UNAMBIGUOUS LOGARITHMIC SPACE BOUNDED COMPUTATIONS

by

Raghunath Tewari

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Vinodchandran Variyam

Lincoln, Nebraska

May, 2011

UNAMBIGUOUS LOGARITHMIC SPACE BOUNDED COMPUTATIONS

Raghunath Tewari, Ph.D.

University of Nebraska, 2011

Adviser: Vinodchandran Variyam

Space complexity investigates the power and limitations of a computational model (e.g. a Turing machine) which has a limited amount of workspace to perform its computation. Particularly interesting is the case when the space is only logarithmic in the input size. Unambiguous computation is a natural restriction of nondeterministic computation, where there is a *unique* accepting path on a 'Yes' instance, and no accepting paths on a 'No' instance. In this dissertation we study the power of unambiguous log-space computations (denoted as UL) and whether it is general enough to contain all of nondeterministic log-space (denoted as NL). This leads us to the study of the *graph reachability* problem, which is known to exactly capture the complexity of NL, and thus exhibiting a UL algorithm for reachability is sufficient to show that NL = UL.

We prove that UL contains certain important restrictions of directed graph reachability. In particular, we show that reachability in planar graphs and certain non-planar graphs are in UL. We give a proof that planar reachability is in UL, by using a result from multi-variable calculus, known as *Green's Theorem*. From another viewpoint, we show that deciding reachability in graphs where the number of paths from the start vertex to any other vertex is bounded by a polynomial, is in UL (this result shows that the complexity class ReachFewL is in UL). We also study and prove an upper bound on the UL hierarchy.

The NL versus UL question led us to another important problem in complexity theory - space complexity of deciding if a graph has a *perfect matching*. We prove that perfect matching in bipartite bounded genus graphs is in SPL (a class which is a generalization of UL and not known to be comparable with NL). We also show that over bipartite planar graphs, the perfect matching problem is in UL.

Embeddings algorithms for graphs on surfaces is well studied in the context of time complexity. Here we give log-space algorithms that provide us certain useful embeddings of planar and bounded genus graphs, on a corresponding surface.

DEDICATION

Dedicated to Ma, Baba and Mamma - they are the reason why I am here.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor Vinodchandran Variyam, without whom this dissertation would not have been possible. His valuable guidance and support forms the backbone of what little I know of complexity theory today. Right from day one of my PhD, Vinod has been a friend, philosopher and guide, in the truest sense of the word. His insistence on mathematical rigor and precision, a value that he also inculcated within me, is something for which I will always be grateful to him. No matter how busy he might be, he has always patiently listened to my harebrained ideas and extracted the useful bits of information, thus providing a sense of direction to my task.

I would also like to thank Matthew Dwyer, Jitender Deogun and Judy Walker for agreeing to serve on my PhD Supervisory Committee and helping me prepare a better dissertation.

I am deeply indebted to Neil Immerman for agreeing to serve on my PhD Supervisory Committee and inviting me to University of Massachusetts – Amherst. Interacting with Neil has been a wonderful learning experience for me. He has been available for virtually any kind of support that I asked of him. Without Neil's careful scrutiny and analysis, this dissertation would not have been at the same level as it currently is.

Any word of thanks to express my gratitude towards Eric Allender, will fall woefully short. I would like to thank Eric for hosting my visit to Rutgers and sharing his knowledge and intuition of complexity theory. I treasure the long conversations that I have had with him on various problems.

I thank Aduri Pavan for his support and encouragement for the past five years, in different aspects. I met Pavan at the beginning of my PhD and since then he has been there beside me as a mentor, collaborator and friend.

I would like to thank V. Arvind and Meena Mahajan for installing the passion for complexity theory during my undergraduate days at Chennai Mathematical Institute. Their valuable comments on some of the drafts of my papers have been extremely helpful. I would particularly like to thank Meena for hosting my visit to Matscience during the summer of 2008.

This dissertation would not have been possible without the help of Samir Datta. Since I first met him in Chennai, I was amazed with Samir's ardent dedication and great humility. He has always been available for discussions, questions, and any kind of support that I desired of him. A great collaborator and a wonderful human being.

My deepest and sincere thanks to Chris Bourke and Raghav Kulkarni for working with me and helping me on various academic and non-academic issues. Collaborating with them have been a great learning experience for me. I would also like to acknowledge the contributions of Derrick Stolee, Prajakta Nimbhorkar, and other colleagues of mine, during my graduate study at University of Nebraska-Lincoln.

I would like to thank Mark Brittenham for providing me valuable and deep insights into concepts in algebraic topology. The results in Chapter 4 would not have been as complete as they are, without the countless discussions that we had.

I would also like to thank the Computer Science and Mathematics faculty at University of Nebraska – Lincoln for the various courses and for being available anytime I wanted, to assist me.

My PhD would have been much more difficult, had the Computer Science departmental staff not been there to help me over the years. With a smile on their face, they have always taken care of my concerns and requests, however small they may be.

I would like to acknowledge the following National Science Foundation grants which partially funded the work done in this dissertation – CCF-0430991, CCF-0830730 and CCF-0916525.

My heartfelt thanks to Paromita for her perpetual support and belief in me. My long stay at Lincoln would not have been as pleasant as it was, without the presence of true friends like Subai and Pinaki. In my life, they aptly justify the saying – "a friend in need, is a friend indeed".

Last but not the least, no amount of appreciation and thankfulness can do justice to the contribution of my family in making this dissertation a reality, and for shaping my life in general. My PhD would have remained a distant dream, had it not been for their relentless and steadfast support. Baba has been my role model and the greatest inspiration of my life. Words cannot express the selflessness of Ma, the unflinching love and care of Mamma and the affection of Sarmistha and Babi. I want to thank Choto Pishemoshai for being the true friend whom I could always trust and fall back upon. I am deeply indebted to Mejo, Sejo, Ranga, Muni, Choto, Bakul, Chhanda, Kabita, Soma, Jhum, Indra, Naren, Maitri, Lopa, Krishnamohan, Mayuri, Rukmini, Ranchhor and Venkatesh for their monumental support and for having faith in me even during the darkest of times. Finally, I would like to express my gratitude for the person who inculcated the passion for mathematics and science within me and whose absence can never be filled – Boro Pishemoshai. I wish he was there in person to share this moment.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Computational complexity theory is the study of resource bounded computations. What are the problems that *can* or *cannot* be solved by a given model of computation, using limited amount of resources? Time, space, nondeterminism and randomness are some of the common resources that we usually consider. Complexity theory has a wide range of applications in computer science and else where, particularly in cryptography (designing efficient protocols that can withstand adversaries), algorithms (designing efficient algorithms), machine learning (studying the hardness of a learning algorithm), mathematics (factoring integers), physics (quantum computation), economics (finding Nash equilibria), etc. Despite several decades of research in this area, some of the most fundamental questions still remain unsolved. In this dissertation, we investigate and make progress on some questions related to *space bounded computations*.

Space complexity theory is the study of space bounded models of computations. What is the power and limitations of a computational model with space as a resource? To formalize the framework of our study, we bound the space of a computational model by a function of the input size. Thus a function uniquely defines a class of problem that are solvable by a given model of computation. Particularly interesting is the case when we focus on computational problems that are decidable using only a logarithmic amount of space (or *log-space* as we shall often call it), in the input size. Various models of computations like deterministic, nondeterministic, etc., can be looked under the lens of log-space computations. We denote the class of problems decidable by a deterministic (respectively nondeterministic) log-space bounded machine by L (respectively NL). Nondeterminism is a generalization of deterministic computations but *is nondeterministic computations any more powerful than deterministic computations, at least in the log-space domain?* This is one of the fundamental questions in space complexity that is unresolved to date. Savitch proved an important result in this context by showing that nondeterministic log-space is in the deterministic version of "quadratic log-space", that is, $\mathsf{NL} \subseteq \mathsf{DSPACE}(\log^2 n)$ (this result is popularly known as *Savitch's Theorem*) [Sav70].[1] Unfortunately there has been no improvement on this problem in its generality, in the last forty years!

Unlike time bounded computations where a majority of researchers believe that $\mathsf{NP} \neq \mathsf{P}$ (the nondeterministic and deterministic versions of polynomial time bounded computations respectively), the case of space bounded is not the same. Savitch's Theorem and the closure of NL under complementation [Imm88, Sze88], are some evidence in support of this hypothesis. Also a fundamental difference between time and space bounded computations is that, *time cannot be reused* whereas *space can*!

1.1 Can nondeterministic log-space computations be made unambiguous?

An interesting class of problems that lie in between deterministic and nondeterministic computations is *unambiguous nondeterminism*. Intuitively, it is a restriction of nondeterminism where we only allow *exactly one* accepting computation on an instance in the language (as opposed to *at least one* accepting computation in the case of general nondeterminism), and no accepting paths

[1] $\mathsf{DSPACE}(f)$ is the class of problems decidable by a deterministic Turing machine whose space is bounded by the function $O(f(n))$.

on an instance outside the language. Thus in some sense it is a *promise* model, where the promise that all instances must have at most one accepting path, must be fulfilled. We denote the class of problems decidable by an unambiguous, nondeterministic, log-space bounded machine by UL. The definition of unambiguous nondeterminism naturally raises the following question - *is unambiguous nondeterminism powerful enough to capture whole of general nondeterminism in the log-space setting?*

In the polynomial time setting introduced the class UP [Val76], the unambiguous version of NP, which proved to be a very useful restriction to study, mainly because of its connection to certain kind of one-way functions [GS88]. In the log-space setting, the class UL was first defined and studied by [BJLR91] and [AJ93]. Since then, UL and related low-space unambiguous classes have been of interest to researchers [BJLR91, AJ93, Lan97, AL98, RA00, ADR05]. This class is particularly interesting because there is increasing evidence that the whole of nondeterministic log-space might be contained in UL (this is in contrast with the polynomial-time setting where there are some evidence that unambiguity is a true restriction to nondeterminism [Rac82, Ko85, GS88]). Reinhardt and Allender showed that the non-uniform version of UL contains NL, that is $\mathsf{NL} \subseteq \mathsf{UL/poly}$ [RA00]. Can this collapse be made uniform? That is, is it true that $\mathsf{NL} = \mathsf{UL}$?

The problem of deciding *reachability* in graphs (also known as the *st-connectivity* problem where we are trying to decide if t is reachable from s) is fundamental in the study of complexity theory. They capture the computational power of several complexity classes, in particular space bounded complexity classes. The general reachability problem for directed graphs is complete for NL and reachability problem in undirected graphs is complete for L [Ete97, Rei08]. Various restricted versions of this problem characterize other low-level complexity classes. Reachability in constant width grid graphs, constant width branching programs characterize the complexity classes AC^0 and NC^1 respectively [BLMS98, Bar89]. Therefore reachability and its various restrictions, become a natural candidate problem in the study of log-space nondeterminism, and exhibiting a UL algorithm for directed reachability would immediately imply that $\mathsf{NL} = \mathsf{UL}$. In this dissertation we make progress towards solving this problem and show that several natural restrictions of directed reachability (e.g. planar reachability , reachability in graphs with polynomially number of paths, etc) are in UL. We also show that certain restrictions of reachability are powerful enough to capture all of NL.

The NL versus UL problem can be *reduced* to an instance of a more general problem, known as the *isolation problem* [MVV87], which deserves attention in its own right. In Section 1.2 we introduce this problem and its connection to certain complexity theoretic questions.

1.2 Isolation problem

Distinguishing a single solution with a certain property, out of a set of solutions, is a basic algorithmic problem with many applications. Very often a computational problem has a "large" solution set and "efficiently" extracting a particular solution out of the set can be a challenging job. More formally, this is known as the isolation problem, and is defined as follows: given a set $U = \{1, \ldots, n\}$ and a non-empty family of subsets of U, say $\mathcal{F}$, define an "efficiently" computable function on U that assigns a polynomially bounded integer value to each element in U, such that the minimum weight set in $\mathcal{F}$ with respect to the weight function, becomes unique. Our notion of efficient computation would typically be computations that can be done in a logarithmic amount of workspace in the input size. Such a computational restriction is sufficient because any logarithmic space computation can only take polynomial amount of time since there are only polynomially many configurations possible. Such a model is necessary because we are interested in looking at space bounded computations. Also the question as to how the family of subsets $\mathcal{F}$ is represented, is very crucial. If F is explicitly given as an input then the problem becomes trivial and not very natural. Thus what we would assume is that $\mathcal{F}$ is provided to us implicitly via a succinct description, that is polynomially bounded in the size of U.

The isolation problem was first explicitly defined and studied by Mulmuley, Vazirani and Vazirani [MVV87]. Unfortunately they were unable to provide a deterministic solution to the problem (which in fact still eludes us) but they did come up with a randomized algorithm to solve the isolation problem. In particular, they showed that if we assign weights to the elements in U from the set of integers $\{1, \ldots, |U|^2\}$, uniformly at random, then with probability greater than $1 - 1/|U|$, the minimum weight set in $\mathcal{F}$ would be unique with respect to the weight function. This is also known as the *isolating lemma* (see Chapter 2 for a formal statement of the theorem). Since its discovery, the isolating lemma has found many applications, mostly in discovering new randomized and non-uniform upper bounds, via isolating minimum weight solutions [MVV87, GW96, ARZ99, RA00].

Note that the isolating lemma does not use any information about the family $\mathcal{F}$. In other words, the result would hold irrespective of the choice of $\mathcal{F}$. Unfortunately this is not possible in the deterministic setting. By a simple counting argument it can be shown that given U, for all weight functions $w : U \rightarrow n^k$, for any fixed k, there exists a collection $\mathcal{F}$ of subsets of U, having at least two subsets with the same minimum weight with respect to the weight function w (see [Agr07] for a proof). Also recently it was shown that if $\mathcal{F}$ is given in terms of certain circuits, then derandomization of the isolating lemma will imply certain circuit lower bounds and hence is a difficult task [AM08]. However, such negative results do not rule out the possibility of bypassing a general solution to the isolating lemma altogether and directly prescribing efficient deterministic weight functions for specific situations so that the minimum weight solution becomes unique.

Showing that $\mathsf{NL} = \mathsf{UL}$ can be reduced to an instantiation of the isolation problem [RA00]. For a class of directed graphs $\mathcal{G}$, isolating a directed path between every pair of vertices would imply that reachability in $\mathcal{G}$ is in UL. In other words, if for any graph G in $\mathcal{G}$, we can construct an edge weight function such that the minimum weight path between every pair of vertices is unique with respect to the weight function, then reachability in the class of graphs $\mathcal{G}$ is in UL. Such class of graphs are also known as *min-unique* graphs.[2] Now if we can show that for a class of graphs

[2]A class of graphs $\mathcal{G}$ is said to be min-unique if for every graph $G \in \mathcal{G}$, there exists an edge weight function that is uniformly computable in log-space, with respect to which between every pair of vertices, if there is a path, then there is a unique minimum weight path.

$\mathcal{G}$, such that reachability in $\mathcal{G}$ is complete for NL and $\mathcal{G}$ is also min-unique with respect to some efficiently computable weight function, then it would imply that $\mathsf{NL} = \mathsf{UL}$.

It is interesting to note that the motivation for Mulmuley, Vazirani and Vazirani in studying the isolation problem was the problem of perfect matching [MVV87]. They gave an efficient, randomized, parallel algorithm for constructing a maximum matching (by isolating a minimum weight perfect matching) in general graphs. This led us to the study of the perfect matching problem. In Section 1.3 we introduce the perfect matching problem and study it in the context of space efficient algorithms.

1.3 Space complexity of perfect matching

The *perfect matching* problem asks the following question: *Does a given undirected graph have a perfect matching?* This question and its variations (eg. constructing a perfect matching, checking uniqueness of a perfect matching, etc.), are some of the most well-studied problems in theoretical computer science. Edmonds gave a polynomial time algorithm to compute a matching of maximum cardinality in a graph [Edm65]. This was known as the *blossom* algorithm and was one of the first examples of a non-trivial algorithm, having polynomial runtime. Subsequently, Valiant showed that counting the number of perfect matchings in a bipartite graph is #P-hard [Val79]. The question of whether or not the perfect matching problem is efficiently parallelizable has yielded powerful tools such as the *isolating lemma* [MVV87] that have found numerous applications elsewhere in complexity theory and theoretical computer science in general (see [LP86] for an excellent introduction to matching and related problems).

In the domain of parallel algorithms and space complexity, the complexity of perfect matching is not very well understood. Whether perfect matching has an efficient parallel algorithm (in other words, is perfect matching in the class NC) - is an important question in parallel complexity. Even for the restricted case of planar graphs or bipartite graphs we do not know if this is true. For the class of logarithmic genus bipartite graphs though, it was shown that the perfect matching problem is in NC [MV00]).

An important log-space *counting* complexity class is SPL, which is the class of problems solvable by an NL machine such that the difference in the number accepting and rejecting paths is 1 for a 'Yes' instance and 0 for a 'No' instance. Equivalently, it is also the class of problems reducible to the determinant with the promise that the determinant is either 0 or 1. SPL is a generalization of log-space unambiguity in some sense. It can easily be shown that UL is contained in SPL (see Chapter 2 for a proof of this claim).

Perfect matching is contained in the non-uniform version of SPL (that is $\mathsf{SPL/poly}$) [ARZ99]. The position of SPL relative to some of the complexity classes that we have seen so far is as follows: SPL is contained in NC but no relation is known between NL and SPL.

It was shown that, proving an SPL upper bound on the perfect matching problem, can be cast as an instance of the isolation problem [ARZ99]. More specifically, for a class of undirected graphs $\mathcal{H}$, isolating a minimum weight perfect matching would imply that the perfect matching problem in the class $\mathcal{H}$ is in SPL (again by assigning an edge weight function with respect to which the minimum weight perfect matching is unique). They in fact prove that perfect matching in general graphs is in $\mathsf{SPL/poly}$ by showing that a random weight function performs the required isolation. Recently, for bipartite planar graphs this non-uniform bound was derandomized, thus proving that perfect matching in bipartite planar graphs is in SPL [DKR10]. In this dissertation we extend the SPL bound to bounded genus bipartite graphs.[3]

As described below, we solve a more general combinatorial problem and show that isolating directed paths and perfect matchings, follows from solution of the problem. More specifically, we show that for a class of bipartite directed graphs $\mathcal{G}$, and for any graph G in $\mathcal{G}$, if we can come up

[3]Note that in both the instances of the isolation problem that we consider (isolating a directed path and isolating a perfect matching), the input is the graph and the structure that we want to isolate (directed path or perfect matching) is only implicitly given.

with a skew-symmetric,[4] polynomially bounded weight function w such that for any simple cycle C in G, $w(C) \neq 0$ then (i) w also isolates directed paths in $\mathcal{G}$, and (ii) in log-space we can compute a weight function w' from w, such that w' isolates perfect matchings in the class of underlying undirected graphs of $\mathcal{G}$.

We also exhibit the first nondeterministic log-space upper bound for the perfect matching problem in bipartite planar graphs. More generally, we show the stronger result that planar, bipartite perfect matching is in unambiguous log-space.

1.4 Organization of the dissertation

In Chapter 2 we give some basic definitions and terminologies that we use in the rest of the dissertation. We also state certain earlier results that we use extensively.

In Chapter 3 we study the isolation problem in planar graphs [TV10]. No better space upper bound was known for the reachability problem in directed planar graphs, other than NL. We obtain a new upper bound on planar reachability (denoted as PLANARREACH) as stated below.

Theorem 1.4.1. PLANARREACH $\in \mathsf{UL} \cap \mathsf{coUL}$.

Green's theorem is a well-known result in multi-variable calculus with a wide range of applications. Here we show that via an application of Green's theorem, one can settle the question of isolation in planar graphs. In particular, we give a weight function with respect to which the class of planar graphs become min-unique. Then, by applying the $\mathsf{UL} \cap \mathsf{coUL}$ algorithm for min-unique graphs [RA00], Theorem 1.4.1 follows. Our isolation result also gave an alternate proof that perfect matching in bipartite planar graphs is in SPL (the bound was proven earlier in [DKR10]).

To prove the isolation result in this chapter, we assume that the graph is given as a straight line embedding. However the proof of the isolation result also works when instead of a straight

[4]A weight function w is said to be *skew-symmetric* if for an edge (u, v), $w(u, v) = -w(v, u)$ (if the edge (v, u) is present).

line embedding, we are provided with a *piecewise straight line embedding* (an embedding where an edge consists of constantly many pieces of straight line segments). We also give a log-space algorithm that outputs a piecewise straight line embedding of a given planar graph.

We also show an extension of our result to a certain class of non-planar graphs by reducing the given instance to reachability question in a graph of smaller size and applying Savitch's theorem [Sav70].

Corollary 1.4.2. *Let $\mathcal{G}$ be a class of directed graphs $G = (V, E \cup E')$ such that (V, E) is planar and $|E'| \leq \mathcal{O}(2^{\sqrt{\log n}})$ where $n = |V|$. Then st-connectivity for any graph in $\mathcal{G}$ can be decided in* $\mathsf{UL} \cap \mathsf{coUL}$.

In Chapter 4, we show that similar isolation results can be shown for graphs on surfaces of bounded genus [DKTV11]. We combine our techniques from Chapter 3 with concepts from algebraic topology to achieve this. We assume that the input graph is given to us as a combinatorial embedding on a surface of bounded genus. From the combinatorial embedding, we give a log-space implementation of the *surface classification theorem* [DH07, Bra21] to get an embedding of the graph on a fundamental polygonal schema. We then prove an isolation result analogous to Chapter 3, that gives us a log-space constructible weight function to isolate the minimum weight perfect matching in bounded genus bipartite graphs and thereby an SPL algorithm for checking if the graph has a perfect matching by derandomizing the $\mathsf{SPL/poly}$ algorithm of [ARZ99]. We state our results formally in the following theorem.

Theorem 1.4.3. *Let G be a bipartite bounded genus graph, given as a combinatorial embedding on a surface of bounded genus. Then,*

- *deciding if G has a perfect matching is in* SPL,
- *constructing a perfect matching in G (if one exists) is in* $\mathsf{FL}^{\mathsf{SPL}}$ *(the functional version of* SPL*), and*

- *deciding if G has a unique perfect matching is in* SPL.

From the isolating result that we prove in this chapter, we also obtain a direct proof of the fact that reachability in bounded genus directed graphs is in UL $\cap$ coUL. Note that Kynčl and Vyskočil had earlier shown that bounded genus reachability reduces to planar reachability [KV10]. Combining their result with Theorem 1.4.1 gives us an alternate proof of the same result.

The intuition as to why we are able to extend to bounded genus graphs, is due to the separate treatment of *surface separating* and *surface non-separating* cycles in the graph. A planar graph only has cycles of the former type but in a higher genus graph, we can have cycles of the latter type also. We use the power of homology theory to give a combinatorial characterization of the latter kind of cycles, that aids us in achieving the desired isolation.

In Chapter 5 we prove that certain non-trivial restrictions of directed reachability are hard for nondeterministic log-space [BTV09, PTV10]. We define three classes of graphs based on certain geometric properties as follows.

- ThreePage is the class of graphs G that can be embedded on a book with 3 pages such that, all vertices of G lie along the spine of the book and the edges lie on exactly one of the three pages without intersection, directed from top to bottom.

- A *three-dimensional monotone grid graph* is a directed graph whose vertices are $[n] \times [n] \times [n]$ with edges connecting a vertex to its immediate neighboring grid point in the positive x, y or z direction. That is an edge is of the form $((i, j, k), (i + 1, j, k))$ (east edge) or $((i, j, k), (i, j + 1, k))$ (north edge) or $((i, j, k), (i, j, k + 1))$ (inward edge), provided the respective coordinates exist.

- Another graph theoretic notion that we study is *geometric thickness*. The geometric thickness of a graph G is defined as the minimal number k such that we can assign planar point locations to the vertices of G, represent each edge as a line segment, and assign each edge to one of the k transparencies so that no two lines cross in any one transparency.

We then prove the following hardness results about NL.

Theorem 1.4.4. *Reachability in the following class of directed graphs is complete for* NL*: (i)* ThreePage*, (ii) three-dimensional monotone grid graphs, and (iii) graphs with geometric thickness* 2.

In this context it may be worthwhile to note that graphs with page number two, two dimensional grid graphs and thickness one graphs are all subclasses of planar graphs, and thus reachability in them are in UL.

In Chapter 6 we study the power and limitations of unambiguous log-space computations with the bigger goal of showing if such computations capture the whole of NL [PTV10]. We also study possible approaches to achieve this goal. ReachFewL is a subclass of NL characterized by the question of reachability in directed graphs where the number of paths from s to any vertex is bounded by a polynomial. ReachFewL was first defined and studied in [BJLR91]. In the following theorem we prove that counting the number of paths in such graphs is in the functional version of UL (that is, FUL).

Theorem 1.4.5. *For any polynomial $q(n)$, there is a nondeterministic log-space bounded Turing machine M so that, for any graph $G = (V, E)$ and two vertices s and t in G, if the number of paths from s to any vertex is bounded by $q(|V|)$, then M will output the number of paths from s to t on a unique path in M (all other paths reject).*

As an immediate corollary of Theorem 1.4.5 we observe that ReachFewL is in UL. We also show that the notion of *min-uniqueness* is not only sufficient but also necessary to show that NL $=$ UL.

Theorem 1.4.6. $\mathsf{NL} = \mathsf{UL}$ *if and only if there is a polynomially bounded* UL*-computable weight function f so that for any directed acyclic graphs G, the weighted graph $f(G)$ is min-unique with respect to s.*

$\mathsf{OptL}[\log \mathsf{n}]$ is the class of functions whose values are the minimum over all outputs of an NL-transducer and the size of the outputs are bounded by $O(\log n)$. This class was first defined and studied by Àlvarez and Jenner who showed that $\mathsf{OptL}[\log \mathsf{n}]$ captures the complexity of certain natural optimization problems [AJ93]. We consider the unambiguous version of $\mathsf{OptL}[\log \mathsf{n}]$, which we call $\mathsf{UOptL}[\log \mathsf{n}]$ where the minimum value is output along a unique computation path, and show that $\mathsf{NL} = \mathsf{UL}$ if and only if $\mathsf{OptL}[\log \mathsf{n}] = \mathsf{UOptL}[\log \mathsf{n}]$. We show two upper bounds on $\mathsf{UOptL}[\log \mathsf{n}]$ in reference to the complexity classes SPL and UL.

Theorem 1.4.7. *1.* $\mathsf{UOptL}[\log \mathsf{n}] \subseteq \mathsf{FL}^{\mathsf{SPL}}[\log n]$.

2. $\mathsf{UOptL}[\log \mathsf{n}] \subseteq \mathsf{FL}^{\mathsf{promiseUL}}$.

Unlike NL, UL is not closed under complementation and therefore it makes sense to study the UL hierarchy. The UL hierarchy is defined as follows: $\mathsf{ULH}_1 = \mathsf{UL}$, $\mathsf{ULH}_{i+1} = \mathsf{UL}^{\mathsf{ULH}_i}$ and $\mathsf{ULH} = \bigcup_i \mathsf{ULH}_i$. We show that the UL hierarchy is contained in $\mathsf{L}^{\mathsf{promiseUL}}$. We also study the $\mathsf{UOptL}[\log \mathsf{n}]$ hierarchy and show that it collapses to $\mathsf{UOptL}[\log \mathsf{n}]$. We formally state the results in the following theorem.

Theorem 1.4.8. *1.* $\mathsf{ULH} \subseteq \mathsf{L}^{\mathsf{promiseUL}}$.

2. $\mathsf{UOptL}[\log \mathsf{n}]^{\mathsf{UOptL}[\log \mathsf{n}]} \leq \mathsf{UOptL}[\log \mathsf{n}]$ *under metric reductions.*

In Chapter 7 we revisit the problem of perfect matching in bipartite, planar graphs. Looking at the problem from a different point of view, we give a UL upper bound on the problem [DKT10]. This is a significant improvement over the known bound of SPL. It is interesting to note here that NL and SPL are incomparable classes and it was not even known whether planar bipartite perfect matching is in NL. Our result settles this question positively. Our result is based on two existing techniques - (i) building upon the algorithm of Miller and Naor [MN89] and proving certain new upper bounds on its space complexity, and (ii) suitably modifying the UL algorithm of Reinhardt and Allender [RA00] to suit our needs. We give the following space bound on perfect matchings in planar bipartite graphs.

Theorem 1.4.9. *Let G be a planar bipartite graph. Deciding if G has a perfect matching and constructing one if it exists, is in* UL.

We also look at another important problem in this chapter - the *even path problem*.[5] For general graphs this problem is NP-complete [LP83], for planar graph the problem is in P [Ned99] and in acyclic graphs the problem is NL-complete. Moreover, a generalization of this problem, known as the *red-blue path problem*,[6] is NL-hard in planar directed acyclic graphs (we write them as DAG in short) [Kul09]. We show the following result.

Theorem 1.4.10. *The even path problem for planar DAGs is in* UL.

To prove Theorem 1.4.10 we use a combination of the two known isolating techniques due to [BTV09] and [Hoa10], in a non-trivial manner. This is the first time the two known deterministic isolation techniques have been combined to obtain a new upper bound. We feel that this approach is very promising and might help in extending our isolation results to more general cases.

Finally in Chapter 8 we give an overview of the progress that we have made so far and what questions still remain unsolved. We also suggest possible techniques and approaches to tackle the open questions.

The results that we show in this dissertation have been published or are in the process of being published in various conferences and journals. They are proven together with Chris Bourke, Samir

[5]Given a directed graph G and two vertices s and t in G, does G have a simple even length path from s to t.

[6]Given a directed graph G whose edges are colored either red or blue, and two vertices s and t in G, does G have a simple path from s to t that alternates between red and blue edges.

Datta, Raghav Kulkarni, Aduri Pavan and N. V. Vinodchandran. In particular, our results in Chapter 3 are from [BTV09, TV10], in Chapter 4 are from [DKTV11], in Chapter 5 are from [PTV10, BTV09], in Chapter 6 are from [PTV10], and in Chapter 7 are from [DKT10].

Chapter 2

Preliminaries

In this chapter we mention the basic definitions and earlier results that we use in this dissertation. In Section 2.1 we define certain graph-theoretic notions and problems. In Section 2.2 we define complexity theory classes and conventions that are used in the rest of the chapters. In Section 2.3 we state certain earlier results that we use later.

2.1 Graph theory

Definition 2.1.1. (Planar graphs)

- A graph G is said to be *planar* if G can be drawn on the plane such that no two of its edges intersect at an intermediate point. Such a drawing is also known as a *planar embedding*.

- A *plane graph* is a planar graph G, together with an embedding of G on the plane.

- A *combinatorial embedding* ϕ is a cyclic ordering of the edges around each vertex. More formally, for a directed graph G,[1] and an edge (u, v), $\phi(u, v) = (u, w)$ where (u, w) is the next outgoing edge from u in a cyclic ordering (say anti-clockwise) of the edges around u.

[1]For an undirected graph, we can replace every edge $\{u, v\}$ with the directed edges (u, v) and (v, u).

Definition 2.1.2. (Dual Graphs) Let $G = (V, E)$ be a directed plane graph and let F be the set of faces of G. Then the *dual* of G is the graph $G^* = (F, E^*)$, where for every edge $e = (u, v) \in E$ the corresponding *dual edge* $e^* = (u^*, v^*) \in E^*$ where u^* and v^* are the faces of G to the east and west of the edge (u, v), if we assume (u, v) to be directed from south to north.[2]

Note that the dual graph can have self-loops and multiple edges.

Definition 2.1.3. (Grid Graphs) A graph $G = (V, E)$ is said to be a *grid graph* if $V = [n] \times [n]$ and $E \subseteq \{((i_1, j_1), (i_2, j_2)) \mid (i_1, j_1), (i_2, j_2) \in V$ and $|i_1 - i_2| + |j_1 - j_2| = 1\}$.

Definition 2.1.4. (Matching) Given an undirected graph $G = (V, E)$, a *matching* M is a subset of E such that no two edges in M have a vertex in common. A *maximum matching* is a matching of maximum cardinality. M is said to be a perfect matching if every vertex is an endpoint of some edge in M.

Definition 2.1.5. (Graph Reachability) *Graph Reachability* is a language consisting of the instances (G, s, t), such that G is a directed graph having two vertices s and t, and there exists a path from s to t in G. Restriction of graph reachability to cases where we study a smaller class of graphs are also well studied. Some of the important restrictions are planarity, bounded genus, etc.

For other general definitions on graph theory, please refer to a standard graph theory text (say [Die10]).

2.2 Complexity theory

Let $[n]$ denote the set $\{1, 2 \ldots, n\}$. A *language* L is a subset of $\{0, 1\}^*$. For a Turing machine M and a string x, we denote the computation of M on x as $M(x)$. A Turing machine M is said to *decide* a language L, if for every $x \in \{0, 1\}^*$, $M(x)$ accepts if x is in L and $M(x)$ rejects if x is

[2]The dual of an undirected graph is defined similarly such that the dual edge is also undirected.

not in L. In this dissertation we shall assume that the Turing machines we consider, do not repeat any configuration. In other words, the configuration graph of a machine on an input, is acyclic.

