Karlsruhe Institute of Technology

Seminar
Gems of Theoretical Computer Science

Summer term 2014

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

Betreuer: Jun.-Prof. Dr. Dennis Hofheinz

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. 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.

Chapter 2

Time and space complexity

2.1 Turing machine models

- deterministic <-> non-determinstic
- definitions for runtime and space usage for both models

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 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 runtime and space usage for both turing machine models, we can use this to classify decision problems based on runtime 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: We show that every turing machine that only needs logarithmic-bounded space and halts, also has a poly-bounded runtime.

2.3.1 Proof

A configuration of a TM is defined as the tuple (T, p, q) where $T : \mathbb{N} \longrightarrow \Gamma$ is the current tape state, $p \in \mathbb{N}$ is the current reading head position 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, that mean

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

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.

3.2 PATH is NL-complete

Proof

3.3 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.

3.4 UPATH <-> PATH

- Motivate why previous proof for NL-completeness fails for UPATH
- Motivate rest of paper: Showing that UPATH is easier than PATH

Chapter 4

RL

4.1 Definition

- requirements
- modified definition of accepting

4.2 Difference between randomized and non-determinstic

- Motivation: $L \subseteq RL \subseteq NL$

4.3 Proof that poly-runtime 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