_h - 1} \max_{j \in [h-1]} \{diam(G_{a_j}^s) + 1\} \\
&= 2^{a_h - 1}\left(diam(G_{a_h}^s) + 1 + \max_{j \in [h-1]} \{diam(G_{a_j}^s) + 1\}\right) \\
&\leq 2^{a_h - 1}\left(diam(G_{a_h}^{s+1}) + 1\right),
\end{aligned}$$

which proves equation 1

We now show that for some s the graph formed by components $\mathcal{G}^s$ contains induced P_n. It easy to see that the value $\sum_{i=1}^{n} 2^i \, (diam(G_i^s))$ increases in each step, and therefore the diameter of some $\mathcal{G}^s$ will eventually be larger than n and so the statement follows.

It remains to bound the number of vertices in the graph in the end. Let $\mathcal{G}^s$ be the components after the last step. Clearly all the vertices of the graph are the vertices of all the components in $\mathcal{G}^{s-1}$ since there are no vertices added in the last step. Each component from G_i^{s-1} has diameter at most $n-2$ (does not contain induced P_n), therefore has at most $2^i \cdot (n-1)$ vertices (by Equation 1). Thus, there are at most $\sum_{i=1}^{k} 2^i \cdot (n-1) \leq n \cdot 2^{k+1}$ vertices in total. □

The next step would be to show that we can bound the amount of vertices needed to force a tree for a small k, since, as we will see later on, we can force several graphs stemming from trees. This approach, however, works only for k of fixed size of three.

Theorem 2.3. For each tree T on n vertices with maximal degree d it holds that $m(T, 3) \leq n + 1 + (d+1)(n-2)$.

Proof. Strategy for *Player A* is constructing one tree A and one forest B simultaneously such that to following conditions hold.

- Tree A and forest B are not connected via edge nor vertex.
- Initially A is singleton and B is a forest consisting of $|V(T)|$ isolated vertices.
- The tree A is isomorphic to induced sub-graph of T and B is isomorphic to T with some removed edges. For each vertex $v \in V(A)$ we denote the corresponding vertex in $V(T)$ by v_T. And for each vertex $v \in V(B)$ we denote the corresponding vertex in $V(T)$ by v_T.

Player A proceeds as follows.

- As the stable set *Player A* always selects one vertex v from tree A and two vertices u, w from forest B, such that $d_A(v) < d_T(v_T)$, $u_T w_T \in E(T)$ and $uw \notin E(B)$.
- If *Player B* adds only the edge uw then the conditions are still satisfied. Otherwise *Player B* connects vertex u or w to the vertex v. Without loss of generality suppose he connects at least u to v. If that

is the case, *Player A* considers the vertex u a part of tree A (that is, he removes it from B and add it to A) and then removes all the neighbors of u in forest B, and if vertex w is connected to either u or v he removes w from forest B as well. Then he replaces all removed vertices from forest B by new isolated vertices.

Once *Player A* cannot select such a stable set, either tree A or forest B is clearly isomorphic to T. Initially there are $n+1$ vertices. For each vertex added to auxiliary tree A, *Player A* has to replace at most $d+1$ vertices in forest B. He needs to do this at most $n-2$ times (Since the replacement step is not needed in the very last step). So he needs at most $n+1+(d+1)(n-2)$ vertices in total. □

3 Paths and trees on polynomial number of vertices

In the previous section we obtained polynomial bounds for paths when k is fixed. In this section we describe a way to force paths and trees in polynomial time in both the number of vertices and k, but the polynomial bounds are worse.

Parameter $m(H,k)$ is clearly increasing in k. Since we want polynomial bound in both n ($n = V(H)$) and k, we can assume that $k = n$ (otherwise we use either larger k or force a larger tree that contains induced former one).

3.1 Paths

We say that *Player A saturates* a vertex set V of a graph G by selecting arbitrary stable set within V for as long as stable set of size k exists within V. We call vertex set $k-$*saturated* if there is no stable-set of size k present.

We use well known Turán's theorem and one more easy observation.

Lemma 3.1 (Turán's theorem)**.** For every graph G it holds that $\alpha(G) \leq \frac{n}{1+d_{avg}(G)}$ where d_{avg} is the average degree of G.

Lemma 3.2. For every graph G there exists an induced sub-graph H such that $d_{min}(H) \geq \frac{d_{avg}(G)}{2}$.

Proof. Inductively remove vertices with minimal degree. □

We show how to force paths in two steps. First we force non-induced paths. Then we build a "grid" from them and use it to force an induced path in it.

Lemma 3.3. For every $n \in \mathbb{N}$ *Player A* can $(2n^2, n)$-force a graph containing a path P_n.

Proof. Let G be a graph on $2n^2$ vertices. Assume *Player A* saturates $V(G)$. Thus by Turán's theorem and the fact that $V(G)$ is saturated

$$d_{avg}(G) \geq \frac{2n^2}{\alpha(G)} - 1 > \frac{2n^2}{n} - 1 \geq 2n.$$

By Lemma 3.2 we immediately obtain H a sub-graph of G with minimal degree n. Such H surely contains n-vertex path. □

Theorem 3.4. $m(P_n, n) \leq 2n^4$.

Proof. By iterating Lemma 3.3 *Player A* can $(2n^4, n)$-force a graph containing n^2 independent non-induced paths of size n.

Therefore this graph contains induced grid with n rows and n^2 columns where there are no edges in between vertices from different columns and every vertex is connected with the vertex directly above it. Then let *Player A* saturate every row of this grid independently. Note that the only edges in this grid are within columns or rows. We call this grid D. We show that this grid, and consequently the whole graph, contains induced P_{2n-1}.

We call a vertex directly above a vertex u in D a *parent* of u. And a *right-neighbors* of a vertex v in D are neighbors of v that are in the same row to the right from v. We call a vertex of the grid *good* if it is either in the first row and has a right-neighbor, or if it has a right-neighbor that has has a good parent.

We will prove that there exists a good vertex v in the last row. If this holds, then we can find by induction an induced path on $2n$ vertices in D starting in v alternating between going right (possibly by more than one column) and up by one row: We start in such vertex v. Suppose the path ends in some good vertex w. By definition, every good vertex that is not in the first layer has a right-neighbor z that has a good parent u. Since every two consecutive vertices in columns are connected, u is connected to z and we can extend the path from w by z and then by good vertex u. Any good vertex in the first layer has a right-neighbor, therefore we can also extend the path by such vertex. Clearly the resulting path alternates between going

right and up by one row, hence consists of $2n$ vertices in total. Since D does not contain any diagonal edge the resulting path is induced.

Therefore it suffice to prove that D contains a good vertex in a last row. We show by induction that i-th row contains at least $n^2 - i \cdot (n-1)$ good vertices. It holds for the first row because otherwise there would be at least n bad vertices and they would form an independent set (because bad vertices does not have right-neighbors) which is not possible since all rows are saturated. Suppose that there are at least $n^2 - i \cdot (n-1)$ good vertices in the i-th row. Look at the vertices right bellow them and denote them by S. By the similar argument as for the first row at most $n-1$ vertices from S has no right-neighbor in S (otherwise the row would not be saturated). Therefore at least $n^2 - i \cdot (n-1) - (n-1)$ vertices from S are good and we are done since the last row contains at least $n^2 - n \cdot (n-1) = n$ good vertices.

□

3.2 Trees

The approach for paths can be generalized for trees. The foundation for paths were non-induced paths, for trees we use *locally induced* trees.

Definition 3.5. Let T be a rooted tree and G be a graph. We say that G contains *locally induced* T if there is a sub-graph S of G isomorphic to T such that the sons of each vertex of S are not connected by any edge in G.

To find locally induced trees in graphs, a large average degree is not sufficient. Therefore we use a different kind of saturation.

Player A saturates family $\mathcal{F}$ of stable vertex subsets of graph G by selecting stable sets containing at most one vertex from each set in $\mathcal{F}$ for as long as there is one within the graph. We call $\mathcal{F}$ *k-saturated* if there is no such stable-set of size k present.

Lemma 3.6. In every *n-saturated* family $\mathcal{F}$ of size at most $2n^3$ consisting of disjoint independent sets of size at most $2n^3$, there are at most $4n^5 - 1$ vertices that has less than n neighbors in every vertex set of $\mathcal{F}$.

Proof. Suppose there would be more than $4n^5 - 1$ vertices that has less than n neighbors in every vertex set of $\mathcal{F}$. Denote the set of these vertices by A. Since every vertex u from A has at most $2n^3 \cdot (n-1)$ neighbors, we can inductively find an independent set of size n as a subset of A: In each step simply pick any remaining vertex u from A, remove all neighbors of u

and remove all vertices from the set from $\mathcal{F}$ the vertex u is in. We remove $2n^3 \cdot (n-1) + 2n^3$ vertices in each step and so we can do at least n steps since $|A| \geq 4n^5 - 1$.

Such independent set cannot exists since $\mathcal{F}$ is *n-saturated*, a contradiction. □

Lemma 3.7. Let T be an arbitrary rooted tree on n vertices. Then *Player A* can $(4n^6, n)$-force a graph containing locally induced T.

Proof. Let G be a graph on $4n^6$ vertices without edges. Let *Player A* saturate a family $\mathcal{F}_0$ consisting of $2n^3$ disjoint independent subsets of $V(G)$ of size $2n^3$. By Lemma 3.6 at most $4n^5 - 1$ vertices from $V(G)$ has less than n neighbors in all sets from $\mathcal{F}_0$. Denote such vertices by B_0 and let $\mathcal{F}_1 = \{S \setminus B_0 : S \in \mathcal{F}_0\}$. By the same argument at most $4n^5 - 1$ vertices from $\bigcup \mathcal{F}_1$ has less than n neighbors in all sets from $\mathcal{F}_1$. Inductively we can form sets B_i and families $\mathcal{F}_i$ as long as there are vertices left. Since we remove at most $4n^5 - 1$ vertices in each step, we can perform at least n such steps.

Now we can find a locally induced tree starting with the root. Let root be an arbitrary vertex from an arbitrary set in $\mathcal{F}_n$. It has at least n pairwise non-adjacent neighbors from one of the sets in $\mathcal{F}_{n-1}$, so we can select children of the root. We can proceed in similar fashion for the rest of the vertices: Select one leaf l of the partially completed tree that is still missing its children. Such leaf is in some vertex set from i-th family $\mathcal{F}_i$. It has n pairwise not-adjacent neighbors (denote this vertex set by K) in some vertex set from $\mathcal{F}_{i-1}$. Select children of l from the vertex set K such that they are disjoint with partially completed tree. Since T has n vertices (and so at most n layers), it is always possible and the construction works. □

Now it's time to use the similar grid construction we used for paths. Previously, we found a path that has every even edge in a column and odd one in a row. We proceed similarly for trees. We just use different column and row structure.

Theorem 3.8. Let T be an arbitrary rooted tree on n vertices. Then $m(T, n) \leq 16 \cdot n^{12}$.

Proof. Let S be an auxiliary tree formed from T by preserving exactly vertices in odd layers (root is in the first layer) of T and adding edges connecting vertices with their grandchildren in T (in other words we contract all edges in between vertices from even layers and their parents), see Figure 2

for an example. Define function $f : V(S) \to V(T)$ mapping vertices of S to their original position in T. Additionally, we order vertices of S by breath-first-search (it is only important that sons have higher number than their parents).

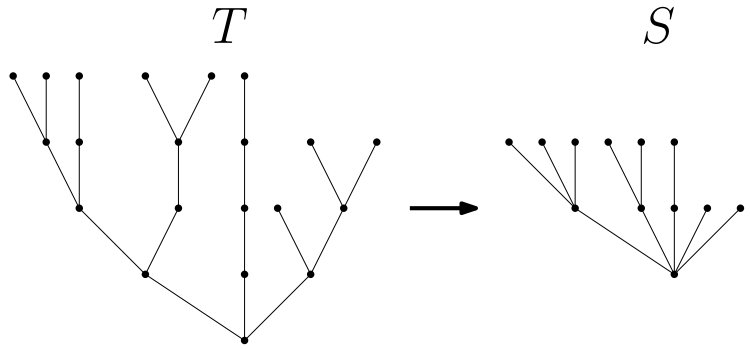


Figure 2: Auxiliary tree S.

By iterating Lemma 3.7 *Player A* can $(16n^{12}, n)$-force a graph containing an induced grid with $4n^6$ columns and $|V(S)|$ rows where there are edges only in between vertices from same columns and every column contains locally induced S with vertices ordered decreasingly (since there are $|V(S)|$ rows, every columns contains only vertices of its copy of S. Therefore roots of copies of S are in the last row). Then, let *Player A* for every row n-saturate a partition of vertices of the row into $2n^3$ sets of size $2n^3$. The only edges in this grid are within columns or rows. We call this grid D. For every vertex u of D, we call vertices above u in the same column a *predecessors* of u, and we call neighbors of u from the same row *siblings* of u.

Similarly as for paths, we call a vertex of the grid *good* if it is either in the first row and has at least n independent siblings, or if it has at least n independent siblings so that all predecessors of these siblings and of v are good. We show that there exists a good vertex in the last row and that we can find an induced copy of T in D.

Using Lemma 3.6 for induction on layers it follows immediately that i-th row contains at least $4n^6 - i \cdot (4n^5 - 1)$ good vertices (we used similar approach in the proof of Lemma 3.4).

To find an induced copy of T in D, we construct a function $g : V(T) \to V(D)$ mapping tree T to its induced copy in D so that for every vertex v from odd layer of T it hold that $g(v)$ is good and $f(g(v)) = v$ (when we look at a vertex $g(v)$ as a vertex of S). We proceed by induction on T.

Firstly, we know that there exists a good vertex v in the last row of D. We set $g(r) = v$ where r is a root of T. Clearly v is good and $f(g(r)) = r$ because we are in the last row. Suppose there is some vertex v from an

odd layer of T for which g is defined on v and $g(v)$ is good but g is not defined on the sons of v. Since $g(v)$ is good, it has at least n independent siblings such that all predecessors of these siblings are good. Thus we can define g for every son u of v so that $g(u)$ is in among these siblings and the image of g does not contain two vertices in a same columns (there are at least n siblings and only n vertices of T). Moreover, every column is a copy of S and for every $g(u)$ it holds that $f(g(u)) = v$, thus we can define g for every son w of every u such that $g(w)$ lies in the same column as $g(u)$ and $f(g(w)) = w$ (also note that $g(w)$ is good because v is good). See that each such $g(w)$ lies in a unique row, because the row corresponds to w in S. The found tree is induced because it is locally induced and every vertex with its sons lies in a unique row or column.

□

4 Cycle Strategies

In this section we present several strategies that force cycles of given length which gives us upper bounds on $m(C_n, k)$. Surprisingly the case of C_4 had to be solved differently.

For integers m, n let $PS(m, n)$ be a graph obtained from $K_{1,m}$ by subdividing all edges $n-2$ times. We denote $PS_{end}(m, n) \subseteq V(PS(m, n))$ the set of m vertices corresponding to the original partity of size m in $K_{1,m}$. Note that the number of vertices in $PS(m, n)$ is equal to $(n-1)m+1$ and since it is a tree, *Player A* can $\left(16\left((n-1)m+1\right)^{12}, (n-1)m+1\right)$-force it.

