Reachability Problems and Space Bounds

Akshar Varma
DA-IICT, Gandhinagar
201301442@daiict.ac.in

Supervisor
Dr. Nutan Limaye
Indian Institute of Technology, Bombay

***Abstract*— Space complexity is an important field of study, to understand the theoretical memory requirements to solve various problems. Such study is necessary due to the increasing availability of large amounts of data, and hence problem sizes, to know which questions can be answered within feasible memory capabilities.**

This project studies certain graph reachability problems within the classes NL, L, UL, coUL, etc. Particularly, for 3D monotone grid graphs, the reachability or st-connectivity problem is looked at, that is whether there is a path from one vertex s to another vertex t. This particular problem is of interest as variants of it span across the space complexity spectrum. Bourke et al. [1] show that the general variant is NL-Complete, hence linking it to the L vs. NL question. We look at variants that are more tractable, and show, using various reductions, that variants of the 3D monotone grid graph reachability (3D-MGGR) problem have some interesting properties.

We show that the 3D-MGGR with $O(1)$ 2D layers is in UL $\cap$ coUL using logspace and AC^0 reductions. We also relax one of the monotonicity conditions within layers and show that problem to have the same complexity. On the other hand, we prove that removing monotonicity between layers makes the problem difficult, specifically we show that the 3D-MGGR problem without monotonicity between layers is NL-Complete even for the case of only 2 layers. To understand the nature of other 3D-MGGR variants, we provide preliminary ideas on defining a restricted class of graphs which would allow the monotonicity conditions to be relaxed further.

These results provide an insight into the nature of complexity classes within NL, and also into how well these problems can characterize these complexity classes. Our reductions show that nuanced variants of 3D-MGGR can have significantly distinct complexities and are worth studying.

***Index Terms*— Space Complexity, 3D Monotone Grid Graph, Reachability, Logspace reductions, AC^0 reductions**

I. Introduction

The broad topic of this work is the field of computational complexity theory, which is focused on understanding the capabilities and limitations of computational models in terms of the time and space required to solve various problems. The study of the complexity of solving problems is a fundamental part of theoretical computer science.

This field of study focuses on Turing machines, motivated by the Church-Turing thesis which states, informally, that a function is computable if and only if it can be computed using Turing machines. The study of complexity of solving problems is a fundamental part of theoretical computer science and it seeks to better understand the nature of problems that can be solved given certain resource constraints, be it the time allowed or the space allowed to solve problems. Both time and space complexity are of importance to understanding the nature of problems; for designing faster and efficient algorithms.

Space complexity focuses on the amount of space that is available to a Turing machine while solving a problem. With the steady increase in the amount of Big Data, problem sizes are also increasing rapidly. This calls for algorithms that use optimal memory to solve problems and the study of space complexity aims to better understand this from a complexity theoretic perspective. Large problem sizes make it preferable to keep the space allowed for computation to be asymptotically lower than the size of the input. It would be quite inefficient if it takes as much memory as the whole graph to answer a question like deciding whether you can reach from vertex *s* to vertex *t* in a (directed) graph. Simply imagining the size of the internet is enough to persuade one to search for algorithms with space complexity that is asymptotically better than linear. That provides some motivation for looking at problems that can be solved using logarithmic space.

We focus on space complexity theoretic exploration of reachability problems in graphs. Since the most difficult reachability problem (directed graph reachability) is **NL**-Complete all of our work is within the complexity class of **NL**. The problem of reachability is interesting, as depending on the class of graphs it showcases vastly different complexities and their study aids in understanding the relationship between the various space complexity classes. Graph reachability problems have been used (like in [2]) to characterize, demarcate and in general better understand complexity classes like **NL**, **coNL** [3, 4] **L** [5], **UL** [6], etc.

We use results based on planar restrictions of graphs to study the complexity of reachability problems in grid graphs, particularly 3D grid graphs with monotonicity restrictions. We provide reductions that show that variants of the problem characterize quite different complexity classes, ranging from **UL** $\cap$ **coUL** to **NL**-Complete and are thus of interest.

The rest of this report is structured as follows. Section II contains basic definitions and other preliminary results useful in later sections. The literature survey done to learn the basics of the field and to search for open problems is mentioned in Section III. Section IV contains the reductions and all the main results done during this bachelor's thesis work. We list future directions for work and concluding remarks in Section V.

II. Preliminaries

We first define strings and languages (from [7]), which are used to model problems and problem instances.

