(Inspiration From)
My REU at DIMACS

Ryan Williams (MIT)

Three Decades of DIMACS, November 2019
My first job where I was *PAID* to think!!

Very exciting. I stayed in a dorm that looked like this:

And I walked to this building, nearly every day:
What I Thought About: Part 1

The Complexity of Reversible Computing

Dieter van Melkebeek (then a DIMACS postdoc) asked:

For every deterministic algorithm using time $t$ and $s$ space, is there an equivalent reversible algorithm using $O(t)$ time and $O(s)$ space?

Deterministic: Each “next step” in the computation is unique

Reversible: Each “previous step” is (also) unique!

Every step taken has a well-defined "reverse", so no "erasing" of any information unless it can be reconstructed.
This Was a Hard Problem!

But there were lots of interesting papers to read...

Reversible $2^{0(S)}$ time and $O(S)$ space

TIME/SPACE TRADE-OFFS FOR REVERSIBLE COMPUTATION*

CHARLES H. BENNETT†

Abstract. A reversible Turing machine is one whose transition function is 1:1, so the description (ID) has more than one predecessor. Using a pebbling argument, this paper shows that $\epsilon > 0$, ordinary multitape Turing machines using time $T$ and space $S$ can be simulated using time $O(T^{1+\epsilon})$ and space $O(S \log T)$ or in linear time and space $O(ST^\epsilon)$. The fact in particular that reversible machines can simulate ordinary ones in quadratic space. This means that all machines that save their input, thereby insuring a global 1:1 relation between IDs, even when the function being computed is many-to-one. Reversible machines that input can of course compute only 1:1 partial recursive functions and indeed provide a notion of such functions. The time/space cost of computing a 1:1 function on such a machine will be a small polynomial to the cost of computing the function and its inverse on an ordinary machine.

Reversible $O(T^{1+\epsilon})$ time and $O(S \log T)$ space, for all $\epsilon > 0$
And I Found Something New!

But there were lots of interesting papers to read...

Reversible \( O(T^{1+\epsilon}) \) time and \( O(S \log T) \) space, for all \( \epsilon > 0 \)

Reversible \( 2^{O(S)} \) time and \( O(S) \) space

Reversible Turing machine is one where the input description (ID) has more than one predecessor and for all \( \epsilon > 0 \), ordinary multitape Turing machines using time \( O(T^{1+\epsilon}) \) and space \( O(S \log T) \) or more in particular that reversible machines can simulate reversible machines that save their input, there are ID IDs, even when the function being computed is an input can of course compute only 1:1 partial functions of such functions. The time-space cost of computing a small polynomial to the cost of computing the

Klaus-Jörn Lange
Universität Tübingen, D-72076 Tübingen, Germany
klaus-lange@informatik.uni-tuebingen.de

and

McKenzie¹ and Alain Tapp¹

Département de recherche opérationnelle, Université de Montréal,
C.P. 6128, Station ‘A’, Montréal, H3C 3J7 Canada
iro.umontreal.ca, tappa@iro.umontreal.ca

This paper describes the simulation of an \( S(n) \) space-bounded deterministic Turing machine by a reversible Turing machine operating in space \( S(n) \). It thus answers a question posed by Bennett in 1989 and refutes the conjecture, made by Li and Vitanyi in 1996, that any reversible simulation of an irreversible computation must obey Bennett’s reversible pebble game rules. © 2000 Academic Press
And I Found Something New!

Space-Efficient Reversible Simulations

Ryan Williams

September 22, 2000

Abstract

We study constructions that convert arbitrary deterministic Turing machines to reversible machines; i.e. reversible simulations. Specifically, we study space-efficient simulations; that is, the resulting reversible machine uses $O(f(S))$ space, where $S$ is the space usage of the original machine and $f$ is very close to linear (say, $n \log n$ or smaller). We generalize the previous results on this reversibility problem by proving a general theorem incorporating two simulations: one is space-efficient ($O(S)$) and is due to Lange, McKenzie, and Tapp[5]; the other is time-efficient ($O(T^{1+\epsilon})$ for any $\epsilon > 0$, where $T$ is the time usage of the original machine) and is due to Bennett[2]. Corollaries of our general theorem give interesting new time-space tradeoffs. One is that for any unbounded space constructible $f(n) = o(T(n))$, there is a reversible simulation using $O(S \log f(S))$ space and $O(f(S)^{1+\epsilon} e^{\min(\epsilon T/f(S))})$ time, for any $\epsilon > 0$. This gives the first reversible simulation that uses time subexponential in $T$, and space subquadratic in $S$.