Theorem 4.1. For every $n, k \in \mathbb{N} : n \geq 3$, it holds that $m(C_{2n-1}, k) \leq n^{O(1)}k^{O(1)}$.

Proof. By theorem 3.8 *Player A* can $(O((n(k-1)+1)^{12}), k)$-force induced $PS(k, n)$. In the next step *Player A* selects $PS_{end}(k, n)$ as the stable set. This connects endpoints of two paths in $PS(k, n)$, forming an induced C_{2n-1}. □

The proof for odd cycles can be generalized for even cycles, with the exception of C_4, while still keeping the bound polynomial.

Theorem 4.2. For every $n \in \mathbb{N} : n \geq 3$, it holds that $m(C_{2n}, k) \leq n^{O(1)}k^{O(1)}$.

Proof. In order for *Player A* to force an even length cycle, he can force an induced path on three vertices in between the endpoints $PS_{end}(p, n-1)$ of path star $PS(p, n-1)$ where $p = m(P_3, k)$. By Theorem 3.4 $p \leq O(k^4)$ and by Theorem 3.8 we know that $m(PS(p, n-1), k) \leq O((n \cdot k^4)^{12})$. □

Theorem 4.3. For the cycle of length four it holds that $m(C_4, k) \leq k^4$.

Proof. Gyárfás et al. [10] proved that for any weakly C_4-free graph the following property holds:

$$\alpha(G)^2\omega(G) \geq V(G) \tag{2}$$

Meaning any graph without induced C_4 contains either a big clique or a big stable set.

Let *Player A* (k^4, k)-force a graph containing k^3 independent induced copies of P_2 (he needs at most k vertices for each path). Let

$$X = \{x_1, x_2, ..., x_{k^3}\}$$

be the set consisting of one endpoint from each forced path, and

$$Y = \{y_1, y_2, ..., y_{k^3} : x_i y_i \in E(G)\}$$

be the set of the remaining endpoints. Let *Player A* saturate X. The resulting graph on X does not contain stable set of size at least k. Therefore by Property 2, either induced C_4 appears in $G[X]$ or $\omega(G[X]) \geq k$, hence some clique K_ω of size at least k would be present on the vertices of set X. Let $\{x_{a_1}, ..., x_{a_k}, ..., x_{a_\omega}\}$ be the vertex set of this clique. Now *Player A* selects the stable set $\{y_{a_1}, ..., y_{a_k}\}$, which forces some edge $y_{a_m} y_{a_l}$. And so there is induced cycle $x_{a_m} y_{a_m} y_{a_l} x_{a_l}$. □

Acknowledgements

The authors would like to thank Sophie Spirkl for many helpful discussions.

References

[1] N. Bousquet, A. Lagoutte, and S. Thomassé. The erdős-hajnal conjecture for paths and antipaths. *J. Comb. Theory Ser. B*, 113(C):261–264, July 2015.

[2] Maria Chudnovsky. The erdős-hajnal conjecture—a survey. 06 2016.

[3] Maria Chudnovsky, Jacob Fox, Alex Scott, Paul Seymour, and Sophie Spirkl. Sparse graphs with no polynomial-sized anticomplete pairs, 2018.

[4] Maria Chudnovsky and Shmuel Safra. The erdős-hajnal conjecture for bull-free graphs. *J. Comb. Theory, Ser. B*, 98:1301–1310, 11 2008.

[5] David Conlon, Jacob Fox, and Benny Sudakov. *Recent developments in graph Ramsey theory*, pages 49–118. London Mathematical Society Lecture Note Series. Cambridge University Press, 2015.

[6] Paul Erdős and András Hajnal. On spanned subgraphs of graphs. *Beitrage zur Graphentheorie und deren Anwendungen, Kolloq. Oberhof (DDR)*, 01 1977.

[7] Paul Erdős and András Hajnal. Ramsey-type theorems. *Discrete Applied Mathematics*, 25(1):37 – 52, 1989.

[8] Paul Erdős, András Hajnal, and János Pach. A ramsey-type theorem for bipartite graphs. *Geombinatorics*, 10, 01 2000.

[9] Jacob Fox and Benny Sudakov. Density theorems for bipartite graphs and related ramsey-type results. *Combinatorica*, 29(2):153–196, Mar 2009.

[10] András Gyárfás, Alice Hubenko, and József Solymosi. Large cliques in -free graphs. *Combinatorica*, 22(2):269–274, Apr 2002.

[11] Anita Liebenau, Marcin Pilipczuk, Paul Seymour, and Sophie Spirkl. Caterpillars in erdős-hajnal. *Journal of Combinatorial Theory, Series B*, 136:33 – 43, 2019.

[12] Hans Jürgen Prömel and Vojtěch Rödl. Non-ramsey graphs are clogn-universal. *Journal of Combinatorial Theory, Series A*, 88(2):379 – 384, 1999.

Notes about Rainbow Cycles

Petra Pelikánová, Sophie Spirkl, Aneta Šťastná

Abstract

Caccetta-Häggkvist conjecture claims that a directed cycle of length at most $\leq \lceil \frac{n}{k} \rceil$ can be found in a digraph D on n vertices with minimal output degree at least k. There exists a generalization of this conjecture by Aharoni, which takes undirected graph G with an edge coloring using n colors such that each color is used on at least k edges and says that then exists a cycle of length at most $\leq \lceil \frac{n}{k} \rceil$ in G, which does not repeat colors. We prove that Aharoni conjecture holds if we are guaranteed to have at least $\mathcal{O}(k^6)$ edges of each color.

Conjecture 1 (Caccetta-Häggkvist)**.** Let D be a directed graph on n vertices with minimal output degree at least k. Then there exists a directed cycle of length at most $\leq \lceil \frac{n}{k} \rceil$ in D.

Caccetta-Häggkvist conjecture is proven for $k \leq 5$ and there are many approximate results, which are described in a summary article by Sullivan [3]. Shen [2] proved that with output degree at least k there is oriented cycle of length at most $\leq \lceil \frac{n}{k} \rceil + 73$.

Aharoni came up with a generalization of this conjecture with substituting the orientation by a coloring of edges. In an edge colored graph we say that a cycle is *rainbow* if all edges of the cycle have distinct colors.

Conjecture 2 (Aharoni)**.** Let G be an undirected graph G on n vertices with edges colored by n colors such that each color is used on at least k edges. Then, there is a rainbow cycle of length at most $\leq \lceil \frac{n}{k} \rceil$ in G.

Caccetta-Häggkvist conjecture is a special case of Aharoni conjecture, which can be observed by coloring all edges going out of a vertex by a unique color. The obtained graph is an instance of Aharoni conjecture, where a subgraph induced by edges of one color is always a star.

Aharoni conjecture is proven for $k = 2$ by Caccetta and Häggkvist [1], but no other results are known. When trying to prove the general case, one might want to ask: What if we had more than k edges of each color? How many would be enough to have a rainbow cycle of length $\leq \lceil \frac{n}{k} \rceil$? It can be proven that $\mathcal{O}(k^6)$ edges of each color suffices for any k, as will be shown in the rest of this article.

First thing which will be useful to know is that there exists a function f of k such that if G has n vertices and $n + f(k)$ edges, then there exists a cycle of length at most $\lceil \frac{n}{k} \rceil$.

Lemma 3. Let $G = (V, E)$, $|V| = n$, $|E| = n + f(k)$, where $f(k) = 128k^3$. Then there exists a cycle of length at most $\leq \lceil \frac{n}{k} \rceil$ in G.

Proof. Pick G_0 a spanning tree of graph G and count its potential function $P_0 = \sum_{u,v \in V(G)} dist_{G_0}(u, v) \leq n^3$ for $\mathrm{dist}_{G_0}(u, v)$ the distance between u and v in the spanning tree G_0. The upper bound holds because maximum distance between two vertices is n and there are n^2 pairs of vertices. If this lemma does not hold, all cycles in G are longer than $\frac{n}{k}$. Now take the shortest cycle in G and denote it C_l, where $l > \frac{n}{k}$ is the length of the cycle. Among all spanning trees of G we choose G_0 such that it contains $C_l \setminus e$ for some edge e.

We create G_1 by adding edge e to G_0 and denote P_1 the potential of G_1. How do P_1 and P_0 differ? An estimate can be done by looking at the segments of C_l of length $\frac{1}{8} \cdot \frac{n}{k}$ next to the edge e, as in Figure 3. The combined length of the segments and edge e is $\frac{1}{4} \cdot \frac{n}{k} + 1 \leq \frac{1}{4} \cdot l$. So when we take vertices u, v from segments on opposite sides of edge e,

$$\begin{aligned} \mathrm{dist}_{G_0}(u, v) &> \frac{3}{4} l \\ \mathrm{dist}_{G_1}(u, v) &\leq \frac{1}{4} l \\ \mathrm{dist}_{G_0}(u, v) - \mathrm{dist}_{G_1}(u, v) &\geq \frac{1}{2} l > \frac{n}{2k} \end{aligned}$$

As there are $\left(\frac{n}{8k}\right)^2$ such tuples u, v, we get

$$P_1 \leq P_0 - \left(\frac{n}{8k}\right)^2 \cdot \frac{n}{2k} = P_0 - \frac{n^3}{128k^3}$$

If there is a cycle of length $\leq \frac{n}{k}$ in G_1, then it was enough to add one edge. Otherwise we take the shortest cycle in G, from which one edge f is missing

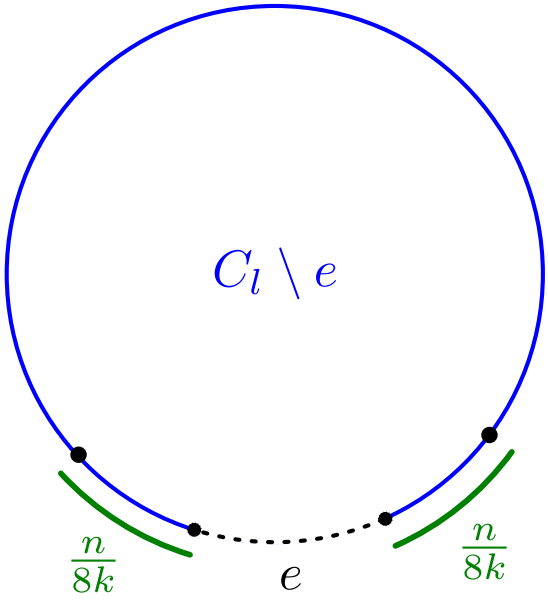


Figure 3: Illustration of which points are used to estimate the change in potential.

in G_1. Note that as it does not have any chord, it has to be included in G_0 up to one edge, and thus at most one edge from it is missing in G_1. We construct G_2 by adding f to G_1 and again observe the potential, which decreases by the same amount. In general, $P_i \leq P_{i-1} - \frac{n^3}{128k^3}$.

How many times we can repeat this until all edges of G are in some G_i? In each step the potential shrinks by $\frac{n^3}{128k^3}$, $P_0 \leq n^3$ and potential cannot be negative as it is a sum of non-negative distances. Thus, we have to stop after

$$\frac{n^3}{\frac{n^3}{128k^3}} = 128k^3$$

steps and we need to add at most $128k^3$ edges to create a cycle of length at most $\frac{n}{k}$. □

We will say that a color *is removed* from G if there is no edge of this color in G. We denote the number of colors on edges of G by $c(G)$ and set of edges of color c by E_c. Note that $E(G) \geq c(G)$.

For a vertex v, the *color degree* of v for color c, denoted by $\deg^c(v)$, is the number of edges of color c incident to v.

Lemma 4. If we remove $f(k) = 128k^3$ vertices from G edge colored by n colors without removing any color from G, then G contains a rainbow cycle of length at most $\lceil \frac{n}{k} \rceil$.

Proof. By removing the subset of vertices of size $f(k)$ without removing any color, we obtain G', where $|V(G')| = n - f(k)$ and $c(G) = n$. In this case we can take a subgraph G'' of G' with only one edge (chosen arbitrarily) of each color. Then $|V(G'')| = n - f(k) = n''$, $|E(G'')| = c(G') = n = n'' + f(k)$. By Lemma 3, the graph G'' contains a cycle of length at most $\lceil \frac{n}{k} \rceil$ and from the construction it contains only one edge of each color, so the cycle has to be rainbow. □

But what if there is no such subset of $f(k)$ vertices which is needed in assumptions of Lemma 4? Here more edges of each color will be useful, so suppose $|E_c| \geq (f(k) + \epsilon)^2$ for each color c, where ϵ will be defined later. Taking a color c, we can ask:

Question 5. Is there a subset S of $f(k)$ vertices such that all edges of color c are incident to it?

a) No. In this case color c remains in G after removing any subset of vertices of size at most $f(k)$, so we call these colors *irremovable*. If the answer is no for all colors, then we get short rainbow cycle from Lemma 4 by picking any subset of vertices of size $f(k)$.

b) Yes. We call such colors *removable*. In this case exists a vertex v_c from the following Lemma 6.

Lemma 6. If $|E_c| \geq (f(k) + \epsilon)^2$ and there is a subset S of vertices such that $|S| = f(k)$ and all edges from E_c are incident to S, then there exists a vertex v_c with $\deg^c(v_c) \geq f(k) + \epsilon$.

Proof. We consider an average color degree of a vertex from S.

$$\frac{\text{number of edges of color } c \text{ incident to } S}{|S|} \geq \frac{(f(k) + \epsilon)^2}{f(k)} \geq f(k) + \epsilon$$

The average $\deg^c(v)$ of v in S is $f(k) + \epsilon$. Thus, there exists a vertex v_c, such that $\deg^c(v_c) = f(k) + \epsilon$. □

The need for $|E_c|$ being quadratic with respect to $f(k)$ stems from Lemma 6, otherwise we could not assure that such a vertex exists. Note that there may be more vertices with $\deg^c(v_c) \geq f(k) + \epsilon$. In such case we choose exactly one such vertex v_c for each c. Then, for such fixed choice of v_c's we say that a color j has a *center* v_{c_j} if and only if there exists $v_{c_j} \neq v_{c_k}$ for all $k \in \{1, \ldots, n\} \setminus \{j\}$. For such colors we can take their centers and apply the Caccetta-Häggkvist conjecture, as we will see later.

Lemma 7. Let G be an edge colored graph with $c(G) = |V(G)| = n$. If there exists $\epsilon > k$ such that each vertex is incident with at least ϵ edges of some color c, which is unique for every vertex, then we can find a rainbow cycle of length at most $\lceil \frac{n}{k} \rceil$.

Proof. If we remove all edges of color c, which are not adjacent to the fixed vertex v with $\deg^c(v) \geq \epsilon$ for all colors, we get a graph G'' where each subgraph induced by one color is isomorphic to a star. In such graph we can give orientation to all edges of one color to go from the center of the star. Finally, we can use Caccetta-Häggkvist conjecture to find a short directed cycle which is equivalent to a rainbow cycle in G.

