Hamiltonian paths

- given a directed graph \( G \) and two of its vertices \( s, t \), is there a path connecting \( s \) with \( t \) that visits all vertices exactly once?
- \( HAMPATH = \{ (G, s, t) | G \text{ is a directed graph, with a Hamiltonian path from } s \text{ to } t \} \)
- (theorem 7.46p286) \( HAMPATH \) is \( NP \)-complete
- reduce from 3\( SAT \)
- (the following figures are from the textbook, Sipser’s “introduction to the theory of computation”)
Hamiltonian paths

Figure 7.49
The high-level structure of $G$

Figure 7.50
The horizontal nodes in a diamond structure

Figure 7.51
The additional edges when clause $c_j$ contains $x_k$

Figure 7.52
The additional edges when clause $c_j$ contains $\overline{x}_i$

Figure 7.53
Zigzagging and zag-zagging through a diamond, as determined by the satisfying assignment

Figure 7.54
This situation cannot occur
**Undirected Hamiltonian path**

- *(theorem 7.55p291) UHAMPATH is NP-complete*
- reduce the directed version to the undirected

**Space complexity**

- characterization of problems in terms of space/memory requirements
- measuring space: use the Turing machine model
- space behaves "better" than time
- space can be re-used

**Space complexity**

- *(definition 8.1p303) The space complexity of a deterministic TM $M$ is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of tape cells that $M$ uses on any input of length $n$*
- the definition also requires that $M$ halts on all inputs

- *(definition 8.2p304) the space complexity classes $SPACE(f(n))$ and $NSPACE(f(n))$ are defined as follows:*
  - $SPACE(f(n)) = \{ L | \text{there exists a deterministic TM that decides } L \text{ in space } f(n) \}$
  - $NSPACE(f(n)) = \{ L | \text{there exists a non-deterministic TM that decides } L \text{ in space } f(n) \}$
Example

- solving SAT brute-force
- given a formula \( \varphi \) on the variables \( x_1, x_2, \ldots, x_n \), try all assignments to the variables and evaluate \( \varphi \) on each assignment
- what is the space complexity of this algorithm?

Space and time

- \( f \)-space computation may run for \( f^{2O(f)} \) time steps at most
- cannot run for more than that because it would repeat a configuration and therefore lead to an infinite loop

Savitch’s theorem

- comparing the power of deterministic and non-deterministic space
- (theorem 8.5p306) for any \( f(n) \geq n \),
  \( \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)) \)
- non-determinism gives little extra power in terms of space complexity
- the equivalent problem for time is the \( P \) versus \( NP \) question
- proof: simulation of a NTM deterministically.
  main idea: re-use space

Solving SAT brute-force

- given a formula \( \varphi \) on the variables \( x_1, x_2, \ldots, x_n \), try all assignments to the variables and evaluate \( \varphi \) on each assignment
- what is the space complexity of this algorithm?
- the space complexity is linear: re-use space
Savitch’s theorem

- straightforward simulation does not work
- $f$-space computation may go for $2^f$ time steps
- we can simulate all possible non-deterministic branches but that requires remembering all non-deterministic choices
- space requirements would be $2^f$ worst case.

Proof of Savitch’s theorem

- consider the yieldability (or reachability) problem: given a NTM $N$, input $w$, configurations $c_1, c_2$ and a number $t$, can $c_1$ yield $c_2$ in $t$ steps?
- if we have a way of solving this problem in limited space, then we can simulate a NTM $N$:
- given $N$ and input $x$, is it possible for the start configuration $c_1$ to yield the accept in the max possible number of steps?

Proof of Savitch’s theorem

- $CANYIELD(c_1, c_2, t)$ :
  - $t = 1$, test $c_1 = c_2$ or $c_1$ leads to $c_2$ directly (check $N$’s transition function)
  - for each configuration $c_m$ of $N$ on input $w$
  - run $CANYIELD(c_1, c_m, t/2)$
  - run $CANYIELD(c_m, c_2, t/2)$
  - if such a mid-point is found, accept
  - otherwise reject

- analysis of space requirements of the simulation
- we need space for the recursion (stack)
- $t$ starts as $2^f(n)$ and is halved at every recursive call
- so the depth of the recursion is $O(\log t)$ or $O(f(n))$
- each level of the recursion needs to store $c_1, c_2, t$ on the stack. that requires $O(f(n)$ space
- total space $O(f^2(n))$
- technical problem: we do not know $f(n)$ beforehand. try all possible values, reusing space
**PSPACE**

- (definition 8.6p308) *PSPACE* is the class of languages that are decidable in polynomial space on a deterministic TM
  \[ PSPACE = \bigcup_k \text{SPACE}(n^k) \]
- Define *NPSPACE*, the non-deterministic version of *PSPACE*
- *PSPACE* = *NPSPACE* by Savitch’s theorem
- Define *EXPTIME* as deterministic exponential time
  \[ P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq \text{EXPTIME} \]

**PSPACE-completeness**

- (definition 8.8p309) A language *B* is called *PSPACE*-complete if it satisfies two conditions:
  1. \( B \in \text{PSPACE} \)
  2. for every \( A \in \text{PSPACE} \), \( A \preceq_P B \)
- If *B* satisfies only (2) then it is called *PSPACE*-hard

**Quantified formulas**

- *SAT* is an *NP*-complete problem
- Involves boolean formulas, but no quantifiers
- Quantified boolean formulas: boolean formulas with existential or universal quantifiers
- Examples:
  - \( \forall x(x + 1 > x) \)
  - \( \forall x \exists y((x \lor y) \land (\neg x \lor \neg y)) \)
  - All variables quantified: fully quantified formulas or sentences
  - Fully quantified formulas are either true or false