Definition 1: **Strings**
A string over some alphabet A, is a finite ordered tuple of elements from A. An alphabet is essentially a set and we generally use the binary alphabet $\{0, 1\}$. □

Definition 2: **Languages or Decision Problems**
We associate a set $L_f = \{x|f(x) = 1\}$ with any Boolean (single bit output) function f and call the set a language or a decision problem. The problem of computing the function value $f(x)$ given input string x, becomes a question of deciding whether the string x belongs to the language, that is, $f(x) = 1$ iff $x \in L_f$. □

To compute the value of a function or equivalently decide whether a string belongs to a set, we need a model of computation. We define one such model, the Turing Machine, as given in Arora and Barak's book [7]. A Turing machine is a model of computation that has one "input tape", one[1] "work tape" and one "output tape". A *tape* is an infinite one-dimensional line of cells, each of which can hold a symbol from a finite set Γ called the alphabet of the machine. There is a *tape head* that can read or write contents to the tape one cell at a time. The *tape head* of the input tape can only read the contents of the input. Further, the machine's computation is divided into discrete steps at which all the *tape heads* can move left or right by one cell.

The manner in which a Turing machines performs computations is based on its current state which belongs to a finite set of states, denoted by Q. The machine keeps a register that stores the current state which is used to determine the next state based on a transition function deciding whether to read/write to cells, whether to move left/right or whether to terminate the computation.

Definition 3: **Turing Machines** (definition from [7])
Formally, a Turing Machine (TM) M is described by a tuple (Γ, Q, δ) containing:

- A finite set Γ of the symbols that M's tapes can contain. We assume that Γ contains a designated "blank" symbol, denoted $\Box$, a designated "start" symbol, denoted $\triangleright$ and the numbers 0 and 1. We call Γ the alphabet of M.
- A finite set Q of possible states M's register can be in. We assume that Q contains a designated start state, denoted q_{start} and a designated halting state, denoted q_{halt}.
- A function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$, where $k \geq 2$, describing the rules M uses in performing each step. This function is called the transition function of M.

Generally, we are concerned with decision problems and thus only allow one bit to be written to the output tape. An input string is accepted by the TM, when it outputs 1, denoting that the string is in the language and analogously a TM rejects a string by outputting 0, denoting that it is not a part of the language. Symbolically, on input x, a TM M, is said to return $M(x)$ and if $M(x) = 1$ then the TM is said to accept the input string and it is said to reject the string if $M(x) = 0$. M *recognizes* a language L_f if on input x, $M(x) = f(x)$. □

These are also known as Deterministic Turing Machines (DTMs) and there exists a model known as Non-deterministic Turing Machines (NDTMs) which allows non-determinism in the computations made by the TM.

Definition 4: **Non-deterministic Turing Machines** (definition from [7])
NDTMs are similar to DTMs, except that they contain two transition functions δ_0 and δ_1 and a special accept state q_{accept}. At each computation step, an NDTM M arbitrarily chooses one of the transition functions and applies that to the current state. An NDTM accepts an input if some sequence of these choices results in $M(x) = 1$ and rejects if all choices result in $M(x) = 0$. □

Let $s : \mathbb{N} \rightarrow \mathbb{N}$ and $L \subseteq \{0, 1\}^*$.

Definition 5: **SPACE**$(s(n))$ (definition from [7])
We say that $L \in \textbf{SPACE}(s(n))$ if there is a constant c and a DTM M deciding L such that at most $c \cdot s(n)$ locations on M's work tapes (excluding the input tape) are ever visited by M's head during its computation on every input of length n. □

Definition 6: **NSPACE**$(s(n))$ (definition from [7])
We say that $L \in \textbf{NSPACE}(s(n))$ if there is a constant c and a NDTM M deciding L such that for inputs of length n, at most $c \cdot s(n)$ locations on M's work tapes are ever visited, regardless of its non-deterministic choices. □

Using the definitions of **SPACE** and **NSPACE**, we define complexity classes that contain problems that can all be solved using some limited amount of resources. In the case of space complexity we are interested in classes that require logarithmic resources for its work tapes. Thus, we define the following two complexity classes.

Definition 7: **L and NL** (definition from [7])
These complexity classes are defined on the basis of the space they use for a problem of size n, as follows:
$\textbf{L} = \textbf{SPACE}(\log(n))$
$\textbf{NL} = \textbf{NSPACE}(\log(n))$ □

Definition 8: **UL**
UL is defined similar to **NL**, except that instead of the Turing Machine accepting on at least one computation path, it should accept on exactly one computation path for a language to be in **UL**. □

Definition 9: **coUL**
A problem is in **coUL** if and only if its complement is in **UL**. Mathematically, $\textbf{coUL} = \{L|\bar{L} \in \textbf{UL}\}$ □

We now define logspace reductions and **NL**-Completeness as given in [7].

Definition 10: **Logspace Reduction** (definition from [7])
A function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is *implicitly logspace computable*, if f is polynomially bounded (i.e., there's some c such that $|f(x)| \leq |x|^c$ for every $x \in \{0, 1\}$) and the languages $L_f = \{\langle x, i\rangle | f(x)_i = 1\}$ and $L_f = \{\langle x, i\rangle | i \leq |f(x)|\}$ are in **L**.