The only drawback is that Caccetta-Häggkvist has not yet been proven in general. Currently best general result is by Shen [2], who proved that if we have output degree k, then we can find a directed cycle of length at most $\lceil \frac{n}{k} \rceil + 73$. However, we can have $\epsilon = k + \delta$, $\delta > 0$ such that

$$\left\lceil \frac{n}{k+\delta} \right\rceil + 73 \leq \left\lceil \frac{n}{k} \right\rceil.$$

As trivially $\left\lceil \frac{n}{k+\delta} \right\rceil + 73 \leq \frac{n}{k+\delta} + 74$ and $\frac{n}{k} \leq \left\lceil \frac{n}{k} \right\rceil$, we can set δ such that the following stronger inequality holds.

$$\begin{aligned} \frac{n}{k+\delta} + 74 &\leq \frac{n}{k} \\ nk + 74k \cdot (k+\delta) &\leq n \cdot (k+\delta) \\ nk + 74k^2 + 74k\delta &\leq nk + n\delta \\ 74k^2 &\leq \delta \cdot (n - 74k) \\ \frac{74k^2}{(n-74k)} &\leq \delta \end{aligned}$$

Thus it is enough to pick $\delta \geq 74k^2$, which means $\epsilon \geq k + 74k^2$ to get a short rainbow cycle. □

Theorem 8. Let G be a graph on n vertices with edge coloring using n colors such that each color is used on at least $(f(k)+\epsilon)^2$ edges. Then there exists a rainbow cycle of length at most $\lceil \frac{n}{k} \rceil$ in G.

Proof. We apply Lemma 6 on every removable colors. For each removable color fix exactly one vertex v_c such that $\deg^c(v_c) \geq f(k)+\epsilon$. As the number of colors is the same as number of vertices, each color which does not have a

unique center v_c causes existence of at least one vertex x which is not v_c for any color c. Note that some vertices x exist due to the irremovable colors. Let t be a number of such vertices x. We consider two cases.

- $t \geq f(k)$: In this case we can delete exactly $f(k)$ vertices which are not v_c for any color c. We deleted exactly $f(k)$ vertices without removing any v_c. The number of edges of color c incident to v_c decreased at most by $f(k) \leq f(k) + \epsilon$. Thus, we did not remove any removable color. Clearly, we also did not remove any irremovable color. As a result, we did not remove any color and thus Lemma 4 can be applied to find a short rainbow cycle.

- $t < f(k)$: In this case we just remove all t vertices, which are not v_c for any c. All remaining vertices are v_c for some color c and furthermore they have lost at most $t \leq f(k)$ edges of this color c. Now we obtained a graph G' where each vertex has been a center for some color c in original graph and so $\deg^c_{G'}(v) \geq \epsilon$. Thus, we can apply Lemma 7.

□

Now it can be concluded that the total number of edges of each color which we need to obtain a rainbow cycle of length at most $\lceil \frac{n}{k} \rceil$ in G is

$$(f(k) + \epsilon)^2 = (128k^3 + 74k^2 + k)^2 \in \mathcal{O}(k^6)$$

References

[1] Louis Caccetta and Roland Häggkvist. On minimal digraphs with given girth. 1978.

[2] Jian Shen. On the caccetta–häggkvist conjecture. *Graphs and Combinatorics*, 18(3):645–654, Oct 2002.

[3] Blair D. Sullivan. A summary of results and problems related to the Caccetta-Häggkvist conjecture. *preprint*, 2006.

On Time of Turing Machines and Depth of Circuits

Pavel Dvořák, Kyle Hess, Lukáš Ondráček, Periklis Papakonstantinou, Jakub Pekárek

Abstract

We simulate Turing machines with running time T by circuits of polynomial size and depth as small as possible. If the depth is $o(T)$ it can be seen as space compression, because of the relation between space of Turing machines and circuits depth. In 1977 Hopcroft et al. [J. ACM 24, 2] proved that any Turing machine with running time T can be simulated by a Turing machine using space $\mathcal{O}(T/\log T)$. Since then, only small progress was done. We break the barrier $\mathcal{O}(T/\log T)$ and prove that any Turing machine with a slight restriction on writing can be simulated by a polynomial size circuit of depth $\tilde{\mathcal{O}}(T/\log^{3/2} T)$, where $\tilde{\mathcal{O}}$ hides $\log\log T$ factor.

We also present a simulation of a Turing machine by a circuit of bounded fan-in, size $\mathcal{O}(T \log T)$ and depth $\mathcal{O}(T)$. This improves the previous simulation by a circuit of size and depth $\mathcal{O}(T \log T)$.

1 Introduction

In this paper we study relations between two standard computational models: Turing machines and circuits[1]. There are two basic algorithm resources: time and/or space, i.e. how many steps an algorithm does to compute the result and how big memory an algorithm needs. More formally, let M be a Turing machine and n be a length of the input. Running time $T = T(n)$ of M is how many steps the machine M does before it halts. Space $S = S(n)$ of M is a number of cells visited by heads of M during the computation. Clearly, if M has k tapes then M cannot use more space than $k \cdot T$ (in each

[1] If you are not familiar with these models see [1] or other complexity book.

step, it can visit only k cells). Thus, $S \leq \mathcal{O}(T)$, but is it really necessary or can we simulate M by a machine M' which uses only $o(T)$ space? Hopcroft et al. [4] proved that any Turing machine M running in time T can be simulated by a Turing machine M' using space $\mathcal{O}(T/\log T)$ (however with huge running time – exponential in T). They use a pebbling game on a configuration graph of M to get the space compression. However, there is a lower bound against this method [7], so there is no hope to improve the result without a new method. Dymond and Tompa [3] used stronger machines (models) to speedup the computation. They proved that a Turing machine of running time T can be simulated by PRAM[2] running in time $\sqrt{T}$ and by alternating Turing machine running in time $T/\log T$, which improves the result of Hopcroft et al. [4].

Another popular computation model are circuits. There are again two basic measures of circuits: size and depth, i.e., how many gates the circuit has and the length of the longest path from input to output. Circuits form a non-uniform model of computation, i.e., we can have for each input size a different circuit (not as Turing machines where we have only one Turing machine for all input size). However, in this paper we study uniform circuit classes. Circuit class is uniform, if there is a Turing machine M such that on input n it generates a circuit for input size n and running time of M is polynomial in n.

If we simulate a Turing machine M by a circuit C or vice versa the size of C polynomially corresponds to the time of M and the depth of C polynomially corresponds to the space of M. We present basic results about the relation between time of Turing machines and circuits size in the next section. Borodin [2] proved that a Turing machine using space S can be simulated by a circuit of depth $\mathcal{O}(S^2)$. On the other way, it is not hard to see that a uniform circuit of depth d can be simulated by Turing machines using space $\mathcal{O}(d)$. Thus, simulating Turing machines by low-depth circuits can be seen as space compression. Williams [8] proved that random access Turing machine of running time T can be simulated by circuits of depth $\mathcal{O}(T/\log T)$ and size poly T.

Another important parameter of circuits is boundedness of fan-in, i.e., if input size of circuit gates is bounded by some constant. Note that any boolean function can be computed by a circuit of depth 2 and unbounded fan-in (and exponential size).

Computation of Turing machine is local, i.e., a content of a cell s in time

[2]Parallel random-access machine.

t depends only on content of three cells in time $(t-1)$ – the content of s and its two neighboring cells. We can generalize this property that a content of cell s in time t depends only on content of $\mathcal{O}(k)$ cells in time $t-k$ for $k < t$. This locality can be transferred to circuits which simulate Turing machines.

Definition 1.1. Let C be a circuit simulating a Turing machine such that gates of C are divided into layers $L_1, \ldots, L_d$ and inputs of gates in a layer L_i are only gates in a layer L_{i-1}. We say that C satisfies *linear-dependency property* if for every i and $k < i$ each gate in a layer L_i depends only on $\mathcal{O}(k)$ gates in a layer L_{i-k}.

We use the linear-dependency property in Section 4 to compress circuits, which allow us to prove the following theorem.

Theorem 1.2. Any multitape Turing machine running in time T can be simulated by an unbounded circuit of polynomial size and depth $\mathcal{O}(T/\log T)$ satisfying linear-dependency property.

Theorem 1.2 is stronger than the result of Hopcroft et al. [4], since the circuit from our proof has polynomial size. However, it is weaker than the result of Williams [8], because random access Turing machine is a stronger model than standard Turing machine. Still, our proof is much simpler than the proof of Williams [8].

By a construction using oblivious machines [1] (as we mention in the next section), we would get a simulation of a Turing machine by a bounded circuit of size and depth $\mathcal{O}(T \log T)$. We improve this simulation in Section 3, where we prove the following theorem.

Theorem 1.3. Any multitape Turing machine running in time T can be simulated by a bounded circuit of size $\mathcal{O}(T \log T)$ and depth $\mathcal{O}(T)$ satisfying linear-dependency property.

As far as we know all space compression or simulating by depth-bounded polynomial size circuits does not break the barrier $\mathcal{O}(T/\log T)$. In Section 5 and 6 we break this barrier for restricted Turing machines to $\tilde{\mathcal{O}}(T/\log^{3/2} T)$ where $\tilde{\mathcal{O}}$ hides $\log\log T$ factor. The first result is for singletape Turing machines with restricted number of reversals, i.e., the changing of head moving direction.

Definition 1.4. A *reversal* of a singletape Turing machine is a step in which the head moved in the opposite direction than its last movement.

Theorem 1.5. Let M be a singletape Turing machine running in time T which performs at most $T/\log T$ reversals. Then, the machine M can be simulated by an unbounded circuit of polynomial size and depth

$$\mathcal{O}(T/\log^{3/2} T).$$

We note that singletape machines are very weak. Paterson [6] proved that even a non-deterministic singletape Turing machines with running time $T \geq n^2$ can be simulated by a deterministic Turing machines using space $\mathcal{O}(\sqrt{T})$. Maass and Shorr [5] proved that any singletape Turing machine with running time $T \geq n^3$ can be simulated by Σ_2-Turing machine with running time $\mathcal{O}(T^{2/3}\log^2 T)$. Thus, if we combine Paterson simulation [6] with the result of Borodin [2] we would not get a circuit of $o(T)$ depth because of quadratic overhead of Borodin's simulation. Still, it is plausible that singletape Turing machines can be simulated by circuits of much smaller depth. However, in a proof of Theorem 1.5 we present our ideas which we use in a proof of our main result, which is very technical. Our main result is for multitape machines where we restricted the number of writes on all but one tape.

Theorem 1.6. Let M be a multitape Turing machine running in time T with one main tape and other tapes performing at most

$$\frac{T}{\sqrt{\log T \cdot \log\log T}}$$

writes. Then, the machine M can be simulated by an unbounded polynomial circuit of depth $\tilde{\mathcal{O}}(T/\log^{3/2} T)$.

2 Cook-Levin Tableau

In this section we quickly recapitulate the standard simulation of Turing machines by circuits. We use some presented ideas in proving our results. For more details about relations between Turing machines and circuits and proofs of results in this section, see [1]. Let M be a Turing machine running in time T. A *timeslice* of a computation of M is a content of the first T cells of the tape with one cell marked with a state of M, indicating the current position of head and the state of the machine. We arrange all T timeslices of the computation into a matrix of size $T \times T$, where each row is a timeslice, and two consecutive rows represent consecutive timeslices. As

a convention, we set the top row to represent the initial configuration, with time going downward. We call this matrix a *computation matrix*.

By locality of Turing machine computation, a content of every cell within the matrix depends only on the three cells in the previous timeslice. This dependence can be expressed as a circuit gadget of constant size and constant depth, encoding the Turing machine transition function. We refer to this gadget, and gadgets with similar semantic function as *transition gadget*.

Using the previous observation, we can directly construct a circuit. For every cell of the computation matrix we use a copy of the same transition gadget representing the transition function of M, except for the borders of the matrix. In the side borders, we adjust the transition gadget to accept less inputs. The top border is adjusted to a function as an input of the whole circuit, and at the bottom border we may either add a constant amount of gates to check if the simulated machine accepted its input, or use the data on the tape as an output. As a consequence we obtain the following lemma.

Lemma 2.1. Any singletape Turing machine running in time T can be simulated by a bounded circuit of size $\mathcal{O}(T^2)$ and depth $\mathcal{O}(T)$ satisfying linear-dependency property.

Consider a multitape Turing machine M. In a similar manner we can represent the computation matrix as a 3-dimensional matrix, where each cell in the matrix corresponds to a specific tape cell of a specific tape in a specific time. The important difference is, that each cell can still be determined by at most 3 cells of the same tape from the previous timeslice, but also by any cells of the other tapes, depending on the positions of their respective heads in given time. Following the construction above, each gadget needs to first obtain the relevant data from the other tapes, which requires an unbounded OR gate (of fan-in $\mathcal{O}(T)$), or a binary tree of OR gates of depth $\mathcal{O}(\log T)$.

Lemma 2.2. Any multitape Turing machine running in time T can be simulated by an unbounded circuit of size $\mathcal{O}(T^2)$ and depth $\mathcal{O}(T)$ or a bounded circuit of size $\mathcal{O}(T^3)$ and depth $\mathcal{O}(T \log T)$.

Another approach is to create a more complex circuit, where each point in time can be represented by multiple timeslices, depending on the position of the heads. To do this, we create one timeslice for every point in time and a k-tuple of positions of head (where k is the number of tapes). Since every k-tuple of head positions can result from only a constant number of k-tuples in the previous timestep, cells in each timeslice are determined by

constant number of cells spread across a constant number of timeslices. The total number of timeslices is T^{k+1} (the k for number of tapes and $+1$ for time coordinate). This gives us a bounded circuit of size $\mathcal{O}(T^{k+2})$ (each time-slice has kT gadgets of constant size) and depth T.

Lemma 2.3. Any k-tape Turing machine running in time T can be simulated by a bounded circuit of size $\mathcal{O}(T^{k+2})$ and depth $\mathcal{O}(T)$ satisfying linear-dependency property.

These approaches used a computation matrix. There is also another approach using oblivious machines, i.e., such machines where position of their heads depend only on the size of input and an actual time step of the computation. It is easy to see that each Turing machine M with running time T can be simulated by an oblivious Turing machine with running time $\mathcal{O}(T^2)$. We just replace each step of M by reading all tapes. However, there is more involved construction which has only logarithmic overhead.

Lemma 2.4. Each Turing machine M with running time T can be simulated by an oblivious Turing machine with running time $\mathcal{O}(T \log T)$.

In the next section we design an improvement of such simulation which we use to prove Theorem 1.3. Oblivious Turing machines can be simulated in much smarter way than using the computation matrices. Instead of having gates for a content of each cell in each time step, in this construction the gates represent *snapshots*. A snapshot is a machine state and symbols read by all heads. By careful inspection of how snapshots depend on each other, one can prove the following result.

Lemma 2.5. Each Turing machine M with running time T can be simulated by a bounded circuit of depth and size $\mathcal{O}(T \log T)$.

3 Oblivious Machines

In this section we use a more careful approach to the ideas from the previous section. In Lemma 2.2, the resulting circuit is not bounded (unless we pay by increased depth) and lacks the linear dependency property and in Lemma 2.3 the circuit size depends on the number of tapes non-linearly. In both cases the cause of the undesired effects stems from the need to account for all possible positions of the head on various tapes. To improve Lemma 2.5, first we follow the naive "quadratic" oblivious construction.