**TQBF**

- True quantified boolean formula *TQBF*
- Given a fully quantified boolean formula, decide whether it is true or false
- \( TQBF = \{ \varphi | \varphi \text{ is a true fully quantified boolean formula} \} \)
- *TQBF* is *PSPACE*-complete
(Theorem 8.9p311) TQBF is PSPACE-complete

- Proof:
  - **TQBF ∈ PSPACE**: try out all possible assignments, reusing space
  - All of PSPACE reduces to TQBF: encode the simulation of any PSPACE computation by a formula

**TQBF ∈ PSPACE**

- Here is a polynomial space algorithm \( T(\varphi) \) for TQBF on input \( \varphi \), \( T(\varphi) \):
  1. If \( \varphi \) has no quantifiers, just evaluate it
  2. If \( \varphi \) is \( \exists x \psi \), call \( T(\psi) \) once with \( x = 0 \) and once with \( x = 1 \)
     accept if any of the two accepts otherwise reject
  3. If \( \varphi \) is \( \forall x \psi \), call \( T(\psi) \) once with \( x = 0 \) and once with \( x = 1 \)
     accept if both of them accept, otherwise reject
- Space required: depth of the recursion is equal to the number of variables, and constant space for each recursive call

**TQBF is PSPACE-hard**

- Let \( A \) be in \( SPACE(n^k) \), decided by a TM \( M \). Reduce it to TQBF as follows
  1. The reduction will map any string \( w \) to a formula \( \varphi \) that is true if \( M \) accepts \( w \)
  2. Construction similar to the Cook-Levin theorem formula
  3. Formula cannot be used in the same way as in Cook-Levin theorem: the running time may be exponential (the computation tableau is too big)
  4. Solution: break up formula into parts and represent each part with the same 'subformula' plus quantifiers

- Construct a formula \( \varphi_{c_1,c_2,t} \)
  - \( \varphi_{c_1,c_2,t} = \exists m_1[\varphi_{c_1,m_1,\frac{t}{2}} \land \varphi_{m_1,c_2,\frac{t}{2}}] \)
  - Reduce formula size
  - \( \varphi_{c_1,c_2,t} = \exists m_1 \forall (c_3,c_4) \in \{(c_1,m_1),(m_1,c_2)\}[\varphi_{c_3,c_4,\frac{t}{2}}] \)
  - Use \( \forall x[\{x = y \lor x = z \} \rightarrow \ldots] \) instead of \( \forall x \in \{y,z\}[\ldots] \)
Other PSPACE-hard problems

- other PSPACE-complete problems include games and finding winning strategies in games
- example: game of GO in an $n \times n$ board
  variant of GO (bounded moves and some simplified rules)

Logspace

- sublinear space bounds, log $n$
- a machine can read the entire input, but does not have enough space to store it
- modify the Turing machine model to allow a read-only input tape, plus a working tape
- the space bound applies on the working tape only
- log-space is an class that contains interesting problems and has robustness properties under model and input encoding variations

L and NL

- $L$ is the class of languages decidable in deterministic logarithmic space
  \[ L = \text{SPACE}(\log n) \]
- NL is the class of languages decidable in non-deterministic logarithmic space
  \[ NL = \text{NSPACE}(\log n) \]

Is there a path?

- the reachability problem on directed graphs is in NL
- for TM with a read only input tape, define the configuration the TM on an input $w$ to include the contents of the work tape, the state, and the positions of all head pointers (including the input tape pointer)
- an $f(n)$-space machine may have at most $n^{2O(f(n))}$ configurations
- with this definition, Savitch’s theorem works for any space bound $f(n) \geq \log n$
Completeness and reductions

- characterizing $L$ and $NL$, using completeness
- (open) question $L = NL$?
- reducibility: polynomial time reducibility is not useful, since all $NL$ problems are reducible to one another
- poly-time reductions are too powerful to reveal interesting properties within $NL$
- use log-space reducibility

log-space reducibility

- (definition 8.21p324) a log-space transducer is a TM with a read only input tape and a write-only output tape that works in $O(\log n)$ space. The transducer computes a function $f : \Sigma^* \rightarrow \Sigma^*$
- $f$ is called a log-space computable function
- a language $A$ is called log space reducible to $B$ written $A \leq_L B$ if $A$ is mapping-reducible to $B$ by a log space computable function

$NL$-completeness

- (definition 8.22p324) a language $B$ is $NL$-complete if
  - $B \in NL$
  - every $A \in NL$ is log-space reducible to $B$
- if only the second property holds, then $B$ is called $NL$-hard
- if any $NL$-complete problem is in $L$ then $NL = L$

Path is $NL$-complete

- (theorem 8.25p325) PATH is $NL$-complete
- proof: construct a directed graph that represents the computation of a log-space computation
- (corollary 8.26p326) $NL \subseteq P$
the following is considered a surprising result

**(theorem 8.27p327)** $NL = co-NL$

- proof: show that $PATH$ is in $NL$
- Immerman-Szelepcsényi theorem: For reasonable $s(n) \geq \log n$, $NSPACE(s(n)) = co-NSPACE(s(n))$

### Complexity classes

- the known relationships among some classes

\[ L \subseteq NL = co-NL \subseteq P \subseteq PSPACE \]

- we know that $NL \neq PSPACE$
- we believe that all these containments are proper

### On NP

- graph theoretic properties and $NP$
- first order logic
  - how do you express reachability in first-order logic?
- existential second order logic $\exists P \varphi$ where $\varphi$ is first-order

### Fagin's theorem (adv)

- the class of all graph-theoretic properties expressible in existential second order logic is precisely $NP$