Definition 2.2.1. (Oracle machines and classes)

- Let $A \subseteq \{0,1\}^*$. Then an *oracle Turing machine* M is a Turing machine with a query tape and special states $q_?$, q_Y and q_N, such that M runs as a usual Turing machine with an oracle access to A. That is when M enters state $q_?$, it determines whether the string on the query tape, y is contained in A or not, and based on the answer it moves to state q_Y or q_N respectively.

- Let $\mathcal{C}$ be a complexity class decided by a Turing machine M and let A be a language. Then $\mathcal{C}^A$ is the class of solvable by an oracle Turing machine M' which is similar to M but with an oracle access to A.

- Let $\mathcal{C}$ and $\mathcal{D}$ be two complexity classes. Then $\mathcal{C}^{\mathcal{D}} = \bigcup_{A \in \mathcal{D}} \mathcal{C}^A$.

Definition 2.2.2. (Space Complexity Classes) Let L be a language.

- L is said to be in $\mathsf{DSPACE}(f)$, if there is a deterministic Turing machine M deciding L and on every input $x \in \{0,1\}^*$, $M(x)$ uses at most $O(f(|x|))$ amount of workspace.

- L is said to be in $\mathsf{NSPACE}(f)$, if there is a nondeterministic Turing machine M deciding L and on every input $x \in \{0,1\}^*$, $M(x)$ uses at most $O(f(|x|))$ amount of workspace.

Definition 2.2.3. (Log-space Complexity Classes) Let L be a language.

- L is said to be in deterministic log-space (L) if there is a deterministic log-space bounded Turing machine deciding L. Alternatively, $\mathsf{L} = \mathsf{DSPACE}(\log n)$.

- L is said to be in nondeterministic log-space (NL) if there is a non-deterministic log-space bounded Turing machine deciding L. Alternatively, $\mathsf{NL} = \mathsf{NSPACE}(\log n)$.

- L is said to be in unambiguous nondeterministic log-space (UL) if there is a nondeterministic log-space bounded Turing machine M, such that for every x in L, $M(x)$ has a unique accepting path, and for every x not in L, $M(x)$ has no accepting path.

We also study certain languages based on the number of accepting and rejecting paths of a NL machine. For a nondeterministic Turing machine M, let $acc_M(x)$ and $rej_M(x)$ denote the number of accepting computations and the number of rejecting computations respectively on an input x. Denote $gap_M(x) = acc_M(x) - rej_M(x)$.

Definition 2.2.4. (The class SPL)

- A language L is in SPL if there exists a log-space bounded nondeterministic machine M so that for all inputs x, $gap_M(x) \in \{0, 1\}$ and $x \in L$ if and only if $gap_M(x) = 1$.

- $\mathsf{FL}^{\mathsf{SPL}}$ is the class of functions computed by a log-space machine with an SPL oracle.

Alternatively, we can define SPL as the class of problems log-space reducible to the problem of checking whether the determinant of a matrix is 0 or not under the promise that the determinant is either 0 or 1. In Proposition 2.2.1 we show that UL is contained in SPL.

Proposition 2.2.1. $\mathsf{UL} \subseteq \mathsf{SPL}$.

Proof. Let L be a language decided by a UL machine M. We construct a nondeterministic log-space machine N based on M as follows: whenever M rejects, N non-deterministically either accepts or rejects. Note that for each rejecting path of M, N has exactly one accepting and one rejecting path.

For an instance $x \in L$, $M(x)$ has exactly one accepting path and the rest of the paths reject. Therefore $gap_N(x) = 1$. Similarly for $x \notin L$, $M(x)$ has no accepting path and therefore $gap_N(x) = 0$. Thus N is an SPL machine and $L \in \mathsf{SPL}$ as required. □

We next define complexity classes defined in terms of circuit families instead of Turing machines. Note that unlike a Turing machine, a circuit can only take as input, a string of a fixed length. Therefore we consider a family of circuits (that is a collection of circuits such that there is exactly one circuit for each natural number) to define a language. A circuit family $\{C_n\}_{n\in\mathbb{N}}$ is said to be *uniform*, if there is a log-space machine, which when provided the input 1^n, outputs C_n. Otherwise the family of circuits is said to be *non-uniform*. The *fan-in* of a gate in a circuit is defined as the number of inputs to that gate. The fan-in of a circuit is the maximum fan-in of the circuit, taken over all gates.

Next we define the complexity class NC, which is the class of problems that have efficient parallel algorithms.[3]

Definition 2.2.5. (The complexity class NC)

- NC^i is the class of decision problems solvable by a uniform family of Boolean circuits, having polynomial size, depth $O(\log^i(n))$, and fan-in 2.
- $\mathsf{NC} = \bigcup_i \mathsf{NC}^i$.

Definition 2.2.6. (Non-uniform Classes) Let $\mathcal{C}$ be a complexity class. Then a language A is said to be in $\mathcal{C}/\mathsf{poly}$ (the non-uniform version of $\mathcal{C}$), if there is a function $f : \mathbb{N} \to \mathbb{N}$ (such that $f(n)$ is bounded by a polynomial in n) and a language $B \in \mathcal{C}$, such that $x \in A$ if and only if $(x, f(|x|)) \in B$.

Definition 2.2.7. (Transducer)

- A Turing machine M is said to be a log-space transducer (or an L-transducer) if M is a log-space bounded, deterministic machine, with a write-only, one-way output tape and M outputs a string when it reaches an accept state.

[3] NC stands for Nick's class, named in honor of Nick Pippenger.

- A Turing machine M is said to be an NL-transducer if M is a log-space bounded, nondeterministic machine, with a write-only, one-way output tape and M outputs a string when it reaches an accept state. Note that along different computation paths, M might output a different string.

Definition 2.2.8. (Log-space Functional Classes) Let $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ be a function. Then,

- f is said to be in FL if there exists an L-transducer M, such that for any $x \in \{0,1\}^*$, $M(x)$ outputs $f(x)$.

- FNL is the class of functions computed by an FL transducer with an oracle access to NL.

Definition 2.2.9. (Log-space reduction) A language A is said to be log-space reducible to another language B if there is a log-space bounded transducer L such that for string $x \in \{0,1\}^*$, x is in A if and only if the output of $L(x)$ is in B.

In this dissertation all reductions considered are log-space reductions, unless otherwise specified.

Definition 2.2.10. (Complete problem) A language L is said to be *hard* for a complexity class $\mathcal{C}$ (denoted as $\mathcal{C}$-hard), if for every language $A \in \mathcal{C}$, A reduces to L. A language L is said to be *complete* for a complexity class $\mathcal{C}$ (denoted as $\mathcal{C}$-complete), if L is $\mathcal{C}$-hard and $L \in \mathcal{C}$.

For definitions of other complexity classes refer to any standard textbooks such as [AB09, Vol99].

Definition 2.2.11. (Matching Problems) We define the following computational problems related to matching:

- PM-Decision: Given a graph G, checking if G has a perfect matching.

- PM-Construct: Given a graph G, constructing a perfect matching, if one exists.

- PM-UNIQUE: Given a graph G, checking if G has a unique perfect matching.
- MIN-WT-PM: Given a graph G together with edge-weights $w : E(G) \to \mathbb{Z}$ such that $|w(e)| \leq n^{O(1)}$, and an integer k, decide if G contains a perfect matching of weight at most k.
- MAX-MATCH: Given a graph G and an integer k, decide if G has a matching of cardinality at least k.

Definition 2.2.12. A *min-unique* graph is a directed graph with positive weights associated with each edge where for every pair of vertices u, v, if there is a path from u to v, then there is a unique minimum weight path from u to v. Here, the weight of a path is the sum of the weights on its edges.

Reinhardt and Allender actually define min-uniqueness for unweighted graphs [RA00], but these two definitions are essentially same in our context as one can replace an edge e with positive integer weight $w(e)$, with a path of length $w(e)$. For completeness, we present a somewhat shorter version of the proof of [RA00] in Section 2.3.1. This proof uses a clever extension of the inductive counting techniques of [Imm88] and [Sze88].

2.3 Earlier results

In this section, we state some earlier results that we use and often refer to in this dissertation.

Theorem 2.3.1 ([Rei08]). *Undirected graph reachability is in* L.

Theorem 2.3.2 ([AM04]). *Given a graph G (say as an adjacency matrix), checking if G is a planar graph and if it is, giving a planar combinatorial embedding of G, reduces to the problem of deciding reachability in undirected graphs.*

Now, combining Theorem 2.3.1 and Theorem 2.3.2 we get that, deciding if a graph is planar and giving a combinatorial embedding of a planar graph is in L.

Theorem 2.3.3 ([MVV87]). *Let $U = \{x_1, \ldots, x_n\}$ be a set and let $\mathcal{F}$ be a non-empty family of subsets of U. Let $w : U \longrightarrow [n^2]$ be a weight function on S. Then,*

$$\Pr_w[\textit{there exists a unique minimum weight set in } \mathcal{F}] \geq \frac{1}{n}.$$

Theorem 2.3.3 is also popularly known as the *isolating lemma*.

Theorem 2.3.4 ([FKS84]). *For every constant c there is a constant c' such that for every set S of n-bit integers with $|S| \leq n^c$ the following holds: There is a $c' \log n$-bit prime number p so that for any $x \neq y \in S$ we have $x \not\equiv y$ $(mod\, p)$.*

2.3.1 Reachability in min-unique graphs is in unambiguous log-space

In this section we give a slight variant of the proof of the UL membership algorithm for min-unique graphs, shown by Reinhardt and Allender [RA00]. We shall make use of this technique to show that various restrictions of graph reachability are in $\mathsf{UL} \cap \mathsf{coUL}$.

Theorem 2.3.5 ([RA00]). *Let $\mathcal{G}$ be a class of graphs and let $G = (V, E) \in \mathcal{G}$. If there is a polynomially-bounded log-space computable function f that on input G outputs a weighted graph $f(G)$ so that*

1. *$f(G)$ is min-unique and*
2. *G has an st-path if and only if $f(G)$ has an st-path.*

then the st-connectivity problem for $\mathcal{G}$ is in $\mathsf{UL} \cap \mathsf{coUL}$.

Proof. Let G be a directed graph with a min-unique weight function w on edges. We first construct an unweighted graph G' from G by replacing every edge e in G with a path of length $w(e)$. It is easy to see that st-connectivity is preserved. That is, there is an st-path in G if and only if there is one in G'. Since G is min-unique, it is straightforward to argue that the shortest path between any two vertices in G' is unique.

Let c_k and Σ_k denote the number of vertices which are at a distance at most k from s and the sum of the lengths of the shortest path to each of them, respectively. Let $d(v)$ denote the length of the shortest path from s to v. If no such path exists, then let $d(v) = |V| + 1$. We have,

$$\Sigma_k = \sum_{\substack{v \in V \\ d(v) \leq k}} d(v) \, .$$

We first give an unambiguous routine (Algorithm 1) to evaluate the predicate "$d(v) \leq k$" when *given* the values of c_k and Σ_k. The algorithm will output the correct value of the predicate (*true/false*) on a unique path and outputs $?$ on rest of the paths.

We argue that Algorithm 1 is unambiguous.

1. If Algorithm 1 incorrectly guesses that $d(x) > k$ for some vertex x then $count < c_k$ and so it returns $?$ in *line* 18. Thus consider the computation paths that correctly guess the set $\{x \mid d(x) \leq k\}$.

2. If at any point the algorithm incorrectly guesses the length l of the shortest path to x, then one of the following two cases occur.

   a) If $d(x) > l$ then no path s to x would be found and the algorithm returns $?$ in *line* 11.

   b) If $d(x) < l$ then the variable sum would be incremented by a value greater than $d(x)$ and thus sum would be greater than Σ_k causing the algorithm to return $?$ in *line* 18.

Thus there will remain only one computations path where all the guesses are correct and the algorithm will output the correct value of the predicate on this unique path. Finally, we note that Algorithm 1 is easily seen to be log-space computable.

```
    Input: $(G, v, k, c_k, \Sigma_k)$
    Output: $true$ if $d(v) \le k$ else $false$
 1  Initialize $count \leftarrow 0;\ sum \leftarrow 0;\ path.to.v \leftarrow false$ ;
 2  foreach $x \in V$ do
 3      Nondeterministically guess if $d(x) \le k$ ;
 4      if guess is Yes then
 5          Guess a path of length $l \le k$ from $s$ to $x$ ;
 6          if guess is correct then
 7              Set $count \leftarrow count + 1$ ;
 8              Set $sum \leftarrow sum + l$ ;
 9              if $x = v$ then set $path.to.v \leftarrow true$ ;
10          else
11              return $?$
12          end
13      end
14  end
15  if $count = c_k$ and $sum = \Sigma_k$ then
16      return $path.to.v$ ;
17  else
18      return $?$;
19  end
```

Algorithm 1: Determining whether $d(v) \le k$ or not.

Next we describe an unambiguous procedure (Algorithm 2) that computes c_k and Σ_k given c_{k-1} and Σ_{k-1}. Algorithm 2 uses Algorithm 1 as subroutine. Other than calls to Algorithm 1, this routine is deterministic, and so it follows that Algorithm 2 is also unambiguous.

We will argue that Algorithm 2 computes c_k and Σ_k. The subgraph consisting only of s ($d(x) \le 0$) is trivially min-unique and $c_0 = 1$ and $\Sigma_0 = 0$. Inductively, it is easy to see that

$$\begin{aligned} c_k &= c_{k-1} + \big|\{v \mid d(v) = k\}\big| \\ \Sigma_k &= \Sigma_{k-1} + k\big|\{v \mid d(v) = k\}\big| \end{aligned}$$

In addition, $d(v) = k$ if and only if there exists $(x, v) \in E$ such that $d(x) \le k - 1$ and

$\neg(d(v) \leq k-1)$. Both of these predicates can be computed using Algorithm 1. Combining these facts we see that Algorithm 2 computes c_k and Σ_k given c_{k-1} and Σ_{k-1}.

```
   Input: $(G, k, c_{k-1}, \Sigma_{k-1})$
   Output: $c_k, \Sigma_k$
1  Initialize $c_k \leftarrow c_{k-1}$ and $\Sigma_k \leftarrow \Sigma_{k-1}$ ;
2  foreach $v \in V$ do
3      if $\neg(d(v) \leq k-1)$ then
4          foreach $x$ such that $(x, v) \in E$ do
5              if $d(x) \leq k-1$ then
6                  Set $c_k \leftarrow c_k + 1$;
7                  Set $\Sigma_k \leftarrow \Sigma_k + k$ ;
8              end
9          end
10     end
11 end
12 return $c_k$ and $\Sigma_k$;
```

Algorithm 2: Computing c_k and Σ_k.

As a final step, we give the main routine that invokes Algorithm 2 to check for st connectivity in a min-unique graph. Since there is an st-path if and only if $d(t) \leq n$, it suffices to compute c_n and Σ_n and invoke Algorithm 1 on (G, t, n, c_n, Σ_n). This procedure is presented as Algorithm 3. To ensure that the algorithm runs in log-space, we do not store all intermediate values for c_k, Σ_k. Instead, only keep the most recently computed values and reuse space. As with Algorithm 2, this procedure is deterministic and so the entire routine is unambiguous. Thus, reachability in min-unique graphs can be decided in $\mathsf{UL} \cap \mathsf{coUL}$. $\square$

```
    Input: A directed graph $G$.
    Output: $true$ if there is a path from $s$ to $t$, $false$ otherwise.
1   Initialize $c_0 \leftarrow 1, \Sigma_0 \leftarrow 0, k \leftarrow 0$;
2   for $k = 1, \ldots, n$ do
3       Compute $c_k$ and $\Sigma_k$ by invoking Algorithm 2 on $(G, k, c_{k-1}, \Sigma_{k-1})$;
4   end
5   Invoke Algorithm 1 on $(G, t, n, c_n, \Sigma_n)$ and return its value ;
```

Algorithm 3: Determining if there exists a path from s to t in G.

Chapter 3

Green's Theorem and Isolation in Planar Graphs

In this chapter we study the problem of isolating directed paths and perfect matchings in planar graphs. We give an introduction to the problem of isolation and its applications in Section 3.1. In the same section we also introduce Green's Theorem, which is an important theorem in multi-variable calculus. In Section 3.2 we define and discuss the log-space computable weight function that achieves the desired isolation. In Section 3.3 we give a reduction from isolating a path to isolating a perfect matching. In Section 3.4 we give a piecewise straight line embedding of a planar graph. In Section 3.5 we show that our result implies certain upper bounds on graph reachability over planar and non-planar graphs.

3.1 Introduction

Green's Theorem is a fundamental result in multi-variable calculus due to 19th century British mathematician George Green. Here we give an application of Green's Theorem to a combinatorial problem, namely the isolation problems in planar graphs. As a consequence we get improved space upper bounds on planar restrictions of reachability and perfect matching problems. We also show an extension of our results to a class of non-planar graphs.

Green's theorem, stated below, relates a certain line integral over a closed curve on the plane to a related double integral over the region enclosed by this curve.

Theorem 3.1.1 (Green's Theorem)**.** *Let C be a closed, piecewise smooth, simple curve on the plane which is oriented counterclockwise. Let R_C be the region bounded by C. Let P and Q be functions of (x, y) defined on a region containing R_C and having continuous partial derivatives in the region. Then*

$$\oint_C (P\,dx + Q\,dy) = \iint_{R_C} \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dA.$$

This fundamental theorem and its generalizations (such as Stokes' Theorem) have deeply influenced the development of several areas of physics and mathematics. Strikingly, Green's Theorem also has a very immediate and elegant practical application in calculating the area of an arbitrary two-dimensional shape. The device known as planimeter, used to calculate the area of an arbitrary shape (such as a region in a map), is based on the following instantiation of Green's Theorem, which we also use in this chapter. If we substitute $Q(x, y) = x$ and $P(x, y) = 0$ in Green's Theorem we get the following theorem.

Theorem 3.1.2 (Area by line integrals)**.** *Let C be a closed, piecewise smooth, simple curve on the plane which is oriented counterclockwise. Let R_C be the region bounded by C. Then,*

$$Area(R_C) = \oint_C x\,dy$$

Refer to any standard text books on calculus (such as [Ste09]) to know more about Green's and other related theorems.

Recall the isolating lemma from Chapter 2. Recently simple log-space computable weight functions were prescribed that isolated directed paths and perfect matchings over grid graphs to yield new deterministic upper bounds [BTV09, DKR10]. Note that grid graphs are a restricted class of planar graphs where the graph completely lies on the two dimensional grid. It was not clear how to extend these weight functions to planar graphs. Here we settle this question.

Our results

Given a directed graph G with a planar embedding, we prescribe a skew-symmetric, log-space computable, polynomially bounded weight function w with the property that, with respect to w, the weight of any simple cycle in G is non-zero. We then use arguments identical to that in [BTV09] to show that such weight functions isolate directed paths - that is, with respect to such weight functions, between any pair of nodes if there is a path, then there is a unique minimum weight path. We also give an efficient construction of a weight function for the class of undirected bipartite planar graphs (based on the earlier weight function), which isolates a perfect matching in such graphs. Our weight function is based on the line integral on the right hand side of Theorem 3.1.2.

The weighting scheme that we prescribe works for any "nice" embedding of the graph on the plane. *Straight line embedding* is such an embedding for planar graphs.

Definition 3.1.1. (Straight line embedding) A straight line embedding of a planar graph G is an embedding where each vertex v in G, is given as a point, (x_v, y_v) on the coordinate axes, and an edge (u, v) is a line between points (x_u, y_u) and (x_v, y_v) so that no two lines intersect other except at the endpoints. Moreover, we will assume that the coordinates are integer points with values bounded by some polynomial in n.

Existence of such embeddings were known earlier [Fár48, dFPP90, Sch90]. For ease of presentation we will assume that the graph is presented as a straight line embedding.

Typically for algorithmic purposes planar graphs are presented in terms of a *combinatorial embedding*. Time efficient algorithms are known that can compute a straight line embedding of a planar graph [dFPP90, Sch90] from a combinatorial embedding. Unfortunately, these algorithms require linear space and at present we do not know how to get a space efficient implementation of them. In Section 3.4 we give a log-space algorithm that gives a *piecewise straight line embedding* of the given planar graph from a combinatorial embedding. This is the first log-space construction known to us, of a piecewise straight line embedding of a given planar graph and might be of independent interest. It will be very clear how the weight function for a straight line embedding can be extended to a piecewise straight line embedding also.

We then show that as a consequence of our isolation result, it follows that (i) deciding reachability in directed planar graphs is in unambiguous log-space [BTV09], and (ii) deciding whether a bipartite planar graph has a perfect matching can be decided in SPL [DKR10]. It is known that the problem of reachability and bipartite matching over planar graphs reduce to their counter parts in grid graphs [ABC$^+$09, DKR10] and hence the weight functions known for grid graphs suffice to derive upper bounds for planar versions of these problems. However, we feel that the application of Green's Theorem to the isolation problem gives it a new dimension and might yield potential strategies to solve the more general cases.

3.2 The weight function

Let $G = (V, E)$ be a graph with a straight line embedding. Let $e = (u, v)$ be a directed edge directed from u to v where u is identified with the point (x_u, y_u) and v is identified with (x_v, y_v). For such a directed edge, define a weight function w as follows (if e is piecewise straight, we calculate the integral over each piece and sum them up):

$$w(e) = 2 \times \oint_e x\, dy = (y_v - y_u)(x_v + x_u)$$

In order to calculate the second equality, we can use the parametric equation of the line segment which is given by $x(t) = (x_v - x_u)t + x_u$ and $y(t) = (y_v - y_u)t + y_u$ where $t \in [0, 1]$. Notice that if the coordinates of the vertices are polynomially bounded, this weight function is also polynomially bounded. For any cycle C in G, weight of C, $w(C)$ is defined as the sum of the weights of the edges in C.

An important property of this weight function is that it is *skew-symmetric*, that is, $w(u, v) = -w(v, u)$. We use this skew-symmetry property in our proofs. We will first show the following lemma which is crucial in proving that this weight function has the required isolation property.

Lemma 3.2.1. *Let G be a directed planar graph, given as a straight line embedding on the plane and let C be any directed simple cycle in G. Let R_C be the region enclosed by C. Then in log-space we can construct a polynomially bounded, skew-symmetric weight function w such that, $|w(C)| = 2 \times Area(R_c)$. In particular, $w(C)$ is non-zero.*

Proof. Let w be the weight function defined above. Note that by definition, w is skew symmetric and polynomially bounded. Let $C = (e_1, e_2, \ldots, e_l)$ be a directed cycle oriented counterclockwise. Then we have

$$\begin{aligned} w(C) &= \sum_i w(e_i) \\ &= 2 \times \sum_i \oint_{e_i} x\,dy \\ &= 2 \times \oint_C x\,dy \\ &= 2 \times \mathrm{A}rea(R_C) \end{aligned}$$

The last equality follows from Theorem 3.1.2. If C is oriented clockwise, we get that $w(C) = -2 \times \mathrm{A}rea(R_C)$. Hence the lemma. $\square$

3.3 Isolating paths and matchings in planar graphs

In this section we show how to isolate a directed path in a planar graph and a perfect matching in a bipartite planar graph, as a consequence of Lemma 3.2.1.

Theorem 3.3.1. *Let G be a planar directed graph with a straight line embedding. Then in log-space we can construct a weight function w, such that, for every pair of nodes u and v, if there is a directed path from u to v, then there is a* unique *path from u to v of minimum weight.*

Proof. Let w be the weight function from Lemma 3.2.1. Suppose there are u, v so that there are two u to v paths P_1 and P_2 of minimum weight. We will assume that the paths do not intersect on vertices other than the end points (otherwise we can find two vertices u' and v' along these paths that satisfies this property using a standard cut-and-paste argument and use these vertices instead). We have $w(P_1) = w(P_2)$. Now consider the graph G' which is same as G except that the path P_2 is reversed so that the set of edges (P_1, P_2^r) becomes a simple cycle in G' (P_2^r denotes the reversed path). Let C denote this cycle. Then $w(C) = w(P_1) + w(P_2^r) = w(P_1) - w(P_2) = 0$. The second equality holds because of the skew-symmetry of the weight function. This contradicts the fact that weight of a simple cycle is not equal to zero by Lemma 3.2.1. $\square$

Now we will consider isolation of perfect matchings in bipartite planar graphs.

Theorem 3.3.2. *Let G be a planar undirected bipartite graph. There exists a log-space computable weight function w', such that, if there is a perfect matching in G, then the minimum weight perfect matching in G is unique.*

Proof. First we define the log-space computable weight function w'. Since for matching we have undirected graphs, we need to give directions to the edges in order to assign weights. First we compute a bipartition of G. This can be achieved in log-space by Reingold's reachability algorithm (say using a universal exploration sequence) for undirected graphs [Rei08]. Thus given a vertex u, we can decide in log-space whether $u \in L$ or $u \in R$, where (L, R) is a bipartition of G. For any undirected edge $\{u, v\}$ so that $u \in L$ and $v \in R$, we first assign direction from u to v. Thus in the corresponding directed graph, denoted by $\vec{G}$, all the edges go from L to R. Then weight of an undirected edge e is $w'(e) = (y_v - y_u)(x_v + x_u)$ with respect to the above-mentioned direction.

Let w' be the weight function defined above. Suppose the theorem is not true and let M_1 and M_2 be two matchings so that $w'(M_1) = w'(M_2)$. Consider $M_1 \oplus M_2$, the symmetric difference of M_1 and M_1. This is nonempty and is a collection of simple alternating (between M_1 and M_2) cycles. Let C be one of the cycles. Let $C_1 = C \cap M_1$ and $C_2 = C \cap M_2$. Then we claim that $w'(C_1) = w'(C_2)$. Suppose $w'(C_1) < w'(C_2)$ then $(M_2 \setminus C_2) \cup C_1$ will be a matching of weight smaller than that of M_2. Let $\vec{C_1}$ and $\vec{C_2}$ be the corresponding set of directed edges. Now consider a directed planar graph $\vec{G'}$ which is same as $\vec{G}$ except that the directions of all the edges in C_2 is reversed. Thus edges of $\vec{C_1}, \vec{C_2}^r$ form a directed cycle $\vec{C}$ in $\vec{G'}$. But $w(\vec{C}) = w(\vec{C_1}) + w(\vec{C_2}^r) = w(\vec{C_1}) - w(\vec{C_2}) = 0$. This contradicts the fact that weight of a simple cycle is not equal to zero by Lemma 3.2.1. $\square$

3.3.1 A sufficient condition for isolating bipartite matching

Note that the above isolation theorems follow, using simple arguments, from a weight function w for directed graphs with the property that weight of any directed cycle is non-zero with respect to w. We state a general result that captures the essentials of the above argument for bipartite matching. A similar theorem holds for isolating directed paths also.

Definition 3.3.1. Given an undirected graph G, let $\overleftrightarrow{G}$ be the directed graph formed by replacing every undirected edge $\{u, v\}$ in G, with the directed edges (u, v) and (v, u). For a class of undirected graphs $\mathcal{G}$, let $\overleftrightarrow{\mathcal{G}}$ be the class of directed graphs $\overleftrightarrow{G}$, such that the undirected graph G is in $\mathcal{G}$.

Theorem 3.3.3. *Let $\mathcal{G}$ be a class of undirected bipartite graphs and let w be a polynomially bounded skew-symmetric, edge weight function defined for every graph $\overleftrightarrow{G} \in \overleftrightarrow{\mathcal{G}}$ such that for any cycle C in $\overleftrightarrow{G}$, $w(C) \neq 0$. Then given a graph $G \in \mathcal{G}$, we can construct a weight function w' in log-space, such that the minimum weight perfect matching in G is unique with respect to w'.*

Proof. Given G, use Reingold's undirected reachability algorithm [Rei08], to construct a bipartition of G, say L and R. Now orient the edges of G as follows to get the graph G': for every edge $e = \{u, v\}$ in G, where $u \in L$ and $v \in R$, replace e with the directed edge $e' = (u, v)$ and the edge $e'' = (v, u)$. By definition $G' \in \mathcal{G}$ and thus $w(G')$ is well defined. We now use w to define a weight on G. For every edge $e \in G$, let $w'(e) = w(e')$.

Now suppose G has two distinct minimum weight perfect matchings, M_1 and M_2, with respect to w'. Then the symmetric difference of M_1 and M_2 is a collection of disjoint, even length, simple cycles, where the edges of the cycle alternate between the matchings M_1 and M_2. Since M_1 and M_2 are distinct, there is at least one cycle. Let $C = (v_1, v_2, \ldots v_{2k}, v_1)$ be one such cycle. Let $e_i = (v_i, v_{(i+1) \bmod k})$ for $i \in [k]$. Without loss of generality assume, $v_1 \in L$ and the edge e_1 is in M_1. Therefore if i is odd (resp. even), then $e_i \in M_1$ (resp $e_i \in M_2$) and e'_i is directed from L to R (resp from R to L). Thus $w'(e_{2i-1}) = w(e'_{2i-1})$ and $w'(e_{2i}) = -w(e'_{2i})$ for $i \in [k]$, due to skew-symmetry of w.

The weight of the restriction of M_1 to C, $w'(M_1 \cap C) = \sum_{i=1}^{k} w'(e_{2i-1})$. Similarly $w'(M_2 \cap C) = \sum_{i=1}^{k} w'(e_{2i})$. Now,

$$\begin{aligned}
w'(M_1 \cap C) - w'(M_2 \cap C) &= \sum_{i=1}^{k} w'(e_{2i-1}) - \sum_{i=1}^{k} w'(e_{2i}) \\
&= \sum_{i=1}^{k} w(e'_{2i-1}) + \sum_{i=1}^{k} w(e'_{2i}) = \sum_{i=1}^{2k} w(e'_i) \\
&\neq 0.
\end{aligned}$$

Therefore either $M_1 \cap C$ or $M_2 \cap C$ has higher weight with respect to w'. Without loss of generality assume its M_2. Thus we get a perfect matching $M' = M_2 \setminus (M_2 \cap C) \cup (M_1 \cap C)$ in G of lesser weight, which is a contradiction. □

3.3.2 Remarks about the weight function

It is clear that there are many other weight functions that will work. In fact any "nice" solution to the differential equation $\left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) = 1$ will yield isolating weight functions. In particular, setting $P(x,y) = \frac{-y}{2}$ and $Q(x,y) = \frac{x}{2}$ to the left hand side of Green's theorem yields the weight function $w(e) = (x_u y_v - x_v y_u)$ which is isolating.

One can easily verify that the weight function we give here is a true extension of the following weight function prescribed in [BTV09] for isolating paths in grid graphs: east and west edges are given 0 weight, a north edge at $((i,j),(i,j+1))$ is given a weight i, and a south edge at $((i,j),(i,j-1))$ is given a weight $-i$. However, if we apply our theorem for the case of isolating matching in grid graphs, we get a different (slightly simpler) weight function than the one prescribed in [DKR10]. We believe that this chapter better explains the reason behind why these weight functions work.

3.4 Piecewise straight line embedding of a planar graph

In this section we give a log-space algorithm to compute a piecewise straight line embedding of a planar graph. All graphs considered in this section are undirected, unless otherwise specified.

Definition 3.4.1. For $p_i \in \mathbb{R}^2$, $(p_1, \ldots, p_{k+1})$ is said to be a piecewise straight line segment, if there is a straight line segment connecting p_i with p_{i+1} for every $i \in [k]$.

Definition 3.4.2. For $k \geq 1$, a k-piecewise straight line embedding of a graph $G = (V,E)$ is a function $f : V \to \mathbb{R}^2$ and a collection of $(k-1)$ functions $g_i : E \to \mathbb{R}^2$ for $i \in [k-1]$, such that every edge $e = (u, v) \in E$ is a piecewise straight line segment, $(f(u), g_1(e), \ldots, g_{l_e-1}(e), f(v))$ for some $l_e \leq k$ and no two embedded edges intersect except possibly at the end points.

Theorem 3.4.1. *Given a combinatorial embedding of a planar graph G, there is a log-space algorithm that computes a* 4*-piecewise straight line embedding of G.*

We will give an embedding of G in the first quadrant of the coordinate plane. We first use Reingold's undirected reachability algorithm [Rei08] to compute a spanning tree T of G rooted at r. Now from G we create a new graph G_T by "cutting" every non-tree edge into two edges. Thus G' would be a tree. We then give a straight line embedding of G' in the first quadrant of the two dimensional Cartesian coordinate system (we shall just refer to it as the coordinate system from now on), such that the leaf end of every "split edge" lies on a circle centered at the origin and containing G'. Next we reconnect the split edges appropriately to avoid intersections. Below we give a more formal description of the algorithm.