Lemma 3.1. Any multitape Turing machine running in time T can be simulated by a bounded circuit of size $\mathcal{O}(T^2)$ and depth $\mathcal{O}(T)$ satisfying linear-dependency property.

Proof. Let M be the initial Turing machine and M' be an oblivious Turing machine with running time $\mathcal{O}(T^2)$ simulating M – the simulation was sketched in the previous section. We construct a circuit analogue of M'. We imagine M' as having all tapes of size $2T+1$, T cells in each direction from the origin with the heads starting in the center. We represent each timeslice of such machine as $\mathcal{O}(T)$ gates. We use transition gadget of M to represent the computation with heads always standing on the central cell. In every slice we use the output of the gadget to determine the movement of each head, and wire the information to each cell of the respective tape in the next timeslice. The cells in the next timestep are constructed as constant size circuits, copying the data either from the cell directly above, or one step to the side, depending on the received signal describing movement of the respective head in the respective timestep. We handle the tape cells around origin in the natural way. □

This construction can be improved by taking standard Turing machine oblivious construction [1] and redesigning it in a non-trivial way, so that all data movement can be parallelized based on local rules. The main idea is that "we do not move the heads, but the tapes under the heads". We split the tapes to blocks of exponentially growing size in both directions from the head positions and we move the content of large blocks rarely. Thus, there is only logarithmic overhead. However, in the standard oblivious construction when we move a content of some block it can be spread over many blocks after the moving. We introduce new rules for moving data to be more local, i.e., the content of moving block ends in up to two different blocks, which are next to the original block. These new rules are much more "friendly" to the circuit simulation and they allows us to design a small circuit.

Theorem. Any multitape Turing machine running in time T can be simulated by a bounded circuit of size $\mathcal{O}(T \log T)$ and depth $\mathcal{O}(T)$ satisfying linear-dependency property.

Proof. First, we describe an oblivious Turing machine M' simulating a fixed Turing machine M. After that, we design a circuit C (which simulates the machine M'), exploiting some properties of circuits such as parallel handling of different parts of the tape. We begin the construction by redesigning

the way in which we interpret the data stored on the tape. We extend the alphabet to allow for storing two symbols per cell, including special symbols of "blank" and "void". We refer to the two symbols as top and bottom. We split the tape into blocks, growing exponentially (by factor of two) in both directions from the starting head position. We refer to the starting head position of M' as to the origin cell, and to the symbols stored in the top or bottom part of cells of each block as a slot. For convenience we number the blocks to the left and right from the origin cell, starting with 0. Therefore both left and right side blocks with number 0 have size 1 (each containing two slots storing exactly 1 symbol), and the blocks with number k (or k-block for short) in general consist of two slots storing exactly 2^k symbols each.

The crucial property of the construction is that one block, consisting of its top and bottom slot, fits exactly into a single slot of the next (bigger) block, and vice versa. For technical reasons, the origin cell represents only a single symbol of M, which is the symbol under the head of M. To the left and right of the origin cell we have blocks of size 1. This allows us to technically extend the previous rule, where the whole content of the origin cell fits exactly into one slot of the blocks on either side.

We represent the content of M in M' as follows. We define an order of slots on the right side of the origin cell (and analogously on the left side) so that slots of smaller blocks precede slots of bigger blocks, and each top slot precedes bottom slot of the same size. The content of the tape to the right of the head of M is represented by the slots to the right of the origin. Similarly for the left side. The content of the tape of M to the left of its head is obtained by a concatenation of all slots in the specified order, skipping the void symbols. The blank symbols represent the cells of M that are empty, and the void symbols are parts of the tape of M' not representing any part of the tape of M. We keep the invariant that each slot is either empty (containing only void symbols) or full (containing only non-void symbols, but including blanks).

The simulation proceeds as follows. In order to represent all $2T+1$ cells of M which can in theory be accessed by M, we use blocks up to number $\lceil \log T \rceil$ on either side. The input is stored only in the top slots of the closest right-side blocks, with the first symbol stored in the origin cell. All the top symbols containing no symbols of the input are blank symbols and all bottom slots contain only void symbols. In the origin cell we have the symbol underneath the head of M. In each step, the transition function of M is used to replace the symbol in the origin cell, and to decide whether

the head of M moves to the left or right (or not at all). Without loss of generality let us assume the case where the head moves to the right. We rearrange the representation of the tape of M so that the current content of the origin cell is shifted into the slots on the left, and a new symbol from the slots on the right enters the origin cell (maintaining consistency and invariants of the representation). By doing so we obtain a representation of the next configuration of M (assuming we also store the new state), and can continue with the next step of the computation. We want to achieve this using only local rules. That is, the movement of slot contents is only influenced by a constant number of surrounding slots. Therefore all of these movements can then be simulated in parallel by circuits of constant fan-in.

The movement rules are following. Suppose we want to simulate step $c \cdot 2^{\ell-2}$ of M for c odd. For each $0 \leq k \leq \ell$ we consider the blocks with numbers k, $k+1$, $k+2$ on each side. Note that for small values of $k \leq 1$, we repeat the rules several times between each step of simulation. The general idea is that we want to prevent the k-block and $(k+1)$-block to be either both completely empty or both completely full. When both are full (all four slots are full), we take data from both slots of the $(k+1)$-block and store them in an empty slot of the $(k+2)$-block (if such empty slot exists). Similarly if both are empty, we take a full slot of the $(k+2)$-block (provided it exists), and use it to fill both slots of the $(k+1)$-block. In the following lemma we prove that these rules guarantee that no three consecutive blocks are either all full or all empty after the application of the rules. This means that whenever the conditions of the rules are satisfied, the corresponding empty/full blocks needed to apply the rules always exist. Even more crucially, the rule only affect the $(k+1)$- and $(k+2)$-blocks and it is not possible for rules to affect the same block twice during the simulation of one step of M.

Now, we describe a circuit C simulating M'. As in the tableau construction we use T layers of circuits of size $\mathcal{O}(T)$ to represent the tape of M' in individual steps of the simulation as timeslices. We use the transition gadget for M' to represent the operations on the origin cell in each step. It remains to show that the slot movements using the rules can be simulated by a constant depth and bounded fan-in circuits between each two steps, and that the total size of the circuit is $\mathcal{O}(T \log T)$. In each step when a rule is to be considered for a block of given number, we can use a constant-size circuit to decide whether a rule is going to be applied or not (as it is enough to test a single cell of any involved slot to decide whether it is empty or full, or we can use auxiliary indicator for each slot), and to decide on the

specific movement of data. Each cell in each block can receive its content from only constantly many cells from the previous timeslice. Therefore the circuits generating the next timeslice are of constant depth and fan-in.

To check that the size of the circuit is as promised, we consider which cell can change their content in each timeslice. Each rule affecting $(k+1)$- and $(k+2)$-blocks is potentially applied (and therefore implemented in the circuit) only once every 2^{k-2} steps. The total number of affected cells is $3 \cdot (2^{k+3})$. On average that is a constant number of gates per timeslice, (per rule) per block. There is a total number of $\mathcal{O}(\log T)$ blocks, so the total number of gates to implement the slot movements is $\mathcal{O}(T \log T)$. □

Lemma 3.2. The movement rules guarantee that no three consecutive blocks are all full or all empty after application of the rules.

Proof. First we show that each block of size k cannot have its slot filled from the lower block twice in less then 2^k steps, unless the $(k+1)$-block pushed its slot back into the k-block between the occurrences. We proceed by induction. The origin cell pushes its symbol into the 0-block at most once every step of the simulation. For an induction step, let us assume that the claim is true for a k-block. In order for a k-block to push its content into a slot of a $(k+1)$-block twice, both slots of the k-block must be filled. For contradiction let us assume the slots are filled by data pushed from $(k-1)$-block. By the induction hypothesis at least 2^{k+1} steps are required for this to happen, and so the required amount of steps is met for the $(k+1)$-block, or it pushes its data back into the k-block.

At the beginning of the simulation each block is exactly half-full. For contradiction, let us consider the earliest timeslice t_2 in which the guarantee is broken by three consecutive blocks getting full, and let k be largest such that the k-, $(k+1)$- and $(k+2)$-blocks are full. To simplify the argument, let us assume the $k \geq 2$. Let t_0 be the last timeslice (no later than t_2) in which the $(k+2)$-block was at most half-full. Note that if before the next timeslice the $(k+2)$-block receives data from the $(k+1)$-block, then the $(k+1)$-block is empty in the timeslice (t_0+1). Similarly if it receives data from the $(k+3)$-block, then by the conditions of the rule, $(k+1)$-block must be empty in t_0, and therefore at most half-full in t_0+1. Let t_1 be the last step before t_2 when the $(k+1)$-block is at most half-full. By the choice of t_1 and t_2, the $(k+1)$-block cannot receive data from the $(k+2)$-block between timeslices t_1 and t_2. Therefore, the remaining slot in $(k+1)$-block is filled from k-block, which is empty in the timeslice t_1+1. By the previous claim, it takes at least 2^{k+1} steps for the k-block to get full without the

$(k+1)$-block becoming half-full again. However, every 2^k steps, the rules are applied which consider the block triplet of $k+1$, $k+2$, $k+3$ and shift the data from $(k+2)$-block to $(k+3)$-block before the timeslice t_2, unless the $(k+3)$-block is also full. In the latter case, we get contradiction with the minimality of both k and t_2 and in the former case we get contradiction with the choice of t_2. Note that for the case where $k \leq 1$, the argument holds the same, with the numbers of steps involved being fractions (multiples of quarters), in which case we reinterpret these as number of timeslices.

We proceed analogously to show that no three consecutive blocks can be empty. First we show that for a k-block to have one of its slots pulled into a lower block twice in a row, at least 2^k steps must pass. We follow the same inductive argument beginning with the fact that the origin cell pulls a symbol from 0-block at most once per step. Then using an analogous argument we show that before any triplet of consecutive blocks (starting on minimal position k) empties completely, a corresponding rule pulling new data from higher blocks must trigger. This again contradicts the emptying of the triplet unless the higher block is also empty. In both cases we get a contradiction with the choices made of k and/or the timeslice. □

4 Circuit Compression

Every boolean function can be expressed as a formula in CNF, which is equivalent to a circuit of unbounded fan-in, constant depth and size $2^n + 1$ where n is the number of inputs.

Following the same concept, let C be a layered bounded fan-in circuit of depth d and size s. Let us denote $L_1, L_2, \ldots, L_{\lfloor d/k \rfloor}$ every k-th layer of C. Every gate in L_i depends on some $g(k)$ gates in the layer L_{i-1}, where $g(k)$ can be in general bounded from above by $2^{\mathcal{O}(k)}$. If we understand each gate in L_i as a boolean function of these $g(k)$ gates in L_{i-1}, we can equivalently compute its value by an unbounded circuit of $\mathcal{O}(1)$ depth and size $\mathcal{O}(2^{g(k)})$, independently of the other gates in L_i.

Applying the idea above, we may replace the circuit C by an equivalent circuit C' in which only the values of gates in layers L_i are computed as described above, and the output is similarly computed from the layer $L_{\lfloor d/k \rfloor}$. The circuit C' is equivalent to C, has unbounded fan-in, depth $\mathcal{O}(d/k)$ and size $\mathcal{O}(s \cdot 2^{g(k)})$.

If the initial circuit C has bounded fan-in, polynomial size and satisfies the linear dependency property, we can bound $g(k)$ by $\mathcal{O}(k)$, and by setting

$k = \mathcal{O}(\log n)$ where n is the number of inputs, we obtain an equivalent circuit of depth $\mathcal{O}(d/\log n)$ and polynomial size.

Theorem (Restating of Theorem 1.2)**.** Any multitape Turing machine running in time T can be simulated by an unbounded circuit of polynomial size and depth $\mathcal{O}(T/\log T)$ satisfying linear-dependency property.

Proof. For a singletape Turing machine, it suffices to use the circuit constructed directly from the Cook-Levin tableau. For multitape case, one of the oblivious constructions needs to be used to obtain a bounded circuit satisfying the linear dependency property. The result is then obtained by the compression described above, setting $k = \mathcal{O}(\log T)$. □

Note that in the above result, the unboundedness is essential. The resulting circuit has unbounded ANDs of fan-in $\mathcal{O}(\log n)$ and unbounded ORs of fan-in $2^{\mathcal{O}(\log n)}$. Replacing these with binary trees of bounded ANDs and ORs would blow up the depth of the circuit by a factor of $\mathcal{O}(\log n)$ negating the depth compression. In essence, the construction can be viewed as a trade-off between depth and (un)boundedness.

5 Singletape Machines

In this section we prove that a singletape Turing machine with bounded number of reversals can be simulated by an unbounded polynomial circuit of depth $\mathcal{O}(T/\log^{3/2} T)$ (Theorem 1.5). To achieve a low-depth circuit, we split the tape into blocks of fixed suitable size (polylog T) and design a small circuits to simulate the blocks in parallel.

5.1 Single-Layer Simulation

For purposes of the following construction, we define a *one-layer block simulation*, or just block simulation for short. A one-layer block simulation represents changes of data within a block happening during one visit of the head of the machine.

To define the simulation more formally, we need to distinguish between input and parameters of the simulation. When constructing a circuit, input is given via input gates, while parameters are hard-wired into the circuit and can be used to design the circuit in a specific way.

The input of a block simulation is the content of the block, and the input parameter is an *entry configuration* of head – entry state of the head

and the direction (left or right) from which the head enters the block. We also allow the head to "not enter" the block, or "enter inside" if the input tape contains specification of a position and state of the head. The block simulation then represents simulation of the Turing machine from the initial state given by input and parameters, until certain conditions are met – for the purposes of this construction, the simulation runs until the head exits the block. Output of the simulation is the final content of the block, exit configuration of the head (state with specification if the head exits to the left or to the right, or exists inside the block) and a validity bit. Validity bit indicates if the input and parameters are consistent (the input tape should specify head position if and only if the parameters specify that the head enters inside the block). Notice that there is no limit on how many steps there can be in one block simulation.

Observation 5.1. Given tape content, for block of size $b = \mathcal{O}(\log T)$, one-layer block simulations for all settings of parameters can be computed by a bounded circuit of polynomial size and depth $\mathcal{O}(\log T)$, or an unbounded circuit of polynomial size and constant depth.

Proof. For a block of b cells, a block simulation with fixed parameters can be viewed as a boolean function from b bits (the tape content) to $b + \mathcal{O}(1)$ bits (the new tape content, the exit configuration, validity bit). Each such function is expressible as a circuit of described properties. Note that there are constantly many possible parameter settings, so all possible block simulations may be computed in parallel increasing the size of the circuit by a constant factor. □

We define *glue* as the information describing what happens on the boundary of the block simulations. For a one-layered block simulation, the glue are the entry and exit head configurations.

Lemma 5.2. Given circuits computing block simulations for all parameters on two neighboring blocks A, B, we can construct a circuit computing all simulations on the joint block $C = A \cup B$ by increasing the depth by a constant (with bounded gates).