A language B is *logspace reducible* to language C, denoted $B \leq_l C$, if there is a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ that is implicitly logspace computable and $x \in B$ iff $f(x) \in C$ for every $x \in \{0, 1\}^*$. □

[1]While multiple work tapes are also possible, they don't provide any extra computational power to the Turing Machine and we can assume that there is only one work tape.

Definition 11: **NL-Completeness** (definition from [7])
A language C is **NL**-Complete is it is in **NL** and for every $B \in \mathbf{NL}$, $B \leq_l C$. □

Definition 12: **First order projections** (definition from [8], based on [9])
These are reductions which are computed by circuits having no gates (except possibly NOT gates), effectively making each output bit either a copy (or a negated copy) of a bit of the input or a constant. □

Definition 13: **Circuits, their sizes, and their depths** (definition based on [7, 10])
A circuit is a directed acyclic graph that with some input gates, output gates and intermediate gates. Each intermediate gate is a Boolean function (AND, OR, NOT, etc.) and can take in any number of incoming edges called inputs (except for unary gates like NOT), and output (outgoing edge) a single bit computing the value of the Boolean function.

The size of a circuit is the number of vertices in the Directed Acyclic Graph. The depth of the circuit is the longest path from an input node to an output node. □

Definition 14: **$\mathbf{AC^0}$ reductions** (definition based on [7, 10])
These are reductions computable by an $\mathbf{AC^0}$ circuit, which have constant depth, polynomial size, and unbounded fan-in for AND and OR gates with NOT gates only at the input. □

Definition 15: **Reachability or st-connectivity**
Given a graph G, a source vertex s and a target vertex t, the reachability problem or the st-connectivity problem is the problem of deciding whether there exists a path from s to t in the graph G. □

Definition 16: **Grid Graphs** (definition based on [8])
Class of graphs where each vertex can be labeled with integer valued coordinates (both 2D and 3D). Further, edges in such graphs can only exist between neighboring grid points. □

Definition 17: **Monotonicity** (based on definition from [1])
Grid Graphs are said to have monotonicity along some axis iff all edges along that axis are in one direction; i.e. if all edges along the X axis are in the positive direction, then it has monotonicity in the X direction. A grid graph having monotonicity in all directions is a monotone grid graph. □

Definition 18: **Layered Grid Graphs** (based on definition from [8])
A grid graph that has monotonicity in some of its direction is called a layered grid graph. □

Definition 19: **Thickness of graphs** (definition from [1], based on [11])
The thickness of a graph G is the minimal number of planar subgraphs whose union is G. □

Definition 20: **Geometric thickness of graphs** (definition from [1], based on [12, 13])
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 k transparencies so that no two line segments cross in any one transparency.

The difference between thickness and geometric thickness is that geometric thickness requires that all vertex placements be consistent across all transparencies. □

III. Initial literature survey

A thorough understanding of Turing machines and related concepts had to first be acquired due to a lack of any previous formal training in Complexity Theory. Sanjeev Arora and Boaz Barak's book, Computational Complexity: A modern approach [7] was used as a textbook to study all the basics required to start working on problems.

The study started by understanding the basics of Turing machines, the relation between strings and machines, simulation of Turing machines using a Universal Turing machine and Uncomputability, among other things. This was followed by various time complexity related complexity classes including **P**, **NP**, **coNP**, **EXP**, **NEXP**, **coEXP**, etc. Relations between these classes were also studied along with the concept of reductions and the locality of computations, using, among others, the Cook-Levin theorem. The concept of diagonalization was studied next and the Time hierarchy theorem, the Non-deterministic Time hierarchy theorem, Ladner's theorem and Oracle machines were looked at.

Having received a broad view of the basics related to the time complexity of Turing machines, space complexity was studied with numerous theorems for space complexity being looked at, analogous to those for time complexity. **PSPACE**-completeness, **NL**-Completeness, etc. were looked at and a thorough understanding of these concepts was acquired via solving exercise problems.

With the required background, other literature was used to find open problems to be focused on. This was primarily done using the PhD thesis of Sambuddha Roy [8], particularly focusing on the complexity theoretic aspects of planar restrictions of graphs. The problems studied in the thesis are those related to subclasses of planar graphs and particularly, the problem of graph reachability is looked at. Such study is motivated by a wish to improve the understanding of the relation between the complexity classes of L and NL, and of intermediate and lower complexity classes. The thesis provided an introduction to the Grid Graph Reachability problem and variants. The proofs in the thesis provided insight into the techniques commonly used for understanding such problems.

Further literature survey was done to find open variants of the reachability problem in grid graphs. Bourke et al. [1] was one paper which dealt with reachability for 3D grid graphs. They prove the general 3D-MGGR problem to be **NL**-Complete which motivated looking at some simpler variants of the problem.