We create $G_T = (V_T, E_T)$ from G as follows. For each edge $e = (u, v)$ in $E \setminus T$, we introduce two new vertices w_e^u and w_e^v. Now replace e with the edges (u, w_e^u) and (v, w_e^v). Denote the newly introduced set of vertices and edges as V_T' and E_T'. Thus $V_T = V \cup V_T'$ and $E_T = T \cup E_T'$. Note that G_T is a tree and every vertex in V_T' is a leaf. We shall think of G_T as a tree rooted at r as well. Next we define the height function, h for every vertex in G_T. For the root node $h(r) = 0$. For every vertex $v \neq r$ in V, $h(v) = h(p) + 1$, where p is the parent node of v in G_T and for every vertex v in V_T', $h(v) = \max\{h(v) : v \in V\}$. Define $h(G_T) = \max\{h(v) : v \in V_T\}$. For a vertex v, let $A(v)$ be the set of leaves u in G_T, such that u is not present in the subtree rooted at v and the path from u to r lies to the left of the path from v to r. Let L be the set of leaves in G_T. Then $\theta(v) = \frac{|A(v)|}{|L|}\frac{\pi}{2}$.

The coordinates of a vertex v, in our embedding would be

$$F(v) = (h(v)\cos(\theta(v)), h(v)\sin(\theta(v))).$$

For every edge $e = (u, v) \in E_T$ draw a straight line segment between $F(u)$ and $F(v)$ to represent the edge. We shall denote the embedding of this edge (line segment) by $F(e)$. Note that, since the function h is defined to be equal to the maximum over all values of h, for vertices in the set V'_T, therefore the vertices in V'_T lie on the concentric circle of radius $h(G_T)$, which(the circle) by definition contains the entire embedded graph G_T.

Next we compare the sets $A(u)$ and $A(v)$ for two vertices u and v.

Lemma 3.4.2. *Let u and v be two distinct vertices in G. (a) If u is an ancestor of v then $A(u) \subseteq A(v)$. (b) If u lies to the left of v, then $A(u) \subsetneq A(v)$. (c) For any descendent w of u, $A(w) \subsetneq A(v)$.*

Proof. (a) follows from the observation that any vertex to the left of a node also lies to the left of any of its descendent. Similarly, if u lies to the left of v, then any node to the left of u also lies to the left of v. This proves (b). (c) follows since any descendent of u lies to the left of v. □

In Lemma 3.4.3 we show that distinct vertices get mapped to distinct coordinates by F. In Lemma 3.4.4 we prove that no two edges of G_T intersect at an intermediate point.

Lemma 3.4.3. *Let u, v be two vertices in G_T. Then $u = v$ if and only if $F(u) = F(v)$.*

Proof. Let u and v be two distinct vertices. If $h(u) \neq h(v)$ then $F(u) \neq F(v)$ since they lie in different concentric cycles around the origin by definition of F. If $h(u) = h(v)$, then it follows from Lemma 3.4.2. □

Lemma 3.4.4. *Let e_1 and e_2 be two edges in G_T. Then $F(e_1)$ and $F(e_2)$ do not intersect except possibly at end points.*

Proof. Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ such that u_i is the parent of v_i. If $u_1 = u_2$ then since $v_1 \neq v_2$, e_1 and e_2 do not intersect non-trivially. Also if v_1 is an ancestor of u_2 then v_1 is an ancestor of v_2 as well and therefore they cannot intersect since they lie in concentric circles of different length around the origin.

We now consider the case when u_1 is not an ancestor or descendent of u_2. Without loss of generality assume u_1 is to the left of u_2, which implies that $A(u_1) \subsetneq A(u_2)$. From Lemma 3.4.2 we get $\theta(u_1) \leq \theta(v_1) < \theta(u_2) \leq \theta(v_2)$. Therefore the line segments $(F(u_1), F(v_1))$ and $(F(u_2), F(v_2))$ do not intersect. $\square$

Next we rejoin the split edges to get back the original graph. After joining, a split edge would be embedded as a piecewise straight line as we describe below. Recall that precisely the non-tree edges in G are the edges that were split.

Suppose $e = (u, v)$ was a non-tree edge in G. Then e was replaced by the edges (u, w_e^u) and (v, w_e^v) by the introduction of two new vertices w_e^u and w_e^v. We remove the vertices w_e^u and w_e^v and the edges (u, w_e^u) and (v, w_e^v) and draw the piecewise straight line segment:
$(F(u), F(w_e^u), \max\{F(w_e^u), F(w_e^v)\}, F(w_e^v), F(v))$ to represent edge e. (where the $\max$ function is defined as $\max\{(a_1, b_1), (a_2, b_2)\} \triangleq (\max\{a_1, a_2\}, \max\{b_1, b_2\})$) We shall denote the embedding of this edge (piecewise line segment) by $F(e)$. In Lemma 3.4.5 we show that non-tree edges do not intersect non-trivially, to complete the proof of Theorem 3.4.1.

Lemma 3.4.5. *Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be two non-tree edges in G. Then the edges $F(e_1)$ and $F(e_2)$ do not intersect non-trivially.*

Proof. We only need to show that the piecewise line segments $(F(w_{e_1}^{u_1}), \max\{F(w_{e_1}^{u_1}), F(w_{e_1}^{v_1})\}, F(w_{e_1}^{v_1}))$ and $(F(w_{e_2}^{u_2}), \max\{F(w_{e_2}^{u_2}), F(w_{e_2}^{v_2})\}, F(w_{e_2}^{v_2}))$ do not intersect.

Case 1 *(One end point of e_1 and e_2 is common)*: Without loss of generality assume $u_1 = u_2 = u(say)$ and $\theta(v_1) \leq \theta(v_2)$. Thus in G_T, $w_{e_2}^u$ lies to the left of $w_{e_1}^u$ which implies $\theta(w_{e_2}^u) < \theta(w_{e_1}^u)$ by Lemma 3.4.2. Also $\theta(w_{e_1}^{v_1}) < \theta(w_{e_2}^{v_2})$ since $w_{e_1}^{v_1}$ and $w_{e_2}^{v_2}$ are children of v_1 and v_2 respectively. This shows the Lemma for Case 1.

Case 2 *(All end points of e_1 and e_2 are distinct)*: Without loss of generality assume $\theta(u_1) \leq \theta(u_2)$ and $\theta(u_i) \leq \theta(v_i)$ for $i \in \{1, 2\}$. Since e_1 and e_2 cannot intersect, therefore if $\theta(v_1) \geq \theta(u_2)$, then $\theta(u_1) \leq \theta(v_2) \leq \theta(v_1)$, and if $\theta(v_1) < \theta(u_2)$, then either $\theta(v_2) \geq \theta(v_1)$. This implies that either

$\theta(w^{u_1}_{e_1}) < \theta(w^{u_2}_{e_2}) < \theta(w^{v_2}_{e_2}) < \theta(w^{v_1}_{e_1})$ or $\theta(w^{u_1}_{e_1}) < \theta(w^{v_1}_{e_1}) < \theta(w^{u_2}_{e_2}) < \theta(w^{v_2}_{e_2})$. Hence the Lemma holds for this case too. □

Note that the coordinates that we assign are real numbers and need not be computable in log-space. To take of this problem we can "inflate" the entire mapping by multiplying each coordinate with a suitable large number (say $|V|^5$) and then taking the floor of each point to get an integral embedding.

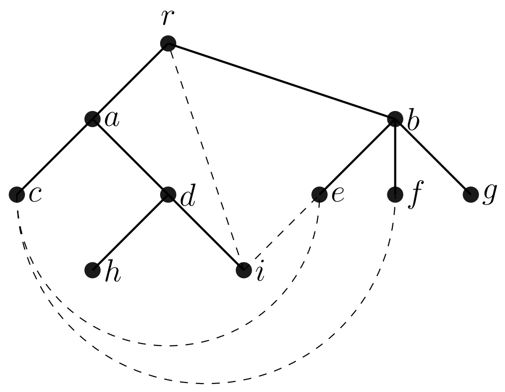


Figure 3.1: Example of a graph G containing a spanning tree T rooted at r (the tree and non-tree are represented by solid and dashed edges respectively).

3.5 Certain upper bounds on graph reachability

In this section we prove an improved upper bound on the complexity of planar reachability and reachability in a certain class of non-planar graphs.

3.5.1 Planar reachability

A natural and important restriction of the reachability problem is when the graphs involved are planar, which we denote by PLANARREACH. The complexity of this problem is not yet settled satisfactorily. The best known upper bound in terms of space complexity is NL. Though it is hard

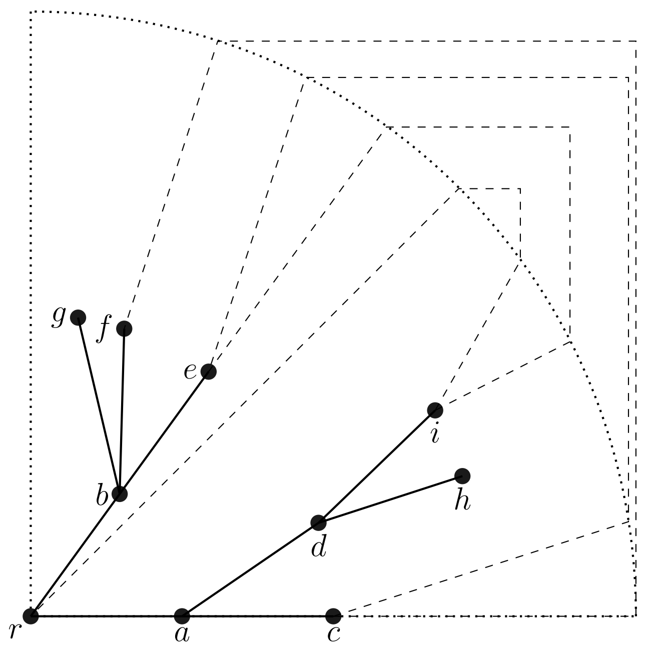


Figure 3.2: Piecewise straight line embedding of G.

for L [Ete97], it is not known whether it is complete for NL. Recently there has been progress in understanding the complexity of PLANARREACH. [ADR05] showed that PLANARREACH reduces to the reachability problem for grid graphs. In the same paper, they also gave a log-space reduction from PLANARREACH to its complement.

In this chapter we make further progress in understanding the space complexity of PLANARREACH. From our planar isolation result, it follows that PLANARREACH can be decided in *unambiguous log-space*.

Theorem 3.5.1. PLANARREACH $\in \mathsf{UL} \cap \mathsf{coUL}$.

Proof. Follows from Theorem 3.3.1 and Theorem 2.3.5 (a proof is given in Chapter 2). □

3.5.2 Extension to a class of non-planar graphs

In this section we present an extension of our main result to a certain class of non-planar graphs.

Let $\mathcal{G}$ be a class of graphs in which reachability can be decided in complexity class $\mathcal{C}$. Let $G = (V, E) \in \mathcal{G}$. Let $G' = (V, E \cup E')$ be the graph G with an additional set of edges, E', added to it. We refer to G as the *main* graph and G' the *augmented* graph. We will show that if $|E'|$ is not too large, then reachability for the augmented graph can be decided in $\mathsf{L}^{\mathcal{C}}$.

Theorem 3.5.2. *Let $G' = (V, E \cup E')$ be a graph such that reachability in $G = (V, E)$ can be decided in $\mathcal{C}$. Then if $|E'| = \mathcal{O}(2^{\sqrt{\log n}})$ then reachability in G' can be decided in $\mathsf{L}^{\mathcal{C}}$.*

Proof. The idea is to reduce reachability in G' to reachability in a smaller graph using a $\mathsf{UL} \cap \mathsf{coUL}$ oracle. Construct a graph whose vertices are labeled by edges in E' and there is directed edge from the vertex (u_1, u_2) to (v_1, v_2) in this graph if there is a path in G from u_2 to v_1. Now since this new graph is only of size $\mathcal{O}(2^{\sqrt{\log n}})$ we can solve reachability in this graph deterministically in log-space using Savitch's theorem.

Formally, let $\mathsf{isPath}(x, y)$ be a boolean predicate that is true if there is a directed path $p : x \rightsquigarrow y$ in the main graph $G = (V, E)$ (that is, there is a path $x \rightsquigarrow y$ that does not use auxiliary edges). $\mathsf{isPath}(x, y)$ is computable in $\mathcal{C}$. Also, let $a_1 = (u_1, v_1), a_2 = (u_2, v_2), \ldots a_m = (u_m, v_m)$ be the auxiliary edges (thus, $|E'| = m$).

We construct a new graph $\tilde{G} = (\tilde{V}, \tilde{E})$ as follows. Let

$$\begin{aligned} \tilde{V} &= \{v_{e_i} | e_i \in E'\} \cup \{\tilde{s}, \tilde{t}\} \\ \tilde{E} &= E_1 \cup E_2 \cup E_3 \end{aligned}$$

where

$$\begin{aligned} E_1 &= \{(\tilde{s}, y) \quad | \; y = (u_j, v_j) \in E' \text{ and } \mathsf{isPath}(s, u_j) \text{ is true}\}, \\ E_2 &= \{(x, y) \quad | \; x = (u_i, v_i), y = (u_j, v_j) \in E' \text{ and } \mathsf{isPath}(v_i, u_j) \text{ is true}\}, \\ E_3 &= \{(x, \tilde{t}) \quad | \; x = (u_i, v_i) \in E' \text{ and } \mathsf{isPath}(u_j, t) \text{ is true}\} \end{aligned}$$

Connectivity from $\tilde{s}$ to $\tilde{t}$ in $\tilde{G}$ can now be accomplished via application of Savitch's Theorem, which requires $\mathcal{O}(\log^2 m)$ space (recall that the size of $\tilde{G}$ is $m + 2$). For $m \leq \mathcal{O}(2^{\sqrt{\log n}})$ this simulation runs in space $\mathcal{O}(\log n)$.

Now it follows from the definition of $\tilde{G}$ that there is a path from s to t in G' if and only if there is a path from $\tilde{s}$ to $\tilde{t}$ in $\tilde{G}$. □

The statement of Theorem 3.5.2 is intentionally general. It is motivated by the possibility of extending our reachability result to non-planar graphs which may have one or more *crossing edges*. In particular, if we are given an embedding of a graphs partitioned into a main graph that is planar and an auxiliary set of crossing edges, as long as there are not many crossing edges, then we can solve st-connectivity in this non-planar graph in $\mathsf{UL} \cap \mathsf{coUL}$.

Corollary 3.5.3. *Let $\mathcal{G}$ be a class of directed graphs $G = (V, E \cup E')$ such that (V, E) is planar and $|E'| \leq \mathcal{O}(2^{\sqrt{\log n}})$ where $n = |V|$. Then st-connectivity for any graph in $\mathcal{G}$ can be decided in* $\mathsf{UL} \cap \mathsf{coUL}$.

Proof. Let G be a planar graph with auxiliary edge set being the crossing edges. We will make reachability queries to the main, planar graph, which can be done in $\mathsf{UL} \cap \mathsf{coUL}$ by Theorem 3.5.1. However, since $\mathsf{L}^{\mathsf{UL} \cap \mathsf{coUL}} = \mathsf{UL} \cap \mathsf{coUL}$, the corollary follows. □

Note that if we were able to extend this result to graphs with n^ϵ auxiliary crossing edges for any $\epsilon > 0$, then it would show that $\mathsf{NL} = \mathsf{UL}$.

3.6 Conclusion

In this chapter we established an new upper bound on the planar reachability problem. The question is can we extend our results to reachability in general graphs, using the tool of multi-variable calculus. We believe that this approach has a lot of promise in yielding positive results in future.

Chapter 4

Isolation in Bounded Genus Graphs

In this chapter we extend our isolation results from Chapter 3, to graphs over bounded genus surfaces. In Section 4.1 we give an introduction of the problem. In Section 4.2 we give the necessary definitions and state results from earlier work, that we use in this chapter. In Section 4.3 we give matching preserving, log-space reductions from a combinatorial embedding of the graph on a surface of genus g, to an embedding on the polygonal schema of the surface. In Section 4.4 we state and prove our main result on isolation in bounded genus graphs, assuming an embedding on the polygonal schema. In Section 4.5 we reduce the non-orientable case to the orientable one. In Section 4.6 we state new complexity theoretic upper bounds that we obtain.

4.1 Introduction

Genus of a surface is a natural topological generalization of planarity. Intuitively a surface is said to have genus g if it has g 'holes' and locally at any point the surface is similar to a planar surface. Therefore a planar surface is one, that has genus zero. A graph is said to have genus g if the minimum genus surface on which the graph can be embedded without crossing edges, has genus g. We formalize these definitions in Section 4.2. In this chapter we extend the deterministic isolation technique of Chapter 3 to isolate a minimum weight perfect matching in bipartite graphs embedded on bounded genus surfaces. This is more interesting in light of the fact that even the existence of a polynomially bounded weight function for bounded genus graphs, that isolates a minimum weight perfect matching, was not known earlier. As a future direction it would be interesting to consider the general bipartite graph $K_{n,n}$, and prove the existence of a polynomially bounded weight function that isolates a minimum weight perfect matching in this case.

4.1.1 Our contribution

Let G be a bipartite graph and let $\overleftrightarrow{G}$ be the directed graph obtained by replacing every undirected edge $\{u, v\}$ of G with the directed edges (u, v) and (v, u). The main technical contribution of this chapter can then be stated (semi-formally) as follows.

Main Technical Result

Given an embedding of a undirected bipartite bounded genus graph G, there is a log-space matching preserving reduction f, and a log-space computable, polynomially bounded, skew-symmetric weight function w for the class of directed graphs, so that the weight of any simple cycle in $\overleftrightarrow{f(G)}$ with respect to w is non-zero.

We use this result to establish (using known techniques) the following new upper bounds. Refer to Chapter 2 for definitions.

New Upper Bounds

For bipartite graphs, combinatorially embedded on surfaces of bounded genus the problems PM-DECISION and PM-UNIQUE are in SPL, and the problem PM-CONSTRUCT is in $\mathsf{FL}^{\mathsf{SPL}}$.

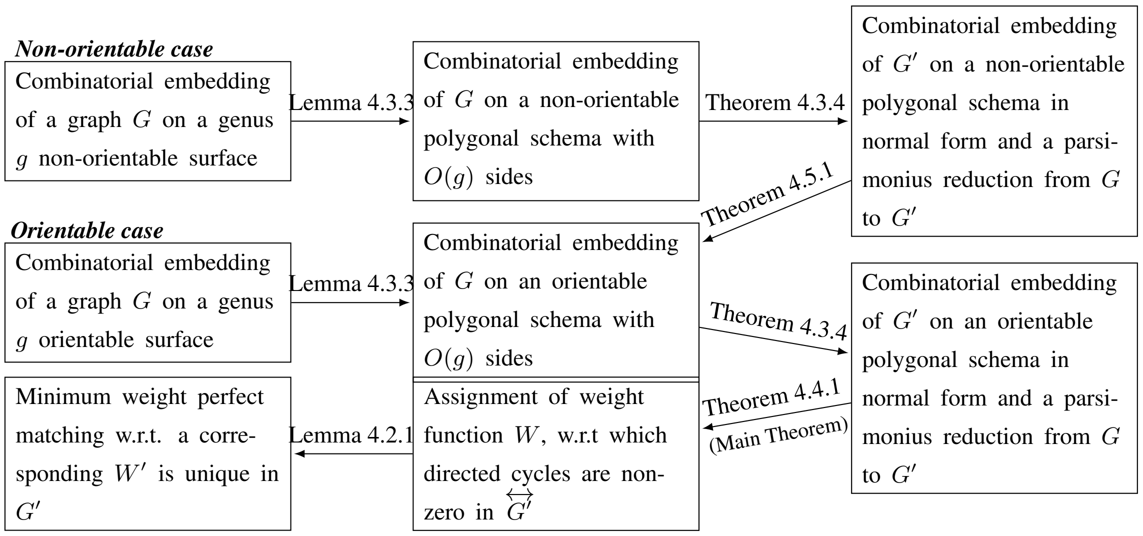
The techniques that we use in this chapter can also be used to isolate directed paths in graphs on bounded genus surfaces. This shows that the reachability problem for this class of graphs can be decided in the unambiguous class UL, extending the results of [BTV09]. But this upper bound is already known since recently Kynčl and Vyskočil show that reachability for bounded genus graphs log-space reduces to reachability in planar graphs [KV10].

Matching problems over graphs of low genus have been of interest to researchers, mainly from a parallel complexity viewpoint. The matching problems that we consider in this chapter are known to be in NC. In particular in [MV00], the authors present an NC^2 algorithm for computing a perfect matching for bipartite graphs on surfaces of $O(\log n)$ genus (readers can also find an account of known parallel complexity upper bounds for matching problems over various classes of graphs in their paper). However, the space complexity of matching problems for graphs of low genus has not been investigated before. In this chapter we take a step in this direction.

Proof Outline

We assume that the graph G is presented as a combinatorial embedding on a surface (orientable or non-orientable) of genus g, where g is a constant. This is a standard assumption when dealing with graphs on surfaces, since it is NP-complete to check whether a graph has genus at most g [Tho89]. We first give a sequence of two reductions to get, from G, a graph G' with an embedding on a genus g 'polygonal schema in normal form'. These two reductions work for both orientable and non-orientable cases. At this point we take care of the non-orientable case by reducing it to the orientable case. These reductions are matching preserving, bipartiteness preserving and computable in log-space. Finally, for $\overleftrightarrow{G'}$ (the directed version of G'), we prescribe a set of $4g+1$ weight functions, $\mathcal{W} = \{w_i\}_{1 \leq i \leq 4g+1}$, so that for any cycle C in $\overleftrightarrow{G'}$, there is a weight function $w_i \in \mathcal{W}$ with respect to which the weight of C is non-zero. Since g is constant, we can take a linear combination of the elements in $\mathcal{W}$, for example $\sum_{w_i \in \mathcal{W}} w_i \times (n^c)^i$ (where n is the number of vertices) for some fixed constant c (say $c = 4$), to get a single weight function with respect which the weight of any cycle is non-zero.

The intuition behind these weight functions is as follows (for some of the definitions, refer to



Non-orientable case
Combinatorial embedding of a graph G on a genus g non-orientable surface
Lemma 4.3.3
Combinatorial embedding of G on a non-orientable polygonal schema with $O(g)$ sides
Theorem 4.3.4
Combinatorial embedding of G' on a non-orientable polygonal schema in normal form and a parsimonius reduction from G to G'
Theorem 4.5.1
Orientable case
Combinatorial embedding of a graph G on a genus g orientable surface
Lemma 4.3.3
Combinatorial embedding of G on an orientable polygonal schema with $O(g)$ sides
Theorem 4.3.4
Combinatorial embedding of G' on an orientable polygonal schema in normal form and a parsimonius reduction from G to G'
Theorem 4.4.1
(Main Theorem)
Assignment of weight function W, w.r.t which directed cycles are non-zero in $\overleftrightarrow{G'}$
Lemma 4.2.1
Minimum weight perfect matching w.r.t. a corresponding W' is unique in G'

Figure 4.1: Outline of the steps. Note that all reductions are matching preserving and log-space computable.

later sections). The set $\mathcal{W}$ is a disjoint union $\mathcal{W}_1 \cup \mathcal{W}_2 \cup \{w\}$ of the sets of weight functions $\mathcal{W}_1$, $\mathcal{W}_2$, and $\{w\}$. Consider a graph G embedded on a fundamental polygon with $2g$ sides. There are two types of cycles in G: *surface separating* and *surface non-separating*. A basic theorem from algebraic topology implies that a surface non-separating cycle will intersect at least one of the sides of the polygon an odd number of times. This leads to $2g$ weight functions in $\mathcal{W}_1$ to take care of all the surface non-separating cycles. There are two types of surface separating cycles: (a) ones which completely lie inside the polygon and (b) the ones which cross some boundary. Cycles of type (a) behave exactly like cycles in the plane so the weight function w designed for planar graphs works (from [DKR10, TV10]). For dealing with cycles of type (b), we first prove that if such a cycle intersects a boundary, it should alternate between 'coming in' and 'going out'. This leads to $2g$ weight functions in $\mathcal{W}_2$ which handle all type (b) cycles.

Figure 4.1.1 gives a pictorial view of the components involved in the proof of our main technical result.

4.2 Preliminaries

In this section we introduce the necessary terminology from algebraic topology that we use in the rest of this chapter. For a more comprehensive understanding of this topic, refer to any standard algebraic topology book such as [Mas91].

4.2.1 Topological graph theory

A *2-manifold* is a topological space such that every point has an open neighborhood homeomorphic to $\mathbb{R}^2$ and two distinct points have disjoint neighborhoods. A 2-manifold is often called a *surface*. The *genus* of a surface Γ is the maximum number g, if there are g cycles $C_1, C_2, \ldots, C_g$ on Γ, such that $C_i \cap C_j = \emptyset$ for all i, j and $\Gamma \setminus (C_1 \cup C_2 \cup \ldots \cup C_g)$ is connected. A surface is called *orientable* if it has two distinct sides, else it is called *non-orientable*. A cycle C in Γ is said to be *non-separating* if there exists a path between any two points in $\Gamma \setminus C$, else it is called *separating*.

A *polygonal schema* of a surface Γ, is a polygon with $2g'$ directed sides, such that the sides of the polygon are partitioned into g' classes, each class containing exactly two sides and glueing the two sides of each equivalence class gives the surface Γ (upto homeomorphism). A side in the ith equivalence class is labelled σ_i or $\bar{\sigma}_i$ depending on whether it is directed clockwise or anti-clockwise respectively. The *partner* of a side σ is the other side in its equivalence class. By an abuse of notation, we shall sometimes refer to the symbol of a side's partner, as the partner of the symbol. Frequently we will denote a polygonal schema as a linear ordering of its sides moving in a clockwise direction, denoted by X. For a polygonal schema X, we shall refer to any polygonal schema which is a cyclic permutation, or a reversal of the symbols, or a complementation (σ mapped to $\bar{\sigma}$ and vice versa) of the symbols, as being the same as X. A polygonal schema is called orientable (resp. non-orientable) if the corresponding surface is orientable (resp. non-orientable).

Definition 4.2.1. An orientable polygonal schema is said to be in *normal form* if it is in one of the following forms:

$$\sigma_1\tau_1\bar{\sigma_1}\bar{\tau_1}\sigma_2\tau_2\bar{\sigma_2}\bar{\tau_2}\ldots\sigma_m\tau_m\bar{\sigma_m}\bar{\tau_m} \quad (4.2.1)$$

$$\sigma\bar{\sigma} \quad (4.2.2)$$

A non-orientable polygonal schema is said to be in normal form if it is of one of the following forms:

$$\sigma\sigma X \quad (4.2.3)$$

$$\sigma\tau\bar{\sigma}\tau X \quad (4.2.4)$$

where, X is a string representing an orientable schema in normal form (i.e. like Form 4.2.1 or 4.2.2 above) or possibly an empty string.

We denote the polygonal schema in the normal form of a surface Γ as $\Lambda(\Gamma)$. We will refer to two orientable symbols σ, τ which form the following contiguous substring: $\sigma\tau\bar{\sigma}\bar{\tau}$ as being clustered together while a non-orientable symbol σ which occurs like $\sigma\sigma$ as a contiguous subtring is said to form a *pair*. Thus, in the first and third normal forms above all symbols are clustered. The first normal form represents a connected sum of torii and the third of a projective plane and torii. In the fourth normal form all but one of the orientable symbols are clustered while the only non-orientable symbol is sort of clustered with the other orientable symbol. This form represents a connected sum of a Klein Bottle and torii. The second normal form represents a sphere.

We next introduce the concept of $\mathbb{Z}_2$-homology. Given a 2-manifold Γ, a *1-cycle* is a closed curve in Γ. The set of 1-cycles forms an Abelian group, denoted as $\mathcal{C}_1(\Gamma)$, under the *symmetric difference* operation, Δ. Two 1-cycles C_1, C_2 are said to be homologically equivalent if $C_1 \Delta C_2$ forms the boundary of some region in Γ. Observe that this is an equivalence relation. Then the *first homology group* of Γ, $H_1(\Gamma)$, is the set of equivalence classes of 1-cycles. In other words, if $\mathcal{B}_1(\Gamma)$ is defined to be the subset of $\mathcal{C}_1(\Gamma)$ that are homologically equivalent to the empty set, then $H_1(\Gamma) = \mathcal{C}_1(\Gamma)/\mathcal{B}_1(\Gamma)$. If Γ is a genus g surface then $H_1(\Gamma)$ is generated by a system of $2g$ 1-cycles, having only one point in common, and whose complement is homeomorphic to a topological disk. Such a disk is also referred to as the *fundamental polygon* of Γ.

An undirected graph G is said to be embedded on a surface Γ if it can be drawn on Γ so that no two edges cross. We assume that the graph is given with a *combinatorial embedding* on a surface of bounded genus. Refer to the book by Mohar and Thomassen [MT01] for details. The genus of a graph G is the minimum number g such that G has an embedding on a surface of genus g (an embedding where every face of G is homeomorphic to a disc). Such an embedding is also called a *2-cell embedding*. A genus g graph is said to be orientable (non-orientable) if the surface is orientable (non-orientable).

Definition 4.2.2. The *polygonal schema of a graph* G is a combinatorial embedding given on the polygonal schema of some surface Γ together with the ordered set of vertices on each side of the polygon. Formally it is a tuple $(\phi, \mathcal{S})$, where ϕ is a cyclic ordering of the edges around a vertex (also known as the *rotation system* of G) and $\mathcal{S} = (S_1, S_2, \ldots, S_{2g})$ is the cyclic ordering of the directed sides of the polygon. Each S_i is an ordered sequence of the vertices, from the tail to the head of the side S_i. Moreover every S_i is paired with some other side, say S'_i in $\mathcal{S}$, such that the jth vertex of S_i (say from the tail of S_i) is the same as the jth vertex of S'_i (from the tail of S'_i).

In the following definition we formalize the class of such graphs.

Definition 4.2.3. We define BddGenBip to be the class of bounded genus, orientable, bipartite graphs given together with an embedding given on the polygonal schema in normal form of the surface in which the graph has an embedding. Moreover, for every graph in this class, no edge has both its end points on the boundary of the polygon.

We also define the directed version of the above class of graphs as follows:

Definition 4.2.4. Given G, define $\overleftrightarrow{G}$ to be the directed graph formed by replacing every edge $\{u, v\}$ in G with the directed edges (u, v) and (v, u). Let DIRBDDGENBIP be the class of directed graphs $\overleftrightarrow{G}$, such that G is in BDDGENBIP.

4.2.2 Necessary prior results

Lemma 4.2.1 is an adaptation of Theorem 3.3.2 from Chapter 3 for the case of bounded genus bipartite graphs.

Lemma 4.2.1. *Let w be a logarithmic space bounded, skew-symmetric edge weight function defined for the class of graphs* DIRBDDGENBIP*, such that for any graph $G \in$ DIRBDDGENBIP and any cycle $C \in G$, $w(C) \neq 0$. Then in log-space we can construct an edge weight function w' for the class of graphs* BDDGENBIP*, such that for any graph $H \in$ BDDGENBIP the minimum weight perfect matching in H is unique with respect to w'.*

Proof. Let H be a graph in BDDGENBIP. By definition, the graph $\overleftrightarrow{H}$ is in DIRBDDGENBIP and hence there exists a log-space computable weight function w with respect to which every cycle in $\overleftrightarrow{H}$ has non-zero weight. Using Reingold's reachability algorithm [Rei08], construct a bipartition (L, R) of H. Now for an edge $e = \{u, v\}$ in H such that $u \in L$ and $v \in R$, set $w'(e) = w(\overrightarrow{e})$ where $\overrightarrow{e}$ is the directed edge (u, v) in $\overleftrightarrow{H}$.

Now suppose H has two distinct minimum weight perfect matchings, M_1 and M_2, with respect to w'. Then the symmetric difference of M_1 and M_2 is a collection of disjoint, even length, simple cycles, where the edges of the cycle alternate between the matchings M_1 and M_2. Since M_1 and M_2 are distinct, there is at least one such cycle. Let $C = (v_1, v_2, \ldots v_{2k}, v_1)$ be such a cycle. Let $e_i = (v_i, v_{(i+1) \mod k})$ for $i \in [k]$. Without loss of generality assume, $v_1 \in L$ and the edge e_1 is in M_1. Therefore if i is odd (resp. even), then $e_i \in M_1$ (resp $e_i \in M_2$) and $\overrightarrow{e_i}$ is directed from L to R (resp from R to L). Thus $w'(e_{2i-1}) = w(\overrightarrow{e_{2i-1}})$ and $w'(e_{2i}) = -w(\overrightarrow{e_{2i}})$ for $i \in [k]$, due to skew symmetry of w.

