Karlsruhe Institute of Technology

Seminar Gems of Theoretical Computer Science

LOGSPACE, RandomWalks on Graphs and Universal Traversal Sequences

Summer term 2014

Institute of Theoretical Informatics (ITI)
Department of Informatics
Karlsruhe Institute of Technology (KIT)

Author: Patrick Niklaus
Betreuer: Jun.-Prof. Dr. Henning Meyerhenke

Institute of Theoretical Informatics (ITI)
Department of Informatics
Karlsruhe Institute of Technology
Am Fasanengarten 5
76128 Karlsruhe

Contents

Chapter 1

LOGSPACE, RandomWalks on Graphs and Universal Traversal Sequences

Patrick Niklaus

In this paper I will present the collected findings of Aleliunas et. al. [2] on the space complexity of deciding whether there exsists a path between two nodes a, b in graphs.

Interestingly the complexity of this decision problem is different for directed and undirected graphs. These differences can be exploited to characterize the two space-bounded complexity classes L and NL.

In the case of an undirected graph, we can employ an algorithm called *random walk* which does just that. Unlike common path finding algorithms like the famous Dijekstra algorithm, this algorithm only needs a logarithmic amount of storage to compute, and is inherently simplistic in its decription: Instead of chosing the next neighbour node to explore based on certain criteria, we just pick one at random and check if we reached our destination. It is not intuetively clear that this algorithm can decide whether there is a path between a and b correctly. Infact it can only do so with a certain probability, it is an *randomized* algorithm. However, by chosing a sufficent number of steps we can make sure it decides correctly with a probability higher than $\frac{1}{2}$, which we can still increase by just running the algorithm multiple times.

Furthermore this is deeply related to finding "routing directions", that visit all nodes in a graph and work in all graphs with a certain size, so called *universal traversal sequences*. We will show that for every graph that is $d - connected$, there exists such a sequences and its length is only polynominal in the number of nodes.

Further Research A lot of research has be done by Aldous et. al on providing tighter lower bounds for the minium number of steps it takes to visit all nodes in graph, both for special graphs e.g. b-ary trees [1] and general undirected graphs [3].

Applications Apart from the theoretical implications for complexity theory, RandomWalk and Universal Traversal Sequences have well known applications in Artificial Intelligence for state space exploration [4].

Chapter 2

Time and space complexity

2.1 Turing machine models

For this paper we use the common formal definition of a *turing machine*, which we will extent to have 3 seperate tapes: A read-only reading tape, a work tape that can be read and written abitarially, and a write-only output tape. We do this, to be able to ignore any reading operations done on the input, which is important to formalize sub-linear space bounds. We also require the reading head to stay only on the non-blank part of the input, to be sure we can not encode any additional information using the input tape.

A *turing machine* is a tuple $M = (Q, \Gamma, b, \Sigma, q_0, F, \delta)$ with:

Q Set of states

Γ Set of tape symbols

$b \in \Gamma$ Blank symbol

$\Sigma \subseteq \Gamma$ Set of input symbols

$q_0 \in Q$ Initial state

$F \subseteq Q$ Set of halting states.

$\delta : \Gamma \times Q \longrightarrow \Gamma \times Q \times \{L, R, N\}^3$ (partial) transition function

We will also use the notion of *turing machine acceptors*, which we can formalize by choosing a subset of the halting states as *accepting* states.

As you might have noticed, this TM definition is that of a determinstic TM. For non-determinstic TM we simply remove the restriction that δ is a (partial) function. Instead δ is a general relation, which for example implies that there can be more than one next configuration.

This results in the property that a computation of a non-determinstic TM is not a single sequence of configurations, but rather a set of sequences. A determinstic TM accepts if it halts in an accepting state. To redefine this for a NTM, we say it accepts an input, if there *exists at least one* sequence of configurations that ends in an accepting state and the NTM halts.

Furthermore we need to expand the notion of running time, which is defined for a determinstic TM as the number of configurations in an computation. For a non-determinstic TM we define the running time as the *maximum* number of configurations that lead to an accepting state, which is equivalent to the depth of the computation tree.

2.2 Decision problems and Complexity classes

A common class of problems are so called *Decision Problems*, that describe (in the most general case) the following problems: Given a language $L \subseteq \Sigma^*$ and a word $w \in \Sigma^*$: Is $w \in L$? To formalize this, we can use *turing machine acceptors*, that halt in an accepting state if $w \in L$ and in an non-accepting state otherwise.