Proof. Suppose A is to the left of B. First we construct circuits computing block simulations of A and B for all sets of parameters. Then for every set of parameters for C we want to combine block simulations of A and B describing a block simulation on C. Suppose we want to compute simulation on C for fixed parameters.

We create a separate subcircuit for every possible glue of A and B, that is, for every possible entry/exit configurations of head that may be specified on the boundary of A and B. For example, a head may enter A from left in state 1, then exit to the right in state 2 (entering B in the same state) and then exit back to the left in state 3. There are only constantly many of these combinations. In each subcircuit we verify for each A and B that the relevant block simulation with given parameters is valid and its output exit configuration matches the glue. If successful, the subcircuit outputs the resulting tape contents of the relevant simulations and the used glue data. Note that for given entry configuration, the exit configuration is uniquely determined, and at most one subcircuit is successful.

Knowing the resulting block contents, it remains to determine the exit head configuration for C. If the head enters C from left, therefore entering into A, and then continues across B and exits C to the right, we use the exit configuration given by B. If the head never enters B, we use the exit configuration given by A. If the head turns and returns from B to A, we output the information that the head exits inside the block C, and modify the relevant cell on the output to reflect this. The remaining cases are handled analogously. □

We say two timeslices are *separated* by a certain event (reversal, number of steps, ...), if simulation of the Turing machine beginning from the first timeslice eventually yields the second timeslice, but no sooner than the event happens in the computation. For convenience, if the computation terminates before the event conditions are met, we allow the second timeslice to represent the final configuration of the Turing machine.

Lemma 5.3. Given a timeslice of a Turing machine, we can compute a timeslice separated by a reversal using a bounded circuit of polynomial size and depth $\mathcal{O}(\log T)$.

Proof. First we split the tape into blocks of size $b = \Theta(\log T)$ and compute all block simulations for all parameters, given the input tape data from the timeslice. Using the previous lemma, we iteratively combine pairs of neighboring blocks into bigger blocks, doubling size every time while increasing the depth of the circuit by only a constant. Eventually we combine all $\Theta(T/\log T)$ blocks increasing depth by $\mathcal{O}(\log T)$ and obtain block simulations of the whole tape. The block simulation with the initial head configuration starting inside the block now contains the output corresponding to the new timeslice.

It remains to observe that the simulation contains at least one reversal. Note that in the original block of size b the head always exits either to the left or to the right (unless computation terminates). We show by induction, that the head can only exit inside block after reversing. Suppose that the head enters A from the left or enters inside and exits A to the right. If we combine A with block B on the right, the head either leaves B to the right, or reverses and exits inside $A \cup B$. In general, A may be first combined with block on left, we get a super-block A' of A which is then combined with super-block B' of B. Similarly, either head leaves B' to the right, or reverses and exists to the left of B', or exits inside B', in which case it also reversed by induction hypothesis. Since in the final block covering the whole tape the head cannot exit to either side, at least one reversal has occurred (or the computation terminated). □

Lemma 5.4. Any singletape Turing machine running in time T can be simulated by a bounded circuit of polynomial size and depth

$$\mathcal{O}\big(\min\{T,\, R \log T\}\big),$$

where R is the maximum number of reversals.

Proof. We construct the circuit as follows. We take the initial machine configuration as the first timeslice and then apply the previous lemma in sufficiently many rounds to obtain a series of timeslices (separated by reversals) so that we are guaranteed that the last one corresponds to the final state of the computation.

In every round, beginning with a timeslice, we split the tape into blocks of size $\log T$ and follow the previous lemma. Since in the new timeslice the head is always positioned on one of the boundaries of the blocks (unless computation terminated), we use a simple trick to improve the construction. In the next round of simulation we shift the blocks by half of their size. Therefore, in the next simulation the head of the Turing machine starts in the middle of a block, and needs to perform at least $b/2 = \Theta(\log T)$ steps before reaching the first boundary. Any reversals happening prior to that are absorbed within a single block simulation, and so this part of the computation is always simulated. In this way we can guarantee that each round simultaneously simulates at least $\Omega(\log T)$ steps and at least one reversal. Thus after $\Theta(\min\{T/\log T,\, R\})$ rounds, the final timeslice represents the end of the calculations. To justify the depth analysis, note that both the block simulations and their gluing can be achieved in bounded

depth $\mathcal{O}(\log T)$. The remaining operations introduce only constant increase of depth per round. □

5.2 Multi-Layer Simulation

Building on the previous construction, we define a *multi-layer block simulation*. For simplicity, we also refer to this as to just a block simulation. A multi-layer block simulation represents changes of data withing a block happening during multiple visits of the head of the machine.

The input of an h-layer block simulation with time-limit t is the initial content of the block. The input parameters are an entry configuration of head, defined as before, two *border configuration* vectors of h additional configurations and a bit indication whether head terminates in this block. Each of border configuration vectors corresponds to one of the block boundaries (left and right), and each entry represents a head state and information whether head is entering or leaving the block. In addition, any suffix of the vector can contain blank entries.

The h-layer block simulation essentially represents a series of one-layer simulations. First the head enters the block as described in the entry configuration and the simulation runs until the head exits the block again (through the left or right boundary). This exit is recorded in the corresponding vector as the first element. Then the head enters from the same side in the state recorded on the same vector as the next element and continues its path until exiting the block again on either side. Note that although elements of every vector are in order, we do not know the order in which the two vectors interleave. If the head does not enter after any exit, then the rest of the vectors are blank. If the head enters the block, but the exit information does not fit into the corresponding vector, then the simulation is terminated, the position and state of the head is recorded on the tape. Furthermore, whenever the simulation performs t steps, it also terminates with the head terminating inside the block, recorded on the tape. Output of the simulation is the final content of the tape, the number of steps performed within the simulation, and a validity bit, defined as before (given the tape content, entry configuration of the head uniquely determines its behavior in the block, and the following exit configuration and its location must be consistent). Note that the parameters of the simulation are of size $\mathcal{O}(h) + \mathcal{O}(\log t)$ bits.

For multi-layered simulations, the *glue* consists of all information in the parameters.

We define the *horizon* as a big portion of tape (typically containing a given number of neighboring blocks) to which a significant portion of the computation of the machine is constrained. The motivation behind horizon is that we simulate only small parts of tape of the machine instead of the whole space to limit degrees of gates by allowing usage of small glues.

Lemma 5.5. Let Turing machine head begin in a given tape cell, and perform S steps. For any size of block b there exists an offset t such that if we split the tape into blocks of size b, there are at most S/b instances of the head crossing between the blocks.

Proof. Consider all b possible offsets $0, 1, \ldots, b-1$. Each step of the head contributes at most one crossing to one of the possible offsets. By Dirichlet's principle one of the offsets must have at most S/b crossings. □

Observation 5.6. Given tape content, for a block of size $b = \mathcal{O}(\log T)$, an $\mathcal{O}(\log T)$-layer block simulations for all settings of parameters can be computed by a bounded circuit of polynomial size and depth $\mathcal{O}(\log T)$, or an unbounded circuit of polynomial size and constant depth.

Proof. Analogously to the one-layer case, we create separate circuits for all fixed settings of parameters and view each simulation as a boolean function. This time we have polynomially many subcircuits of polynomial size and promised depth. □

Lemma 5.7. Given a representation of a timeslice limited to one horizon of width $a = \Theta(\log^{3/2} T)$ with the head starting in the middle half, we can compute a portion of new timeslice, limited to the same horizon, separated from the previous timeslice by at least $\Omega(\sqrt{\log T})$ reversals or $\Omega(\log^{3/2} T)$ steps. We achieve this using a bounded circuit of polynomial size and depth $\mathcal{O}(\log T)$, or an unbounded circuit of polynomial size and constant depth.

Proof. We proceed similarly to the previous construction based on one-layer blocks. We fix the size of blocks to be $b = \Theta(\log T)$, and use block simulations with $h = \Theta(\sqrt{\log T})$ layers and time limit $t = \log^2 T$. First we create separate circuits for each of the b possible offsets of splitting horizon into individual blocks. In each offset we obtain $\Theta(\sqrt{\log T})$ blocks of size b. For each block we enumerate and compute all of the block simulations with h layers.

Unlike before, we do not combine the blocks sequentially, but rather guess all of the glue at once, branching the circuit based on all possible glues.

For $a/b = \Theta(\sqrt{\log T})$ block simulations with $h = \mathcal{O}(\sqrt{\log T})$ layers, the overall glue is expressed by $\mathcal{O}(\log T)$ bits. For every choice of glue we verify that the individual block simulations agree with the glue. For at most one unique choice of glue the verification succeeds. Combining the outputs of the successfully glued block simulations, we obtain a simulation of the horizon (for a given offset). Furthermore, each block simulation outputs $\mathcal{O}(\log t) = \mathcal{O}(\log \log T)$ bits describing the number of steps simulated. Together we have $\mathcal{O}(\sqrt{\log T} \log \log T)$ bits, and we can sum these numbers into a single $\mathcal{O}(\log \log T)$-bit number, again viewing the problem as a general boolean function with low number of input bits. We then compare the number with a fixed constant of size $\Theta(\log^{3/2} T)$.

Once we obtain the outputs for each offset within the area and verify if at least $\Omega(\log^{3/2} T)$ steps were performed, we choose any offset with positive verification and use its output. Since the head starts roughly in the middle on the horizon, it must perform at least $a = \Theta(\log^{3/2} T)$ steps to reach the boundary of the horizon. Suppose this does not happen. By Lemma 5.5, at least one setting of the offset simulates at least $\Theta(\log^{3/2} T)$ steps without any of the border vectors filling completely. The only remaining terminating condition is the time-limit, in which case $\mathcal{O}(\log^2 T)$ steps must be simulated within a single block. In all cases we are guaranteed that the desired number of steps was simulated by at least one of the subcircuits.

The construction described above can be build using either a bounded circuit of depth $\mathcal{O}(\log T)$, or an unbounded circuit of constant depth (both of polynomial size). □

Now we are ready to prove the following theorem, from which follows Theorem 1.5.

Theorem 5.8. Any singletape Turing machine running in time T can be simulated by an unbounded circuit of polynomial size and depth

$$\mathcal{O}(T/\log^{3/2} T + R/\sqrt{\log T}),$$

where R is the maximum number of reversals, and by a bounded circuit of polynomial size and depth $\mathcal{O}(T/\sqrt{\log T} + R\sqrt{\log T})$.

Proof. Again we proceed similarly as in the previous construction. We compute a series of timeslices (separated by sufficient number of steps and/or reversals) to simulate the Turing machine.

Suppose we are given a timeslice. First we define the width of the horizon to be $a = \Theta(\log^{3/2} T)$. We tile the tape into horizons in two (or generally

constantly many) ways, so that one set of horizons start at the cell 0, and the other set of horizons are shifted by $a/2$ cells. By this we guarantee that no matter where the head is, it will always be in the middle half of one horizon while the number of horizons covering any portion of the tape is constant. To achieve this on the borders of the tape, we allow the borders to be covered by horizons reaching beyond the tape. For the unbounded case, we can create a set of horizons for every possible offset.

We simulate every horizon following the previous lemma, so that, if the head is indeed in the middle half in the beginning, we simulate either at least $\Theta(\log^{3/2} T)$ steps or $\Theta(\sqrt{\log T})$ reversals. To compose the new timeslice, each tape cell receives its data from all the horizons covering it (constantly many). To choose the correct data, the horizons without the head starting in the correct position are ignored (we may define horizons to output a bit indicating this). If no horizon satisfies this condition, the cell simply copies data from the previous timeslice. Note that (in the setting with two tiling sets of horizons), at most one horizon satisfies this condition, and so the next timeslice differs from the previous by exactly the data computed by the corresponding horizon simulation.

Based on the guarantees of the simulations, to get a timeslice corresponding to the final configuration we need to compute the following number of timeslices:

$$\Omega\left(T/\log^{3/2} T + R/\sqrt{\log T}\right).$$

□

6 Multitape Machines

In this section we prove Theorem 1.6, i.e., each Turing machine of bounded number of writes on all tapes but one can be simulated by an unbounded polynomial circuit of depth $\tilde{\mathcal{O}}(T/\log^{3/2} T)$. The idea extends the ideas from the previous section to work for multitape Turing machines. For better understanding we first prove the result for Turing machines where is only one read-write tape and other tapes are read-only. After that, we will prove the full result.

6.1 Read-Only Case

As a first step toward our goal we consider a model of a Turing machine where all except one tape are read-only, and only one main tape is read-

write. We refer to the tapes as the *main* tape and the *read-only* tapes.

High-level idea: We simulate only a part of the tape – a *horizon*. We guess enough information about the computation to split the simulation into smaller *blocks* on the tape which can be simulated in parallel. This includes good enough knowledge of the movements of all heads, which allows us to fetch the necessary read-only data to feed into the individual simulations without using all of the data (which would mean too many inputs). Since we want to simulate more than $\mathcal{O}(\log T)$ steps in one simulation, the overall number of read bits may be too high. We further split the simulation of each block time-wise into *chunks*, each of which reading at most $\mathcal{O}(\log T)$ bits, and then simulate these in parallel as well.

We will require a few parameters:

1. $A = \log^{3/2} T / \sqrt{\log \log T}$ – size of horizon in number of tape cells (also limit of number of steps of horizon simulation).
2. $B = \sqrt{\log T \log \log T}$ – size of a block in number of cells (also limit of number of steps in one period).
3. $C = \log T$ – limit of number of steps of one chunk.

In this construction, a block simulation is defined similarly to the previous constructions, with more complicated conditions. We set a time-limit of the block simulation to A steps. We define *period* of the horizon simulation as follows. Every time a head crosses a boundary between blocks, a period ends (and a new one begins). Furthermore, once B steps are performed since the beginning of the current period, the period also ends. Note that the boundaries of periods are defined in respect to the whole horizon, however as a slight abuse of notation, periods can be viewed both as a time-wise portions of the horizon simulation, as well as time-wise portions of the block simulations. When we consider periods as parts of blocks, we say a period is active if the head is inside the given block during this period.

Lemma 6.1. For a given horizon with head starting in the middle, there exists an offset of blocks such that the first A steps are simulated within at most $\mathcal{O}(A/B)$ periods in every block simulation.

Proof. By Lemma 5.5, there exists an offset such that the first A steps cross between the blocks at most $O(A/B)$ times. The other way how period may end is by reaching the limit of B steps, which may happen at most $\mathcal{O}(A/B)$ times. □

We define a *computation guide* as an extended idea of glue, gluing together a horizon simulation from smaller individual pieces. For every transition between periods within a horizon simulation, the computation guide specifies the state of the head. For every period, and every tape, the computation guide specifies the relative movement of the head during the period (the total number of steps to the right minus the total number of steps to the left) and for the main tape the overall number of steps.

Note that if we know the initial head position, the information from the computation guide allows us to determine in which block the main head is located in each of the periods, and the exact locations of all of the heads at the moment of transition between periods. We do not compute these by a circuit, but rather as before will hard-wire these parameters into the simulations and therefore we can assume all of these to be pre-computed by the construction of the circuit.