The weight of the restriction of M_1 to C, $w'(M_1 \cap C) = \sum_{i=1}^{k} w'(e_{2i-1})$. Similarly $w'(M_2 \cap C) = \sum_{i=1}^{k} w'(e_{2i})$. Now,

$$\begin{aligned} w'(M_1 \cap C) - w'(M_2 \cap C) &= \sum_{i=1}^{k} w'(e_{2i-1}) - \sum_{i=1}^{k} w'(e_{2i}) \\ &= \sum_{i=1}^{k} w(\overrightarrow{e_{2i-1}}) + \sum_{i=1}^{k} w(\overrightarrow{e_{2i}}) = \sum_{i=1}^{2k} w(\overrightarrow{e_i}) \\ &\neq 0. \end{aligned}$$

Therefore either $M_1 \cap C$ or $M_2 \cap C$ has higher weight with respect to w'. Without loss of generality assume its M_2. Thus we get a perfect matching $M' = M_2 \setminus (M_2 \cap C) \cup (M_1 \cap C)$ in H of lesser weight, which is a contradiction. □

Lemma 4.2.2 ([ARZ99]). *For any weighted graph G assume that the minimum weight perfect matching in G is unique and also for any subset of edges $E' \subseteq E$, the minimum weight perfect matching in $G \setminus E'$ is also unique. Then deciding if G has a perfect matching is in* SPL. *Moreover, computing the perfect matching (in case it exists) is in* $\mathsf{FL}^{\mathsf{SPL}}$.

Sketch of proof. Let w_{max} and w_{min} be the maximum and minimum possible weights respectively, that an edge in G can get. Then any perfect matching in G will have a weight from the set $W = \{k : k \in \mathbb{Z}, n \cdot w_{min} \leq k \leq n \cdot w_{max}\}$. Similar to [ARZ99], there exists a GapL function f, such that for some value of $k \in W$, $|f(G,k)| = 1$ if G has a perfect matching of weight k, else $f(G,k) = 0$ for all values of k. Note that in [ARZ99] the authors actually give a GapL/poly function since the weight function for the graphs (which are unweighted to begin with) are required as an advice in their GapL machine. Here we consider weighted graphs, thus eliminating the need for any advice. Now consider the function

$$g(G) = 1 - \prod_{k} \left(1 - (f(G,k))^2\right).$$

By definition, $g(G) = 1$ if G has a perfect matching, else it is 0.

To compute a perfect matching in G, we will construct a log-space transducer that makes several queries to the function f defined above. For a graph G' having a unique minimum weight perfect matching (say M'), the weight of M' can be computed by iteratively querying the function $f(G', k)$ for values of $k \in W$ in an increasing order, starting from $n \cdot w_{min}$. The value k, for which the function outputs a non-zero value for the first time, is the weight of M'. We denote this weight by $w_{G'}$. First compute w_G. For an e in G, define the graph $G^{-e} = G \setminus \{e\}$. Now compute $w_{G^{-e}}$ for every edge e in G. Output the edges e for which $w_{G^{-e}} > w_G$. The set of outputted edges comprise a perfect matching (in fact the minimum weight perfect matching) because deleting an edge in this set had increased the weight of the minimum weight perfect matching in the resulting graph. □

4.3 Embedding on a polygonal schema in normal form

In this section we give a log-space, matching-preserving reduction that takes a combinatorial embedding of a bounded genus bipartite graph and outputs a graph in the class BDDGENBIP.

Theorem 4.3.1. *Given a 2-cell combinatorial embedding of a graph G of bounded genus, there is a log-space transducer that constructs a graph $H \in$ BDDGENBIP, such that, there is a perfect matching in G iff there is a perfect matching in H. Moreover, given a perfect matching M' in H, in log-space one can construct a perfect matching M in G.*

Proof. Given a combinatorial embedding of a graph G we apply Theorem 4.3.4 to get a combinatorial embedding of G' on a polygonal schema in normal form. At this point, there are no vertices lying on the boundary of the polygonal schema, only edges crossing it. From G' we construct another graph $G'' \in$ BDDGENBIP as follows: for each such edge $e = (u, v)$, such that u and v lie on different segments of the polygon, we introduce internal vertices $u' = v', v''$ on the edge e(converting it to a path $u, u' = v', v'', v$) so that u', v' lie on the boundary of the polygon on the sides nearer to u, v respectively. Now, due to this construction, the edge e gets replaced by a path of length 3 and thus the number of perfect matchings in G'' is preserved.[1] If G was orientable, we output $H = G''$ and we are done.

If G was non-orientable, we apply Theorem 4.5.1 to get a graph G_2 along with its embedding on an orientable polygonal schema (need not be in the normal form), such that there is a one to one correspondence between perfect matchings in G_1 and G_2. Once again we apply Theorem 4.3.4 to get a graph say G'_2 on a polygonal schema in normal form. Then we repeat the same procedure as above to get the desired orientable graph in BDDGENBIP as earlier. □

4.3.1 Combinatorial embedding to a polygonal schema

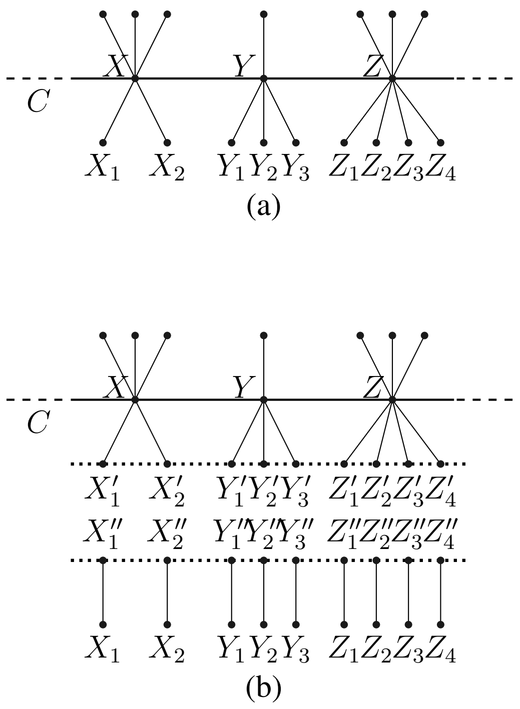
Lemma 4.3.2 ([ABC$^+$09]). *Let G be a graph embedded on a surface, and let T be a spanning tree of G. Then there is an edge $e \in E(G)$ such that $T \cup \{e\}$ contains a non-separating cycle.*

Notice that in [ABC$^+$09] the graph was required to be embedded on an orientable surface but the proof did not use this requirement.

Definition 4.3.1. Given a cycle (or path) C in an embedded graph G, define by $G \text{✂} C$ the graph constructed by "cutting" the edges incident on the cycle from the right. In other words, the neighbors of $u \in C$ (which are not on the cycle) can be partitioned into two sets, arbitrarily called left and right. For every neighbor v of u which lies to the right of C, cut the edge (u, v) into two pieces (u, x_{uv}) and (y_{uv}, v) where x_{uv}, y_{uv} are (new) spurious vertices. We add spurious edges between consecutive spurious vertices along the cut and label all the newly formed spurious edges with the label L_C along the left set and L_C^{-1} along the right set. (see Figure 4.2).

Also, if C is a path, its endpoints will lie on two paths. Consider the first path - if the two edges on either side of C on this path have the same label L_1. This can be broken into two cases - firstly,

[1]That is if e was part of a matching in G' then we include the edges (u, u') and (v, v'') in the matching in G'', and if e was not part of a matching in G' then we include the edge (v', v'') in the matching in H, the rest being the same.



X
Y
Z
C
X_1 X_2 $Y_1 Y_2 Y_3$ $Z_1 Z_2 Z_3 Z_4$
(a)
X'_1 X'_2 $Y'_1 Y'_2 Y'_3$ $Z'_1 Z'_2 Z'_3 Z'_4$
X''_1 X''_2 $Y''_1 Y''_2 Y''_3$ $Z''_1 Z''_2 Z''_3 Z''_4$
X_1 X_2 $Y_1 Y_2 Y_3$ $Z_1 Z_2 Z_3 Z_4$
(b)

Figure 4.2: An example of the cut operation ✂, cutting graph G along cycle (or path) C. (a) Part of graph G and cycle C. (b) Part of the resulting graph G✂C, with the dotted lines representing the spurious edges.

if the left and right side of this endpoint are the same (in other words, the path is a cycle). In this case, we just keep the same label L_1. When the left and right side of this endpoint are distinct, we will need to split the label into two or three new labels as detailed below and similarly for the other path and common label L_2. We will only describe the case when L_1, L_2 are both defined - the other cases are similar and simpler.

First assume that $L_1 \neq L_2$ and $L_1 \neq L_2^{-1}$. Then we will split remove labels L_1, L_2 and replace them by four new labels say $L'_{1,C}, L''_{1,C}$ and $L'_{2,C}, L''_{2,C}$, respectively for the two sides of the intersection. If, on the other hand, L_1 is the same as L_2 or its inverse - then there are two subcases. Firstly, if the path C is between two copies of the same vertex then we replace L_1 by two new labels $L'_{1,C}, L''_{1,C}$ one for either side of the cut. L_2 being a copy or an inverse copy of L_1 splits automatically. The second case is if C is between two distinct points on two copies or

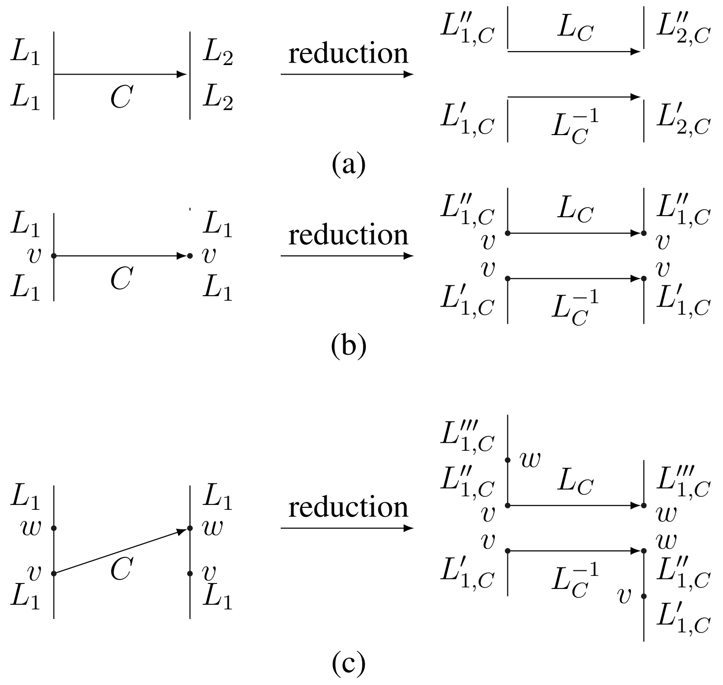


Figure 4.3: Cutting along a path C when (a) $L_1 \neq L_2$ and $L_1 \neq L_2^{-1}$, (b) $L_1 = L_2$ or $L_1 = L_2^{-1}$ and C is between copies of the same vertex v, and (c) $L_1 = L_2$ or $L_1 = L_2^{-1}$ and C is between distinct vertices v and w.

inverse copies. Then we split L_1 into three parts according to the two points. The rotation system is modified appropriately. We illustrate this in Figure 4.3.

Notice that in the process of cutting, for every new label L_C we are adding at most 4 new labels.

Given a graph G_i embedded on a surface, potentially with spurious edges, we can find C_{i+1}, a non-separating cycle (which does not use a spurious edge) by invoking Lemma 4.3.2. Define G_{i+1} to be $G_i ✂ C_{i+1}$.

Starting with $G_0 = G$ of genus g and repeating the above operation at most g times, we get a planar graph H with at most $2g$ spurious faces (which consist of spurious vertices and edges).

Now find a spanning tree of this graph which does not use a spurious edge - that such a tree exists follows from noticing that the graph without spurious edges is still connected. Find a tree path connecting any two spurious faces. Cut along this path to combine the two spurious faces into one larger spurious face. Repeat the operation till all the spurious faces are merged into one spurious face and re-embed the planar graph so that it forms the external face.

It is easy to see that the procedure above can be performed in log-space, provided that g is constant. Thus we have sketched the proof of the following:

Lemma 4.3.3. *Given the combinatorial embedding of a bounded genus graph we can find a polygonal schema for the graph in log-space.*

4.3.2 Normalizing a polygonal schema

We adapt the algorithmic proof of Brahana-Dehn-Heegaard (BDH) [Bra21, DH07] classification theorem as described in Vegter-Yap [VY90] so that it runs in log-space for bounded genus graphs. The algorithm starts with a polygonal schema and uses the following five transforms $O(m)$ times to yield a normalized polygonal schema, where the original polygonal schema has $2m$ sides.

A. Replace $X\sigma\bar{\sigma}$ by X (Example given in Figure 4.4).

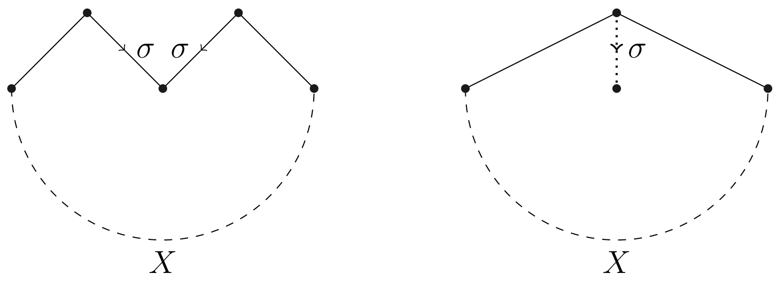


Figure 4.4: Reduction A (pasting along σ)

B. Replace $\sigma\tau X\bar{\tau}Y$ by $\rho X\bar{\rho}\sigma Y$ (Example given in Figure 4.5).

C. Replace $\sigma X\sigma Y$ by $\tau\tau Y^* X$, where Y^* is reverse complement of Y (Example given in Figure 4.6).

D. Replace $\sigma X\tau Y\bar{\sigma}U\bar{\tau}V$ by $\rho\pi\bar{\rho}\bar{\pi}UYXV$ (Example given in Figure 4.7).

E. Replace $\sigma_1\sigma_1 X\sigma_2\sigma_3\bar{\sigma}_2\bar{\sigma}_3 Y$ by $\tau_1\tau_1\tau_2\tau_2\tau_3\tau_3 XY$ (Example given in Figure 4.8).

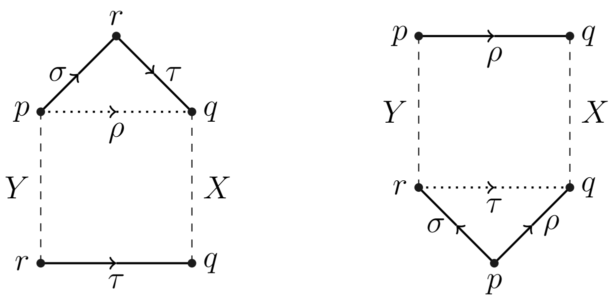


Figure 4.5: Reduction B (Cutting along ρ followed by pasting along τ). Note that the number of vertices in the equivalence class of r, reduces by 1.

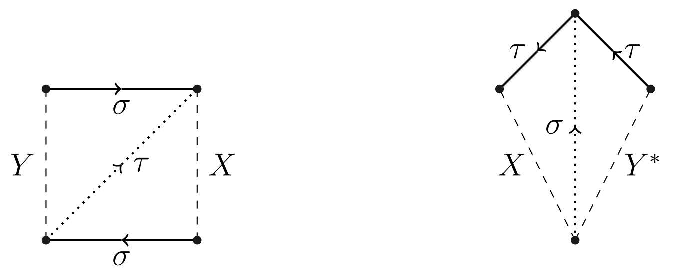


Figure 4.6: Reduction C (Cutting along τ followed by pasting along ρ)

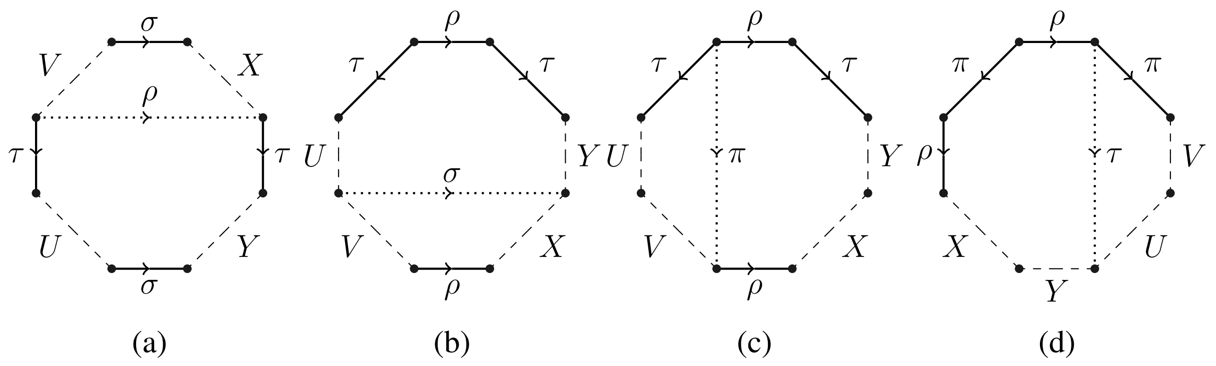


Figure 4.7: Reduction D (a) Cutting along ρ. (b) Pasting along σ. (c) Cutting along π. (d) Pasting along τ.

F. Replace $\sigma\sigma\tau\tau X$ by $\sigma\rho\bar{\sigma}\rho X$ (Example given in Figure 4.9).

The procedure is to

1. Use reductions A,B,C several times to ensure that all the sides of the polygonal schema have

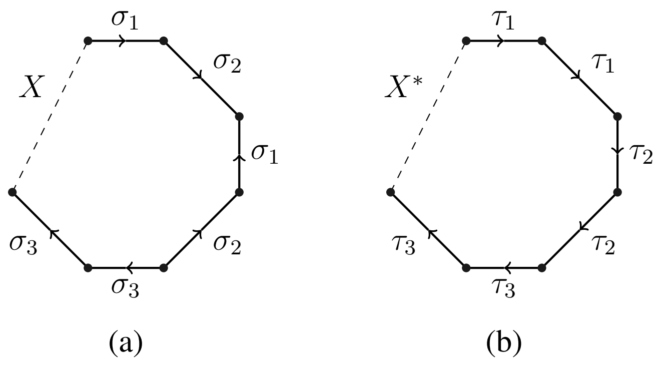


Figure 4.8: Reduction E (a) Initial polygonal schema (b) Polygonal schema obtained after applying Reduction E (replacing (i) σ_1 with $\bar{\tau}_3\bar{\tau}_2$, (ii) σ_2 with $\tau_1\tau_2$ and (iii) σ_3 with $\tau_1\tau_2\tau_3X^*$).

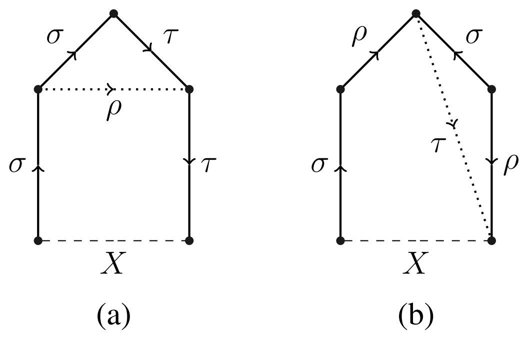


Figure 4.9: Reduction F (a) Cutting along ρ. (b) Pasting along τ.

a common endpoint.

2. a) Orientable case: Use transform D repeatedly to bring the polygon in normal form.

   b) Non-orientable case:

      - Use reductions C,D to convert the schema into a form where the orientable symbols are clustered and non-orientable symbols are paired
      - Use reduction E repeatedly (in the forward direction) to eliminate all orientable symbols.
      - Use Reduction E in the *reverse* direction repeatedly to eliminate all but at most one non-orientable symbol.

- Use Reduction F, if necessary, to ensure that there is at most one non-orientable symbol.

Possibly, the only step requiring any explanation is the last one. We apply Reduction E in reverse with X as the empty string to replace three non-orientable symbols by two orientable ones forming a cluster of 4 and a single non-orientable one which forms a pair. The way we apply the reduction, ensures that both the orientable and the non-orientable parts are contiguous.

Finally we will be left with a string in one of the first two normal forms or a string of the form $\sigma\sigma\tau\tau X$ (where X is an orientable schema in normal form) in which case Reduction F is applicable.

To see that the above procedure can be carried out in L it suffices to prove that each of the above reductions can be carried out in L, the number of reductions is bounded by a constant and we can decide in L when to carry out a reduction.

The Vegter-Yap paper does careful book-keeping in order to ensure that the number of operations in Step 1 is linear in the original genus. We can alternatively, follow the brute force approach and keep on applying Reductions A,B,C while the sides of the polygon do not have a common end-point. This will require at most linear number of applications of the first two reductions.

Observe that for the orientable case, each application of reduction D reduces the number of unclustered symbols by two. Thus we are done in $O(m)$ applications of this reduction. Similarly, each application of reduction C reduces the number of unpaired non-orientable symbols by one and as before every application of reduction D reduces the number of unclustered orientable symbols by two. So in $O(m)$ steps all the orientable symbols are clustered and the non-orientable symbols are paired. Now every application of reduction E in the forward direction gets rid of two orientable symbols so in $O(m)$ steps all the orientable symbols are removed. Finally $O(m)$ applications of reduction E in reverse lead to removal of all but one non-orientable symbols.

To see that each of the steps is in L observe that each of the steps involves one or more of the following operations:

- find a path through the interior of the polygon between two points on its boundary
- cut along a path
- paste two paired sides of (a cut) polygon together

We know how to do the second operation in L while the third, being the reverse of the second one is even easier, since we just have to identify corresponding spurious vertices and then excise them out of the corresponding edge. The first operation is just an undirected reachability question in the graph (minus its boundary) hence is in L by Theorem 2.3.1.

Finally, a determination of when to apply a particular reduction is easily seen to be in L for all but, possibly, reduction D. In this case, for an orientable symbol σ separated from its mate $\bar{\sigma}$ on both sides, sequentially test for each other symbol τ if it lies in one of the two stretches that σ and its mate divide the schema into, while its mate $\bar{\tau}$ lies in the other. Having found the first such τ suffices to enable a use of the reduction.

Thus, using the above argument and Lemma 4.3.3 we have sketched the proof of the following theorem:

Theorem 4.3.4. *Given a combinatorial embedding of a bounded genus (say $g \geq 0$) graph G, in log-space we can find a polygonal schema for the graph in normal form of genus $O(|g|)$ in magnitude, and also the corresponding combinatorial embedding.*

4.4 Isolation in bounded genus graphs

In this section we establish new upper bounds on the space complexity of certain matching problems on the class of bounded genus bipartite graphs, given as a combinatorial embedding.

Theorem 4.4.1 (Main Theorem). *Given an orientable graph $G \in$ DIRBDDGENBIP in terms of its polygonal schema, there exists a log-space computable, polynomially bounded, skew-symmetric weight function $w : E(G) \to \mathbb{Z}$, such that for any cycle $C \in G$, $w(C) \neq 0$.*

Proof of Theorem 4.4.1. Let $(\phi, \mathcal{S})$ be the polygonal schema of the given graph G, where $\mathcal{S} = (S_1, S_2, \ldots, S_{2g})$. We shall denote the pair of a side S_i by S'_i. Let $\mathcal{T} = \{T_1, T_2, \ldots T_g\}$ be the set of distinct sides of the polygon, that is no two elements in $\mathcal{T}$ are pairs of each other. We define w as a linear combination of the following $2g + 1$ weight functions defined below.[2]

For each edge $e = (u, v)$ in G, define $2g + 1$ weight functions as follows:

- For each $i \in [g]$,

$$w_i(e) = \begin{cases} 1 & \text{if } u \text{ lies on the side } T_i \\ -1 & \text{if } v \text{ lies on the side } T_i \\ 0 & \text{otherwise} \end{cases} \tag{4.4.1}$$

- For each $i \in [g]$,

$$w'_i(e) = \begin{cases} j & \text{if } u \text{ lies on the side } T_i \text{ at index } j \text{ from the tail of } T_i \\ -j & \text{if } v \text{ lies on the side } T_i \text{ at index } j \text{ from the tail of } T_i \\ 0 & \text{otherwise} \end{cases} \tag{4.4.2}$$

- w_{TV} is defined to be the weight function defined by Tewari and Vinodchandran in [TV10] by assuming G to be a planar graph.

Firstly note that each of the three kinds of weight functions that we defined above are skew-symmetric. Therefore a linear combination of them is also skew-symmetric.

Now think of G as a planar graph by ignoring the identification of vertices along respective sides of the polygon, which we shall call G_{planar}. G_{planar} is bipartite since G is bipartite. Now

[2]We can construct such a w in log-space since g is constant.

w_{TV} is the weight function defined in Lemma 3 in [TV10]. Therefore for every cycle $C \in G$ that does not cross any of the sides S_i (and thus is a valid cycle in G_{planar}), $w_{TV}(C) \neq 0$ [TV10].

Let C be a simple cycle in G that crosses the boundary of the polygon at some point. That is, there exists a vertex v on side T_i such that the cycle enters v from the side T_i and exits from the vertex corresponding to v on side T_i'. Let E_j^C be the set of edges of C that lie on the side T_j. From the definition of w_j it follows that $w_j(C) = w_j(E_j^C)$ (same thing holds for w_j' as well). Now if there exists a k such that $|E_k^C|$ is odd, then $w_k(C) \neq 0$. Otherwise, if for all k, $|E_k^C|$ is even then by Lemma 4.4.3 it follows that the edges of E_k^C, alternate between going out and coming into the side T_k if at all it passes through the side T_k. Existance of one such side is guaranteed since C crosses the boundary of the polygon. Without loss of generality we assume that side is T_1. Now by using Lemma 4.4.4 we get that $w_1'(E_1^C) \neq 0$ and thus $w_1'(C) \neq 0$. (See below for Lemma 4.4.3 and 4.4.4) □

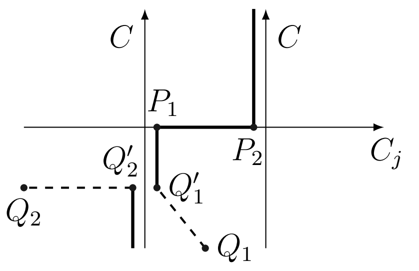


Figure 4.10: Construction of a path from Q_1 to Q_2 in $\Gamma \setminus C$ (the dotted path is a path between Q_1 and Q_1' (resp. between Q_2 and Q_2').

To establish Lemma 4.4.3 we use an argument (Lemma 4.4.2) from homology theory. For two cycles (directed or undirected) C_1 and C_2, let $I(C_1, C_2)$ denote the number of times C_1 and C_2 cross each other (that is one of them goes from the left to the right side of the other, or vice versa).

Next we adapt the Lemma 4.4.2 from Cabello and Mohar [CM07]. Here we assume we are given an orientable surface (Cabello and Mohar gives a proof for a graph on a surface).

Lemma 4.4.2. *Given a genus g orientable, surface Γ, let $\mathcal{C} = \{C_i\}_{i\in[2g]}$ be a set of cycles that generate the first homology group $H_1(\Gamma)$. A cycle C in Γ is non-separating if and only if there is some cycle $C_i \in \mathcal{C}$ such that $I(C, C_i) \equiv 1(\text{mod}2)$.*

Proof. Let $\tilde{C}$ be some cycle in Γ. We can write $\tilde{C} = \sum_{i\in[2g]} t_i C_i$ since $\mathcal{C}$ generates $H_1(\Gamma)$. Define $I_{\tilde{C}}(C) = \sum_{i\in[2g]} t_i I(C, C_i)(\mod 2)$. One can verify that $I_{\tilde{C}} : \mathcal{C}_1(\Gamma) \to \mathbb{Z}_2$ is a group homomorphism. Now since $\mathcal{B}_1(\Gamma)$ is a normal subgroup of $\mathcal{B}_1(\Gamma)$, $I_{\tilde{C}}$ induces a homomorphism from $H_1(\Gamma)$ to $\mathbb{Z}_2$.

Any cycle is separating if and only if it is homologous to the empty set. Therefore if C is separating, then $C \in \mathcal{B}_1(\Gamma)$ and thus every homomorphism from $H_1(\Gamma)$ to $\mathbb{Z}_2$ maps it to 0. Hence for every $i \in [2g]$, $I(C, C_i) \equiv I_{C_i}(C) = 0$.

Suppose C is non-separating. One can construct a cycle C' on Γ, that intersects C exactly once. Let $C' = \sum_{i\in[2g]} t'_i C_i$. Now $1 \equiv I_{C'}(C) \equiv \sum_{i\in[2g]} t'_i I(C, C_i)(\text{mod}2)$. This implies that there exists $i \in [2g]$ such that $I(C, C_i) \equiv 1(\text{mod}2)$. □

Lemma 4.4.3. *Let C be a simple directed cycle on a genus g orientable surface Γ and let $\mathcal{C} = \{C_i\}_{i\in[2g]}$ be a system of $2g$ directed cycles on Γ, having exactly one point in common and $\Gamma \setminus \mathcal{C}$ is the fundamental polygon, say Γ'. If $I(C, C_i)$ is even for all $i \in [2g]$ then for all $j \in [2g]$, C alternates between going from left to right and from right to left of the cycle C_j in the direction of C_j (if C crosses C_j at all).*

Proof. Suppose there exists a $j \in [2g]$ such that C does not alternate being going from left to right and from right to left with respect to C_j. Thus if we consider the ordered set of points where C intersects C_j, ordered in the direction of C_j, there are two consecutive points (say P_1 and P_2) such that at both these points C crosses C_j in the same direction.

Let Q_1 and Q_2 be two points in $\Gamma \setminus C$. We will show that there exists a path in $\Gamma \setminus C$ between Q_1 and Q_2. Consider the shortest path from Q_1 to C. Let Q'_1 be the point on this path that is as close to C as possible, without lying on C. Similarly define a point Q'_2 corresponding to Q_2. Note that it is sufficient for us to construct a path between Q'_1 and Q'_2 in $\Gamma \setminus C$. If both Q'_1 and Q'_2 locally lie on the same side of C, then we get a path from Q'_1 to Q'_2 not intersecting C, by traversing along the boundary of C. Now suppose Q'_1 and Q'_2 lie on opposite sides (w.l.o.g. assume that Q'_1 lies on the right side) of C. From Q'_1 start traversing the cycle until you reach cycle C_j (point P_1 in Figure 4.10). Continue along cycle C_j towards the adjacent intersection point of C and C_j, going as close to C as possible, without intersecting it (point P_2 in Figure 4.10). Essentially this corresponds to switching from one side of C to the other side without intersecting it. Next traverse along C to reach Q'_2. Thus we have a path from Q'_1 to Q'_2 in $\Gamma \setminus C$. We give an example of this traversal in Figure 4.10. This implies that C is non-separating.

It is well known that $\mathcal{C}$ forms a generating set of $H_1(\Gamma)$, the first homology group of the surface. Now from Lemma 4.4.2 it follows that $I(C, C_l) \equiv 1(\text{mod}2)$ for some $l \in [2g]$, which is a contradiction. □

Lemma 4.4.4. *Let G be a graph in* DIRBDDGENBIP *with C being a simple cycle in G and E_1^C being the set of edges of C that lie on the side T_1. Assume $|E_1^C|$ is even and the edges in E_1^C alternate between going out and coming into the polygon. Let $i_1 < i_2 < \ldots < i_{2p-1} < i_{2p}$ be the distinct indices on T_1 where C is incident upon. Then, $\left|w'_1(E_1^C)\right| = |\sum_{k=1}^{p}(i_{2k} - i_{2k-1})|$ and thus non-zero unless E_1^C is empty.*

Proof. Let $e_j = (u_j, v_j)$ for $j \in [2p]$ be the $2p$ edges of G incident on the side T_1. Without loss of generality assume that u_1 lies on T_1 (that is the edge e_1 is directed away from the side T_1). Therefore by Lemma 4.4.3, for every i in $[2p]$ such that i is odd, u_i is incident on T_1 (that is the edge e_i is directed away from the side T_1) and for every i such that i is even, v_i is incident on T_1 (that is the edge e_i is directed towards the side T_1). Hence $w'_1(E_1^C) = i_1 - i_2 + i_3 - i_4 \ldots - i_{2p}$. Now removing the assumption at the beginning of this proof would show that the LHS and RHS of the above equation are equal modulo absolute value as required. □

It is interesting to note here that similar method does not show that bipartite matching in non-

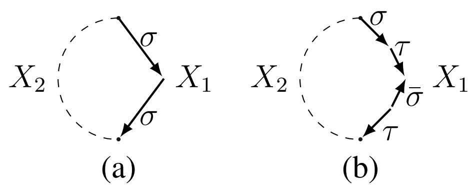


Figure 4.11: **(a)** $\Lambda(\Gamma)$ when the surface is a sum of an orientable surface and the projective plane. **(b)** $\Lambda(\Gamma)$ when the surface is a sum of an orientable surface and the Klein bottle.

orientable bounded genus graphs is in SPL. The reason is that Lemma 4.4.3 crucially uses the fact that the surface is orientable. In fact, one can easily come with counterexample to Lemma 4.4.3 if the surface is non-orientable.