Since we have defined a notion of running time and space usage for both turing machine models, we can use this to classify decision problems based on running time and space usage. In general we say, a language L is in a complexity class A if and only if the corresponding decision problem satisfies a certain constraint. For this paper, we are mainly interested in four complexity classes (note when we say *polynominally-bounded* we mean: bounded by a polynominal in the input length n):

P All decision problems that can be solved by a *deterministic* turing machine using only polynominally-bounded many steps.

NP All decision problems that can be solved by a *non-deterministic* turing machine using only polynominally-bounded many steps.

L All decision problems that can be solved by a *determinstic* turing machine using only log-bounded space for the compuation.

NL All decision problems that can be solved by a *non-determinstic* turing machine using only log-bounded space for the compuation.

Interestingly, simlar to the open question $P = NP$ it is also still undecied whether $L = NL$.

2.3 $NL \subseteq P$

In this section we want to assert where NL is placed in the complexity hierarchy. It should be clear that $L \subseteq NL$ and $P \subseteq NP$, so we are mainly interested in placing NL in that hierarchy. As it turns out, we can show that $NL \subseteq P$. Actually we can show much more:

Theorem 1. *Every turing machine that only needs logarithmic-bounded space for its computation and halts, also has a poly-bounded running time.*

Proof. A configuration of a TM is defined as the tuple (T, o, p, q) where $T : \mathbb{N} \longrightarrow \Gamma$ is the current working tape state, $o, p \in \mathbb{N}$ which are the current head position on the input tape and on the working tape respectively and $q \in Q$ is the current state.

For a TM that only needs logarithmic-bounded space, we know that there are only $|\Gamma|^{O(\log n)} \in n^{O(1)}$ posibilities for T, as the number of symbols that can be non-blank is bounded by $O(\log n)$. Since the number of states is finite and the head position on the tape can be bounded by $O(\log n)$ as well (for the input tape it is always bounded by n), we can give a bound on the number of configurations:

$n^{O(n)} \cdot n \cdot O(\log n)) \cdot |Q| \subset O(n^k)$ for a suitably chosen $k \in \mathbb{N}$.

Now we need to see that this results in a poly bounded running time for a NTM. Since we have an upper bound to the number of configurations, any cycle-free sequence of configuration that leads to a halting state is shorter than our bound. So a NTM only needs to iterate all configurations until a halting configuration is found, or the length of the sequence is longer than our bound. This can be done in polynominal running thus, $NL \subseteq P$.

$\square$

This proof is given for deterministic TM, since every nondeterministic TM can be transformted to an equivalent determinstic TM (with exponentially more states, but note that is still constant with regard to the input size).

2.4 Reducibilty and NL-completeness

In the case of $P = NP$ it has been proven valuable to search for certain "hard" problems that characterize the complexity class. Similary to the notion of *NP-Completeness*, that is certain problems that are at least as difficult as any other decision problem in that complexity class, we try to define *NL-Completeness*.

A common technique to show that a decision problem is NP-complete, is to use reductions, for example the *Polynominal-Time Many-One Reduction*. A decision problem A is said to be *poly-time many-one reducible* to a decision problem B if and only if there exists a function f that can be computed in poly-time, for which the follwing requirement holds:

$w \in A \Leftrightarrow f(w) \in B$

Naively applying the same technique here will not work, since the poly-time constraint on the function is much too loose. Since the transforming function has no log-space constraint, it could be used to solve all decision problems in NL thus yielding a trivial reduction.

So, for NL-Completeness we need to add a space constraint on the transformtion function: A decision problem A is said to be *log reducible* to B ($A \leq_{log} B$) if and only if there exists a function f that can be computed using logarithmic-bounded space (and thus is also poly-time canstraint as we saw in the previous section) and satisfies the following requirement:

$w \in A \Leftrightarrow f(w) \in B$

We say B is *NL-complete* iff $A \leq_{log} B$ for all decision problems $A \in NL$.

Chapter 3

PATH and UPATH

The complexity class NL has a prominent member: the reachability problem in graphs. It actually turns out that this problem is different for directed and undirected graphs, which we will explore in this section.

3.1 PATH

Given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$ the decision problem $PATH$ can be formulated as:
$(G, a, b) \in PATH \Leftrightarrow$ there exsits a path from a to b in G.

3.1.1 NL-completeness

Theorem 2. *PATH is NL-complete.*
Based on [5] Solution to Excerise 5.3.