In the periods between searching for open problems, some background was gained in the related fields of circuit complexity and basics of first-order projections; all of which were helpful during later reductions. This background work included reading the initial parts of Vollmer's book, Introduction to Circuit Complexity [10], and Immerman's book, Descriptive Complexity [9], of which the latter was used as a reference to understand some of the proofs used in Roy's thesis. Barrington's works [14, 15] also came up while searching for problems related to those mentioned in Roy's thesis and further motivated looking at problems related to grid graph reachability.

IV. 3D-MGGR AND VARIANTS

A three-dimensional grid graph is a directed graph whose vertices belong to $[n] \times [n] \times [n]$, with edges connecting only immediate neighboring grid points. We identify positive X and Y directions with the north and east cardinal directions and negative X and Y directions with south and west cardinal directions respectively. An edge in the positive Z direction $((i, j, k) \rightarrow (i, j, k + 1))$ is an upward edge and an edge in the negative Z direction is a downward edge. We call a three-dimensional grid graph monotone (3D-MGG) if there are only north, east and upward edges.

Bourke et al. refer to the st-connectivity problem (or the reachability problem) for such a graph as 3D-MGGR and show that it is **NL**-Complete [1]. We look at some variants of this problem and use reductions to determine their complexity.

A. 3D-MGGR$_{O(1)}$ $\in$ ***UL*** $\cap$ ***coUL***

We modify the 3D-MGGR problem by restricting the number of XY-planes (called layers) in the 3D grid to be some $k = O(1)$, and give a many-one logspace reduction from this restricted version of the 3D-MGGR problem to Directed Planar Reachability (DPR) problem. Since Bourke et al. prove that DPR is in **UL** $\cap$ **coUL**, this reduction shows that 3D-MGGR with $O(1)$ layers (3D-MGGR$_{O(1)}$) is also in **UL** $\cap$ **coUL**. Further, our reduction followed by Roy's logspace reduction from DPR to 2D GGR (Section 4.4 of his thesis [8]), suffices to give a logspace reduction from the 3D-MGGR$_{O(1)}$ problem to the 2D GGR problem.

The main idea in the reduction is to note that for a path from s to t to exist, there must be a sequence of upward edges (often referred to simply as "sequence") through which the path goes from layer to layer on the way from s to t. Between two such upward edges, the path needs to find a subpath within a particular layer. The reduction creates a graph which looks at all possible sequences of upward edges and within each such sequence looks at whether there is a subpath for each required layer.

Theorem 1: 3D-MGGR$_{O(1)}$ many-one logspace reduces to DPR. Therefore, 3D-MGGR$_{O(1)}$ $\in$ **UL** $\cap$ **coUL**.

Proof: Since we only have $O(1)$ layers, the grid dimensions are $m \times m \times k$, where $k = O(1)$ is a constant independent of the problem size. Vertices in each layer belong to an $m \times m$ sized 2D grid. Since we are concerned with only the st-connectivity problem in monotone graphs, we can trivially restrict s and t to lie on diagonally opposite corners of the 3D grid by neglecting the remaining part of the grid. For each layer, we make a set containing all the vertices from that layer which have edges that leave the layer (going upward in monotone graphs) to get the sets $L_1, L_2, \ldots, L_k$, each of which is ordered by first comparing the row number and then the column number of the elements. Let $l_1, l_2, \ldots, l_k$ be the cardinality of these sets, with $l_i \leq m^2$, $\forall i \in [k]$.

Any path from s to t will use some sequence of upward edges and we let the total number of possible choices for such sequences be $a = \prod_{i=1}^{k} l_i$. Given the i^{th} sequence, we label the vertices incident on the upward edges as $t_{i,j}$ and $s_{i,j+1}$ when the edge goes from layer j to layer $j + 1$ (for completeness we label s as $s_{i,1}$ and t as $t_{i,k}$). The $s_{i,j}$ vertices act as sources and $t_{i,j}$ act as targets within the layer j. To see whether the i^{th} sequence can be used for the overall path, we check if there is some subpath from $s_{i,j}$ to $t_{i,j}$ (denoted by $t_{i,j} \rightsquigarrow s_{i,j}$) for all possible layers j. For a given i, this can symbolically be denoted as determining whether the Boolean expression $\forall j \in [k]\ \exists(t_{i,j} \rightsquigarrow s_{i,j})$ returns true or false. Similarly, the overall condition can be expressed simply as determining whether $\exists i \in [a]\ \forall j \in [k]$ such that $\exists(t_{i,j} \rightsquigarrow s_{i,j})$ returns true or false. The condition for a given sequence is realized in graphs by making an "AND" gadget over the subpaths within each layer. The final condition is realized by making a graph which acts as an "OR" gadget over all possible sequences.