We define *chunks* of computation to be (non-overlapping) groups of C/B consecutive periods time-wise covering a block simulation. Note that in each chunk, at most $C = \mathcal{O}(\log T)$ steps are performed.

Observation 6.2. There exists a circuit validating chunk of a block simulation given the following:

- Fixed parameters: positions of each head and state of the machine for every boundary of all active periods and the initial and the final content of the main tape
- Input: content of C cells around heads on each read-only tape

Proof. In every active period we know the initial positions of head and the state of the machine. Assuming we successfully simulated all of the previous active periods, we also know the current content of the block (on the main tape). Simulating step by step, we know the positions of all heads in every step, and so we know all of the bits read at those steps from all tapes. Therefore we have all the information to perform the simulation step by step. □

Lemma 6.3. Given a timeslice representation as an input and computation guide and the initial and final block contents as fixed parameters, we can validate chunk using a circuit of unbounded polynomial size and constant depth.

Proof. According to Observation 6.2, all of the necessary information is available to us. We only need the B bits from the input representing the

given block, and $\mathcal{O}(C)$ bits from the read-only tapes that may be read during the given chunk. In total we have $\mathcal{O}(\log T)$ input bits, and therefore, viewing the computation as a boolean function, the simulation can be computed using a circuit of the promised properties. □

Lemma 6.4. Given a timeslice representation as an input and computation guide as fixed parameter, we can validate block simulation and compute the final block content using a circuit of unbounded polynomial size and constant depth.

Proof. Any block contains at most $\mathcal{O}(A/B)$ periods split into at most

$$\mathcal{O}(A/C)$$

chunks. We simulate all chunks in parallel by guessing all of the remaining information needed for their validation. We branch the circuit, each branch corresponding to a guess of content of a block in the beginning of the block, at the end of the block, and between every two chunks. In total we guess $\mathcal{O}(A/C \cdot B) = \mathcal{O}(\log T)$ bits and so we obtain a polynomial number of subcircuits. Each subcircuit now validates in parallel all chunks, following Lemma 6.3, and compares the actual initial content of block with the given content of block.

If exactly one of the possibilities validates successfully, it gives as an output the guessed final content of the block. The block simulation circuit simply forwards this content to its output. It remains to see that supposing the computation guide is valid, exactly one subcircuit succeeds. Given a timeslice, the following computation is uniquely defined. Let us fix the decomposition of the block simulation into periods (this is determined uniquely by the position of the horizon and the offset of the block within horizon). It is easy to see that given all the information, the simulation behaves in a unique way and at most one choice of parameters agrees with the definition of the Turing machine and the enforced conditions. Furthermore, such a choice always exists. □

Lemma 6.5. Given a representation of a timeslice limited to one horizon of size A with the head starting in the middle cell and fixed positions of all heads, we can compute a portion of new timeslice, limited to the same horizon, separated from the previous timeslice by A steps. We achieve this using an unbounded circuit of polynomial size and constant depth.

Proof. Similarly to the previous constructions, we first branch based on a choice of a suitable block offset. For every offset we branch based on a choice of computation guide. Given a computation guide, we can validate all blocks in parallel and by concatenation of the obtained data compose the final content of the horizon. We only need to wire the relevant data from the read-only tapes to input gates of the individual circuits. Since we assume that the initial position of each head is a parameter, we can hard-wire these.

Each branching is at most polynomial, and every subcircuit is of unbounded polynomial size and constant depth. As before, for each offset we also validate the number of steps simulated. For each offset the number of steps of all blocks are described by a total of

$$B \cdot \log A = \mathcal{O}(\sqrt{\log T}(\log \log T)^{3/2})$$

bits, which can be summed into an $\mathcal{O}(\log A)$-bit number when viewed as a boolean construction. Furthermore, the number of steps for all offsets are described by $B \cdot \mathcal{O}(\log A) = \mathcal{O}(\sqrt{\log T}(\log \log T)^{3/2})$-bits in total, and once again we can pick the best possible offset viewing the choice as a general boolean function. The output of the chosen offset is then forwarded to the output of the horizon simulation circuit. □

Lemma 6.6. Every multitape Turing machine with one main tape and other read-only tapes can be simulated by an unbounded polynomial circuit of depth $\mathcal{O}(T \frac{\sqrt{\log \log T}}{\log^{3/2} T})$.

Proof. As in the previous constructions we construct a series of circuits generating timeslices of the Turing machines. For every timeslice, we branch into individual simulations parameterized by the positions of all heads, this gives us at most $\mathcal{O}(T^k)$ different machine simulations. In each simulation we establish a horizon around the known position of the head and use Lemma 6.5 to perform the horizon simulation. We compose the new timeslice from the output of the horizon simulation and copy of the rest of the tape from the previous timeslice. Since every horizon simulation (except for the last one) is guaranteed to perform $\Theta(A)$ steps, we need at most $O(T/A)$ iterations, each of unbounded constant depth and polynomial size. □

6.2 Read-Write Case

Having constructed the previous machinery for a simple case, next we try to relax the conditions on the non-main tapes. We define parameter W as

the number of times the head writes to a non-main tape, that is the number of steps where the symbols read and written by the machine, according to the transition function, differ on any of the non-main tapes.

Observation 6.7. Every multitape Turing machine with one main tape and other tapes performing at most W writes can be simulated by an unbounded polynomial circuit of depth $\mathcal{O}(W + T\sqrt{\log \log T}/\log^{3/2} T)$.

Proof. We use exactly the same construction as previously, except whenever the horizon simulation attempts to write into any of the read-only tapes, we end the simulation early. This is correct as between any two write events, the computation behaves in a read-only manner to the non-main tapes. As each horizon simulation then produces a (the relevant portion of) timeslice separated from the previous one by either a write event or at least A steps, the result follows.

To make the construction technically sound, we need to address several issues. We extend the definition of the computation guide to allow it to specify less periods (with the rest of the elements being blank), the number of steps of the last period before it ends interrupted by a write event, and the description of the write event (a vector of symbols written to each respective cell, with possibly blank entries). We naturally adjust the block simulations and chunk verification to allow performing the blank periods (copying the input data onto the output). Finally, we use time-limited period simulations wherever in the circuit the period is parameterized to end after a fixed number of steps. We define a computation guide to be invalid if a write event happens at any other point during the computation other than the last specified step, or if the write is specified to occur but does not.

Note that given the initial timeslice, the valid computation guide is still unique and always exists. To reconstruct the new timeslice, we obtain the data from the horizon simulation as previously, and apply the writes specified by the valid computation guide. The result follows by iteratively constructing the new timeslices until the obtained timeslice is guaranteed to be the final configuration. □

In general, we can bound $W \leq T$, where the equality is easily obtained by some machines. The previous result then guarantees only depth $\mathcal{O}(T)$ due to the high cost of each write. As a next step we bring down this cost by allowing the horizon simulations to work with a low number of writes. Theorem 1.6 is a corollary of the following theorem.

Theorem 6.8. Every multitape Turing machine with one main tape and other tapes performing at most W writes can be simulated by an unbounded polynomial circuit of depth

$$\mathcal{O}\left(W\frac{\log\log T}{\log T}+T\frac{\sqrt{\log\log T}}{\log^{3/2}T}\right).$$

Proof. Once again we extend the computation guide to contain also information about $\log T/\log B$ writes. To compress the description, we describe the writes in a sequence. We encode every write in respect to a given period, specifying a symbols written (to all non-main tapes), time step of the event and position relative to the initial position of the head within the given period. In this way every write event is described by $\mathcal{O}(\log B)$ bits in total. We encode the sequence to implicitly associate the elements with the periods correctly. We order the write events in their chronological order. Between every two events we put a single 0 bit, indicating that the following bits encode a write event. The total number of 0s added does not increase the size of the string asymptotically. Furthermore, every time a period ends in the chronological order, we insert a 1 bit. The number of these 1s determines uniquely the period for each write event. The number of these 1 bits in the sequence is bounded by the maximum number of periods $\mathcal{O}(A/B)=\mathcal{O}(\log T/\log\log T)$. In total the bitstring describing the write events is of length $\mathcal{O}(\log T)$.

We extend the definition of validity of the computation guide. A computation guide is valid if the simulation reaches A steps, or the simulation ends one step before a write event and the computation guide cannot contain more write events. Note that under this definition, the valid computation guide always exists and is unique. Both of these conditions can be tested. Since each horizon simulation outputs the number of steps performed as a binary string, which can be tested by an appropriate boolean function. Similarly we may try to simulate the step following the last step in the horizon simulation to test whether it indeed performs a write or not.

The information describing write events is hard-wired as a parameter into the horizon simulation. Within each period simulation it is possible to deduce the exact step and position in which each write event happens as well and to account for all of the previous writes affecting the input data from non-main tapes. Finally, we may use a separate piece of circuit to construct the new contents of the non-main tapes. Since the writes are hard-coded, we can simply copy the original content and replace exactly the changed

cells with their newest values trivially. This way each horizon simulation can finish after either simulating $\Theta(A)$ steps, or after at least $\log T / \log B$ write events in the non-main tapes and outputs the new contents of the horizon on the non-main tapes. The result is obtained analogously to the previous constructions. □

Note that the previous construction can be slightly improved. In essence, there is no inherent distinction between the main and non-main tapes, other than the fact that in the simulation the non-main tapes have a limited number of writes. We can exploit this and further branch the horizon simulation circuits, to try in parallel all choices of main tape and then use the simulation which performs the most steps before the simulation limits are met. Since the number of tapes is constant, this comes at a cost of constant overheads in both depth and size. However in general this does not guarantee any reduction on the number of writes that need to be encoded, as it is possible for the write events to always happen on all tapes simultaneously.

A write event is said to be *authorized* if it is recorded in a computation guide and *unauthorized* otherwise. In our next construction we adjust the construction so that there may be unauthorized write events happening within the horizon simulation. We will make sure that every unauthorized write event correctly affects the new content of the non-main tapes. However the unauthorized write events will not affect any of the simulations running in parallel with the simulation which causes it.

A read event on a given tape is said to be *effective* if the head reads a symbol and the output of the transition function depends on this symbol. A read event is ineffective otherwise.

A write event e is said to be *effective*, in respect to a computation guide, if there are two distinct chunk simulations w and r running in parallel, such that e happens within w and r performs an effective read event on the same cell in a step following e in a sequential order of the simulated computation and furthermore no other write event changes the content of the cell between e and w. The write event is *ineffective* otherwise.

There are several examples of ineffective write events. If a cell is written to, but it is not read again withing the same horizon simulation (or for at least A steps), then it is clearly ineffective. Similarly, if a cell is read before the horizon simulation ends, but only within the same chunk, it is also ineffective. The important nuisance of the definition is that the (in)effectiveness of a write event depends on the time-wise position of the horizon, the space-wise position of the horizon, offset of its blocks and gen-

erally any movement of the head (dependent on the data on the tapes). Because of this, the following theorem is only implicitly defined.

Theorem 6.9. Every multitape Turing machine with one main tape can be simulated by an unbounded polynomial circuit of depth

$$\mathcal{O}\left(F\frac{\log\log T}{\log T}+T\frac{\sqrt{\log\log T}}{\log^{3/2} T}\right),$$

where F is the number of effective write events in the non-main tapes.

Proof. Within each horizon simulation there are effective and ineffective write events. The effective write events must be authorized to maintain consistency of the parallel simulations, while the ineffective write events should always be kept unauthorized to increase efficiency of the simulations.

As the first step, we deal with the unauthorized writes. We extend each active period simulation to also output the new contents of the parts of the non-main tapes it was presented on input (data vectors). Also for every tape, the period simulation outputs a triplet of binary mask vectors, indicating which cells were written to, which cells were involved in an effective read event and which cell were involved in unauthorized write event. Note that since we understand the chunk simulation as a single unit, the periods following within the same chunk easily take into account previously performed write events. All of the previously mentioned vectors are forwarded to the output of chunk simulation.

On the level of horizon simulation, once all chunk simulations provide their outputs, we use the head position information from the computation guide to design a circuit which for every cell within the horizon checks receives all of the relevant bits from all relevant masks and data vectors. Since every cell receives at most constant amount of bits per (active) period simulation and there are at most $O(A/B) = \mathcal{O}(\log T)$ periods in total, we can now process all the data of each cell by an arbitrary function and represent it as a constant depth boolean function. The function we need simply considers all of the information in chronological order (in respect to the order of active periods within the computation being simulated) to determine the final content of the cell (based on the last write performed), tests for unauthorized effective write events and checks that every authorized write event was effective.

In this way the horizon simulation outputs the new contents of all tapes, correct in respect to the unauthorized ineffective write events, and can val-

idate the computation guide in respect to the following conditions. Additionally to the existing conditions, we say that a computation guide is invalid if unauthorized effective write event or authorized ineffective write event occurs during the simulation. This is easily tested by the previous construction. Furthermore, if the simulation is prescribed by the computation guide to terminate after a specific number of steps smaller than A, the guide is invalid if performing one more step would not change unauthorized ineffective write event into an effective one or if the computation guide could contain all of the additional necessary authorizations within the limit of the number of authorizations. Note that since the computation guide specifies the final positions of all heads, we can speculatively simulate the next step (given the new contents of all tapes), check that effective read events occur on some of the k relevant cells and that the corresponding cells would indeed experience new unauthorized effective write events. Note that the number of potential new effective write events is at most one per cell, and so we can count all instances and compare to the number of blank spaces taking space of potential additional authorizations in the computation guide using a constant size circuit.

We claim that the computation guide is still unique. If the simulation can successfully reach A steps, then the corresponding computation guide is clearly unique. If the simulation terminates earlier (exceeding the limit on authorizations), then the point of termination is uniquely defined by the above conditions, and so is the relevant computations guide. Furthermore, in this case the computation guide is missing at most $k - 1$ authorizations from the allowed maximum. Only the output of the unique valid computation guide is considered for every choice of block offset and the choice of the main tape. Out of these possibilities, the best is chosen based on the highest number of performed steps, analogously to the previous constructions. As before, we can see that using the horizon simulation described above, we may construct a new timeslice separated from the previous one by either A steps or at least $\frac{\log T}{\log \log T}$ effective write events. The result is obtained as before. □

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

[2] Allan Borodin. On relating time and space to size and depth. *SIAM JOURNAL ON COMPUTING*, pages 733–744, 1977.

[3] Patrick W. Dymond and Martin Tompa. Speedups of deterministic machines by synchronous parallel machines. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 336–343, New York, NY, USA, 1983. Association for Computing Machinery.

[4] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977.

[5] Wolfgang Maass and Amir Schorr. Speed-up of turing machines with one work tape and a two-way input tape. *SIAM J. Comput.*, 16(1):195–202, February 1987.

[6] Michael S. Paterson. Tape bounds for time-bounded turing machines. *J. Comput. Syst. Sci.*, 6(2):116–124, April 1972.

[7] Uwe Schöning and Randall Pruim. *The Pebble Game*, pages 203–212. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[8] Ryan Williams. Parallelizing time with polynomial circuits. *Theor. Comp. Sys.*, 48(1):150–169, January 2011.