Example: Reversible $2^{O\left(\frac{T}{\text{poly}(S)}\right)}$ time and $O(S \log S)$ space

Similar results published by [Buhrman, Tromp, Vitanyi, ICALP 2001]
The Complexity of the SAT Problem

Lance Fortnow and Dieter had recently improved work of Lipton and Viglas from the previous year

Satisfiability Problem (SAT)
Input: An encoding of a propositional Boolean formula $F$ (composed of AND, OR, NOT) over Boolean variables $x$
Decide: Is there a 0-1 assignment to $x$ such that $F(x) = 1$?

Theorem [Cook-Levin 70’s] SAT is NP-Complete
Corollary $\text{SAT} \in P \iff P = \text{NP}$
P vs NP is currently out of reach \textit{(as far as I know...)}
But there \textit{has} been some progress on:

\begin{itemize}
  \item \textbf{Is LOGSPACE = NP?}
  \item \textbf{Is there a SAT algorithm that treats its input as read-only and uses only O(log n) additional memory, on formulas encoded in n bits?}
\end{itemize}

We believe: \textbf{NO WAY!}

\begin{itemize}
  \item \textbf{LOGSPACE} \subseteq \textbf{P} \subseteq \textbf{NP}
  \item so \textbf{LOGSPACE} \neq \textbf{NP} is necessary for \textbf{P} \neq \textbf{NP}
\end{itemize}
Progress on LOGSPACE vs NP

LOGSPACE $\neq$ NP
\[ \iff \forall k, \text{ SAT can’t be solved in } n^k \text{ time and } O(\log n) \text{ space} \]

Theorem [Fortnow ’97] There’s an $\epsilon > 0$ such that
SAT can’t be solved in $n^{1+\epsilon}$ time and $n^{o(1)}$ space

Theorem [Lipton Viglas ’99]
SAT can’t be solved in $n^{\sqrt{2}}$ time and $n^{o(1)}$ space

Theorem [Fortnow, Van Melkebeek ’00]
SAT can’t be solved in $n^\phi$ time and $n^{o(1)}$ space, $\phi = 1.618 \ldots$

- These theorems hold for many computational models
  (Random access machines, Programming languages, etc.)

- Theorems hold for other NP-hard problems as well
  (Vertex Cover, Hamiltonian Path, Max Cut, etc.)
(1) Assume you have a “great” algorithm for a hard problem e.g. a SAT algorithm using $n^{1.4}$ time and $O(\log n)$ space.

(2) Use the great algorithm as a subroutine in algorithms for solving harder problems, e.g., in the poly-time hierarchy.

(3) Obtain algorithms so great they do the impossible e.g. show all $n^2$ time algorithms can be re-implemented to run in $n^{1.9}$ time (this is false, by diagonalization!)

The strategy is an algorithmic design problem! I marveled at this idea for a very long time. It has been very influential in some of my work.
[Lipton-Viglas’99] Suppose SAT can be solved in \( n^c \) time and log space.

Then, because SAT is complete, every problem solvable in nondeterministic \( n^k \) time is solvable in *essentially* \( n^{ck} \) time and log space.

1. For all \( k \), \( \text{NTIME}[n^k] \subseteq \text{TIME-SPACE}[n^{ck}, \log n] \)

All \( L \in \text{TIME-SPACE}[n^{ck}, \log n] \) can be simulated in \( \sim n^{ck/2} \) time by a machine \( M \) that starts working *existentially* (guessing a computation path), then “alternates” to work *universally* (trying all paths in parallel).

2. \( \text{TIME-SPACE}[n^{ck}, \log n] \subseteq \Sigma_2\text{-TIME}[n^{ck/2} \log n] \)

Applying our SAT algorithm, we can replace the ‘universal’ part of \( M \) with a *deterministic* \( n^{c^2k/2} \) time computation, to obtain

3. \( \Sigma_2\text{-TIME}[n^{ck/2} \log n] \subseteq \text{NTIME}[n^{c^{2k/2}}] \)

Together, (1), (2), (3) imply \( \text{NTIME}[n^k] \subseteq \text{NTIME}[n^{c^{2k/2}}] \).

When \( c < 2^{1/2} \) this is false, by the *nondeterministic* time hierarchy!
The Influence on My Work

I learned all of this while at DIMACS! But it took me a few more years to UNDERSTAND it...