The reduction takes a monotone 3D grid graph and produces a planar graph by creating these "AND" and "OR" gadgets. We make "AND" gadgets for each sequence, one-by-one, keeping track of the current sequence of upward edges by representing each sequence as a k-dimensional vector which in its i^{th} element keeps track of which element of L_i is being used as the starting vertex for the upward edge. We start at the lexicographically smallest such vector and simply cycle through all valid vectors. First we create s' and t' which become the new source and target vertices respectively for the DPR problem. For each sequence, we use copies of some vertices from the 3D grid to make our planar graph and the selection of vertices is done on the basis of the upward edges used in that particular sequence and hence on the basis of vertices of the form $s_{i,j}$ and $t_{i,j}$. We use the labels of the vertices from the 3D grid when we actually mean their copies.

For each vector, we start by making an edge between s' and a copy of $s_{i,1}$ and between a copy of $t_{i,k}$ and t'. For every pair $s_{i,j}$ and $t_{i,j}$ in that sequence, the rectangular grid (of the j^{th} layer) defined by these two vertices is replicated and each rectangle is connected to the next one using (copies of) the corresponding upward edges ($t_{i,j}$ to $s_{i,j+1}$). This completes the "AND" gadget for that vector; there will be a path from $s_{i,1}$ to $t_{i,k}$ only if there are subpaths within all the layers. Doing this for each sequence and asking the reachability question for s' and t' completes the "OR" construction as there is a path from s' to t' if and only if there is a path through any one of the sequences.

Since $k = O(1)$, the k-dimensional vectors can be stored using space logarithmic in n and can be used to keep track of which sequence's graph is being made at the moment. The additional information needed to create any of the "AND" gadgets also uses only $O(\log(n))$ space and hence the overall reduction is easily seen to be a many-one logspace reduction. ■

Corollary 1: 3D-MGGR$_{O(1)}$ is also reducible to DPR with relaxed monotonicity conditions such that only one of the north-south and east-west directions is monotonous at a time. We denote such problems as layered grid graph reachability problems and denote the existence of monotonicity in a particular direction using subscripts. 3D-L$_{NU}$GGR$_{O(1)}$ (monotonous in North and Upward directions) and 3D-L$_{EU}$GGR$_{O(1)}$ (monotonous in East and Upward directions) both reduce to DPR via a slightly modified reduction.

Proof: If the north-south direction (the east-west direction) has monotonicity, then instead of a rectangle defined by the $s_{i,j}$ and $t_{i,j}$ vertices, we use the part of the grid between the rows (the columns) containing those vertices and include all the columns (the rows) until the boundary of the 2D grid. The underlying concept of the reduction stays the same, it is merely the construction of the "AND" gadget that gets slightly modified with the rest of the reduction carrying forward almost exactly as before. ■

B. 3D-MGGR$_{O(1)}$ $\leq^{\mathbf{AC^0}}$ DPR

We now give a reduction from 3D-MGGR$_{O(1)}$ to DPR that can be computed using an $\mathbf{AC^0}$ circuit. In an $\mathbf{AC^0}$ reduction, we are allowed to use polynomial sized, $O(1)$ depth circuits using unbounded fan-in AND and OR gates (only inputs can have NOT gates). This reduction uses less resources than the logspace reduction and is hence a "weaker" reduction, which in turn makes our result slightly "stronger".

Before we prove the theorem, we note a few functions that are computable using an $\mathbf{AC^0}$ circuit. Since we have AND and OR gates, slightly more complex Boolean functions, like XOR, are possible using constant depth circuits. These functions can then be combined to compare two integers and determine whether they are equal or not. Checking inequalities $(<, >, \leq, \geq)$ requires a slightly more complex circuit[2] but that is also possible due to the unbounded nature of the fan-in of $\mathbf{AC^0}$ circuit gates. We can also compute the sum and difference of two integers. One additional function that is used in our reduction and can be computed by an $\mathbf{AC^0}$ circuit (using a combination of the previous functions mentioned) is sorting of a constant number of integers.

Theorem 2: 3D-MGGR$_{O(1)}$ $\mathbf{AC^0}$ reduces to DPR.

Proof: We construct an instance of DPR similar to that in the proof of Theorem 1. Instead of enumerating all valid sequence of upward edges using k counters to store a k-dimensional vector, we use a flag for all possible vectors to see if it is a valid one. For each valid vector, we use an $\mathbf{AC^0}$ circuit to make copies of vertices and then complete the necessary edges to get the "AND" gadget as before.