4.5 Reducing the non-orientable case to the orientable case

In this section we tackle the non-orientable case by giving a matching preserving reduction from a graph embedded on a non-orientable surface to a graph embedded on an orientable surface.

Let G be a bipartite graph embedded on a genus g non-orientable surface. As a result of Theorem 4.3.4 we can assume that we are given a combinatorial embedding (say Π) of G on a (non-orientable) polygonal schema, say $\Lambda(\Gamma)$, in the normal form with $2g'$ sides. (Here g' is a function of g.)

Let $Y = (X_1, X_2)$ be the cyclic ordering of the labels of the sides of $\Lambda(\Gamma)$, where X_2 is the 'orientable part' and X_1 is the 'non-orientable part'. More precisely, for the polygonal schema in the normal form, we have: X_1 is either (σ, σ) (thus corresponds to the projective plane) or it is $(\sigma, \tau, \bar{\sigma}, \tau)$ (thus corresponds to the Klein bottle). See Figure 4.11.

Now let G be a bipartite graph embedded on a non-orientable polygonal schema $\Lambda(\Gamma)$ with $2g'$ sides. We will construct a graph G' embedded on an *orientable* polygonal schema with $4g' - 2$ sides such that G has a perfect matching iff G' has a perfect matching. Moreover, given a perfect matching in G' one can retrieve in log-space a perfect matching in G. This is illustrated in Theorem

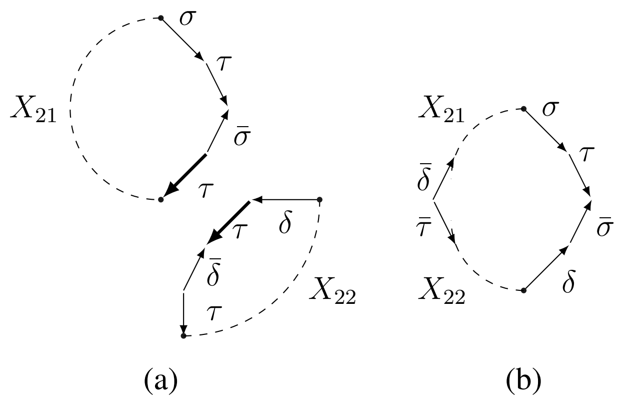


Figure 4.12: *Klein bottle.* **(a)** The two copies of $\Lambda(\Gamma)$ with the side that is being glued shown in dark. **(b)** Polygonal schema obtained after the glueing operation.

4.5.1.

Theorem 4.5.1. *Let G be a bipartite graph given with its embedding on a non-orientable polygonal schema in normal form $\Lambda(\Gamma)$, with $2g'$ sides as above. One can construct in log-space, another graph G' together with its embedding on the polygonal schema of an orientable surface Γ' of genus $4g'-2$ such that: G has a perfect matching iff G' has a perfect matching. Moreover, given a perfect matching in G', one can construct in log-space a perfect matching in G.*

Proof. We first show the case when Γ is the sum of an orientable surface and a Klein bottle. Consider the polygonal schema formed by taking two copies of $\Lambda(\Gamma)$ and glueing the side τ of one copy with its partnered side τ of the other copy. We relabel the edge labelled σ in the second copy with some unused symbol δ to avoid confusion. The entire reduction is shown in Figure 4.12. Let G' be the resulting graph.

Note that the polygonal schema obtained as a result represents an orientable surface and has constantly many sides. Also every vertex and edge in G has exactly two copies in G' and G' is also bipartite. Let M be a matching in G. Let M' be the union of the edges of M from both the copies of G . Its easy to see that M' is a matching in G'. Now consider a matching M' in G'. The *projection* of M' to G gives a subgraph of G where every vertex has degree (counted

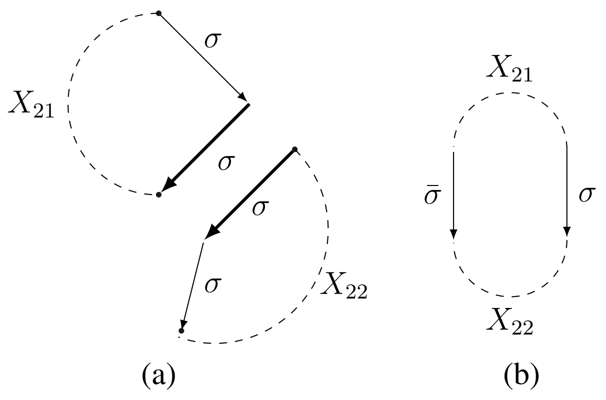


Figure 4.13: *Projective plane.* **(a)** The two copies of $\Lambda(\Gamma)$ with the two pair of sides that are being glued shown in dark. **(b)** Polygonal schema obtained after the glueing operation.

with multiplicity) exactly two. Since G is bipartite, one can obtain a perfect matching within this subgraph.

Now consider the case when Γ is the ‘sum’ of an orientable surface and a projective plane, i.e., following the notation above X_1 corresponds to the labels of a polygonal schema for the projective plane and X_2 corresponds to the labels of a polygonal schema of an orientable surface. Take two copies of $\Lambda(\Gamma)$, and glue σ of one copy with its partner σ in the other copy. We show this operation in Figure 4.13. The rest of the proof is similar to the Klein bottle case. □

Thus we see that the non-orientable case can be reduced to the orientable case. The resulting polygonal schema need not be in the normal form. Once again we apply Theorem 4.3.4 to get a combinatorial embedding on a polygonal schema in the normal form.

4.6 New complexity upper bounds

Theorem 4.6.1. *For a graph embedded on a bounded genus surface,*

(a) PM-Decision *is in* SPL,

(b) PM-Construct *is in* $\mathsf{FL}^{\mathsf{SPL}}$ *and*

(c) PM-UNIQUE *is in* SPL.

Proof. As a result of Theorem 4.3.1, we can assume that our input graph G is orientable and is in the class BDDGENBIP. This implies that the directed graph $\overrightarrow{G}$ is in the class DIRBDDGENBIP. Using Theorem 4.4.1 and Lemma 4.2.1 we get a log-space computable weight function W, such that the minimum weight perfect matching in G with respect to W is unique. Moreover, for any subset $E' \subseteq E$, Theorem 4.4.1 is valid for the subgraph $\overrightarrow{G} \setminus E'$ also, with respect to the same weight function W. Now (a) and (b) follows from Lemma 4.2.2. Checking for uniqueness can be done by first computing a perfect matching, then deleting an edge from the matching and rechecking to see if a perfect matching exists in the new graph. If it does, then G does not have a unique perfect matching, else G does have a unique perfect matching. □

Theorem 4.4.1 also gives an alternative proof of directed graph reachability for bounded genus graphs.

Theorem 4.6.2 ([BTV09, KV10]). *Directed graph reachability for bounded genus graphs is in* UL.

Proof. From Theorem 4.4.1, it follows easily that the class of graphs DIRBDDGENBIP is min-unique with respect to the weight function in Theorem 4.4.1. The proof is similar to the proof of Theorem 3.3.1 in Chapter 3. Now applying Theorem 2.3.5 from Chapter 2, we get the desired result. □

4.7 Conclusion

In this chapter we gave new upper bound on the problem of perfect matching in bounded genus graphs via an extension of the isolating lemma. Can we use the framework of algebraic topology to advance the isolation results to graphs of logarithmic genus?

Chapter 5

Hardness Results for Nondeterministic Logarithmic Space

In this chapter we exhibit three new complete problems for NL. All three problems are restriction of the directed reachability problem and have distinct geometric and combinatorial properties. The goal of studying these variants of reachability is that they might give new avenues of showing improved upper bound on NL. In Section 5.1 we show that graphs over three pages are NL-complete. In Section 5.2 we show that three dimensional grid graphs are NL-complete. In Section 5.3 we show that graphs of thickness two are NL-complete.

5.1 Three pages are sufficient for nondeterministic log-space

We show that the reachability problem for directed graphs embedded on 3 pages is complete for NL. It can be shown that the reachability problem for graphs on 2 pages is equivalent to reachability in grid graphs and hence is in UL by the result of [BTV09]. It is also interesting to note that graphs embedded on 1 page are outer-planar and hence reachability for directed graphs on 1 page is complete for L [ABC$^+$09].

Definition 5.1.1. ThreePage is the class of all graphs G that can be embedded on 3 pages as follows: all vertices of G lie along the spine and the edges lie on exactly one of the three pages without intersection. Moreover all edges are directed from top to bottom. THREEPAGEREACH is the language consisting of tuples (G, s, t, f), such that $G \in$ ThreePage, s and t are two vertices in G and there exists a path from s to t in G, and f is an embedding of G on 3 pages (that is, f defines the ordering of the vertices along the the spine and in which page an edge lies on).

Theorem 5.1.1. THREEPAGEREACH *is complete for* NL.

Proof. To show that THREEPAGEREACH is in NL we need to verify that, given an instance (G, s, t, f), if f is an embedding of G on 3 pages. Note that, for any two edges (u_1, v_1) and (u_2, v_2) in G that lies in the same page, the edges cross each other if and only if either (i) u_2 lies in between u_1 and v_1, or (ii) v_2 lies in between u_1 and v_1, in the ordering of the vertices along the spine. This condition can be checked in NL and therefore whether f is indeed an embedding of G on 3 pages or not can also be verified in NL.

To show that THREEPAGEREACH is hard for NL, assume that we are given a topologically sorted DAG G, with $(u_1, u_2, \ldots, u_n)$ being the topological ordering of the vertices of G. We want to decide if there is a path in G from u_1 to u_n. We define an ordering on the edges of G, say $\mathcal{E}(G)$. Given two edges e_1 and e_2, (i) if the head of e_1 precedes the head of e_2, then e_1 precedes e_2 in the ordering, (ii) if the head of e_1 is the same as the the head of e_2, then e_1 precedes e_2 in the ordering if tail of e_1 precedes tail of e_2. It is easy to see that $\mathcal{E}(G)$ can be constructed in log-space given G and in any path from s to t, if edge e_1 precedes e_2, then e_1 precedes e_2 in $\mathcal{E}(G)$ as well. Let m be the number of edges in G.

For any integer k, let $[k]$ denote the set of integers $\{1, \ldots, k\}$. We create $2m$ copies of each vertex in G and let v_i^j denote the jth copy of the vertex u_i, for $i \in [n]$ and $j \in [2m]$. We order the vertices along the spine of H from top to bottom as follows:
$(v_1^1, v_2^1, \ldots, v_n^1, v_n^2, v_{n-1}^2, \ldots, v_1^2, v_1^3, v_2^3, \ldots, v_n^3, \ldots, v_n^{2m}, \ldots, v_1^{2m})$.

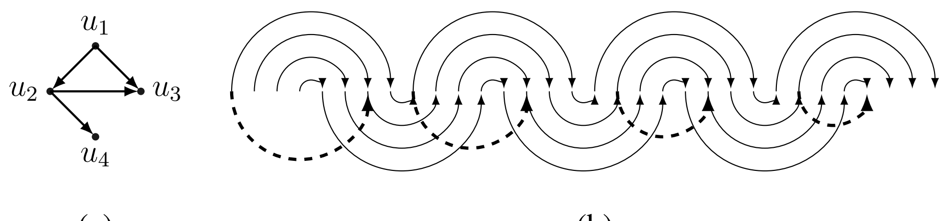


Figure 5.1: (a) Graph G. (b) The corresponding graph H. The dashed edges of H are on page 3.

Next we need to connect all the $2m$ vertices corresponding to each u_i from the top to bottom. We use the first 2 pages to do that. Put the edge (v_i^j, v_i^{j+1}) in H, for each $i \in [n]$ and each $j \in [2m-1]$, using page 1 when j is odd and page 2 when j is even. For the kth edge in $\mathcal{E}(G)$, say $e_k = (u_{k_1}, u_{k_2})$, put the edge $(v_{k_1}^{2k-1}, v_{k_2}^{2k})$ in H, using page 3. It is clear that this can be done without any two edges crossing each other. We give an example of this reduction in Figure 5.1. The claim is, there exists a path from u_1 to u_n in G if and only if there exists a path from v_1^1 to v_n^{2m} in H.

Suppose there exists a path p from u_1 to u_n in G. Let $p = (e_{i_1}, \dots e_{i_l})$. For each $j \in [l]$, corresponding to e_{i_j} there exists an edge in page 3 of H by construction, say f_j. Also by construction and the ordering $\mathcal{E}(G)$, the tail of f_j lies above the head of f_{j+1} along the spine of H. Further, since the head of $e_{i_{j+1}}$ is the same as the tail of e_{i_j} for $j \in [l-1]$, there exists a path from the tail of f_j to the head of f_{j+1} (using edges from pages 1 and 2). Thus we get a path from v_1^1 to v_n^{2m} in H.

To see the other direction, let ρ be a path from v_1^1 to v_n^{2m} in H. Let $\rho_3 = (\alpha_1, \alpha_2, \dots, \alpha_r)$ be the sequence of edges of ρ that lie on page 3. Note that each of the edges in ρ_3 has a unique pre-image in G by the property of the reduction. This defines a sequence of edges p' in G by taking the respective pre-images of the edges in ρ_3. Now the sub-path of ρ from v_1^1 to the head of α_1 uses only edges from page 1 and 2 and thus by construction the head of α_1 is a vertex $v_1^{l_1}$ (for some $l_1 \in [2m]$). A similar argument establishes that the tail of α_r is a vertex $v_n^{l_2}$ (for some $l_2 \in [2m]$) and also that the tail of α_i and the head of α_{i+1} are the copies of the same vertex in G, for $i \in [r-1]$. Therefore p' is a path from u_1 to u_n in G. $\square$

5.2 Three dimensional grid graphs

A *three-dimensional monotone grid graph* is a directed graph whose vertices are $[n] \times [n] \times [n]$ with edges connecting a vertex to its immediate neighboring grid point in the positive x, y or z direction. That is an edge is of the form $((i,j,k),(i+1,j,k))$ (east edge) or $((i,j,k),(i,j+1,k))$ (north edge) or $((i,j,k),(i,j,k+1))$ (inward edge), provided the respective coordinates exist. We refer to the st-connectivity problem in such graphs as 3D-MGGR and show that it is complete for NL.

Theorem 5.2.1. 3D-MGGR *is complete for* NL.

Proof. We use the fact that the standard NL-complete reduction which generates the configuration graph of a log-space Turing machine can be easily modified to get a topologically sorted DAG. We simply prepend a timestamp to each configuration which results in a layered acyclic graph. Each layer can use the canonical ordering to induce a total topological order. Thus, without loss of generality, we will reduce such a DAG to a 3-D monotone grid graph while preserving st-connectivity.

Let $G = (V, E)$ be a DAG with topologically sorted vertices $V = \{v_1, \ldots, v_n\}$. That is, if (v_i, v_j) is an edge, then $i < j$. We construct a 3D layered grid graph G' as follows. For each vertex v_i, we make i copies at positions (i,i,k) for $k = 1, \ldots, i$. We also connect each of the i copies by an edge in the positive z-direction ($(i,i,k) \to (i,i,k+1)$ for $k = 1, \ldots, i-1$).

Now, each xy plane (identified by $k \in \{1, \ldots, n\}$) will serve to preserve connectivity for each vertex v_k. We start by connecting vertices in an eastward direction, $(l,k,k) \to (l+1,k,k)$ for $l = k, \ldots, n-1$. Next, we connect vertices in a northward direction depending on the edges in the original graph G as follows. If $(v_k, v_l) \in E$ then we draw a path from (l,k,k) to (l,l,k) thus connecting (the last copy of) v_k to the k-th copy of v_l. Otherwise, we do nothing. An example of this construction from a complete DAG of size 4 can be found in Figure 5.2.

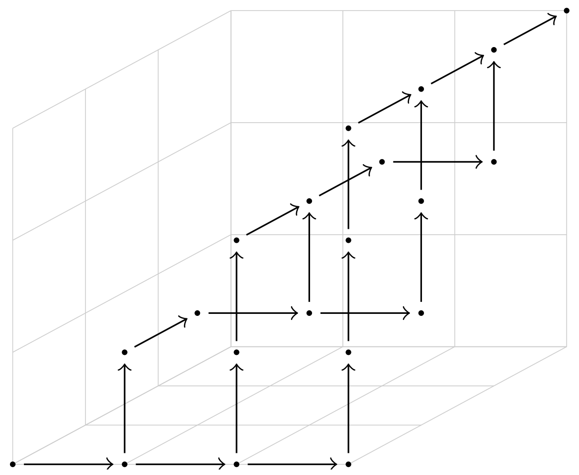


Figure 5.2: A mapped graph resulting from a complete DAG on $n = 4$ vertices.

The resulting graph is bounded within the $n \times n \times n$ cube. Furthermore, since each edge only requires an index look-up, the construction is clearly log-space computable.

Finally, without loss of generality we can assume that $s = v_1$ and $t = v_n$ and so we map s to the single copy of v_1 and t to the back most copy of v_n at coordinates (n, n, n). We claim that there exists a path $s \rightsquigarrow t$ in G if and only if there exists a path $(1, 1, 1) \rightsquigarrow (n, n, n)$ in G'. The construction clearly preserves st-connectivity. $\square$

5.3 Thickness two graphs

The usual graph-theoretic notion of thickness of a graph G is defined as the minimal number of planar subgraphs whose union is G. Intuitively, we can think of thickness as the minimal number of transparencies required to draw the graph so that no edges cross within any single transparency. Clearly, a graph is planar if and only if it has thickness-one. Surprisingly, however, thickness-two suffices to capture all of NL.

We'll actually show that completeness holds for an even more restrictive notion of thickness called *geometric thickness* [HSV95, DEH00]. The geometric thickness of a graph G is defined as the minimal number k such that we can assign planar point locations to the vertices of of G, represent each edge as a line segment, and assign each edge to one of k transparencies so that no two lines cross in any one transparency. The difference between these two notions is that geometric thickness requires that all vertex placements be consistent across all transparencies.

Theorem 5.3.1. *The st-connectivity problem for (geometric) thickness-two graphs is complete for* NL. *Moreover, each transparency is a layered grid graph.*

Proof. We will make use of the 3D monotone grid graph that results by applying the reduction in Theorem 5.2.1. We start by embedding each xy-layer (identified as L_k, $1 \leq k \leq n$) in the first transparency. We do so by laying each layer above the previous layer, shifting each layer by one unit. That is, the lower left corner of each layer L_k is oriented at $(k, (k-1)n+1)$ (the upper-right corner is thus at $((k+n), ((k-1)n+1+n)))$. This results in a $2n \times n^2+1)$ sized grid.

We now embed the inward z-edges using the second transparency. We will do so by routing them inside the grid defined by the xy planes. In order to do this, we first expand the grid by 3: each unit square is replaced by a 3×3 grid, leading to a *fine-grid.* Thus, each (i, j) coordinate in the first grid maps to

$$((3i-2),(3j-2))$$

in the fine grid.

We now have room to route the z-edges through the second transparency. Consider the z edges between layer L_k and L_{k+1}: $(i, i, k) \rightarrow (i, i, k+1)$ in the original 3D-mGG. The initial vertex is now oriented at

$$(3(k+i)-2, 3((k-1)n+1+i)-2) = ((3k+3i-2),(3nk-3n+3i+1))$$

and the final vertex is at

$$(3(k+1+i)-2, 3(kn+1+i)-2) = ((3k+3i+1), (3kn+3i+1))$$

in the expanded fine-grid. We will capture the connectivity of this edge by routing a path between these two grid points on the second transparency, avoiding contact with other z-edge/paths in the second transparency. First, we travel east one edge in the fine-grid:

$$((3k+3i-2), (3nk-3n+3i+1)) \rightarrow ((3k+3i-1), (3nk-3n+3i+1))$$

We then travel north until we have cleared the sub-grid corresponding to L_k; that is to

$$y = 3(kn+1)-2+1 = 3kn+2$$

on the fine-grid. We then travel east again for one edge;

$$((3k+3i-1), (3kn+2)) \rightarrow ((3k+3i), (3kn+2))$$

and continue north again to the same row as the final vertex:

$$y = (3kn+3i+1)$$

At this point, we simply travel east again one more edge and arrive at $((3k+3i+1), (3kn+3i+1))$, the intended final vertex.

This reduction is illustrated (cf. Figure 5.3) for the first few layers resulting from the complete DAG on 5 vertices from Figure 5.2.

It is not difficult to see that the reduction results in only two layers each of which avoids any

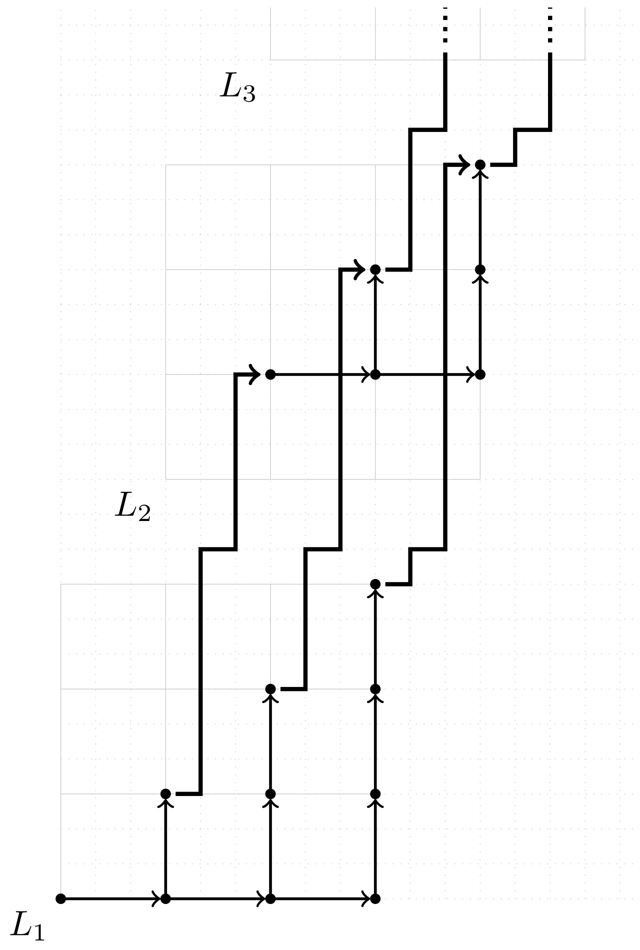


Figure 5.3: First two layers of the thickness-two reduction on the DAG in Figure 5.2. Lighter directed edges correspond to the first transparency while the darker (routing) paths are on the second transparency.

edge crossings and is clearly a layered grid graph. Moreover, the reduction is clearly log-space computable. □

In fact, the reduction in Theorem 5.3.1 is even stronger: each layer is actually a directed forest. Moreover, the reduction actually gives us a thickness-two embedding using only log-space.

5.4 Conclusion

The results in this chapter give us possible directions of establishing an improved upper bound on graph reachability. Although various restrictions of graph reachability characterize various log-space complexity classes, as of now we do not know the existence of a UL-complete problem. Coming up with a restriction of reachability (or even some other computational problem) that characterizes UL, might throw more light into the structure of the class UL.

Chapter 6

Graphs with *few* Paths and Unambiguous Hierarchies

In this chapter we study the power and limitations of unambiguous log-space computations. We give an introduction of the problems that we study in this chapter, in Section 6.1. In Section 6.2 we define the various log-space classes that we consider. In Section 6.3 we prove that counting the number of paths in a certain class of graphs is in UL. In Section 6.4 we investigate the relation between min-uniqueness and the NL versus UL problem. In Section 6.5 we study certain unambiguous hierarchies and prove upper bounds on them.

6.1 Introduction

Historically, several researchers have investigated the complexity class UL (for example, [BJLR91, BDHM92, BHS93, AJ93]) in different contexts. Reinhardt and Allender showed that UL contains NL non-uniformly. In [ARZ99], Allender, Reinhard, and Zhou showed that, under the (very plausible) hardness assumption that deterministic linear space has functions that can not be computed by circuits of size $2^{\epsilon n}$, the constructions given by Reinhardt and Allender can be *derandomized* to show that $\mathsf{NL} = \mathsf{UL}$ [ARZ99]. As the reachability problem for directed graphs is complete for NL, it is natural to investigate the space complexity of reachability for subclasses of directed graphs and indeed the recent progress has been in this direction. In [BTV09], it is shown that reachability for directed planar graphs is in UL. Subsequently, Thierauf and Wagner showed that reachability for $K_{3,3}$-free and K_5-free graphs can be reduced to planar reachability in log-space [TW09]. Kynčl and Vyskočil showed that reachability for bounded genus directed graphs also reduces to the planar case [KV10]. Thus reachability for these classes of graphs is also in UL.

6.1.1 Our Results

We extend the study of reachability to certain other important subclasses of directed graphs in this chapter.

Reachability in graphs with few paths

FewL, the log-space analog of the polynomial time class FewP [All86, CH90], is the class of languages that are decided by nondeterministic log-space machines that have the property that on any input there are at most polynomially many accepting paths [BJLR91, BDHM92]. Is $\mathsf{FewL} = \mathsf{UL}$? As $\mathsf{FewL} \subseteq \mathsf{NL}$, this is a very interesting restriction of the $\mathsf{NL} = \mathsf{UL}$ question (it is known that FewL is in $\mathsf{L}^{\mathsf{promiseUL}}$ [All06]). While we are unable to show that $\mathsf{FewL} \subseteq \mathsf{UL}$, we prove new *unambiguous* upper bounds for complexity classes related to FewL.

As one of our main results, we show that counting the number of s-t paths in graphs where the number of paths from s to any node is bounded by a polynomial is in the unambiguous function class FUL.

This result immediately implies a new upper bound $\mathsf{ReachFewL} \subseteq \mathsf{UL}$. $\mathsf{ReachFewL}$ is a restriction of FewL [BJLR91]. A nondeterministic machine M is called a *reach-few* machine, if on any input x and any configuration c of $M(x)$, the number of paths from the start configuration to

c is bounded by a polynomial. $\mathsf{ReachFewL}$ is the class of languages decided by a reach-few machine that is log-space bounded. Our result improves on the previous known trivial upper bound of $\mathsf{ReachFewL} \subseteq \mathsf{FewL}$.

The class $\mathsf{ReachFewL}$ was also investigated by Buntrock, Hemachandra, and Siefkes [BHS93] under the notation $\mathsf{NspaceAmb}(\log n, n^{O(1)})$. In [BHS93], the authors define, for a space bound s and an unambiguity parameter a, the class $\mathsf{NspaceAmb}(s(n), a(n))$ as the class of languages accepted by $s(n)$ space bounded nondeterministic machines for which the number of paths from the start configuration to any configuration is at most $a(n)$. They show that $\mathsf{NspaceAmb}(s(n), a(n)) \subseteq \mathsf{Uspace}(s(n) \log a(n))$ (hence $\mathsf{NspaceAmb}(\log n, O(1)) \subseteq \mathsf{UL}$). Our method can be used to show that $\mathsf{NspaceAmb}(s(n), a(n)) \subseteq \mathsf{Uspace}(s(n) + \log a(n))$, thus substantially improving their upper bound.

Even though our techniques do not lead to a new upper bound on FewL, we show a new upper bound for LFew (LFew is the counting verstion of FewL and is analogous to the class Few [CH90] in the polynomial-time setting). We show that $\mathsf{LFew} \subseteq \mathsf{UL}^{\mathsf{FewL}}$. This puts LFew in the second level of FewL hierarchy.

Complexity of Min-uniqueness

Our second consideration is the notion of *min-uniqueness* which is a central notion in the study of unambiguity in the log-space setting. Min-uniqueness was first used by Wigderson to show that $\mathsf{NL} \subseteq \oplus\mathsf{L}/\mathsf{poly}$ [Wig94]. For a directed graph G and two nodes s and t, G is called *st-min-unique* if the minimum length s to t path is unique (if it exists). G is min-unique with respect to s, if it is sv-min-unique for all vertices v. While st-min-uniqueness was sufficient for Wigderson's result, Reinhardt and Allender used the stronger version of min-uniqueness to show that $\mathsf{NL} \subseteq \mathsf{UL}/\mathsf{poly}$. In particular, they essentially showed that a log-space algorithm that transforms a directed graph into a min-unique graph with respect to the start vertex can be used to design an unambiguous algorithm for reachability. This technique was subsequently used in [BTV09] to show that reachability for planar directed graphs is in UL. These results strongly indicate that understanding min-uniqueness is crucial to resolving the NL versus UL problem.

Our second set of results is aimed at understanding min-uniqueness from a complexity-theoretic point of view. First we observe that min-uniqueness is necessary to show that $\mathsf{NL} = \mathsf{UL}$: if $\mathsf{NL} = \mathsf{UL}$, then there is a UL algorithm that gives a reachability preserving mapping from any directed graph to another graph that is min-unique with respect to the start vertex. It is an easy observation that Reinhardt and Allender's technique will work even if the algorithm that makes a directed graph min-unique is only UL computable. Thus min-uniqueness is necessary and sufficient for showing $\mathsf{NL} = \mathsf{UL}$.

OptL is the function class defined by Àlvarez and Jenner in [AJ93] as the log-space analog of Krentel's OptP [Kre88]. OptL is the class of functions whose values are the maximum over all the outputs of an NL-transducer. Àlvarez and Jenner showed that this class captures the complexity of some natural optimization problems in the log-space setting.

Consider $\mathsf{OptL}[\log n]$, the restriction of OptL where the function values are bounded by a polynomial. Àlvarez and Jenner considered this restriction and showed that $\mathsf{OptL}[\log n] = \mathsf{FL}^{\mathsf{NL}}[\log n]$ [AJ93]. Tantau showed that "given a directed graph G and two nodes s and t, computing the length of the shortest path from s to t" is complete for $\mathsf{OptL}[\log n]$ [Tan03].

Here we define a new unambiguous function class $\mathsf{UOptL}[\log n]$ (unambiguous OptL: the minimum is output on a unique computation path) and ask the following question: can we investigate the notion of min-uniqueness in the context of complexity classes? We show that $\mathsf{NL} = \mathsf{UL}$ is equivalent to the question whether $\mathsf{OptL}[\log n] = \mathsf{UOptL}[\log n]$.

SPL, the 'gap' version of UL, is an interesting log-space class first studied in [ARZ99]. The authors showed that the PM-DECISION is contained in a non-uniform version of SPL. They also show that SPL is powerful enough to contain FewL. We show that $\mathsf{UOptL}[\log n] \subseteq \mathsf{FL}^{\mathsf{SPL}}[\log n]$. Thus any language that is reducible to $\mathsf{UOptL}[\log n]$ is in the complexity class SPL. This contrasts with the equivalence $\mathsf{OptL}[\log n] = \mathsf{FL}^{\mathsf{NL}}[\log n]$. We also show that the class LogFew reduces to

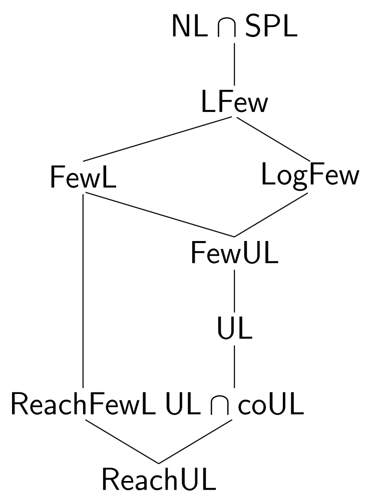


Figure 6.1: Relations known before.

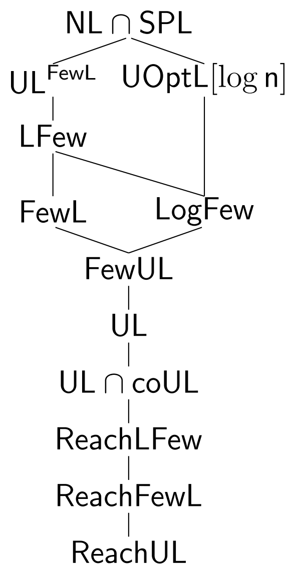


Figure 6.2: New relations.

$\mathsf{UOptL}[\log n]$ (refer to the next section for the definition of LogFew).

Finally, we also observe that $\mathsf{UOptL}[\log n]$ is contained in $\mathsf{FL}^{\mathsf{promiseUL}}$. A very interesting open question is to show that FewL reduces to $\mathsf{UOptL}[\log n]$.

Figures 6.1 and 6.2 depict the relations among various unambiguous and 'few' classes known before and new relations that we establish in this chapter, respectively. Definitions of these complexity classes are given in subsequent sections.

Unambiguous hierarchies

Since it is not known whether UL is closed under complement, it is interesting to investigate ULH; the unambiguous log-space hierarchy over UL. We first consider the hierarchy over $\mathsf{UOptL}[\log n]$ and then show that the $\mathsf{UOptL}[\log n]$ hierarchy collapses: $\mathsf{UOptL}[\log n]^{\mathsf{UOptL}[\log n]} \leq \mathsf{UOptL}[\log n]$. Since $\mathsf{UL} \leq \mathsf{UOptL}[\log n]$ (with relativization) it follows from this collapse result and the result that $\mathsf{UOptL}[\log n] \subseteq \mathsf{FL}^{\mathsf{promiseUL}}$, in fact $\mathsf{ULH} \subseteq \mathsf{L}^{\mathsf{promiseUL}}$.