Proof. It is clear that $PATH \in NL$: Non-deterministically guess a path from a to b. This can be done by non-deterministically generating a sequence of n nodes in the graph and checking to see if adjacent pairs in the sequence are adjacent in the graph and if both a and b occur in the sequence. Note: This works since each cycle-free path in a graph has at most n nodes, so if there is one the NTM will find it. The only space needed on the working tape is the space to store a pair of nodes and a counter, which is all bounded by $\log n$. Now let $A \in NL$ via a $O(\log n)$ space-bounded machine M. As we saw in the proof of Theorem 1, this machine has at most polynomially many configurations on an input of length n. The desired reduction of A to $PATH$ outputs for any x the graph in which each such configuration is a node, and the there is an edge from configuration c_i to c_j if c_j is a configuration that could follow c_i in the computation on input x. This can be done, for example, by producing for each configuration the finite list of possible successor configurations. Note: This can be computed by a TM that also has logarithmic space-bound, even if the output is linear in the input since, since we do not consider the output tape! By adding a single end node, for all accepting configurations, we can use $PATH$ to decide if there exists a path from the start configuration to the end-node. If so, we accept, otherwise we reject. □

3.2 UPATH

Similar to $PATH$ we define $UPATH$ as:

$(G, a, b) \in PATH \Leftrightarrow$ there exsits a path from a to b in G.
where G is an *undirected* graph.

3.3 UPATH vs. PATH

One might be tempted to adapt the previous proof of Theorem 2 for NP-completeness of $PATH$ to $UPATH$, but this will not work, since the given reduction will create invalid results for undirected graphs: It is often not possible to transition from c_i to c_j, even if you can transition from c_j to c_i. Thus there can be a lot of invalid computation paths in the undirected graph, which $UPATH$ would accept.

This might already be a hint, that $UPATH$ is an easier problem than $PATH$, in fact we will see, that using randomization, we can solve $UPATH$ using no non-determinism at all.

Chapter 4

RL

To differenciate $PATH$ and $UPATH$ even more, we will introduce yet another complexity class called RL.

RL contains all the decision problems that can be decided by a TM that only needs logarithmic-bounded spaced for the computation, but is allowed to execute a *random* decision at each step *and only uses polynominal-bounded many steps.*

Since we now have a *randomized* acceptor M, we need to redefine the notion of accepting once more:
$x \in A \Rightarrow Pr[M \text{ accepts } x] \geq \frac{1}{2}$
$x \notin A \Rightarrow Pr[M \text{ accepts } x] = 0$

4.1 randomized vs. non-determinstic

One might wonder what the relation between RL and NL is. For this it is important not to confuse the concepts of *random* (as used by RL) and *non-determinstic* (as used by NL). Non-determinism allows the TM to guess the *correct* transition in each step. A randomized TM can use a random value to determine the next transition, but for the same random value it will always act deterministically. Thus we see that every determinstic TM is a randomized TM that takes exactly zero random decisions and every randomized TM can be simulated by non-deterministic TM, by replacing random decisions with non-deterministic transisions to the corresponding states that could be chosen.

To conclude we see that: $L \subseteq RL \subseteq NL$

4.2 Proof that poly-running time bound is required

- By counter example

Chapter 5

Random Walk

5.1 The algorithms

- definition
- show that it can be computed using $\log n$ space

5.2 Application on PATH

- Show why this algorithm does not work for deciding PATH

5.3 Correctness for UPATH

Show that RandomWalk satisfies the properties of a decider for UPATH in RL:

1. Show that $P_v = \lim n \longrightarrow \infty \frac{|\{i \leq n | v = v_i\}|}{n}$
   - Proof that $P_{(u,v)} = \frac{1}{2 \cdot e}$
   - Proof that $P_v = d(v) \cdot P_{(u,v)}$
2. Show that the expected number of steps of a random walk of a to visit all nodes once is bounded by a polynominal.
3. Show that b appears with probability higher than $\frac{1}{2}$ on such a random walk.

Chapter 6

Universal Traversal Sequence

6.1 Definition

- Visit all nodes in a d-regular Graph (why d-regular: formalism, node numbering..)

6.2 Relation to previous proof

- usage of probability applicifcation (only finitly many d-regular graphs of a certain size)

References

[1] David J Aldous. Random walk covering of some special trees. *Journal of mathematical analysis and applications*, 157(1):271–283, 1991.

[2] Romas Aleliunas, Richard M Karp, Richard J Lipton, Laszlo Lovasz, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 218–223. IEEE, 1979.

[3] Uriel Feige. A tight lower bound on the cover time for random walks on graphs. *Random Structures & Algorithms*, 6(4):433–438, 1995.

[4] Michal Kouckỳ. Universal traversal sequences with backtracking. *Journal of Computer and System Sciences*, 65(4):717–726, 2002.

[5] Uwe Schöning and Randall Pruim. *Gems of theoretical computer science.* Springer, 1998.