We have the instance of 3D-MGGR$_{O(1)}$ consisting of the graph $G = (V, E)$, the source vertex s and the target vertex t. Each vertex in V is labeled using a 3-tuple containing the X, Y, and Z coordinates of the grid point where the vertex lies. Each edge is labeled using a pair of such 3-tuples. The instance of DPR will consist of a graph $H = (V', E')$, a new source vertex s' and a new target vertex t'. Vertices in V' are copies of the original vertex labeled using the k-dimensional vector defining the sequence that the particular vertex belongs to (which copy is it), and the 3-tuple providing the original position of the vertex. Edges are labeled using the pair of labels of the vertices it belongs to, and these are exactly as those present in the reduction in the proof of Theorem 1.

We use a small circuit (referred to here as a "flag" circuit) to check for each vertex $v_i \in V$, if it has an upward edge. If it has, then the "flag" circuit for v_i outputs 1, else it outputs 0. The various"flag" circuit's output gates are labeled using the labels of the corresponding vertex. These outputs act as inputs to "valid vector" circuits which output whether the set of input vertices are a valid sequence of upward edges. The input to the "valid vector" circuits are made by making all possible choices of k "flag" circuits out of all the km^2 "flag" circuits. Thus there are $\binom{km^2}{k}$ "valid vector" circuits.

Each "valid vector" circuit checks if the set of input vertices form a valid sequence of upward edges. The first check is whether s and t are both a part of the set. Since there are exactly k inputs, we can easily make the rest of the checks in $\mathbf{AC^0}$. For the vector to be a valid choice, it should be reachable by the vertex before it. This is checked by first sorting all the vertices (first on Z coordinate, then Y, then X) which happens in $\mathbf{AC^0}$ since k is constant. Then for each vertex we check if the next vertex belongs to the next level and if the X and Y coordinates of the next vertex are at least as much as that of the current vertex. If all these checks pass, then the vector represents a valid sequence of upward edges and the "valid vector" circuit outputs a 1, else it outputs 0.

Once we have all the valid vectors, we go on to make copies of the vertices and the required edges. For each vector, we make all the vertices as we did in the reduction for Theorem 1. These vertices are labeled with the k-dimensional vector followed by the 3-tuple that originally represented it in G. We then simply check if a particular edge must exist or not. For all edges that might start at s', we only allow those that go to $s_{i,1}$, similarly we only allow edges to go to t' if they start at $t_{i,k}$, and do this for all possible vectors i. All edges from $t_{i,j}$ for $j < k$ must go to $s_{i,j+1}$. All edges that are part of the rectangular subgrid of a layer are checked on the basis of the vector label, using circuits to check whether vertices are within the rectangular subgrid; this requires simple inequalities to be checked.

Each of these are simple checks that can all be done using $\mathbf{AC^0}$ circuits. Since all components of the construction can be done in $\mathbf{AC^0}$, the overall construction can be done in $\mathbf{AC^0}$. Thus, H is constructed similar to the construction in the proof of Theorem 1. Since we have already shown how this construction preserves reachability, this reduction is an $\mathbf{AC^0}$ reduction and thus 3D-MGGR$_{O(1)}$ $\leq^{\mathbf{AC^0}}$ DPR. ■

C. 3D-L$_{NE}$GGR$_2$ *is* ***NL*-Complete**

Now we show that if we remove the monotonicity conditions only in the Z direction, even then the reachability problem becomes **NL**-Complete. For this we provide a many-one logspace reduction from a known **NL**-Complete problem to 3D-L$_{NE}$GGR$_2$ (layered along North and East directions) with monotonicity only in X and Y directions.

Theorem 3: (Due to Bourke et al. [1])
The st-connectivity problem for (geometric) thickness-two graphs is **NL**-Complete. Moreover, each transparency is a monotone grid graph. □

Theorem 4: The 3D-L$_{NE}$GGR$_2$ problem with monotonicity only in the X and Y directions is **NL**-Complete.

[2]For example, for checking whether $a < b$, we use the following circuit. For each bit a_i, check if all bits of higher significance are equal to corresponding bits of b, and if they are, then check if a_i is a 0 while b_i is a 1. If the overall result for any bit is 1, then $a < b$.

Proof: We know that the reachability problem for geometric thickness two graphs with each transparency being a monotone grid graph, is **NL**-Complete. We take an arbitrary instance of such a problem, say graph G along with source vertex s and target vertex t and construct a corresponding instance of 3D-L_{NE}GGR$_2$, say graph H, with new source vertex s' and new target vertex t'.

Let the vertex set of G be V and the edges in the two transparencies be E_1 and E_2. We know that V can be arranged into a grid and thus each vertex has a label of the form (i, j) where i denotes its position along the X axis and j denotes the position along the Y axis. The edges in both E_1 and E_2 are monotone in nature due to which we can assume that s and t are on diagonally opposite ends of the grid. To construct H, we will make two copies of all vertices and put one each in the two XY-plane layers of H. The edges from a transparency get mapped to the copies of vertices in the corresponding layer. Finally, we connect all pairs of copies of vertices ($\forall u \in V$, copy of vertex u from layer 1 with the corresponding copy in layer 2) with bidirectional edges in the Z direction.