Slowdown for the Geodesic-Biased Random Walk

Mikhail Beliayeu, Petr Chmel, Bhargav Narayanan, Jan Petr

Abstract

Given a connected graph G with some subset of its vertices excited and a fixed target vertex, in the *geodesic-biased random walk* on G, a random walker moves as follows: from an unexcited vertex, she moves to a uniformly random neighbour, whereas from an excited vertex, she takes one step along some fixed shortest path towards the target vertex. We show, perhaps counterintuitively, that the geodesic-bias can slow the random walker down significantly: there exist connected, bounded-degree n-vertex graphs with excitations where the expected hitting time of a fixed target is at least $\exp(\sqrt[4]{n}/100)$.

1 Introduction

In this paper, we investigate a model of excited random walk on a connected graph, namely *geodesic-biased random walk*, where the excitations are designed to decrease the hitting time of a fixed target vertex. The model originates in the theoretical computer science and computational biology communities [8, 7, 5], and was brought to our attention by Sousi [17]. By way of context, let us mention that various matters relating to hitting times — recurrence and return times [4, 18, 2, 3], speed [15, 9, 16] and slowdown [13, 14] — have been investigated in a number of different models of excited random walk; for a broad overview, see [12, 10].

Geodesic-biased random walk is defined on a connected n-vertex graph G. Having fixed a starting vertex $a \in V(G)$, a target vertex $b \in V(G)$ and a subset $\mathcal{X} \subset V(G)$ of excited vertices, a random walker walks from a until she hits b as follows: from an unexcited vertex of G, she moves to a

uniformly random neighbour, whereas from an excited vertex, she takes one step along some predetermined shortest path to the target vertex b. Our focus here is the hitting time $\tau_a(b, \mathcal{X})$ i.e., the first time at which the walker hits b starting from a when the set of excited vertices is $\mathcal{X}$.

When every vertex is excited, i.e., $\mathcal{X} = V(G)$, the geodesic-biased walk reduces to a deterministic walk along a shortest path to the target vertex, in which case we have $\mathbb{E}[\tau_a(b, V(G))] = O(n)$. On the other hand, when no vertices are excited, i.e., $\mathcal{X} = \emptyset$, the geodesic-biased walk reduces to the simple random walk on G, and an old result of Lawler [11] gives a uniform polynomial bound (see also [1, 6]) for the expected hitting time of $\mathbb{E}[\tau_a(b, \emptyset)] = O(n^3)$. Many of the existing results in the literature [8, 7, 5] show that the expected hitting time of a fixed target in the geodesic-biased walk, for various graphs G and random choices of the set $\mathcal{X}$ of excited vertices, is significantly smaller than Lawler's uniform bound. Motivated by this, we shall investigate how much the geodesic-bias can decrease the hitting time of a fixed target.

While the geodesic-bias ostensibly aims to decrease hitting times, it is actually not hard to construct examples where the expected hitting time of a fixed target in the geodesic-biased walk is *slightly larger* than the expected hitting time in the analogous simple random walk. To wit, consider a graph where two vertices a and b are connected by two paths of lengths 2 and 3, with the middle vertex of the shorter path being attached to a 'trap', say a large clique; here, it is not hard to see that exciting a increases the expected hitting time of b, since the random walker ends up spending more time in the 'trap'. However, the digraph formed by taking a shortest path from each vertex to a fixed target is acyclic, so one cannot string together multiple such 'traps' in a cyclic fashion; in particular, such constructions cannot hope to slow the geodesic-biased walk down by more than a constant factor in comparison to the simple random walk.

In the light of the above discussion, it is natural to ask if the results in [8, 7, 5] are indicative of a broader phenomenon, and if there is a uniform polynomial bound for the expected hitting time of a fixed target in the geodesic-biased walk, much like Lawler's bound [11] for the simple random walk. Our first result shows, perhaps surprisingly, that this is not the case: even a single excitation can cause an exponential slowdown.

Theorem 1.1. For infinitely many $n \in \mathbb{N}$, there exists a connected graph

G on n vertices with $a, b \in V(G)$ such that

$$\mathbb{E}[\tau_a(b, \{a\})] = \Omega\left(\exp\left(\frac{\sqrt[4]{n}\log n}{100}\right)\right).$$

The construction proving Theorem 1.1 produces graphs of unbounded degree. In the context of the simple random walk, bounded-degree graphs are known to behave somewhat differently from those of unbounded degree; for example, as shown by Lawler [11], expected hitting times in a bounded-degree n-vertex graph are $O(n^2)$. Our second result, also in the spirit of Theorem 1.1, shows that exponential slowdown is unavoidable on graphs of bounded degree as well, though more excitations are required in this case.

Theorem 1.2. For infinitely many $n \in \mathbb{N}$, there exists a connected graph G on n vertices of maximum degree 3 with $a, b \in V(G)$ and a set $\mathcal{X} \subset V(G)$ of $O(\sqrt{n})$ excited vertices such that

$$\mathbb{E}[\tau_a(b, \mathcal{X})] = \Omega\left(\exp\left(\frac{\sqrt[4]{n}}{100}\right)\right).$$

This paper is organised as follows. We give the proofs of Theorems 1.1 and 1.2 in Section 2. We conclude with a discussion of some open problems in Section 3.

2 Proofs of the main results

In this section, we prove our two main results. It will be helpful to have some notation. As is usual, we write $[n]$ for the set $\{1, 2, \ldots, n\}$. In the geodesic-biased random walk on a graph G, when the target vertex b and set $\mathcal{X}$ of excited vertices are clear from the context, we abbreviate the expected hitting time $\tau_x(y, \mathcal{X})$ of y from x by $T(x, y)$.

We shall make use of a well-known Chernoff-type bound.

Proposition 2.1. Let $X = X_1 + X_2 + \cdots + X_n$, where $X_1, X_2, \ldots, X_n$ are independent Bernoulli random variables. Writing $\mu = \mathbb{E}[X]$, we have

$$\mathbb{P}(X \geq (1+\delta)\mu) \leq \exp\left(\frac{-\delta^2\mu}{2+\delta}\right)$$

for all $\delta > 0$. □

We also require the following well-known gambler's ruin estimate.

Proposition 2.2. The probability that the simple random walk on the interval $\{0, 1, \ldots, n\}$ started at 1 visits n before it visits 0 is $1/n$. □

We are now ready to give the proof of Theorem 1.1.

Proof of Theorem 1.1. We build an infinite family of graphs as follows. We fix $k \in \mathbb{N}$, set $m = \lfloor \sqrt{k} \rfloor$, and consider a graph G as follows: we start with a path of length $m+1$ between a and b, say $a, v_1, v_2 \ldots, v_m, b$, and then connect each v_i to a by k disjoint paths of length $i+1$ as shown in Figure 1. Formally, we take

$$V(G) = \{a, b\} \cup \{v_1, v_2, \ldots, v_m\} \cup \bigcup_{j=1}^{m} \bigcup_{i=1}^{j} R_{i,j},$$

where $R_{i,j} = \{r_{i,j,l} : l \in [k]\}$, and specify $E(G)$ as follows:

- $\forall i \in [m-1] : \{v_i, v_{i+1}\} \in E(G)$,
- $\forall j \in [m], \forall i \in [j-1], \forall l \in [k] : \{r_{i,j,l}, r_{i,j+1,l}\} \in E(G) \wedge \{r_{i,1,l}, a\} \in E(G) \wedge \{r_{i,k,l}, v_i\} \in E(G)$,
- $\{a, v_1\} \in E(G)$ and $\{v_m, b\} \in E(G)$.

We consider the geodesic-biased random walk on this graph with target b and $\mathcal{X} = \{a\}$. The unique shortest path to b from a is the path $a, v_1, v_2 \ldots, v_m, b$, so the random walker always moves to v_1 from a.

Lemma 2.3. For $1 \leq j \leq m+1$, we have $T(a, v_j) \geq \frac{k^{j-1}}{4^{j-1} \cdot (j-1)!}$.

Proof. We will prove this lemma by induction. For $j = 1$, we have $T(a, v_1) = 1$ and the bound clearly holds. Now, assume the lemma holds for j and note that $T(a, v_{j+1}) = T(a, v_j) + T(v_j, v_{j+1})$, as we can only reach v_{j+1} from v_j. We may then bound $T(v_j, v_{j+1})$ by

$$\begin{aligned} T(v_j, v_{j+1}) &= 1 + \frac{T(v_{j-1}, v_{j+1})}{k+2} + \frac{T(v_{j+1}, v_{j+1})}{k+2} + \frac{k \cdot T(R_{j,j}, v_{j+1})}{k+2} \\ &\geq \frac{k}{k+2} T(R_{j,j}, v_{j+1}) \end{aligned}$$

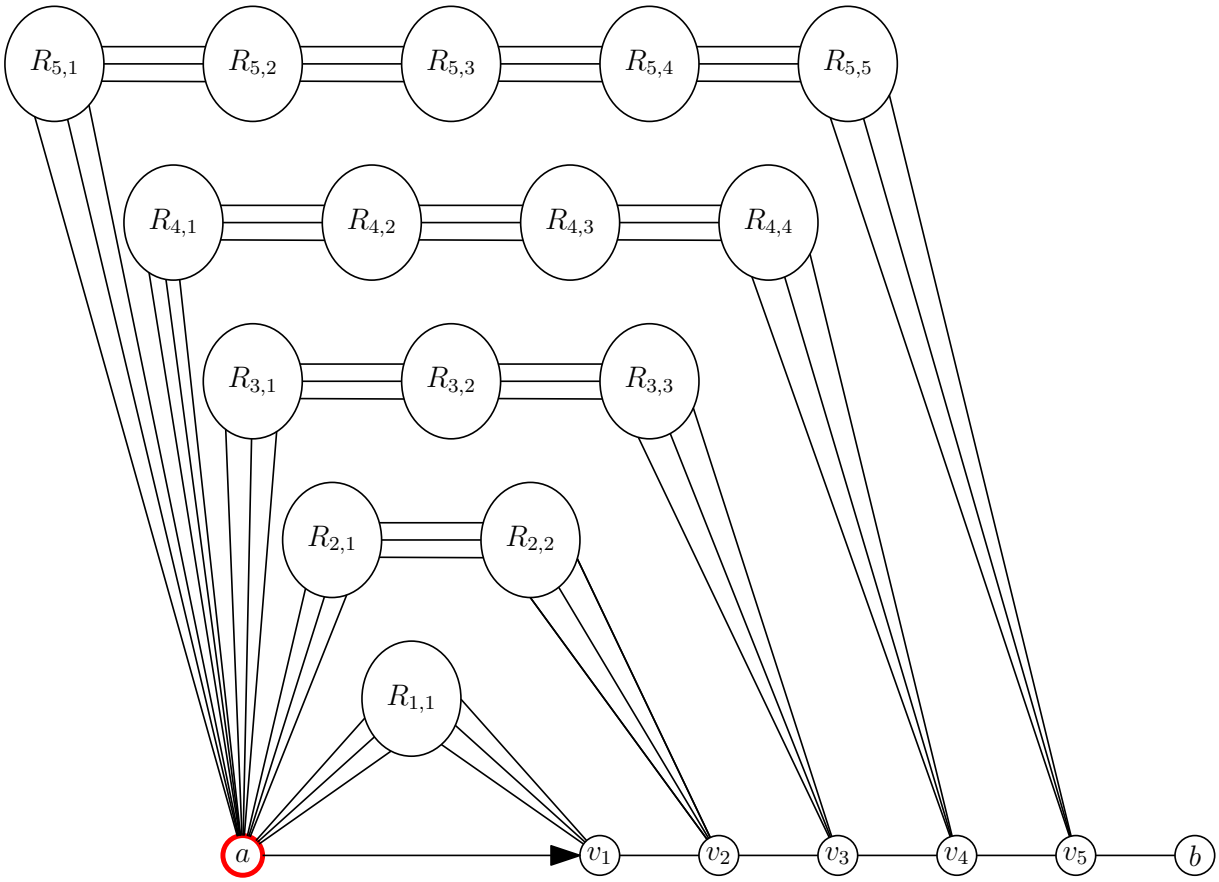


Figure 1: The construction with $m = 5$.

From Proposition 2.2, it follows that the probability of walking from $R_{j,j}$ to v_j before a is $j/(j+1)$, and the complementary event has the probability $1/(j+1)$. We then see that

$$T(R_{j,j}, v_{j+1}) \geq \frac{1}{j+1} T(a, v_{j+1}) + \frac{j}{j+1} T(v_j, v_{j+1}).$$

Using this bound, we obtain

$$\begin{aligned} T(v_j, v_{j+1}) &\geq \frac{k}{k+2}\frac{1}{j+1} T(a, v_{j+1}) + \frac{k}{k+2}\frac{j}{j+1} T(v_j, v_{j+1}) \\ \frac{k+2j+2}{(k+2)(j+1)} T(v_j, v_{j+1}) &\geq \frac{k}{(k+2)(j+1)} T(a, v_{j+1}) \\ T(v_j, v_{j+1}) &\geq \frac{k}{k+2j+2} T(a, v_{j+1}) \end{aligned}$$

Combining the above bound with the bound on $T(a, v_{j+1})$, we get

$$\begin{aligned} T(a, v_{j+1}) &\geq T(a, v_j) + \frac{k}{k+2j+2} T(a, v_{j+1}) \\ \frac{2j+2}{k+2j+2} T(a, v_{j+1}) &\geq T(a, v_j) \\ T(a, v_{j+1}) &\geq \frac{k+2j+2}{2j+2} T(a, v_j) \geq \frac{k}{4j} T(a, v_j) \end{aligned}$$

By the induction hypothesis, we now conclude that

$$T(a, v_{j+1}) \geq \frac{k}{4j} \frac{k^{j-1}}{4^{j-1} \cdot (j-1)!} = \frac{k^j}{4^j \cdot j!};$$

the result follows. □

From Lemma 2.3, we conclude that $T(a,b) \geq k^m/4^m m!$; since $m = \lfloor \sqrt{k} \rfloor$, standard bounds for the factorial show that

$$T(a,b) \geq \frac{1}{4} \left(\frac{\sqrt{k}}{4} \right)^{\sqrt{k}-1}$$

and since $n = |V(G)| = \Theta(m^2 k) = \Theta(k^2)$, we deduce that

$$T(a,b) = \Omega\left(\exp\left(\frac{\sqrt[4]{n} \log n}{100} \right) \right),$$

proving the result. □

Next, we present the (slightly more involved) proof of Theorem 1.2.

Proof of Theorem 1.2. To prove the result, we build an infinite family of graphs as follows. We fix $m \in \mathbb{N}$, and consider a graph G constructed as follows: as before, we start with a path of length $m+1$ between a and b, say $a, v_1, v_2 \ldots, v_m, b$, and then attach a path of length $2m+2$ to each v_i, and finally chain the ends of these paths to a by another path as shown in Figure 2. Formally, we set

$$V(G) = \{a, b\} \cup \{v_1, v_2, \ldots, v_m\} \cup \{s_1, s_2, \ldots, s_m\} \cup \bigcup_{j=1}^{2m+1} \bigcup_{i=1}^{m} \{r_{i,j}\}$$

and specify $E(G)$ as follows:

- $\forall i \in [m-1] : \{v_i, v_{i+1}\} \in E(G) \wedge \{s_i, s_{i+1}\} \in E(G)$,
- $\forall j \in [2m], \forall i \in [m] : \{r_{i,j}, r_{i,j+1}\} \in E(G) \wedge \{r_{i,1}, s_i\} \in E(G) \wedge \{r_{i,2m+1}, v_i\} \in E(G)$,
- $\{a, v_1\} \in E(G)$, $\{v_m, b\} \in E(G)$, and $\{a, s_1\} \in E(G)$.