We use a combination of existing techniques for proving our results.

6.2 Log-space complexity classes

We assume familiarity with the basics of complexity theory and in particular the log-space bounded complexity class NL. We call a nondeterministic log-space machine an NL machine. For an NL machine M, recall the definitions of $acc_M(x)$, $rej_M(x)$ and $gap_M(x)$ from Chapter 2.

We are interested in various restrictions of NL machines with few accepting paths. In the literature (for instance [BJLR91, BDHM92, AJ93, ARZ99]) various versions of unambiguity and fewness have been studied. We first define them all here.

Definition 6.2.1. (Unambiguous machines) A nondeterministic log-space machine M is

- *reach-unambiguous* if for any input and for any configuration c, there is at most one path from the start configuration to c. (The prefix 'reach' in the term indicates that the property should hold for all configurations reachable from the start configuration.)

- *unambiguous* if for any input there is at most one accepting path.

- *weakly unambiguous* if for any input and any accepting configuration c there is at most one path from the start configuration to c.

Definition 6.2.2. (Unambiguous classes)

- ReachUL – class of languages that are decided by reach-unambiguous machines with at most one accepting path on any input.

- UL – class of languages that are decided by unambiguous machines.

- FewUL – class of languages that are decided by weakly unambiguous machines.

- LogFew – class of languages L for which there exists a weakly unambiguous machine M and a log-space computable predicate R such that $x \in L$ if and only if $R(x, acc_M(x))$ is true.

We could define a 'reach' version of FewUL. But that coincides with $\mathsf{ReachUL}$ as shown in [BJLR91]. The following containments are easy: $\mathsf{ReachUL} \subseteq \mathsf{UL} \subseteq \mathsf{FewUL} \subseteq \mathsf{LogFew}$. It is also known that FewUL is $\mathsf{L}_d(\mathsf{UL})$ (log-space disjunctive truth-table closure of UL) [BJLR91].

By relaxing the unambiguity condition to a polynomial bound on the number of paths, we get analogous 'few' classes.

We are interested in graphs with a bound on the number of paths. We use the following notation due to Buntrock, Hemachandra and Siefkes [BHS93] to quantify *ambiguity* in a graph.

Definition 6.2.3. For a directed acyclic graph G and a node s, we say G is k-ambiguous with respect to s, if for any node v, the number of paths from s to v is bounded by k.

Definition 6.2.4. (Few machines) A nondeterministic log-space machine M is a

- *reach-few* machine if there is a polynomial p so that on any input x the configuration graph of M on x is $p(|x|)$-ambiguous with respect to the start configuration.

- *few* machine if there is a polynomial p so that on any input x there are at most $p(|x|)$ accepting paths.

Definition 6.2.5. (Few classes)

- $\mathsf{ReachFewL}$ – class of languages that are decided by reach-few machines.

- $\mathsf{ReachLFew}$ – class of languages L for which there exists a reach-few machine M and a log-space computable predicate R such that $x \in L$ if and only if $R(x, acc_M(x))$ is true.

- FewL – class of languages that are decided by few-machines.

- LFew – class of languages L for which there exists a few machine M and a log-space computable predicate R such that $x \in L$ if and only if $R(x, acc_M(x))$ is true.

As mentioned earlier, $\mathsf{ReachFewL}$ is the same class as $\mathsf{NspaceAmb}(\log n, n^{O(1)})$ defined in [BHS93]. In [BJLR91], the authors observe that $\mathsf{ReachFewL} \subseteq \mathsf{LogDCFL}$. This is because a depth first search of a reach-few machine can be implemented in $\mathsf{LogDCFL}$.

The following containments follow from the definitions: $\mathsf{ReachFewL} \subseteq \mathsf{FewL} \subseteq \mathsf{LFew}$. It is also clear that all the above-defined classes are contained in LFew and it is shown in [ARZ99] that $\mathsf{LFew} \subseteq \mathsf{NL}$. Thus all these classes are contained in NL. We also consider the class SPL – the 'gap' version of UL. SPL is contained in $\oplus\mathsf{L}$ (in fact all 'mod' classes) and it is big enough to contain LFew [ARZ99]. A nonuniform version of SPL contains the matching problem [ARZ99]. We use the facts that $\mathsf{L}^{\mathsf{UL}\cap\mathsf{coUL}} = \mathsf{UL} \cap \mathsf{coUL}$ and $\mathsf{FUL}^{\mathsf{UL}\cap\mathsf{coUL}} = \mathsf{FUL}$ in our chapter. The proof of these uses standard techniques, refer to [TW10] for a proof of the former equivalence and the latter equivalence can be shown similarly.

We will use *metric reductions* for functional reducibility. A function f is log-space metric reducible to function g, if there are log-space computable functions h_1 and h_2 so that $f(x) = h_1(x, g(h_2(x)))$.

6.3 Reachability in graphs with few paths

In this section we show new upper bounds on the space needed by an unambiguous machine for reachability problems over graphs with a polynomial number of paths. Our main technical tool is the following theorem due to Reinhardt and Allender.

The following theorem from [RA00] states that the reachability problem can be solved unambiguously for classes of graphs that are min-unique with respect to the start vertex. Moreover, we can also check whether a graph is min-unique unambiguously. We give a proof of a version of Theorem 6.3.1 in Chapter 2.

Theorem 6.3.1 ([RA00]). *There is an unambiguous nondeterministic log-space machine M that given a directed graph G and two vertices s and t as input, does the following:*

1. *If G is not min-unique with respect to s, then M outputs 'not min-unique' on a unique path.*
2. *If G is min-unique with respect to s, then M accepts on a unique path if there is a directed path from s to t, and rejects on a unique path if there are no paths from s to t.*

We can also define the notion of min-uniqueness for weighted graphs. But this is equivalent to the above definition for our purposes if the weights are positive and polynomially bounded as we can replace an edge with weight k, with a path of length k. In fact we will some times use this definition for weighted graphs without explicitly mentioning it. Thus for showing that $\mathsf{NL} = \mathsf{UL}$ it is sufficient to come up with a positive and polynomially bounded weight function that is UL-computable and makes a directed graph min-unique with respect to the start vertex. For graphs with a polynomial number of paths, we can use known hashing techniques to make the graph min-unique. In particular, we use the hashing theorem due to Fredman, Komlós and Szemerédi based on primes [FKS84]. The statement of the theorem is given in Chapter 2 in Theorem 2.3.4.

All our upper bounds in this section are based on the following theorem.

Theorem 6.3.2. *For any polynomial $q(n)$, there is a nondeterministic log-space bounded Turing machine M so that, for any reachability instance $\langle G, s, t\rangle$, if G is $q(n)$-ambiguous with respect to s, then M will output the number of paths from s to t on a unique path (all other paths rejects).*

Proof. First we show that the reachability question in a $q(n)$-ambiguous graph can be decided in an unambiguous manner. We do this by making such graphs min-unique with respect to s and applying Theorem 6.3.1.

Theorem 6.3.3. *For any polynomial $q(n)$, there is a nondeterministic log-space bounded Turing machine M so that, for any reachability instance $\langle G, s, t\rangle$, if G is $q(n)$-ambiguous with respect to s, then M will accept on a unique path if there is a path from s to t. If there are no s to t paths, M will reject on a unique path. All other paths will output '?'.*

Proof. (*of Theorem 6.3.3*). Let $\langle G, s, t\rangle$ be an instance of reachability. Consider the edges of G in the lexicographical order. For the i^{th} edge give a weight 2^i. This is a very good weight function that assigns every path with unique weight. The problem is that it is not polynomially bounded. We will give a polynomial number of weight functions that are log-space computable and polynomially bounded so that for one of them G will be min-unique with respect to s. Since by Theorem 6.3.1 it is possible to check whether a given weight function makes the graph min-unique using a UL$\cap$coUL computation, we can go through each weight function sequentially. Let p_j be the j^{th} prime number. Then the j^{th} weight function (for $1 \leq j \leq q'(n)$ for an appropriate polynomial q' dictated by Theorem 2.3.4) is $w_j(e_i) = 2^i(\mod p_j)$. It follows from Theorem 2.3.4 that under some w_j all paths from s to t will have different weights. Hence the graph is min-unique under this weight function. □

(*Proof of Theorem 6.3.2 cont.*) Let $\mathcal{G}$ be the class of weighted graphs which are $q(n)$-ambiguous with respect to a fixed vertex s, such that every path starting at s has a distinct weight. Let A be the 'promise language' consisting of tuples (G, s, t, i), given the promise that $G \in \mathcal{G}$ such that there exists a path of length i from s to t. In particular, such a graph G is min-unique. Note that A is in promiseUL,[1] that is, there exists an NL machine that has zero or one accepting path on every input that satisfies the promise. Also note that, given a $q(n)$-ambiguous graph G, with respect to one of the weight functions defined in Theorem 6.3.3, G is in $\mathcal{G}$.

In the above proof, a 'good' weight function actually does more than making the graph min-unique: it makes weights of every path distinct. With this stronger property we can count the number of paths by making queries of the form "is there a path of length i from s to t" to the language A, for all $i \leq N$ and by counting the number of positive answers (where N is the maximum weight possible and is bounded by a polynomial). It is important to observe that whenever we make a query to A, the query does satisfy the necessary promise.

[1]We define promiseUL later in Definition 6.4.4

But among polynomially many weight functions we have to reject those that do not give distinct weights to paths from s to t. Theorem 6.3.1 can only be used to reject weight functions that do not make the graph min-unique. It is possible that some weight function makes the graph min-unique with respect to s but the graph may still have two paths from s to t of the same weight. We use the unambiguous machine for deciding reachability in order to check this more strict condition.

Let G' denote the standard layered graph of G: G' will have n layers. For a vertex u of G there will be copy u_i in the i^{th} layer of G'. There is an edge from u_i in the i^{th} layer to v_{i+1} in the $(i+1)^{th}$ layer if (u, v) is an edge in G. Notice that no new paths are added in this layered graph. The following claim is straightforward to see.

Claim 6.3.4. *If G is k-ambiguous, then G' is also k-ambiguous. Moreover, there is an s to t path of length i in G if and only if there is an s_1 to t_i path in G'.*

Hence deciding reachability in G' can also be done unambiguously and checking whether G has an s to t path of weight exactly i can be done by reachability from s_1 to t_i in G'.

In order to check whether w is a 'bad' weight function, we need to check whether there are two paths from s to t of the same weight. We can use the following equivalence for checking this condition. α and β are integer values bounded by a polynomial.

$$\begin{aligned} w \text{ is bad} \quad \Leftrightarrow \quad & \exists\alpha\exists(e = uv)\exists\beta[\exists \text{ a path of length } \beta \text{ from } s \text{ to } u] \wedge [\exists \text{ a path of weight} \\ & \alpha - w(e) - \beta \text{ from } v \text{ to } t] \wedge [\exists \text{ a path of length } \alpha \text{ from } s \text{ to } t \text{ in } G - \{e\}] \end{aligned}$$

Note that the total number of possible values of α, β and e are bounded by polynomials in n. Thus we can decide whether a weight function w is 'bad' or not by making polynomially many reachability queries (that is for each choice of α, β and e). Once we get a good weight function w, we can again use reachability queries to compute the number of distinct paths from s to t using a deterministic log-space machine. Now using the fact that $\mathsf{L}^{\mathsf{UL}\cap\mathsf{coUL}} = \mathsf{UL} \cap \mathsf{coUL}$ [TW10] we get the desired result. This proves Theorem 6.3.2. $\square$

Theorem 6.3.5. *Let $L \in \mathsf{ReachFewL}$ be accepted by a reach-few machine M. Then the $\#L$ function $acc_M(x)$ is computable in $\mathsf{FL}^{\mathsf{UL} \cap \mathsf{coUL}}$.*

Proof. By definition, the configuration graph of a machine accepting a $\mathsf{ReachFewL}$ language is $q(n)$-ambiguous for some fixed polynomial q. □

Corollary 6.3.6. $\mathsf{ReachFewL} \subseteq \mathsf{ReachLFew} \subseteq \mathsf{UL} \cap \mathsf{coUL}$

Our method can be used to show that $\mathsf{NspaceAmb}(s(n), a(n)) \subseteq \mathsf{Uspace}(s(n) + \log a(n))$. This substantially improves the earlier known upper bound in [BHS93] that
$\mathsf{NspaceAmb}(s(n), a(n)) \subseteq \mathsf{Uspace}(s(n) \log a(n))$.

Theorem 6.3.7. *For a space bound $s(n) \geq \log n$ and ambiguity parameter $a(n)$ computable in space $s(n)$ so that $a(n) = 2^{O(s(n))}$, $\mathsf{NspaceAmb}(s(n), a(n)) \subseteq \mathsf{Uspace}(s(n) + \log a(n))$.*

Theorem 6.3.8. *Let $L \in \mathsf{FewL}$ be accepted by a few-machine M. Then the $\#L$ function $acc_M(x)$ is computable in $\mathsf{FUL}^{\mathsf{FewL}}$.*

Proof. For an input x, let G denote the configuration graph of $M(x)$ and let c_s be the start configuration and c_t be the unique accepting configuration. Let p be the polynomial bounding the number of paths from c_s to c_t. First we will present an $\mathsf{FL}^{\mathsf{FewL}}$ computation that outputs a graph H that is $p(n)$-ambiguous with respect to c_s which preserves the number of paths from c_s to c_t. Combining this reduction with the unambiguous machine from Theorem 6.3.2 we will get the required upper bound.

We say a configuration c is *useful* if it is in some c_s to c_t path and c is *useless* if it is not useful. In the reduced graph H, we will remove all the useless nodes from G, and the edges incident on them. Clearly, all the c_s-to-c_t paths in G will be preserved in H. Moreover, H will be $p(n)$-ambiguous.

We will design a FewL language for detecting whether a configuration is useful or not. For a configuration $c \in G$, consider the following graph G_c. Take two copies of G: G_1 and G_2. In G_1 delete all the outgoing edges from c. In G_2, delete all the incoming edges to c. Now add a directed edge from the copy of c in G_1 to the copy of c in G_2 to get a single graph H. The following claim is easy to verify.

Claim 6.3.9. *c is useless if and only if there is a path from c_s to c_t in G'_x. Moreover, if c is useful, then the number of paths from c_s to c_t is at most $p(n)$.*

Thus the following language $L' = \{(x, c) \mid \text{ there is a path from } c_s \text{ to } c_t \text{ in } H_{x,c}\}$ is in FewL. The log-space machine, for each configuration c, checks whether c is useful or not by querying L' and delete it from G if it is useless. The output graph G' will not have any useless nodes and hence will be $p(n)$-ambiguous. □

As a corollary, we get the following new upper bound on the complexity class LFew. Earlier it was known to be in $\mathsf{NL} \cap \mathsf{SPL}$ [ARZ99].

Corollary 6.3.10. $\mathsf{LFew} \subseteq \mathsf{UL}^{\mathsf{FewL}}$

Similar ideas together with the fact that planar reachability is in $\mathsf{UL} \cap \mathsf{coUL}$ [BTV09] also gives the following upper bound on couting the number of paths in planar directed acyclic graphs with a polynomial bound on the number of paths. This improves the upper bound of UAuxPDA for this problem given by Limaye, Mahajan and Nimbhorkar [LMN10].

Theorem 6.3.11. *For any polynomial p, there is an unambiguous machine M that given a planar directed acyclic graph G and two nodes s and t as input, (a) outputs the number of s to t paths if the number of such paths are bounded by $p(n)$ (b) outputs "more than $p(n)$ paths" if there are more than $p(n)$ s to t paths.*

Proof. Consider the edge weight functions defined in Theorem 6.3.3. With respect to each weight function, we can check whether the number of s to t paths is bounded by $p(n)$ in an unambiguous manner by considering two cases: (i) if none of the weight functions are good, which can be checked unambiguously, then clearly there are more than $p(n)$ number of paths, (ii) if a weight function is good, then we can in fact count the number of paths from s to t and see if it is greater than $p(n)$ by an unambiguous algorithm that makes reachability queries (in $\mathsf{UL} \cap \mathsf{coUL}$ for planar graphs). Since $\mathsf{FUL}^{\mathsf{UL} \cap \mathsf{coUL}}$ is in FUL the theorem follows. $\square$

6.4 Complexity of min-uniqueness

Let f be a polynomially bounded, positive-valued, edge weight function (that is a function that takes an edge as input and outputs an integer which we call the weight of the edge). Then by an abuse of notation, for any directed graph G, we shall denote $f(G)$ to be the weighted directed graph obtained by taking every edge e in G and replacing it with the weighted edge having weight $f(e)$.

Theorem 6.3.1 states that min-uniqueness is sufficient for showing $\mathsf{NL} = \mathsf{UL}$. Next we prove that if $\mathsf{NL} = \mathsf{UL}$ then there is a UL-computable weight function that makes any directed acyclic graph min-unique with respect to the start vertex. Thus min-uniqueness is necessary and sufficient for showing $\mathsf{NL} = \mathsf{UL}$.

Theorem 6.4.1. $\mathsf{NL} = \mathsf{UL}$ *if and only if there is a polynomially-bounded* UL*-computable weight function f so that for any directed acyclic graphs G, $f(G)$ is min-unique with respect to s.*

Proof. The reverse direction follows from the above theorem due to Reinhardt and Allender. For the other direction the idea is to compute a spanning tree of G rooted at s using reachability queries. Since NL is closed under complement, under the assumption that $\mathsf{NL} = \mathsf{UL}$, reachability is in UL (since $\mathsf{UL} = \mathsf{coUL}$ under the above assumption). Thus the following language $A = \{(G, s, v, k) \mid$ there is a path from s to v of length $\leq k\}$ is in UL.

The tree can be described as follows. We say that a vertex v is in level k if the minimum length path from s to v is of length k. A directed edge (u, v) is in the tree if for some k (1) v is in level k (2) u is the lexicographically first vertex in level $k - 1$ so that (u, v) is an edge.

It is clear that this is indeed a well defined tree and deciding whether an edge $e = (u, v)$ is in this tree is in $\mathsf{L}^A \subseteq \mathsf{UL}$.

Now for each edge in the tree give a weight 1. For the rest of the edges give a weight n^2. It is clear that the shortest path from a vertex with respect to this weight function is min-unique with respect to s and it is computable using a UL-transducer. $\square$

Àlvarez and Jenner [AJ93] define OptL as the log-space analog of Krental's OptP [Kre88]. They show that OptL captures the complexity of some natural optimization problems in the log-space setting (e.g. computing lexicographically maximum path of length at most n from s to t in a directed graph). They also consider $\mathsf{OptL}[\log n]$ where the function values are bounded by a polynomial (hence has $O(\log n)$ bit representation). Here we revisit the class OptL [AJ93] and study it in relation to the notion of min-uniqueness.

Definition 6.4.1. Let M be an NL-transducer. An output on a computation path of M is valid if it halts in an accepting state. For any input x, $opt_M(x)$ is the minimum value over all valid outputs of M on x. If all the paths reject, then $opt_M(x) = \infty$. Further, M is called *min-unique* if for all x either $M(x)$ rejects on all paths or $M(x)$ outputs the minimum value on a unique path.

Definition 6.4.2. A function f is in OptL if there exists a NL-transducer M so that for any x, $f(x) = opt_M(x)$. A function f is in UOptL if there is a min-unique nondeterministic transducer M so that for any x, $f(x) = opt_M(x)$. Define $\mathsf{OptL}[\log n]$ and $\mathsf{UOptL}[\log n]$ as the restriction of OptL and UOptL where the output of the transducers are bounded by $O(\log n)$ bits.

Remark 6.4.3. We can also define the class $\mathsf{OptL}[\log n]$ (similarly $\mathsf{UOptL}[\log n]$) in terms of a function that gives the maximum value over all valid outputs of M on x, instead of the minimum value. It is easy to see that the two classes (corresponding to maximum and minimum) of functions we thus obtain are equivalent via a log-space reduction by subtracting the value of the $f(x)$ from a sufficiently large polynomial whose value is greater than the maximum possible output value of f. For the sake of convenience, we use both the notions in this paper.

We next observe that if we restrict the output to be of $O(\log n)$ bits, the classes OptL and UOptL coincide if and only if $\mathsf{NL} = \mathsf{UL}$.

We will need the following proposition shown in [AJ93]. $\mathsf{FL}^{\mathsf{NL}}[\log n]$ denotes the subclass of $\mathsf{FL}^{\mathsf{NL}}$ where the output length is bounded by $O(\log n)$.

Proposition 6.4.2 ([AJ93])**.** $\mathsf{OptL}[\log n] = \mathsf{FL}^{\mathsf{NL}}[\log n]$.

Theorem 6.4.3. $\mathsf{OptL}[\log n] = \mathsf{UOptL}[\log n]$ *if and only if* $\mathsf{NL} = \mathsf{UL}$.

Proof. $\mathsf{NL} = \mathsf{UL} \Rightarrow \mathsf{OptL}[\log n] = \mathsf{UOptL}[\log n]$: Since NL is closed under complement, if $\mathsf{NL} = \mathsf{UL}$ then $\mathsf{NL} = \mathsf{UL} \cap \mathsf{coUL}$. Hence $\mathsf{OptL}[\log n] = \mathsf{FL}^{\mathsf{NL}}[\log n] = \mathsf{FL}^{\mathsf{UL}\cap\mathsf{coUL}}[\log n]$. For a function $f \in \mathsf{OptL}$, let M be the FL machine that makes queries to a language $L \in \mathsf{UL} \cap \mathsf{coUL}$ and computes f. Let N be the unambiguous machine that decided L. The min-unique transducer M' will simulate M and whenever a query y is made to L, it will simulate N on y and continue only on the unique path where it has an answer. In the end M' will output the value computed by M on a unique path.

$\mathsf{OptL}[\log n] = \mathsf{UOptL}[\log n] \Rightarrow \mathsf{NL} = \mathsf{UL}$: Let $L \in \mathsf{NL}$. Since NL is closed under complement, there is a nondeterministic machine M that on input x accepts on some path and outputs '?' on all other paths if $x \in L$, and rejects on some paths and outputs '?' on all other paths if $x \notin L$. We will show that $L \in \mathsf{coUL}$. Consider the NL-transducer which on input x simulates $M(x)$ and outputs 1 if M accepts and outputs 0 if M rejects and outputs a large value on paths with '?'. Let N be a min-unique machine that computes this OptL function. Thus if $x \notin L$ then $N(x)$ has a unique path on which it outputs 0 (and there may be paths on which it outputs 1). If $x \in L$ then there is no path on which it outputs 0. Now consider the machine N' that simulates N and if N outputs 0 then it accepts. For all other values N' rejects. Clearly this is an unambiguous machine that decides $\overline{L}$. □

As $\mathsf{UOptL}[\log n] \subseteq \mathsf{OptL}[\log n]$, $\mathsf{UOptL}[\log n]$ is in $\mathsf{FL}^{\mathsf{NL}}[\log n]$. Here we show that $\mathsf{UOptL}[\log n]$ can be computed using a SPL oracle. Thus if NL reduces to $\mathsf{UOptL}[\log n]$, then $\mathsf{NL} \subseteq \mathsf{SPL}$.

Theorem 6.4.4. $\mathsf{UOptL}[\log n] \subseteq \mathsf{FL}^{\mathsf{SPL}}[\log n]$

Proof. Let $f \in \mathsf{UOptL}[\log n]$ and let M be the min-unique NL-transducer that witnesses that $f \in \mathsf{UOptL}[\log n]$ and let p be the polynomial bounding the value of f. Consider the following language L:

$$L = \{(x, i) \mid f(x) = i \text{ and } i \leq p(|x|)\}.$$

We will show that $L \in \mathsf{SPL}$. Then in order to compute f a log-space machine will ask polynomially many queries (x, i) for $1 \leq i \leq p(n)$.

Consider the following machine N: on input x and $i \leq p(n)$, it simulates M on input x and accepts if and only if M halts with an output at most i. Let $g(x, i)$ be the number of accepting paths of N on input (x, i). Notice that for $i < f(x)$, $g(x, i) = 0$, for $i = f(x)$ then $g(x, i) = 1$, and for $i > f(x)$, $g(x, i) \geq 1$.

Now consider the GapL function $h(x, i) = g(x, i) \prod_{j=1}^{i-1}(1 - g(x, j))$ (to know more about closure properties of GapL functions see [AO94]). It follows that $h(x, i) = 1$ exactly when $f(x) = i$. For the rest of i, $h(x, i) = 0$. Thus $L \in \mathsf{SPL}$. □

An interesting question is whether FewL reduces to UOptL. We are not able to show this, but we show that the class LogFew reduces to UOptL.

Theorem 6.4.5. $\mathsf{LogFew} \leq \mathsf{UOptL}[\log n]$ *(under metric reductions)*

Proof. In this proof we define the class $\mathsf{UOptL}[\log n]$ as a maximization class (see Remark 6.4.3). Let L be a language in LogFew. Let M be a weakly unambiguous machine that decides L. Consider the NL-transducer N that on input x computes the number of accepting paths of $M(x)$: $N(x)$ guesses an integer l so that $1 \leq l \leq p(n)$ (where p is the polynomial bounding the number of accepting configurations) and then guesses l distinct accepting paths to l accepting configurations, in a lexicographically increasing order, and accepts and outputs l if all of them accept. Clearly N

outputs $acc_M(x)$ on exactly one computation path and all other paths that accept will have output less than $acc_M(x)$. □

Definition 6.4.4. (Promise classes)

- A *promise language* is a tuple (I_y, I_n), where I_y and I_n are disjoint subsets of $\{0,1\}^*$ (collectively known as the promise instances).
- A promise language (I_y, I_n) is said to be in $\mathsf{promiseUL}$ if there is an NL machine M, such that M has a unique accepting path for instances in I_y, and no accepting paths for instances in I_n, but could have any number of accepting paths for all other instances.
- A language A is said to be *consistent* with a promise language (I_y, I_n), if $x \in I_y \Longrightarrow x \in A$ and $x \in I_n \Longrightarrow x \notin A$.
- f is said to be in $\mathsf{FL}^{\mathsf{promiseUL}}$ if there exists a promise language (I_y, I_n) in $\mathsf{promiseUL}$ and a log-space transducer M such that for every language A consistent with (I_y, I_n), $f(x) = M^A(x)$ for all $x \in \{0,1\}^*$.

We next show that a function in $\mathsf{UOptL}[\log n]$ is also contained in $\mathsf{FL}^{\mathsf{promiseUL}}$.

Theorem 6.4.6. $\mathsf{UOptL}[\log n] \subseteq \mathsf{FL}^{\mathsf{promiseUL}}$

Proof. In this proof we define the class $\mathsf{UOptL}[\log n]$ as a maximization class (see Remark 6.4.3). Let f be a $\mathsf{UOptL}[\log n]$ function computed by a $\mathsf{UOptL}[\log n]$ machine. We will first define a $\mathsf{promiseUL}$ problem.

The instances of the problem are: $\langle M, x, k\rangle$ where M is a nondeterministic log-space bounded transducer, k is an integer and x is an input to M. The promise language (I_y, I_n) of 'Yes' and 'No' instances is defined as follows.

$$\begin{aligned} I_y &= \{\langle M, x, k\rangle \text{ so that } M \text{ is a } \mathsf{UOptL}[\log n] \text{ machine and } opt_M(x) = k\} \\ I_n &= \{\langle M, x, k\rangle \text{ so that } M \text{ is a } \mathsf{UOptL}[\log n] \text{ machine and } opt_M(x) < k\} \end{aligned}$$

Now an NL machine on input $\langle M, x, k\rangle$ simulates $M(x)$ and accepts if the output is k and rejects otherwise.

On I_y instances it accepts on a unique path. I_n instances it rejects on all paths.

Now consider a $\mathsf{UOptL}[\log n]$ function computed by a machine M. An FL machine asks queries $\langle M, x, k\rangle$ starting from the largest possible k and comes down until it gets a yes answer at which point it outputs that k. The FL machine is only asking queries in the promised region. $\square$

6.5 Unambiguous hierarchies

Since UL is not known to be closed under complement, it is interesting to study the complexity class hierarchy over UL which can be defined in natural way: $\mathsf{ULH}_1 = \mathsf{UL}$ and $\mathsf{ULH}_{(i+1)} = \mathsf{UL}^{\mathsf{ULH}_i}$. Then $\mathsf{ULH} = \bigcup_i \mathsf{ULH}_i$. We show that $\mathsf{ULH} \subseteq \mathsf{L}^{\mathsf{promiseUL}}$. For showing this we in fact first show that the hierarchy over $\mathsf{UOptL}[\log n]$ collapses. We then use the fact that $\mathsf{UOptL}[\log n] \subseteq \mathsf{FL}^{\mathsf{promiseUL}}$ from Theorem 6.4.6.

We assume RST-relativizations when dealing with nondeterministic log-space oracle classes. When the machine enters the query state it behaves deterministically until the entire query is written. One important consequence of this is that, since the number of configurations of a log-space machine is polynomial, the total number of queries that such a machine can make on any input is polynomially bounded. Moreover, the set of all potential queries that can be asked by a nondeterministic machine on any specific input can be computed by a deterministic log-space machine.

Theorem 6.5.1. $\mathsf{UOptL}[\log n]$ *hierarchy collapses. That is,* $\mathsf{UOptL}[\log n]^{\mathsf{UOptL}[\log n]} \leq \mathsf{UOptL}[\log n]$ *under metric reductions.*

Proof. We use an enhanced census technique, similar to the ones that are used to prove collapses of hierarchies over log-space classes [Hem89, SW88, Ogi95, ABO99]. But since we are dealing with function classes we need to be a bit more careful. Also, in this proof we define the class $\mathsf{UOptL}[\log n]$ as a maximization class (see Remark 6.4.3).

Let f be a $\mathsf{UOptL}[\log n]$ function computed by an oracle machine M making oracle calls to a $\mathsf{UOptL}[\log n]$ function g. Let N be a $\mathsf{UOptL}[\log n]$ machine computing g. We will show that f reduces to a $\mathsf{UOptL}[\log n]$ function h. An important consideration (which makes the proof a bit more complicated) is that f and g could be partial and on the inputs where the value is not defined, the corresponding machines reject on all paths (and hence do not have the unambiguous behavior).

Consider the computation of $M(x)$. Let $Q_x = \{q_1, \ldots, q_{n^c}\}$ be all the potential queries of $M(x)$ to the function g. Let $D_x = \{q \mid q \in Q_x$ and $g(q)$ is defined$\}$. Let $S_x = \sum_{q \in D_x} g(x)$. That is, S_x is the sum of all the values of the function g on queries on which g is defined. This value is polynomially bounded.

Consider the function h, which has two components, defined as $h(x) = \langle S_x, f(x) \rangle$ (obtained by concatenating S_x and $f(x)$). Clearly given x and $h(x)$, we can decode $f(x)$ in log-space and hence $f \leq h$. We will show that $h(x)$ is a $\mathsf{UOptL}[\log n]$ function.

Consider the following nondeterministic transducer M_h which operates in two phases, on input x. In the first phase, on input x, M_h tries to compute the sum S_x. Towards this M_h initializes a sum $S = 0$ and guesses a subset $A \subseteq Q_x$ of potential queries and simulates N on this subset: that is, M_h generates the potential queries $q \in Q_x$ one by one, for each of q, it guesses 0 or 1. If it guessed 0 (meaning g is not defined) it goes to the next query. If the guess is 1 then it guesses a computation path ρ of N on q. If the path rejects, M_h rejects. Otherwise it updates $S = S + \rho_N(x)$ (that is, it adds the value computed by $N(q)$ on this path ρ to S). We claim that after the first phase, M_h will have computed S_x on a unique path, and all other paths the value computed will be less than S_x. M_h will output this sum S as the first component of the function. Since the highest value S_x is output on a unique path, for the second phase we will ignore the computation on any path that is a continuation of the paths that compute a value $S < S_x$.

In the second phase M_h will start simulating $M(x)$. For each query q asked by M, M_h will simulate the answer as follows (if $g(q)$ is defined, then M_h could have just guessed a path of N and continued; but a problem arises on queries for which the oracle function is undefined and hence the computation does not have an unambiguous behavior; we need to take care of this). M_h again guesses a subset A of queries as in phase one and for each of the queries in A, it also guesses a computation path ρ, of N and computes the sum S of values for each of the queries. It continues the computation only on the unique path where $S = S_x$. For this path if q is not in A, then M_h treats the answer to the query as "not defined" and continues. If q is in A then M_h treats the value computed by $N(q)$ on the path ρ as answer to the oracle query q. Finally, M_h outputs $\langle S_x, v\rangle$ where v is the value that N computes on a path. $\square$

Corollary 6.5.2. $\mathsf{ULH} \subseteq \mathsf{L}^{\mathsf{promiseUL}}$

Proof. Follows from Theorem 6.4.6 and Theorem 6.5.1. $\square$

6.6 Conclusion

In this chapter, we proved certain important results on log-space unambiguity. We established that min-uniqueness is *the* only way to show that $\mathsf{NL} = \mathsf{UL}$. Our results raised some very intriguing questions as well. We showed that $\mathsf{ReachFewL} \subseteq \mathsf{UL}$, but is FewL also in UL? Or at the very least, is FewL in $\mathsf{UOptL}[\log \mathsf{n}]$? Also is UL closed under complement?