We define $x(u)$ to be a function that returns the X coordinate of a vertex u and similarly $y(u)$ to be a function that returns the Y coordinate. Using these we proceed to define our instance of H. Let V' be the vertex set of H, we define $V' = V_1 \cup V_2$ where V_1 and V_2 are given by Equation 1 and Equation 2 respectively, and correspond to the vertices in layer one and layer two respectively. Similarly, we define the edge set of H as $E' = E_1' \cup E_2' \cup E_3$ where E_1 and E_2 contain the edges from layer 1 and layer 2 respectively (Equations 3 and 4) and E_3 has all the bidirectional edges in the Z direction between corresponding vertices (Equation 5). Finally, we assign[3] s' to be the copy of s in V_1 and t' to be the copy of t in V_2.

$$V_1 = \Big\{ \big(x(u),\, y(u),\, 1\big) \;\big|\; \forall u \in V \Big\} \tag{1}$$

$$V_2 = \Big\{ \big(x(u),\, y(u),\, 2\big) \;\big|\; \forall u \in V \Big\} \tag{2}$$

$$E_1' = \Big\{ \big((x(u),\, y(u),\, 1),\, (x(v),\, y(v),\, 1)\big) \;\big|\; \forall (u,\, v) \in E_1 \Big\} \tag{3}$$

$$E_2' = \Big\{ \big((x(u),\, y(u),\, 2),\, (x(v),\, y(v),\, 2)\big) \;\big|\; \forall (u,\, v) \in E_2 \Big\} \tag{4}$$

$$E_3 = \big\{ \big((x(u),\, y(u),\, 1),\, (x(u),\, y(u),\, 2)\big) \;\big|\; \forall u \in V \big\} \tag{5}$$

Any path from s to t will use some sequence of edges from E and these will result in some sequence of chosen transparencies throughout the path. Since we are mapping all the edges within a transparency into edges in different layers, to replicate the same path, we only need a way to move from one layer to another. This is provided by the bidirectional edges defined in E_3, which allows us to move from a vertex in one layer to the corresponding vertex in the other layer.

Thus, our mapping of vertices and the bidirectional edges in the instance H suffices to retain the reachability between vertices from the instance G. Although every edge from the original path in G may need a corresponding additional bidirectional edge in the path in H, a maximum of twice the original number of edges is enough.

Our reduction is clearly possible in logspace and we also showed that it preserves reachability. Thus we have shown that 3D-L_{NE}GGR$_2$ is **NL**-Complete. ■

We note that the reduction mentioned above is actually a first order projection. In our reduction we are copying edges (mapping vertices and edges from transparencies to layers) or we are using constants (the bidirectional edges between all pairs), both of which are allowed in first order projections.

[3]Due to the bidirectional edges between the two layers, assigning the source and target vertices to either of the layers can be done without any loss of generality.

D. Further relaxation of restrictions and some open problems

We have seen that for monotone 3D grid graphs, restricting the number of layers to $O(1)$ allows us to answer the reachability question in **UL** $\cap$ **coUL**. At the same time, without monotonicity in the Z direction, we get an **NL**-Complete problem even after restricting the number of layers to 2. In this section we discuss some preliminary ideas regarding variants that allow us to further restrict monotonicity, albeit at the cost of certain other restrictions.

Most of these restrictions are motivated by the idea of an origami like unfolding of the 3D grid graphs. This can be visualized for two layers by imagining a folded sheet of paper. A folded sheet of paper, models a 3D grid graph with two layers. On unfolding it at the fold, we see that edges that were going upward/downwards are now reduced to those within the plane and hence the 3D grid graph reachability problem gets reduced to a planar reachability problem. One can easily imagine numerous extensions to this for graphs which have monotonicity in, say the X direction. For example, if all edges in the Z direction from one layer, lie on one column, and these columns are all in increasing order as we go up the layers, then we can perform a similar unfolding operation on such graphs to again get a planar reachability problem. This particular structure can be visualized by imaging Japanese handheld fans [16] and how they unfold.

Some interesting open problems exist here, regarding how to mathematically define more general classes which allow some such manipulations to be performed to achieve easy reductions. More work needs to be done to see how these ideas can be extended and/or combined to obtain a much broader class of graphs which may allow us to remove the restriction on the number of layers. For example, the Japanese fans structure can be reduced to planar reachability even if the number of layers is unrestricted. These could be combined with ideas for classes of graphs which allow us to completely do away with most monotonicity conditions. The first example of a folded sheet of paper is a simple example of such classes.