We consider the geodesic-biased random walk on this graph with target b and $\mathcal{X} = \{a, s_1, s_2, \ldots, s_m\}$. Notice that our choice of path lengths ensures that the random walker moves deterministically from s_i to s_{i-1} (or to a in the case of s_1), and from a to v_1.

Lemma 2.4. We have $T(v_1, b) \geq \exp(\sqrt{m}/10)/(m^{3/2}+1)$.

Proof. We proceed via a renewal argument. Observe that $T(v_1, b) \geq 1 + q \cdot T(v_1, b)$, where q is the probability of the event that the random walker visits a before b after leaving v_1. It will be more convenient to work with the complementary event, namely, that the random walker visits b before a after leaving v_1; we write $p = 1 - q$ for the probability of this event. From the previous inequality, we then have $T(v_1, b) \geq 1/(1-q) = 1/p$.

Now, we shall estimate p, the probability that the geodesic-biased walk starting at v_1 hits b before a. To do so, we consider the Markov chain $(x_t)_{t \geq 0}$ induced by the geodesic-biased walk on the states $a, v_1, \ldots, v_m, b$ with a and b being absorbing; of course, p is exactly the probability that this induced chain started at v_1 reaches the absorbing state b before it hits the absorbing state a.

For each non-absorbing state v_i, there are three possibilities for the next state of the induced chain hit by the random-walker: v_{i-1}, v_{i+1} or a. The

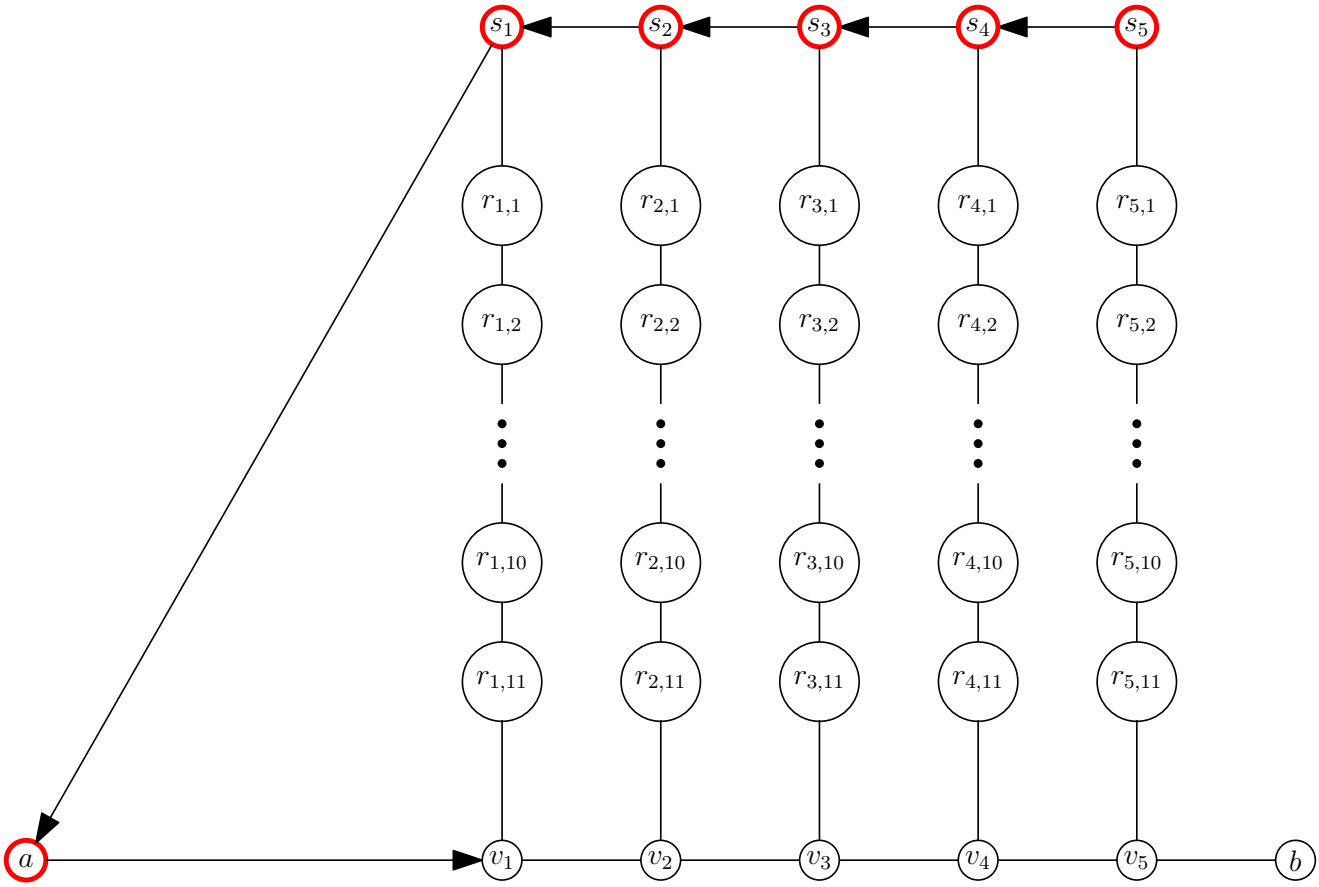


Figure 2: The bounded-degree construction with $m = 5$.

probabilities of these transitions are as follows: we write ε for the probability of returning to a via s_i, and note that the other two transitions have the same probability, i.e.,

$$\mathbb{P}[x_{t+1} = v_{i+1} \,|\, x_t = v_i] = \mathbb{P}[x_{t+1} = v_{i-1} \,|\, x_t = v_i] = \frac{1-\varepsilon}{2}.$$

We may calculate ε, the probability of *retracing*, i.e., returning to a via s_i, as follows. The probability of reaching s_i before v_i starting from $r_{i,2m+1}$ is, by Proposition 2.2, exactly $1/(2m+2)$. It then follows that $\varepsilon = \frac{1}{3}(\frac{2m+1}{2m+2}\varepsilon + \frac{1}{2m+2})$, from which we get $\varepsilon = 1/(4m+5)$.

We shall estimate $p = p_s + p_l$ by separately estimating p_s, the probability of the chain hitting b before a starting from v_1 in at most $m^{3/2}$ steps, and p_l, the probability of the chain hitting b before a starting from v_1 and taking more than $m^{3/2}$ steps to do so.

First, we dispose of 'long' excursions. We claim that $p_l \leq (1-\varepsilon)^{m^{3/2}}$; indeed, if the chain does not hit either of a or b in the first $m^{3/2}$ steps, then the chain does not, in particular, retrace on any of the first $m^{3/2}$ steps. Thus

$$p_l \leq (1-\varepsilon)^{m^{3/2}} \leq \left(1 - \frac{1}{4m+5}\right)^{m^{3/2}} \leq \exp\left(\frac{-\sqrt{m}}{10}\right).$$

Next, we focus on the 'short' excursions. Note that we may write $p_s = \sum_{t=0}^{m^{3/2}} p(t)$, where

$$p(t) = \mathbb{P}[\{x_t = b\} \wedge \{\forall\, 1 \leq i < t : (x_i \neq a \wedge x_i \neq b)\}].$$

We may then bound $p(t)$ by conditioning on the chain never retracing to get

$$p(t) \leq \mathbb{P}[\{x_t = b\} \wedge \{\forall\, 1 \leq i < t : (x_i \neq a \wedge x_i \neq b)\} \,|\, \text{No Retrace}].$$

This upper bound may be interpreted in terms of the simple random walk on the integers; indeed, conditional on never retracing, the chain is isomorphic to the simple random walk on the integer line. Concretely, consider the simple random walk $\{y_t\}_{t\geq=0}$ on the integers and note that

$$\begin{aligned}
&\mathbb{P}[\{x_t = b\} \wedge \{\forall\, 1 \leq i < t : (x_i \neq a \wedge x_i \neq b)\} \,|\, \text{No Retrace}] \\
&\quad = \mathbb{P}[\{y_0 = 1 \wedge y_t = m+1\} \wedge \{\forall 1 \leq i < t : (y_i \neq 0 \wedge y_i \neq m+1)\}] \\
&\quad \leq \mathbb{P}[\{y_0 = 1 \wedge y_t = m+1\}] \leq \mathbb{P}[\{y_0 = 1 \wedge y_t \geq m+1\}].
\end{aligned}$$

The last probability above is easy to estimate since the simple random walk on the integers may be viewed as a sum of independent Bernoulli random variables, so by applying Proposition 2.1 (with $\delta = m/t$) to such a representation of the random walk on the integers, we obtain

$$\mathbb{P}[\{y_0 = 1 \wedge y_t \geq m+1\}] \leq \exp\left(\frac{-m^2}{4t+2m}\right) \leq \exp\left(\frac{-\sqrt{m}}{10}\right),$$

where the second inequality holds for all $t \leq m^{3/2}$. Consequently, we have

$$p_s \leq m^{3/2} \exp\left(\frac{-\sqrt{m}}{10}\right).$$

Combining the above estimates for p_s and p_l and the fact that $T(v_1, b) \geq 1/(p_s + p_l)$ now yields the required bound. □

The theorem immediately follows from the above lemma. Indeed,

$$T(a, b) = 1 + T(v_1, b),$$

and writing the above bound for $T(v_1, b)$ in terms of

$$n = |V(G)| = 2 + m(2m+3)$$

proves the result. □

3 Conclusion

Our results raise a few different natural questions; we discuss two such problems below.

There remains the question of determining the right order of uniform bound for the expected hitting time of a fixed target in the geodesic-biased walk: we have shown that on a connected n-vertex graph, this may be as large as $\exp(n^{1/4} \log n/100)$, while it is more or less trivial to show a uniform upper bound of $\exp(n \log n)$; it would be interesting to close this gap and pin down the truth.

Another problem that we have been unable to resolve concerns bounded-degree graphs. While we have exhibited exponential slowdown for the geodesic-biased walk on bounded-degree graphs, our constructions nonetheless require an unbounded number of excitations, which leads to the following: in the geodesic-biased walk on a bounded-degree graph with a bounded number of excitations, is there a uniform polynomial bound on the expected hitting time of the fixed target?

References

[1] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science*, pages 218–223. IEEE, New York, 1979.

[2] G. Amir, I. Benjamini, and G. Kozma. Excited random walk against a wall. *Probab. Theory Related Fields*, 140:83–102, 2008.

[3] I. Benjamini, G. Kozma, and B. Schapira. A balanced excited random walk. *C. R. Math. Acad. Sci. Paris*, 349:459–462, 2011.

[4] I. Benjamini and D. B. Wilson. Excited random walk. *Electron. Comm. Probab.*, 8:86–92, 2003.

[5] L. Boczkowski, A. Korman, and Y. Rodeh. Searching a tree with permanently noisy advice. In *26th European Symposium on Algorithms*, volume 112 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 54, 13. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2018.

[6] G. Brightwell and P. Winkler. Maximum hitting time for random walks on graphs. *Random Structures Algorithms*, 1:263–276, 1990.

[7] E. Fonio, Y. Heyman, L. Boczkowski, A. Gelblum, A. Kosowski, A. Korman, and O. Feinerman. A locally-blazed ant trail achieves efficient collective navig- ation despite limited information. *eLife*, page 2016;5:e20185, 2016.

[8] N. Hanusse, D. Ilcinkas, A. Kosowski, and N. Nisse. Locating a target with an agent guided by unreliable local advice. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 355–364, New York, NY, USA, 2010. ACM.

[9] M. Holmes. On strict monotonicity of the speed for excited random walks in one dimension. *Electron. Commun. Probab.*, 20:no. 41, 7, 2015.

[10] E. Kosygina and M. P. W. Zerner. Excited random walks: results, methods, open problems. *Bull. Inst. Math. Acad. Sin. (N.S.)*, 8:105–157, 2013.

[11] G. F. Lawler. Expected hitting times for a random walk on a connected graph. *Discrete Math.*, 61:85–92, 1986.

[12] R. Pemantle. A survey of random processes with reinforcement. *Probab. Surv.*, 4:1–79, 2007.

[13] J. Peterson. Large deviations and slowdown asymptotics for one-dimensional excited random walks. *Electron. J. Probab.*, 17:no. 48, 24, 2012.

[14] J. Peterson. Extreme slowdowns for one-dimensional excited random walks. *Stochastic Process. Appl.*, 125:458–481, 2015.

[15] C. D. Pham. Monotonicity and regularity of the speed for excited random walks in higher dimensions. *Electron. J. Probab.*, 20:no. 72, 25, 2015.

[16] C. D. Pham. The infinite differentiability of the speed for excited random walks. *C. R. Math. Acad. Sci. Paris*, 354(11):1119–1123, 2016.

[17] P. Sousi. Personal communication, June 2015.

[18] M. P. W. Zerner. Multi-excited random walks on integers. *Probab. Theory Related Fields*, 133:98–122, 2005.

Note about the Project "The Minimum Circuit Size Problem"

Azucena Garvia-Bosshard, Amulya Musipatla
Mentor: Eric Allender

Our project focus was the Minimum Circuit Size Problem (MCSP), the problem of determining whether a Boolean function can be computed using a (Boolean) circuit of a certain size. We also studied a closely related problem, MKTP, which analyzes Kolmogorov complexity in place of circuit complexity. These problems have gained attention as promising candidates for NP-intermediate problems.

Over the summer, we improved the known hardness result for MKTP by Allender and Hirahara [1], that MKTP is hard for DET (computing determinant of a matrix) under non-uniform NC0 reductions, and showed that MKTP is in fact hard for DET under non-uniform projections. In other words, the existing reduction had each output bit relying on a constant number of input bits while in our reduction each output bit relies on at most one input bit. Clearly this type of reduction is much more restrictive which helps motivate interest in this result.

Proving this relationship required an intermediate step of showing that Graph Isomorphism (GI) is also hard for DET, and then reducing GI to MKTP. We showed the first step by modifying the method Torán [2] used to prove that Graph Isomorphism is hard for DET under log-space reductions, and showing that every step of this reduction could be simulated with a projection. This immediately followed for some steps of his reduction but required a more explicit construction for other parts. We also showed that the reduction from Graph Isomorphism to MKTP constructed by Allender and Hirahara is already a projection. Combined with the first step, this proved our result.

Our improvement in the known reduction gives us more intuition on the difficulty of MKTP and loosens requirements on separating complexity

classes. We believe this result should follow for MCSP.

References

[1] Eric Allender and Shuichi Hirahara. New insights on the (non-)hardness of circuit minimization and related problems. *ACM Trans. Comput. Theory*, 11(4), September 2019.

[2] Jacobo Torán. On the hardness of graph isomorphism. *SIAM J. Comput.*, 33(5):1093–1108, May 2004.