Chapter 7

Bipartite Planar Matching in Unambiguous Logarithmic Space

In this chapter we study the perfect matching and some related problems in bipartite planar graphs. We give a brief introduction of the problems that we study in Section 7.1. In Section 7.2 we state the necessary definitions and terminology that we use in the rest of this chapter. In Section 7.3, we present the algorithms to decide the existence of a perfect matching and compute one if it exists, in a bipartite planar graph. In Section 7.4, we show that the algorithms discussed in Section 7.3 can be done in unambiguous log-space. In Section 7.5 we study some related problems and establish certain upper bounds on them as well. In particular we show that the even path problem is in UL.

7.1 Introduction

Historically, matching problems have been occurring as central problems in algorithms and complexity theory. Edmonds *blossom* algorithm [Edm65] for computing a maximum cardinality matching (MAX-MATCH) was one of the first examples of a non-trivial polynomial time algorithm. It had a considerable share in initiating the study of *efficient computation*, including the class P it-self; Valiant's #P-hardness [Val79] for counting perfect matchings in bipartite graphs provided surprising insights into the counting complexity classes. The question of whether or not a graph has a perfect matching is also well-studied (we denote this problem as PM). The study of whether or not PM is well parallelizable has yielded powerful tools such as *isolating lemma* [MVV87] that have found numerous applications elsewhere.

The rich combinatorial structure of matching problems combined with their potential to serve as central problems in the field invites their study from several perspectives. The focus of this chapter is on the space complexity of the matching problems. The best known upper bound for PM (and other matching problems mentioned above) is *non-uniform* SPL [ARZ99] whereas the best hardness known is NL-hardness [CSV84].

7.1.1 Matching problems in planar graphs

A well known example where planarity is a boon is that of counting perfect matchings. The problem in planar graphs is in P [Kas67] as opposed to being #P-hard in general graphs [Val79]. Counting perfect matchings in planar graphs can in fact be done in NC [Vaz88]; thus PM-DECISION in planar graphs is in NC. In contrast, constructing a perfect matching (PM-CONSTRUCT) in NC remains an outstanding open question, whereas the bipartite planar case is known to be in NC [MN89].

The space complexity of matching problems in planar graphs was first studied by Datta, Kulkarni, and Roy [DKR10] where it was shown that finding a minimum weight perfect matching (MIN-WT-PM) in bipartite planar graphs is in SPL. This result was recently generalized to bounded genus bipartite graphs by Datta, Kulkarni, Tewari, and Vinodchandran [DKTV11]. Kulkarni [Kul09] shows that MIN-WT-PM in planar graphs (not necessarily bipartite) is NL-hard. The only known hardness for PM in planar graphs is L-hardness (see for instance [DKLM10]). For bipartite planar graphs, nothing better than L-hardness is known for any matching problem.

Table 7.1: Space Complexity of Matching Problems in Planar Graphs

Problem in Planar Graphs	**Upper Bound**	**Hardness**
PM-CONSTRUCT	PSPACE [1]	L
MAX-MATCH	PSPACE	L
MIN-WT-PM	$L^{C_=L}$ [Kas67]	NL [Kul09]
PM-DECISION	$L^{C_=L}$ [Kas67]	L
bipartite MAX-MATCH	$L^{C_=L}$ [Hoa10]	L
bipartite MIN-WT-PM	SPL [DKR10]	L
bipartite HALL-OBS (CONSTRUCT)	NL (new)	L
bipartite HALL-OBS (DECISION)	coUL (new)	L
bipartite PM	UL (new)	L
bipartite UPM	UL (new)	L [DKLM10]

Recall the problem of reachability in directed graphs. We know that directed reachability is NL-complete. It turns out that reachability in planar graphs reduces (in log-space) to PM in bipartite planar graphs [DKLM10]; the former was proved to be in UL $\cap$ coUL by Bourke, Tewari, and Vinodchandran [BTV09]. In this chapter we show that the latter is in UL, leaving the coUL bound as an intriguing open question. Table 7.1.2 records the current knowledge (including the current results) about the space complexity of matching problems in planar graphs.

7.1.2 The Hall-obstacle problem

Hall's Theorem (see for instance [LP86]) asserts that a bipartite graph $G = (A \cup B, E)$ has a perfect matching if and only $|A| = |B|$ and for every $S \subseteq A : |N(S)| \geq |S|$, where $N(S) := \{v \in B \mid \exists u \in A : (u, v) \in E\}$. A *Hall-obstacle* in a bipartite graph $G = (A \cup B, E)$ is a set $S \subseteq A$ such that $|N(S)| < |S|$. We consider the following computational problems related to Hall-obstacle:

- HALL-OBS (DECISION) : decide if a bipartite G contains a Hall-obstacle.

- HALL-OBS (CONSTRUCT) : construct a Hall-obstacle in a bipartite G (if exists).

7.1.3 Our results

Theorem 7.1.1. *In bipartite planar graphs:*

(a) PM-DECISION *and* PM-CONSTRUCT *are in* UL *(the latter is in the functional version of* UL*);*

(b) HALL-OBS *(*DECISION*) is in* coUL*;*

(c) HALL-OBS *(*CONSTRUCT*) is in* NL.

We build on two key algorithms: (i) Miller and Naor's algorithm [MN89] for perfect matching in bipartite planar graphs; (ii) Reinhardt and Allender's [RA00] UL algorithm for computing the shortest path in *min-unique* graphs. Miller and Naor reduce the PM-DECISION in planar graphs to the following problem in directed planar graphs: NEG-CYCLE (DECISION) problem - given a directed graph with polynomially bounded edge-weights, decide whether or not the graph contains a negative weight cycle. We observe that this reduction works in log-space and NEG-CYCLE problem is in NL. This immediately yields an NL bound for perfect matching in bipartite planar graphs. To see that the bound can be brought down to UL in part (a), we first provide a technical extension of (ii) when the graph does not contain any negative weight cycles (it may contain negative weight edges though). A simple but subtle combination of (i) and (ii) then yields the desired result. Part (b) is just the complement of PM-DECISION and hence trivially in coUL. For part (c), we argue that via a simple adaptation of Miller and Naor's algorithm, the NEG-CYCLE directly corresponds to a Hall-obstacle. To the best of our knowledge this is the first time a bound on the space complexity (and the parallel complexity) of constructing Hall-obstacle in bipartite planar graphs is being noted. As opposed to [KMV08] and [DKR10], our space bounded algorithms do not require determinant computation as a subroutine, instead we make use of a variant of planar reachability. However, for the weighted case we do not know how to improve upon the SPL bound in [DKR10].

Let EXACT-PM (DECISION) denote the problem of deciding, given an integer k, whether or not a graph G with edges colored Red or Blue contains a perfect matching with exactly k Red edges. This problem was first posed by Papadimitriou and Yannakakis [PY82]. It is known to be in

RNC [MVV87] but not known to be in P. We consider the following relaxation of the EXACT-PM problem: let EVEN-PM (DECISION) denote the problem of deciding whether or not a graph G with edges colored Red or Blue contains a perfect matching with even number of Red edges. In this chapter, we observe the following:

Theorem 7.1.2. *(a)* EVEN-PM *in bipartite graphs is in* P;
(b) EVEN-PM *in bipartite planar graphs is in* NL.

We also consider EVEN-PATH problem: deciding whether or not there is a directed path of even length between two specified vertices. EVEN-PATH is NP-complete [LP83] but restricted to planar graphs it is in P [Ned99]. In DAG, the problem is NL-complete. EVEN-PATH problem in planar DAG can be viewed as a relaxation of the following problem, which is NL-complete in planar DAG [Kul09]: RED-BLUE-PATH problem - given a directed graph with edges colored Red or Blue, decide whether or not there is a (simple) path between two specified vertices such that any two consecutive edges in the path are of two different colors. Here we show that:

Theorem 7.1.3. EVEN-PATH *in planar DAG is in* UL.

A motivation to study whether EVEN-PATH in planar DAG is in UL is the hope to develop new techniques to prove RED-BLUE-PATH in planar DAG is in UL, and thus NL $=$ UL. Indeed, our proof of the UL bound for EVEN-PATH in planar DAG combines two different *isolation* techniques ([BTV09], [Hoa10]) in a non-obvious way. In the process, we also give a simple log-space procedure to obtain generalized BTV weights (see Section 7.5.2) in planar graphs without going through any piecewise linear embedding of the graph. Our procedure is inspired by the subroutine of computing *pseudo-flow* in Miller and Naor's algorithm and it might be of independent interest.

7.2 Preliminaries

See for instance [Vol99] for definitions of standard complexity classes. It is known that $\mathsf{UL} \subseteq \mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{P}$ and $\mathsf{UL} \subseteq \mathsf{SPL}$ (See Proposition 2.2.1). It is also known that $\mathsf{SPL} \subseteq \oplus L \subseteq \mathsf{NC}$. As of now, NL and SPL as well as NL and $\oplus L$ are incomparable.

7.2.1 Flow terminology

An undirected *edge* is a two element unordered set $\{u, v\}$ such that $u, v \in V$. An undirected graph $G = (V, E)$ consists of a set V of *vertices* and a set E of *undirected edges*. An *arc* is an ordered tuple $(u, v) \in V \times V$. A directed graph $\overrightarrow{G} = (V, \overrightarrow{E})$ consists of a set V of *vertices* and a set $\overrightarrow{E} \subseteq V \times V$ of arcs. Given an undirected graph $G = (U, V)$, its directed version is a directed graph $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ where $\overleftrightarrow{E} := \{(u, v) \mid \{u, v\} \in E\}$.

Definition 7.2.1 (Capacity demand graph)**.** A *capacity-demand graph* is a triple (G, c, d) where $G = (V, E)$; every arc $(u, v) \in \overleftrightarrow{E}$ is assigned a real value $c(u, v)$ called the *capacity* of the arc and every vertex $v \in V$ is assigned a real value $d(v)$ called the *demand* at the vertex.

Definition 7.2.2. (Flow)

- A *pseudo-flow* in a capacity-demand graph (G, c, d) is a function $f : \overleftrightarrow{E} \to \mathbb{R}$ such that:

  1. for every arc $(u, v) \in \overleftrightarrow{E}$, we have:

     $$\text{(skew-symmetry) } f(u, v) = -f(v, u),$$

     and

  2. for every vertex $v \in V$, we have:

     $$\text{(demands met)} \quad \sum_{w \in V : (v,w) \in \overleftrightarrow{E}} f(v, w) = d(v).$$

- A *flow* in a capacity-demand graph (G, c, d) is a function $f : \overleftrightarrow{E} \to \mathbb{R}$ such that:

    1. f is a pseudo-flow in (G, c, d), and
    2. for every $(u, v) \in \overleftrightarrow{E}$, we have:

$$\text{(capacity constraints satisfied)}\ f(u, v) \leq c(u, v).$$

- A *zero-demand graph* (G, c) is a capacity-demand graph in which the demand at every vertex is zero.

It is important to note that, in this chapter we only study flows that have integral values. Therefore the term flow means *integral flow*, unless otherwise specified.

Definition 7.2.3 (Directed Dual)**.** Let G be an undirected planar graph. Fix an embedding of G in the plane. Note that, this induces an embedding on $\overleftrightarrow{G}$ where one can imagine the arc (u, v) lying on top of the arc (v, u). Let G^* denote the dual of G with respect to the embedding. The directed dual of G is the directed version of G^* denoted by $\overleftrightarrow{G^*}$. The arcs of $\overleftrightarrow{G}$ and that of $\overleftrightarrow{G^*}$ are in one to one correspondence as follows: an arc $(u, v) \in \overleftrightarrow{G}$ corresponds to the arc $(u^*, v^*) \in \overleftrightarrow{G^*}$, where u^* and v^* are the faces in $\overleftrightarrow{G}$ to the left and and right of the arc (u, v) respectively.

7.2.2 Important results

Below we state some results from earlier papers that we use to prove our theorems.

Proposition 7.2.1 (folklore, see for instance [Has81])**.** *Let (G, c) be a zero-demand graph. Let f be a flow in (G, c). If $C^* = (e_1^*, \ldots, e_k^*)$ is a directed cycle in $\overleftrightarrow{G^*}$, then*

$$\sum_{e: e^* \in C^*} f(e) = 0.$$

Proof. The cycle C^* defines a *cut*, $V = V_1 \cup V_2$, in G, where the vertices in V_1 lie inside C^* and the vertices in V_2 lie outside[2], and for every arc e_i^*, without loss of generality assume that the corresponding primal arc $e_i = (u_i, v_i)$ has $u_i \in V_1$ and $v_i \in V_2$. Since G is a planar embedding, the set of arcs $\{e_i\}_{i\in[k]}$ are the only cut edges from V_1 to V_2. Now,

$$\begin{aligned}
0 = \sum_{u\in V_1} d(v) &= \sum_{u\in V_1} \sum_{v\in V:(u,v)\in \overleftrightarrow{E}} f(u,v) \\
&= \sum_{u\in V_1} \left(\sum_{v\in V_1:(v,w)\in \overleftrightarrow{E}} f(u,v) + \sum_{v\in V_2:(v,w)\in \overleftrightarrow{E}} f(u,v) \right) \\
&= \sum_{u\in V_1} \sum_{v\in V_1:(v,w)\in \overleftrightarrow{E}} f(u,v) + \sum_{u\in V_1} \sum_{v\in V_2:(v,w)\in \overleftrightarrow{E}} f(u,v) \\
&= 0 + \sum_{u\in V_1} \sum_{v\in V_2:(v,w)\in \overleftrightarrow{E}} f(u,v) \\
&= \sum_{e:e^*\in C^*} f(e).
\end{aligned}$$

Thus the proposition follows. □

Lemma 7.2.2 ([MN89]). *If (G, c) be a zero-demand graph. There exists a flow in (G, c) if and only if $\overleftrightarrow{G^*}$ has no negative weight cycle with respect to weights c.*

Proof. Suppose there exists a flow f in (G, c). Let C^* be a negative weight cycle in $\overleftrightarrow{(G^*)}$ with respect to the weight function c. Then by Proposition 7.2.1 we have that

$$\begin{aligned}
0 &= \sum_{e:e^*\in C^*} f(e) \\
&\le \sum_{e:e^*\in C^*} c(e) \\
&= \sum_{e^*\in C^*} c(e^*) \\
&< 0
\end{aligned}$$

[2]Note that vertices in G correspond to faces in G^*.

Thus we get a contradiction which implies that $\overleftrightarrow{G^*}$ does not have any negative weight cycle.

To see the other direction, we construct a flow f in G, assuming that $\overleftrightarrow{G^*}$ does not have any negative weight cycle. Fix a vertex s^* in $\overleftrightarrow{G^*}$. For any vertex u^* in $\overleftrightarrow{G^*}$, define $\mathrm{dist}^c(s^*, u^*)$ to be the shortest distance of u^* from s^* with respect to the weight function c. Since $\overleftrightarrow{G^*}$ does not have any negative weight cycle, p is well defined and finite for any vertex reachable from s^*. Now for an arc (u, v) in G, define $f(u, v) = \mathrm{dist}^c(s^*, u^*) - \mathrm{dist}^c(s^*, v^*)$. By definition, $f(u, v) = -f(v, u)$. Also for any vertex $v \in G$, $d(v) = \sum_{(v,w)|(v,w)\in\overleftrightarrow{E}} f(v, w) = 0$ by Proposition 7.2.1, since the dual arcs corresponding to the set of outgoing arcs from v, constitute a cycle. Finally, since dist is a distance function, $\mathrm{dist}^c(s^*, v^*) \leq \mathrm{dist}^c(s^*, u^*) + c(u^*, v^*)$, which implies that $f(u, v) \leq c(u, v)$. Therefore f is a flow in G. □

7.3 Algorithms for the perfect matching problem in planar bipartite graphs

Let $G = (A \cup B, E)$ be a bipartite planar graph. Assume that $|A| = |B|$, otherwise G does not have a perfect matching. In this section we show that deciding if G has a perfect matching reduces to the problem of deciding if a polynomially weighted planar graph has a negative weight cycle. We also show how to construct a perfect matching in G, if one exists. Our techniques are based on the paper by Miller and Naor, which gives an NC algorithm for the problem [MN89]

We first show how to convert the graph G into a capacity-demand graph (G, c, d), so that there is a one-to-one correspondence between flows in (G, c, d) and perfect matchings in G. The conversion is a straightforward extension of the usual matching to flow reduction. The only difference is that, here we have multiple sources and sinks instead of a single source/sink, in order to preserve planarity.

Proposition 7.3.1. *Let $G = (A \cup B, E)$ be a bipartite planar graph. Let c be the capacity function* *defined as follows: for every edge $\{u,v\} \in E$ where $u \in A$ and $v \in B$, $c(u,v) = 1$ and $c(v,u) = 0$. Let d be the demand function defined as follows: for every vertex $v \in A$, $d(v) = 1$, and for every vertex $v \in B$, $d(v) = -1$. Then there exists a one-to-one correspondence between flows in the capacity-demand graph (G,c,d), and perfect matchings in G. Also given a flow f, the corresponding perfect matching M can be constructed in log-space.*

Proof. Let M be a perfect matching in G. For every edge $\{u,v\} \in M$ where $u \in A$ and $v \in B$, set $f(u,v) = 1$ and $f(v,u) = -1$. By definition f satisfies the capacity constraint. It also satisfies the demand constraint, since M is a perfect matching and therefore every vertex has degree one, when restricted to M. Therefore f is a flow.

To start with, let $M = \emptyset$. Let f be a flow in (G,c,d) (note that f takes integral values only). For every edge $(u,v) \in \overleftrightarrow{E}$, such that $u \in A$ and $v \in B$, $f(u,v)$ is either 1 or 0. This is because, $0 = -c(v,u) \leq -f(v,u) = f(u,v) \leq c(u,v) = 1$. If $f(u,v) = 1$, add the edge $\{u,v\}$ to M. Now M cannot have two edges of the form $\{u,v\}$ and $\{u,w\}$, since that would violate the demand constraint at u. Also M cannot have a unmatched vertex, since that would again violate the demand constraint. Therefore M is a perfect matching. Also it is easy to see that the construction of M is possible in log-space. □

7.3.1 Existence of a perfect matching

We next give an algorithm (Algorithm 4) to decide if G has a perfect matching. The idea is to construct a capacity-demand graph (G,c,d) similar to Proposition 7.3.1 and see if a flow exists in it. To see the existence of a flow we use Lemma 7.2.2 and see if a negative weight cycle exists in a suitably weighted planar graph. A crucial step in Algorithm 4 is to construct a pseudo-flow in the capacity-demand graph, for which we use the subroutine, Algorithm 5.

Lemma 7.3.2. *Algorithm 4 outputs 'Yes' if and only if the input graph has a perfect matching*

```
Input: A bipartite planar graph $G = (A \cup B, E)$
Output: Yes if $G$ has a perfect matching; No otherwise
1 Construct a capacity-demand graph $(G, c, d)$ as follows: for each vertex $v \in A$, set
  $d(v) = 1$ and for each vertex $v \in B$, set $d(v) = -1$. For an edge $\{u, v\}$ such that
  $u \in A, v \in B$, set $c(u, v) = 1$ and $c(v, u) = 0$;
2 Construct a pseudo-flow $f'$ in $(G, c, d)$ by using Algorithm 5;
3 Construct a zero-demand graph $(G, c - f')$;
4 Output Yes if $\overleftrightarrow{G^*}$ has no negative weight cycle with respect to weights $(c - f')$;
  Output No otherwise;
```

Algorithm 4: Deciding if a bipartite planar graph has a perfect matching (adapted from [MN89])

Proof. By definition of d, $\sum_v d(v) = 0$ in G. Thus we can use Algorithm 5 to construct a pseudo-flow, f', in (G, c, d). If the Algorithm outputs 'Yes', then by Lemma 7.2.2 there exists a flow (say f'') in the zero demand graph $(G, c - f')$. We claim that $f = f' + f''$ is a flow in (G, c, d). f is skew-symmetric since both f' and f'' are skew symmetric. To see that f meets the capacity constraint observe that, $f(u, v) = f'(u, v) + f''(u, v) \leq f'(u, v) + (c - f')(u, v) = c(u, v)$. To see that f meets the demand constraint, for any vertex $u \in V$, observe that, $\sum_{(u,v)\in\overleftrightarrow{E}} f(u, v) = \sum_{(u,v)\in\overleftrightarrow{E}} f'(u, v) + \sum_{(u,v)\in\overleftrightarrow{E}} f''(u, v) = \sum_{(u,v)\in\overleftrightarrow{E}} f'(u, v) = d(v)$.

If the Algorithm outputs 'No', then there does not exist a flow in $(G, c - f')$ and hence in (G, c, d). □

To see correctness of Algorithm 5, observe that $f'(u, v) = -f'(v, u)$, since we have the promise that $\sum_v d(v) = 0$. For any vertex $v \in V$, $\sum_{(v,w)\in\overleftrightarrow{E}} f(v, w) = \sum_{w\in V\setminus\{v\}} -d(w) = d(v)$. Thus f' is a pseudo-flow.

7.3.2 Constructing a perfect matching

We next see how a perfect matching can be constructed in G, if one exists. We give an algorithm (Algorithm 6) that constructs a flow in (G, c, d). Then by applying Proposition 7.3.1 we get a

```
    Input: A capacity-demand graph $(G, c, d)$
    Promise: $\sum_v d(v) = 0$
    Output: A pseudo-flow in $(G, c, d)$
1 Compute a spanning tree $T$ in $G$;
2 For any arc $(u, v) \notin \overleftrightarrow{T}$, set $f'(u, v) = 0$;
3 For an arc $(u, v) \in \overleftrightarrow{T}$, removing the edge $\{u, v\}$ separates the tree $T$ into two
  subtrees. Let $T_u$ denote the subtree containing $u$ and $T_v$ denote the subtree
  containing $v$. For any $(u, v) \in \overleftrightarrow{T}$, set $f'(u, v) = \sum_{w \in T_u} d(w)$;
```

Algorithm 5: Constructing a pseudo-flow in a zero demand graph (adapted from [MN89])

perfect matching in G. The fact that f is indeed a flow in (G, c, d) follows from Lemma 7.3.2.

```
    Input: A planar bipartite graph $G = (A \cup B, E)$
    Promise: $\overleftrightarrow{G^*}$ has no negative weight cycle with respect to $w = c - f'$
    Output: A Perfect Matching in $G$
1 Fix a vertex $s^* \in \overleftrightarrow{G^*}$;
2 Set $f''(u, v) := \mathrm{dist}^w(s^*, v^*) - \mathrm{dist}^w(s^*, u^*)$;
3 Set $f = f'' + f'$;
4 For $u \in A, v \in B$ output "$u$ is matched to $v$" if and only if $f(u, v) = 1$;
```

Algorithm 6: Constructing a perfect matching in a bipartite planar graph (given that one exists) (adapted from [MN89])

7.3.3 Constructing a Hall-obstacle

As we know existence of an obstacle is an equivalence condition for a bipartite graph *not* to have a perfect matching. Therefore if G does not have a perfect matching, then it is an interesting problem to construct a Hall-obstacle in G. In this subsection, we note the correspondence between the Hall-obstacles in a bipartite planar graph and the negative weight cycles in a related planar graph with suitable weights.

Given a bipartite planar graph $G = (A \cup B, E)$, Algorithm 7 outputs a Hall-obstacle if G does not have a perfect matching, else it outputs 'Does not exist'.

```
    Input: A bipartite planar graph $G = (A \cup B, E)$
    Output: A set of vertices that forms a Hall-obstacle; 'Does not exist' otherwise
1   Construct a capacity-demand graph $(G, c, d)$ as follows: for each vertex $v \in A$, set
    $d(v) = 1$ and for each vertex $v \in B$, set $d(v) = -1$. For an edge $\{u, v\}$ such that
    $u \in A, v \in B$, set $c(u, v) = 1$ and $c(v, u) = 0$;
2   Construct a pseudo-flow $f'$ in $(G, c, d)$ by using Algorithm 5;
3   if there is a negative weight cycle in $\overleftrightarrow{G^*}$ with respect to weights $c \cdot n^5 - f'$ then
4       Let $C^*$ be a negative weight cycle in $\overleftrightarrow{G^*}$ with respect to weights $c \cdot n^5 - f'$;
5       Let $(V_1 = A_1 \cup B_1, V_2 = A_2 \cup B_2)$ be the directed cut in $\overleftrightarrow{G}$ corresponding to $C^*$
        where $V_1$ corresponds to the set of faces of $\overleftrightarrow{G^*}$ that are in the interior of $C^*$ or
        equivalently the vertices on $\overleftrightarrow{G}$ that are on one side of the cut corresponding to
        $C^*$;
6       Output $A_1$;
7   else
8       Output 'Does not exist';
9   end
```

Algorithm 7: Constructing a Hall-obstacle in a bipartite planar graph

Lemma 7.3.3. *Let $G = (A \cup B, E)$ be a bipartite planar graph and C^* be a cycle in $\overleftrightarrow{G^*}$. Let f' be a pseudo flow in (G, c, d) (where (G, c, d) is constructed as in Algorithm 7). Then,*

$$f'(C^*) = |A_1| - |B_1|.$$

Proof. Let F^* be the set of faces in $\overleftrightarrow{G^*}$ that lie in the interior of C^*. Since f' is skew-symmetric, $f'(C^*) = \sum_{v^* \in F^*} f'(v^*) = \sum_{v \in V_1} f'(v) = |A_1| - |B_1|$. □

Lemma 7.3.4. *Let (a, b) be an edge in $\overleftrightarrow{G}$, such that $a \in A_1$ and $b \in B_2$. Then, moving b from B_2 to B_1 does not increase the weight of the cut (and the corresponding cycle in the dual) with respect to the weights $c - f'$.*

Proof. By moving a vertex b from B_2 to B_1, f' decreases by 1 by Lemma 7.3.3, and c decreases at least by 1. $\square$

Lemma 7.3.5. *$\overleftrightarrow{G^*}$ has a negative weight cycle with respect to weights $c - f'$ if and only if $\overleftrightarrow{G^*}$ has a negative weight cycle with respect to weights $c \cdot n^5 - f'$ (that is, the total weight contribution from c is zero).*

Proof. Let C^* be a negative weight cycle with respect to weights $c - f'$ in $\overleftrightarrow{G^*}$. Applying Lemma 7.3.4, we can get a cycle C' in $\overleftrightarrow{G^*}$, such that there are no edges $(a, b) \in \overleftrightarrow{G}$, with $a \in A$ and $b \in B$, such that a is inside C' and b is outside. Thus $(c \cdot n^5 - f')(C') < 0$, since by construction, $c(C') = 0$. The reverse direction follows trivially since c is non-negative. $\square$

Theorem 7.3.6. *(Theorem 7.1.1 (c))* HALL-OBS (CONSTRUCT) *in a bipartite planar graphs is in* NL.

Proof. Constructing a negative weight cycle with respect to the weights $c \cdot n^5 - f'$ is in NL. The set A_1 forms a Hall-obstacle since $N(A_1) \subseteq B_1$ and $|A_1| > |B_1|$ (see Lemma 7.3.3). $\square$

7.4 Complexity analysis and an unambiguous log-space bound

Suppose we have a directed graph G with polynomially bounded weights on its edges. The weights could be positive or negative. Let s be a fixed vertex in G. Let $\text{len}(u, v)$ denote the length of the minimum length path from u to v. Let V_k be the set of vertices in G, at a distance at most k from s. That is,

$$V_k := \{v \mid \text{len}(s, v) \leq k\}.$$

Let $\text{dist}_k^w(u, v)$ denote the weight of the minimum weight walk of length at most k from u to v, with respect to the weight function w. We also denote $\text{dist}^w(u, v) = \text{dist}_\infty^w(u, v)$. Note that $\text{dist}_k^w(u, v)$ could be negative. Let Σ_k^w be the sum of the minimum weight walks (with respect to the weight function w) from s to every vertex in V_k. That is,

$$\Sigma_k^w := \sum_{v \in V_k} \mathrm{dist}_k^w(s, v).$$

7.4.1 Weighting scheme A

First we describe a log-space subroutine (Algorithm 8), that when given a planar graph G defines a skew-symmetric edge weight function w_A, on the directed graph $\overleftrightarrow{G}$, such that the weight of any cycle in $\overleftrightarrow{G}$ is non-zero. The weight function is a generalization of the weight function in [BTV09] to planar graphs. Tewari and Vinodchandran had given a log-space construction of a similar weight function by an application of Green's Theorem [TV10].

```
Input: A planar graph $G$
Output: An edge weight function, $w_A : \overleftrightarrow{E} \longrightarrow [n]$ such that for any simple cycle
        $\overrightarrow{C}$ in $\overleftrightarrow{G}$ $w_A(\overrightarrow{C}) \neq 0$
1 Compute a spanning tree $T$ in $G$;
2 For any arc $e \in \overleftrightarrow{T}$, set $w_A(e) = 0$;
3 Let $R$ denote the spanning tree in $G^*$ consisting of the edges that do not belong to $T$.
  Fix a root $r$ for $R$ (say the unbounded face) and let $\overrightarrow{R}$ denote the orientation of $R$
  where each edge is oriented towards $r$;
4 An arc $e^* = (u, v) \in \overrightarrow{R}$ separates the tree $R$ into two subtrees. Let $\alpha_u$ denote the
  number of vertices in the subtree containing $u$. Set $w_A(u, v) = \alpha_u$ and
  $w_A(v, u) = -\alpha_u$;
5 Set $w_A(e) = w_A(e^*)$ for every $e \in \overleftrightarrow{E}$ where $e^*$ is the (directed) dual edge of $e$;
```

Algorithm 8: Weighting Scheme A

Lemma 7.4.1 (adaptation of [BTV09]). *Let G be a planar graph and $\overrightarrow{C}$ be a simple cycle in $\overleftrightarrow{G}$. Then $w_A(\overrightarrow{C})$ is equal to the number of faces in the interior of $\overrightarrow{C}$. In particular, $w_A(\overrightarrow{C}) \neq 0$.*

Proof. It is enough to prove the case when $\overrightarrow{C}$ is anti-clockwise because the clockwise case follows due to the skew-symmetry of w_A. Also it suffices to show that for a facial cycle (anti-clockwise)

$\overrightarrow{F}$ in $\overleftrightarrow{G}$, $w(\overrightarrow{F}) = 1$. This is because, for a simple cycle $\overrightarrow{C}$:

$$w(\overrightarrow{C}) = \sum_{\overrightarrow{F} \in \mathrm{Int}(\overrightarrow{C})} w(\overrightarrow{F}).$$

But $w(\overrightarrow{F})$ equals the sum of the weights of dual edges (in $\overleftrightarrow{G^*}$) outgoing from the dual vertex $F^* \in \overleftrightarrow{G^*}$), so it suffices to show that for every vertex $u \in \overleftrightarrow{G^*}$):

$$\sum_{v:(u,v) \in \overleftrightarrow{G^*}} \alpha_v = 1.$$

Observe that the number of nodes in the subtree rooted at u is one more than sum of the number of vertices in the subtrees rooted at v for various v, such that (u, v) is a dual edge. Hence the above equation follows. □

Note that a planar graph G is min-unique with respect to the weight function w_A. In the rest of this chapter, we refer to the weight function defined in Algorithm 8 as w_A.

7.4.2 Computing the *distance* function

In this section we give a UL algorithm that computes the distance function, $\mathrm{dist}^w(s, v)$ in a planar graph. The idea is to extend the UL algorithm of Reinhardt and Allender [RA00], to work when the graph contains negative weight edges but no negative weight cycles.

Let w be a polynomially bounded weight function defined on the arcs of $\overleftrightarrow{G}$, such that w can assign negative weights to the edges in $\overleftrightarrow{G}$, but there is no negative weight cycle in $\overleftrightarrow{G}$ with respect to w. Let $W(u, v) = w_A(u, v) \cdot n^5 + w(u, v)$. We show in Lemma 7.4.2 that min-uniqueness with respect to the weight function w_A implies min-uniqueness with respect to the weight function W as well.