Studying such methods to relax restrictions, both on the number of layers as well as on the monotonicity would allow us to better understand the nature of the reachability problem which in turn sheds light on the relations between various complexity classes.

V. Concluding remarks and possible directions for future work

This project was aimed at studying problems from a space complexity perspective. The primary problem looked at was the reachability problem, which has varied complexities depending on the class of graphs that are looked at. The focus of this project was on grid graphs, particularly on 3D grid graphs; with an aim of better understanding the space complexity landscape within **NL**.

We looked at variants of the **NL**-Complete 3D-MGGR, including 3D-MGGR$_{O(1)}$ and 3D-L$_{NE}$GGR$_2$. These problems are of interest as the general 3D-MGGR was shown to be **NL**-Complete [1] while changing the conditions in the problem resulted in quite different complexity results. We have shown using both logspace and $\mathbf{AC^0}$ reductions that 3D-MGGR$_{O(1)}$ $\in$ **UL** $\cap$ **coUL**. Further, we show that merely removing the monotonicity conditions in the Z direction are enough to result in an **NL**-Complete problem even after restricting the number of layers to 2. We show this using what is actually a first order projection a weaker form of reductions. All of our results show that 3D-GGR is an interesting problem to be looked at from a space complexity point of view.

The results described in this thesis showcase the interesting properties of 3D grid graphs, how the monotonicity and number of layers affects the complexity of reachability. One shortcoming of most of our reductions is that the large blow up in the sizes of the resultant instances after our reduction. In Theorem 1, we see a blow up of the order of $m^{O(k)}$ which is a large quantity. If this can be reduced to the form of $m^{O(1)} \times f(k)$, then we can get stronger results. Such a form could potentially allow k to increase, for example, $f(k) = 2^{2^k}$ can allow $k = \log\log(n)$. These (and those mentioned in Section IV-D) are all open problems that would shed more light on the behavior of 3D grid graphs with respect to reachability.

Apart from these, other problems can also be looked at for the class of 3D grid graphs. Reachability in undirected 3D grid graphs might be useful to study to better understand complexity classes within **L**. Problems other than reachability, for example, length of paths, may also be of interest for this class of graphs. 3D grid graphs thus show a lot of potential for further study.

Acknowledgments

I would like to express my gratitude to Dr. Nutan Limaye for giving me the opportunity to work under her supervision and for her guidance and helpful feedback throughout this bachelor's thesis work. I would also like to thank Prof. Rahul Muthu for his continued support as the on-campus mentor during this period.

References

[1] Chris Bourke, Raghunath Tewari, and NV Vinodchandran. "Directed planar reachability is in unambiguous log-space". In: *ACM Transactions on Computation Theory (TOCT)* 1.1 (2009), p. 4.

[2] Avi Wigderson. "The complexity of graph connectivity". In: *Mathematical Foundations of Computer Science 1992* (1992), pp. 112–132.

[3] Neil Immerman. "Nondeterministic space is closed under complementation". In: *SIAM Journal on computing* 17.5 (1988), pp. 935–938.

[4] Róbert Szelepcsényi. "The method of forced enumeration for nondeterministic automata". In: *Acta informatica* 26.3 (1988), pp. 279–284.

[5] Omer Reingold. "Undirected connectivity in log-space". In: *Journal of the ACM (JACM)* 55.4 (2008), p. 17.

[6] Eric Allender, Samir Datta, and Sambuddha Roy. "The directed planar reachability problem". In: *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer. 2005, pp. 238–249.

[7] Sanjeev Arora and Boaz Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009.

[8] Sambuddha Roy. "Complexity theoretic aspects of planar restrictions and obliviousness". PhD thesis. Rutgers, The State University of New Jersey, 2006.

[9] Neil Immerman. "Descriptive complexity and model checking". In: *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer. 1998, pp. 1–4.

[10] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer Science & Business Media, 2013.

[11] Alan Gibbons. *Algorithmic graph theory*. Cambridge University Press, 1985.

[12] Michael B Dillencourt, David Eppstein, and DanielS Hirschberg. "Geometric thickness of complete graphs". In: *International Symposium on Graph Drawing*. Springer. 1998, pp. 102–110.

[13] Joan P Hutchinson, Thomas Shermer, and Andrew Vince. "On representations of some thickness-two graphs". In: *Computational Geometry* 13.3 (1999), pp. 161–171.

[14] David A Mix Barrington et al. "Searching constant width mazes captures the AC 0 hierarchy". In: *Annual Symposium on Theoretical Aspects of Computer Science*. Springer. 1998, pp. 73–83.

[15] DA Mix Barrington et al. "On monotone planar circuits". In: *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*. IEEE. 1999, pp. 24–31.

[16] Wikipedia. *Hand fan — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-April-2017]. 2017. URL: `https://en.wikipedia.org/w/index.php?title=Hand_fan&oldid=776203989`.