**Theorem [W’05, W’07]**
SAT cannot be solved in $n^2 \cos(\pi/7)$ time and $n^{o(1)}$ space

$\sim 1.801$

(largest root of $x^3 - x^2 - 2x + 1 = 0$)

All above lower bounds (and more) can be unified under a common formal proof system

I wrote code for a “theorem-prover” to find new proofs...

**Theorem [BW’12]** The $n^2 \cos(\pi/7)$ time lower bound for SAT is the best possible for this proof system!

OPEN: Improve this lower bound!
The Influence on My Work

Slowly, I began viewing every lower bound problem with a potential “algorithmic argument”!

I want to prove: “There is no good algorithm for X”
I will do this by assuming there is a good algorithm for X,
And deriving $\text{NTIME}(t) \subseteq \text{NTIME}(o(t))$ [contradiction]

Of course, much of the time, such an approach makes no sense...

But I had a suspicion this thinking could be used to attack a challenging question of Eric Allender (the next speaker):

Is $\text{NEXP} \subseteq \text{ACC}^0$?

Can all problems with exponentially-long answers checkable in exponential time be solved with nonuniform, polynomial-size circuit families of $O(1)$ depth with a limited gate basis? Almost certainly NO!!
Theorem (2011): \( \text{NEXP} \not\subset \text{ACC}^0 \)

Let \( \mathbb{C} \) be a “typical” circuit class (like \( \text{ACC}^0 \))

Thm A [W’11] (algorithm design \( \rightarrow \) lower bounds)
If SAT on \( n \)-input \( n^k \)-size \( \mathbb{C} \)-circuits is in \( O(2^n/n^k) \) time, \( \forall k \),
then \( \text{NEXP} \) does not have poly-size \( \mathbb{C} \)-circuits.

A generic way to get lower bounds from algorithms!

Thm B [W’11] (algorithm)
\( \exists \varepsilon, \text{SAT for ACC}^0 \) circuits of \( 2^{n^\varepsilon} \) size is in \( O(2^{n-n^\varepsilon}) \) time.
(Used a well-known representation of ACC\(^0\) from \( \sim 1990 \), that people long suspected should imply lower bounds)
Theorem A: Interesting Circuit SAT algorithms tell us about the *limitations* of circuits.

- SAT? YES/NO
- Runs in $\frac{2^n}{n^{10}}$ time when brute-force takes $2^n$

∃ “interesting” $f$

∀ $f$ can’t have small circuits
Idea of Theorem A

Thm A [W’11] (algorithm design $\rightarrow$ lower bounds)
If SAT on $n$-input $n^k$-size $\mathbb{C}$-circuits is in $O(2^n/n^k)$ time, $\forall k$, then NEXP does not have poly-size $\mathbb{C}$-circuits.

Idea: Show that if we assume both:

(1) NEXP has poly-size $\mathbb{C}$-circuits,
   AND

(2) there is a faster SAT algorithm for $\mathbb{C}$-circuits

Then we can show $\text{NTIME}[2^n] \subseteq \text{NTIME}[o(2^n)]$

(contradicts the nondeterministic time hierarchy!)
Proof Ideas in Theorem A

Assuming
(1) NEXP has poly-size $\mathbb{C}$-circuits
(2) a faster SAT algorithm for $\mathbb{C}$-circuits

Then: \( \text{NTIME}[2^n] \subseteq \text{NTIME}[o(2^n)] \)
(contradicts the nondeterministic time hierarchy!)

Take any problem $L$ in nondeterministic $2^n$ time. Given an input $x$, we “compute” $L$ on $x$ by:
1. Guessing a witness $y$ of $O(2^n)$ length.
2. Checking $y$ is a witness for $x$ in $O(2^n)$ time.

Want to “speed-up” both parts 1 and 2, using the above two assumptions.
Proof Ideas in Theorem A

Assuming (1) NEXP has poly-size \( \mathbb{C} \)-circuits
(2) a faster SAT algorithm for \( \mathbb{C} \)-circuits

Then: \( \text{NTIME}[2^n] \subseteq \text{NTIME}[o(2^n)] \)
(contradicts the nondeterministic time hierarchy!)

Take any problem \( L \) in nondeterministic \( 2^n \) time.
Given an input \( x \), we will “compute” \( L \) on \( x \) by:

1. **Use (1) to guess a witness \( y \) of \( o(2^n) \) length**
   (Easy Witness Lemma [IKW02]:
   if NEXP is in P/poly, then \( L \) has “small witnesses”)

2. **Use (2) to check \( y \) is a witness for \( x \) in \( o(2^n) \) time**

   Technical: Use the fact that NEXP has complete problems under very strong reductions [FLvMV].
Happy Birthday, DIMACS!