Lemma 7.4.2. *If G is min-unique with respect to w_A, then G is also min-unique with respect to W.*

Proof. Let u and v be two vertices in G. Suppose there are two minimum weight paths P_1 and P_2, from u to v with respect to the weight function W. This implies $w(P_1) = w(P_2)$ and $w_A(P_1) = w_A(P_2)$. Also there are no paths from u to v of weight lesser than $w_A(P_1)$. This is a contradiction to the min-uniqueness of G with respect to w_A. □

Given a planar graph G, a source vertex s and a vertex v, Algorithm 11 computes the value of $\text{dist}^w(s, v)$. It uses the following two subroutines. (i) Algorithm 9 is an unambiguous procedure, which given the tuple $(G, s, v, k, |V_k|, \Sigma_k^W)$, computes the value of $\text{dist}_k^W(s, v)$ if v is in V_k, else it assigns the value ∞ to it, on a unique computation path and all other paths reject. (ii) Algorithm 10 computes the values $|V_k|$ and Σ_k^W, given the tuple $(G, s, k, |V_{k-1}|, \Sigma_{k-1}^W)$.

We then combine Algorithm 11 with the perfect matching algorithms in Section 7.3 (Algorithm 5 and 6) to obtain the UL bound for perfect matching in bipartite planar graphs.

Lemma 7.4.3. *Given a directed planar graph G, with polynomially bounded weights w on its arcs such that there are no negative weight cycles, the shortest distance $\text{dist}^w(u, v)$ between any pair of vertices in G with respect to w can be computed in* UL.

Proof. Since there are no negative weight cycles then every minimum weight walk between any pair of vertices is a path. We set t to be equal to n in Algorithm 11 and observe that $\text{dist}_n^w(u, v) = \text{dist}^w(u, v)$, since the maximum length of a path is at most n. Also, G is min-unique with respect to w_A and therefore with respect to W also. The UL bound follow from the fact that, between a pair of vertices u and v, if there is a path from u to v, then a unique computation path outputs the value of the minimum weight path from u to v (all other paths halt and reject). If there is no path from u to v, all paths halt and reject. □

```
Input: The tuple $(G, s, v, k, |V_k|, \Sigma_k^W)$
Promise: $G$ is min-unique with respect to $W$
Output: $\mathrm{dist}_k^W(s, v)$ ( $< \infty$ if $v \in V_k$; $\infty$ otherwise)
Initialize $c \leftarrow 0;\ s \leftarrow 0;\ \mathrm{dist}_k^W(s, v) \leftarrow \infty$ ;
foreach $x \in V$ do
    Guess a walk of length at most $k$ from $s$ to $x$;
    if Guess fails then
        Halt and reject
    else
        Let $p$ be the weight of the walk;
        Set $c = c + 1; s = s + p$;
    end
    if $x = v$ then Set $\mathrm{dist}_k^W(s, v) \leftarrow p$
end
if $c = |V_k|$ and $s = \Sigma_k^W$ then
    Output $\mathrm{dist}_k^W(s, v)$;
else
    Halt and Reject;
end
```

Algorithm 9: An unambiguous procedure to compute the value of the function $\mathrm{dist}_k^W(s, v)$ in a planar graph G.

Lemma 7.4.4. *Given a directed planar graph G with polynomially bounded weights w on its arcs, deciding if G contains a negative weight cycle is in* coUL.

Proof. Let N denote the sum of the absolute values of the weights on the arcs. If G has a negative weight cycle, then for some pair of vertices u, v and some index i such that, $Nn \leq i \leq 2Nn$, $\mathrm{dist}_i^w(u, v) < \mathrm{dist}_{i-1}^w(u, v)$ (where $\mathrm{dist}_i^w(u, v) < \infty$). Therefore, going through the different possible values of u, v and i, gives us coUL upper bound. □

Theorem 7.4.5. *(Theorem 7.1.1 (a))* PM-DECISION *and* PM-CONSTRUCT *in bipartite planar graphs are in* UL.

Proof. We know that if there are no negative weight cycles in $\overleftrightarrow{G^*}$ with respect to the weight func-

```
    Input: The tuple $(G, s, k, |V_{k-1}|, \Sigma^W_{k-1})$
    Promise: $G$ is min-unique with respect to $W$
    Output: $(|V_k|, \Sigma^W_k)$
 1  Initialize $c \leftarrow |V_{k-1}|;\ s \leftarrow \Sigma^W_{k-1}$ ;
 2  foreach $v \in V$ do
 3      if $v \in V \setminus V_{k-1}$ then
 4          Set $\mathrm{dist}^W_k(s, v) \leftarrow \min_{x:(x,v)\in E(G)}[\mathrm{dist}^W_{k-1}(s, x) + W(x, v)]$;
 5      end
 6      if $\mathrm{dist}^W_k(s, v) < \infty$ then
 7          Set $c \leftarrow c + 1$ and $s \leftarrow s + \mathrm{dist}^W_k(s, v)$;
 8      end
 9  end
10  Set $|V_k| \leftarrow c$ and $\Sigma^W_k \leftarrow s$;
11  Output $(|V_k|, \Sigma^W_k)$;
```

Algorithm 10: A procedure to compute the values $|V_k|$ and Σ^W_k in a planar G.

```
    Input: The tuple $(G, w, s, v, t)$, where $G$ is a directed graph on $n$ vertices,
           $w : E(G) \to \mathbb{Z}$ such that $|w(e)| \leq n^{O(1)}$ is an edge weight function, $s$ and $v$
           are vertices in $G$ and $t$ is a positive integer
    Output: $\mathrm{dist}^w_t(s, v)$
 1  Compute the weight function $w_A$ using Algorithm 8;
 2  Initialize $V_0 \leftarrow \{s\}$ and $\Sigma^W_0 \leftarrow 0$;
 3  for $k \in [t]$ do
 4      Compute $(|V_k|, \Sigma^W_k)$ from $(|V_{k-1}|, \Sigma^W_{k-1})$;
 5  end
 6  Compute $\mathrm{dist}^W_t(s, v)$ from $(|V_n|, \Sigma^W_n)$;
 7  Output the coefficient of $n^5$ in $\mathrm{dist}^W_t(s, v)$ (by ignoring the lower order bits from
    $w_A$);
```

Algorithm 11: Unambiguous log-space procedure to compute the value $\mathrm{dist}^w_t(s, v)$ for a positive integer t and weight function w.

tion $c - f'$, then there exists perfect matching in G. Therefore from Lemma 7.4.4, we get the UL bound. Correctness follows from the correctness of Algorithm 4 and 6.

Therefore if there are no negative weight cycles in $\overleftrightarrow{G^*}$ then we get a valid flow and thus a valid perfect matching along a unique accepting path; otherwise we get f that is not a valid flow and we reject. □

Corollary 7.4.6. *(Theorem 7.1.1 (b))* HALL-OBS (DECISION) *in bipartite planar graphs is in* coUL.

7.5 Improved upper bound on some related problems

In this section we study certain generalizations of the perfect matching problem and the graph reachability problem - namely the EVEN-PM problem and the EVEN-PATH problem respectively.

7.5.1 Even perfect matching in bipartite planar graphs is in nondeterministic log-space

Definition 7.5.1 (Even-PM)**.** Given a graph with each edge colored either Red or Blue, an Even-PM is a perfect matching that contains an even number of Red edges. Let EVEN-PM denote the problem of deciding whether or not there exists such a perfect matching.

Theorem 7.5.1. *(Theorem 7.1.2 (a))* EVEN-PM *in bipartite graphs is in* P.

Proof. Given a bipartite graph $G = (V, E)$, first find a perfect matching M in it. If M is Even-PM we are done, otherwise construct an auxiliary directed graph H with respect to M as follows: edge (u, v) is in H if and only if there exists a vertex w such that $\{u, w\} \in M$ and $\{w, v\} \in E \setminus M$. Define a weight function w' on H as follows: if the matching edge $\{u, w\}$ as well as the non-matching edge $\{w, v\}$ are of the same color then $w'(u, v) = 0$, else $w'(u, v) = 1$.

Claim 7.5.2. *G has an Even-PM if and only if H has a cycle of odd weight.*

Proof. Let M' be an Even-PM in G. Then since M is not an Even-PM, $M \neq M'$. The symmetric difference of M and M', $M \oplus M'$ is a collection of even length disjoint cycles, whose edges alternate between M and M'. Also since M and M' have a different parity of red edges, there exists one cycle C in $M \oplus M'$ which has a different parity of red edges from M and M'. This implies that C has an odd number of red edges. Now consider the corresponding cycle C' in H. By definition of w', $w'(C')$ is odd.

Let C' be a cycle of odd weight in H. Then consider the corresponding even length cycle C in G, where every alternate edge of C belongs to M. Let C_M and $C_{\overline{M}}$ be the edges of C belonging to M and $\overline{M}$ respectively. Then by definition of w', the parity of red edges in C_M and $C_{\overline{M}}$ are different. Consider the perfect matching $M' = (M \setminus C_M) \cup C_{\overline{M}}$, formed by replacing the edges of C_M with the edges $C_{\overline{M}}$. By construction, M' is an Even-PM. $\square$

(*Proof of Theorem 7.5.1 cont.*) By Claim 7.5.2 we can detect and construct an Even-PM in G by finding an odd weight cycle in H. Computing a perfect matching in a bipartite graph is in P and checking if a graph has an odd weight cycle is in NL. Therefore, the complexity of the entire procedure is P. $\square$

Corollary 7.5.3. *(restatement of Theorem 7.1.2 (b))* EVEN-PM *in bipartite planar graphs is in* NL.

7.5.2 The Even-Path problem in planar DAGs is in unambiguous log-space

Definition 7.5.2 (Red-Blue-Path)**.** Given a directed graph with each edge colored either Red or Blue, a *Red-Blue-Path* from s to t is a (simple) directed path from s to t such that every two consecutive edges are of different colors. The RED-BLUE-PATH problem is the problem of deciding whether or not there is a Red-Blue-Path from s to t.

Definition 7.5.3 (Even-Path)**.** Given a directed graph and two vertices s and t, an *Even-Path* from s to t is a simple, directed path from s to t containing an even number of edges. The EVEN-PATH problem is the problem of deciding whether or not there is an Even-Path from s to t.

It is important to mention here that the even length path that we consider, is a simple path, that is it does not contain any repeated vertices.

Theorem 7.5.4 ([Kul09])**.** RED-BLUE-PATH *in planar DAG is* NL*-complete.*

In this section, we prove that the EVEN-PATH problem (which can be viewed as a relaxation of the RED-BLUE-PATH problem as a path starting with say Red edge and ending with say Blue edge is always of even length) in planar DAG is in fact in UL. Our proof involves a combination of two different isolation techniques that are currently available.

In Lemma 7.5.5 we show that the number of minimum weight, even length paths in a planar DAG is bounded by a polynomial in n. We then apply the isolation technique obtained from hashing by primes, from Theorem 2.3.4 in Chapter 2, to get the desired UL bound.

Lemma 7.5.5. *Let G be a planar DAG and u and v be any two vertices in G. Then with respect to the weight function w_A, (a) if P_1 and P_2 are two minimum weight Even-Paths from u to v, then $P_1 \oplus P_2$ divides the plane into at most two bounded regions; (b) no three minimum weight Even-Paths from u to v share a common vertex w other than u and v, such that the path segments between the vertices u and w and between w and v are not identical. (c) there are at most $2n^4$ minimum weight Even-Paths from u to v.*

Proof. (a) For the sake of contradiction let C_1, C_2 and C_3 be any three bounded regions of $P_1 \oplus P_2$. Let P_{ij} be the restriction of the i-th path to the j-th for $i \in \{1, 2\}$ and $j \in \{1, 2, 3\}$. Observe that $w_A(P_{1j}) \neq w_A(P_{2j})$ since C_j is a simple cycle and by Lemma 7.4.1 we have that $w_A(C_j) \neq 0$. Now the parity of the lengths of the path segments $P_{1,j}$ and $P_{2,j}$ are different since if they were the same, we could replace the higher weighted segment with the lower weighted one and get an even length path of lesser weight. This implies that $w_A(C_1 + C_2 + C_3)$ is odd since the weight of each C_i is odd. Let $P'_i = \bigcup_j P_{ij}$ for $i \in \{1, 2\}$. Therefore either $w_A(P'_1)$ is odd or $w_A(P'_2)$, but not both. Without loss of generality lets assume $w_A(P'_1)$ is odd. For each j pick the path segment between P_{1j} and P_{2j} that has lesser weight to create a set say P'. Now $w_A(P')$ is strictly smaller than both $w_A(P'_1)$ and $w_A(P'_2)$. If $w_A(P')$ is odd then replace P'_1 with P' and if $w_A(P')$ is even then replace P'_2 with P' to get a path of smaller weight and same parity. This is a contradiction. Thus $P_1 \oplus P_2$ has at most two bounded regions.

(b) Let P_1, P_2 and P_3 be three minimum weight paths from u to v that share a common vertex (say w) such that the segments of each of the three paths between the vertices u and w and between w and v are distinct. In other words, if P'_i and P''_i are the segments of P_i between the vertices u and w and between w and v respectively (for $i \in \{1, 2, 3\}$), then $\{P'_i\}$ are pairwise non-identical and so are $\{P''_i\}$. There exists at least two path segments between P'_1, P'_2 and P'_3 whose lengths have the same parity. Without loss of generality assume its P'_1 and P'_2. Now if $w_A(P'_1) \neq w_A(P'_2)$ then since they have the same parity we can pick the lesser weight path between P'_1 and P'_2 and similarly the lesser weight path between P''_1 and P''_2 and append them to get an even path of weight less than either that of P_1 or P_2 from u to v. Thus we can assume $w_A(P'_1) = w_A(P'_2)$. By Lemma 7.4.1, this implies that $P'_1 \oplus P'_2$ as at least two bounded regions. Moreover since P''_1 and P''_2 are also not identical, therefore $P''_1 \oplus P''_2$ has at least one one bounded region. Thus $P_1 \oplus P_2$ has at least 3 bounded regions, thus contradicting part (a).

(c) Let a, b, c and d be four vertices in G and let $\mathcal{P}_{a,b,c,d}$ be the set of all minimum weight even length paths from u to v that pass through the vertices a, b, c and d in that order and are vertex disjoint between the vertices a and b and between the vertices c and d respectively. Then by part (b), $\mathcal{P}_{a,b,c,d}$ will have at most 2 paths. Since the total number of such tuples is at most n^4, therefore the number of minimum weight, even length u-v paths is bounded by $2n^4$. $\square$

We next reduce the problem of finding an even length path in a DAG to finding a simple path in a corresponding graph. Construct a directed graph G' (possibly a multi-graph) from G as follows: the vertex set of G' is the vertex set of G. An edge (v_i, v_j) is in G' if and only if there exists a vertex v_k in G and the edges (v_i, v_k) and (v_k, v_j) are in G. The weight w of an edges in G' is the sum of the weights of the corresponding two edges in G.

Now Lemma 7.5.6 follows by definition of G' and part (c) of Lemma 7.5.5.

Lemma 7.5.6. *(a) G has an Even-Path from u to v if and only if G' has a directed path from u to v; (b) the number of minimum weights paths from u to v in G' with respect to w_A is at most $2n^4$.*

7.5.3 Weighting scheme B

Weighting scheme B is based on a well known hashing scheme based on primes, due to Fredman, Komlós and Szemerédi [FKS84]. The statement of the theorem is given in Chapter 2 in Theorem 2.3.4.

Let p_i be the i^{th} prime number. Consider the lexicographical ordering of the edges of G' and denote the j^{th} edge in this ordering by e_j. Define the i^{th} weight function (for $1 \leq i \leq q(n)$ and an appropriate polynomial $q(n)$ dictated by Theorem 2.3.4), $w_{B_i}(e_j) = 2^j (\mod p_i)$.

Hoang used this scheme to give better upper bounds for perfect matching in certain classes of graphs [Hoa10]. Pavan, Tewari and Vinodchandran showed that reachability in graphs where the number of paths from s to any vertex is bounded by a polynomial is in UL, by applying this hashing scheme. We use Theorem 2.3.4 here to define a weight function with respect to which G' in min-unique.

Lemma 7.5.7 (Adapted from [PTV10])**.** *There exists an $i \leq q(n)$ such that the graph G' with respect to the weight function $W_i = w_A \cdot n^{10} + w_{B_i}$ is min-unique.*

Proof. Let $\mathcal{P}_v$ be the set of minimum weight paths from s to a vertex v in G', with respect to w_A. Then by Lemma 7.5.6, $|\mathcal{P}_v|$ is bounded by $2n^4$. It follows from Theorem 2.3.4 that with respect to some w_{B_i}, all paths in $\bigcup_v \mathcal{P}_v$ will have distinct weights. Therefore G' is min-unique with respect to W_i for some i. $\square$

For each $i \in [q(n)]$, check if G' is min-unique with respect to W_i or not. Once we have an appropriate i, we can decide reachability in G' in UL [RA00]. By Lemma 7.5.6 a path in G' corresponds to an Even-Path in G and thus we have Theorem 7.5.8.

Theorem 7.5.8. *(Theorem 7.1.3)* EVEN-PATH *in planar DAG is in* UL.

7.6 Conclusion

In this chapter we gave a UL upper bound on some important graph theoretic problems like perfect matching in bipartite planar graphs and deciding an even length path in planar DAGs. In light of these results, the following questions might be worth looking at. (a) Similar to directed planar reachability, is bipartite planar perfect matching in coUL as well? (b) Is perfect matching in higher genus graphs in UL as well? (c) Can we show that RED-BLUE-PATH problem in planar DAGs is in UL?

Chapter 8

Conclusion

In this chapter, we give a brief overview of the results that we have discussed in this dissertation and their importance to complexity theory in general. We also look at the scope of extending these results and solving certain important open problems in the area of small space complexity.

8.1 Our progress so far

In our attempt to study the power of unambiguity in log-space, we have made some definite progress. We have shown that some important and interesting subclasses of general reachability are in UL. Our result that planar reachability is in UL, made a significant impact on the area of small space computations. Thierauf and Wagner first applied it to show that reachability in $K_{3,3}$-free graphs and K_5-free graphs are in UL [TW09] and then to show that isomorphism testing for 3-connected planar graphs can be done in UL [TW10]. Limaye, Mahajan and Nimbhorkar used it to show that the shortest and longest paths in planar DAGs can be computed in UL [LMN10]. Datta, Kulkarni and Roy showed that the weight function that we used to show that planar graphs are min-unique (a crucial step in showing that planar reachability is in UL), can be easily modified to obtain a deterministic weight function that isolates a perfect matching over bipartite planar graphs (thus showing that bipartite, planar perfect matching is in SPL) [DKR10]. Datta, Kulkarni, Limaye and Mahajan used our result to show that testing the uniqueness of a perfect matching over bipartite planar graphs is in $\oplus$L. Kynčl and Vyskočil had shown that bounded genus reachability reduces to planar reachability [KV10]. Combining their result with the fact that planar reachability is in UL, immediately gave a UL upper bound on the reachability problem over bounded genus graphs.

In an attempt to generalize our results beyond planar graphs we studied the isolation problem and its connection to showing reachability in UL. In the planar case, we obtained a nice application of Green's Theorem on the isolation problem. This connection gave us more intuition as to what was happening in the planar case. Also it immediately implied that perfect matching in bipartite planar graphs is in SPL. This result was already known due to [DKR10], but our method was much simpler and it showed the upper bound without involving the machinery of grid graphs. Moreover, the isolation result provided us a way to go past the class of grid graphs, which was unknown till then. Using algebraic topological techniques, we generalized our isolation result to the class of bounded genus graphs. This immediately gave the following upper bounds - (i) bounded genus reachability is in UL and (ii) perfect matching in bounded genus bipartite graphs is in SPL. Although the former was already known due to [BTV09, KV10], the latter result was a completely new upper bound on this problem.

We knew that matching in bipartite planar graphs is in SPL, but even an NL upper bound was unknown for this problem (note that SPL and NL are incomparable classes as of now). We showed that matching over bipartite planar graphs is in UL which is a vast improvement on the space complexity of the problem.

In the process of achieving the above results, we gave some new embedding algorithms for certain classes of graphs, that run in log-space. We gave a piecewise straight-line embedding of a planar graph and a combinatorial embedding on the fundamental polygon for a bounded genus graph (given the promise that the genus is bounded). Both these embedding algorithms are the first of their kind.

Pursuing our study of unambiguous log-space computations, we showed that a certain restriction of the complexity class FewL ($\mathsf{ReachFewL}$), is in UL. In fact we proved a stronger statement that even the number of accepting paths in such computations can be counted in UL. This result immediately gave an affirmative answer to a question by [LMN10], who asked whether counting the number of paths in a planar DAG, with a polynomial bound on the number of paths, can be done in UL? We also showed that achieving min-uniqueness is necessary and sufficient to show that $\mathsf{NL} = \mathsf{UL}$.

Our progress in the above problems furthered our knowledge of unambiguous log-space computations, and more generally about nondeterminism in small space computations.

8.2 Open questions

Our research has also raised a number of questions that are interesting and important to the area of small space computations. Is $\mathsf{NL} = \mathsf{UL}$? How close are we towards answering this conundrum? This is one of the key questions in understanding nondeterministic log-space computations. We showed several new complete problems for NL in this dissertation and hope that they turn out to be useful in settling the above question. Showing that $\mathsf{FewL} \subseteq \mathsf{UL}$ might also be a good intermediate step in this direction.

More generally, can we extend our isolating results to general graphs (or even an interesting subclass like graphs of logarithmic genus)? This not only would show that $\mathsf{NL} = \mathsf{UL}$ but would also imply that bipartite matching is in SPL (in particular in NC), another longstanding open problem.

Our UL bound on the space complexity of planar bipartite perfect matching problem naturally raises the question whether generalizations of planarity (like higher genus) also admit a UL algorithm for the perfect matching problem. This might be a bit of a challenge because the UL algorithm heavily uses planarity. Nevertheless, the possibility of improvement cannot be ruled out.

It is known that perfect matching in bipartite planar graphs is L-hard. This brings forth the question that can we improve our UL upper bound to match the lower bound for the planar bipartite case (that is L).

In contrast to our earlier result on planar reachability, where we show that planar reachability is in $\mathsf{UL} \cap \mathsf{coUL}$, in the case of bipartite planar matching, we are only able to show that deciding if a perfect matching exists is in UL. Can we show the coUL bound as well on the problem? Again this might be a starting point to aim for an L upper bound. Also, the complexity of computing a minimum weight perfect matching in bipartite planar graphs, is still in SPL. Can we bring the complexity down to NL?

Finally, what is the best deterministic space upper bound on NL? Can we improve Savitch's Theorem even by an arbitrarily small amount, that is, can we show that $\mathsf{NL} \subseteq \mathsf{DSPACE}(\log^{2-\epsilon} n)$, for any $\epsilon > 0$?

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1 edition, 2009.

[ABC$^+$09] Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, and Sambuddha Roy. Planar and grid graph reachability problems. *Theory Comput. Syst.*, 45(4):675–723, 2009.

[ABO99] Eric Allender, Robert Beals, and Mitsunori Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Comput. Complex.*, 8:99–126, November 1999.

[ADR05] Eric Allender, Samir Datta, and Sambuddha Roy. The directed planar reachability problem. In *Proceedings of the 25th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 2005.

[Agr07] Manindra Agrawal. Rings and integer lattics in computer science. A lecture series at the Annual Workshop in Computational Complexity, Barbados, 2007.

[AJ93] Carme Àlvarez and Birgit Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[AL98] Eric Allender and Klaus-Jörn Lange. $\mathsf{RUSPACE}(\log n) \subseteq \mathsf{DSPACE}(\log^2 n/\log\log n)$. *Theory of Computing Systems*, 31:539–550, 1998.

Special issue devoted to the 7th Annual International Symposium on Algorithms and Computation (ISAAC'96).

[All86] Eric Allender. The complexity of sparse sets in P. In *Proc. of the conference on Structure in complexity theory*, pages 1–11, 1986.

[All06] Eric Allender. NL-printable sets and nondeterministic Kolmogorov complexity. *Theor. Comput. Sci.*, 355(2):127–138, 2006.

[AM04] Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189:117–134, 2004.

[AM08] V. Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Proceedings of RANDOM '08*, pages 276–289, 2008.

[AO94] Eric Allender and Mitsunori Ogihara. Relationships among PL, #L, and the determinant. In *Structure in Complexity Theory Conference, 1994., Proceedings of the Ninth Annual*, pages 267 –278, 1994.

[ARZ99] Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[BDHM92] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-mod class. *Mathematical Systems Theory*, 25(3):223–237, 1992.

[BHS93] Gerhard Buntrock, Lane A. Hemachandra, and Dirk Siefkes. Using inductive counting to simulate nondeterministic computation. *Information and Computation*, 102(1):102–117, 1993.

[BJLR91] Gerhard Buntrock, Birgit Jenner, Klaus-Jörn Lange, and Peter Rossmanith. Unambiguity and fewness for logarithmic space. In *Proceedings of the 8th International Conference on Fundamentals of Computation Theory (FCT'91)*, Volume 529 Lecture Notes in Computer Science, pages 168–179. Springer-Verlag, 1991.

[BLMS98] David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *15th International Symposium on Theoretical Aspects of Computer Science (STACS)*, Volume 1373 in Lecture Notes in Computer Science, pages 74–83. Springer, 1998.

[Bra21] H. R. Brahana. Systems of circuits on two-dimensional manifolds. *The Annals of Mathematics*, 23(2):144–168, 1921.

[BTV09] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):1–17, 2009.

[CH90] Jin-Yi Cai and Lane Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 1990.

[CM07] Sergio Cabello and Bojan Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. *Discrete Comput. Geom.*, 37(2):213–235, 2007.

[CSV84] Ashok K. Chandra, Larry Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.

[DEH00] Michael B. Dillencourt, David Eppstein, and Daniel S. Hirschberg. Geometric thickness of complete graphs. *Journal of Graph Algorithms and Applications*, 4(3):5–17, 2000.

[dFPP90] Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[DH07] Max Dehn and Poul Heegaard. Analysis situs. *Enzyklopädie der mathematischen Wissenschaften mit Einschluß ihrer Anwendungen*, III.AB(3):153–220, 1907.

[Die10] Reinhard Diestel. *Graph Theory*. Springer-Verlag, 2010.

[DKLM10] Samir Datta, Raghav Kulkarni, Nutan Limaye, and Meena Mahajan. Planarity, determinants, permanents, and (unique) matchings. *ACM Trans. Comput. Theory*, 1:10:1–10:20, March 2010.

[DKR10] Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010.

[DKT10] Samir Datta, Raghav Kulkarni, and Raghunath Tewari. Perfect matching in bipartite planar graphs is in UL. Technical Report TR10-151, Electronic Colloquium on Computational Complexity, 2010. Submitted to conference.

[DKTV11] Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space Complexity of Perfect Matching in Bounded Genus Bipartite Graphs. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 579–590, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Edm65] Jack Edmonds. Paths, trees and flowers. *Canad. J. Math.*, 17:449–467, 1965.

[Ete97] Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *Journal of Computer and System Sciences*, 54(3):400–411, June 1997.

[Fár48] István Fáry. On straight line representation of planar graphs. *Acta Univ. Szeged. Sect. Sci. Math.*, 11:229–233, 1948.

[FKS84] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984.

[GS88] Joachim Grollmann and Alan L. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 17:309–335, April 1988.

[GW96] Anna Gal and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Structures and Algorithms*, 9:1–13, 1996.

[Has81] Refael Hassin. Maximum flow in (s, t)-planar networks. *Information Processing Letters*, 13:107, 1981.

[Hem89] Lane A. Hemachandra. The strong exponential hierarchy collapses. *J. of Computer and System Sciences*, 39(3):299–322, 1989.

[Hoa10] Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150, 2010.

[HSV95] Joan P. Hutchinson, Thomas C. Shermer, and Andrew Vince. On representations of some thickness-two graphs. In *GD '95: Proceedings of the Symposium on Graph Drawing*, pages 324–332, London, UK, 1995. Springer-Verlag.

[Imm88] Neil Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[Kas67] Pieter W. Kasteleyn. Graph theory and crystal physics. *Graph Theory and Theoretical Physics*, 1:43–110, 1967.

[KMV08] Raghav Kulkarni, Meena Mahajan, and Kasturi R. Varadarajan. Some perfect matchings and perfect half-integral matchings in NC. *Chicago Journal of Theoretical Computer Science*, 2008(4), September 2008.

[Ko85] Ker-I Ko. On some natural complete operators. *Theoretical Computer Science*, 37:1 – 30, 1985.

[Kre88] Mark Krentel. The complexity of optimization problems. *J. of Computer and System Sciences*, 36:490–509, 1988.

[Kul09] Raghav Kulkarni. On the power of isolation in planar graphs. Technical Report TR09-024, Electronic Colloquium on Computational Complexity, 2009.

[KV10] Jan Kynčl and Tomáš Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Trans. Comput. Theory*, 1(3):1–11, 2010.

[Lan97] Klaus-Jörn Lange. An unambiguous class possessing a complete set. In *Proceedings of the 14th STACS*, pages 339–350. Springer, 1997.

[LMN10] Nutan Limaye, Meena Mahajan, and Prajakta Nimbhorkar. Longest paths in planar DAGs in unambiguous log-space. *Chicago Journal of Theoretical Computer Science*, 2010(8), June 2010.

[LP83] Andrea Lapaugh and Christos Papadimitriou. The even-path problem for graphs and digraphs. *Networks Volume 14, Issue 4 , Pages 507 - 513*, 1983.

[LP86] László Lovász and Michael D. Plummer. *Matching Theory*, volume 29. North-Holland Publishing Co, 1986.

[Mas91] William S. Massey. *A Basic Course in Algebraic Topology*. Springer-Verlag, 1991.

[MN89] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 112 –117, 1989.

[MT01] Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. John Hopkins University Press, 2001.

[MV00] Meena Mahajan and Kasturi R. Varadarajan. A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs. In *ACM Symposium on Theory of Computing*, 2000.

[MVV87] Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[Ned99] Zhivko Prodanov Nedev. Finding an even simple path in a directed planar graph. *SIAM Journal on Computing, Volume 29 , Issue 2, Oct 99, 685-695*, 1999.

[Ogi95] Mitsunori Ogihara. Equivalence of NC^k and AC^{k-1} closures of NP and other classes. *Information and Computation*, 120(1):55 – 58, 1995.

[PTV10] A. Pavan, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in logspace. Technical Report TR10-009, Electronic Colloquium on Computational Complexity, 2010. To appear in Computational Complexity.

[PY82] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, 1982.

[RA00] Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal of Computing*, 29:1118–1131, 2000. An earlier version appeared in FOCS 1997, pp. 244–253.

[Rac82] Charles Rackoff. Relativized questions involving probabilistic algorithms. *Journal of the ACM*, 29(1):261–268, 1982.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):1–24, 2008.

[Sav70] W. Savitch. Relationships between nondeterministic and deterministic time complexities. *J. Comput. and Systems Sciences*, 4(2):177–192, 1970.

[Sch90] Walter Schnyder. Embedding planar graphs on the grid. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[Ste09] James Stewart. *Calculus (6th ed.)*. Thomson, Brooks/Cole, 2009.

[SW88] Uwe Schöning and Klaus Wagner. Collapsing oracle hierarchies, census functions and logarithmically many queries. In Robert Cori and Martin Wirsing, editors, *STACS 88*, volume 294 of *Lecture Notes in Computer Science*, pages 91–97. Springer Berlin / Heidelberg, 1988.

[Sze88] Robert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tan03] Till Tantau. Logspace optimisation problems and their approximation properties. Technical Report TR03-077, Electronic Colloquium on Computational Complexity, 2003.

[Tho89] Carsten Thomassen. The graph genus problem is NP-complete. *J. Algorithms*, 10(4):568–576, 1989.

[TV10] Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. Technical Report TR10-151, Electronic Colloquium on Computational Complexity, 2010.

[TW09] Thomas Thierauf and Fabian Wagner. Reachability in $K_{3,3}$-free graphs and K_5-free graphs is in unambiguous log-space. In *17th International Conference on Foundations of Computation Theory (FCT)*, Lecture Notes in Computer Science 5699, pages 323–334. Springer-Verlag, 2009.

[TW10] Thomas Thierauf and Fabian Wagner. The isomorphism problem for planar 3-connected graphs is in unambiguous logspace. *Theor. Comp. Sys.*, 47:655–673, October 2010.

[Val76] Leslie Valiant. The relative complexity of checking and evaluating. *Information Processing Letters*, 5:20–23, 1976.

[Val79] Leslie Valiant. The complexity of computing the permanent. *SIAM J. Comput.*, 8:189–201, 1979.

[Vaz88] Vijay Vazirani. NC algorithms for computing the number of perfect matchings in $K_{3,3}$–free graphs and related problems. In *Proceedings of SWAT '88*, pages 233–242, 1988.

[Vol99] Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Springer-Verlag, 1999.

[VY90] Gert Vegter and Chee-Keng Yap. Computational complexity of combinatorial surfaces. In *Proceedings of the 6th Annual Symposium on Computational Geometry*, pages 102–111, 1990.

[Wig94] Avi Wigderson. NL/poly $\subseteq$ $\oplus$L/poly. In *Proceedings of the 9th Structures in Complexity conference*, pages 59–62, 1994.