Complexity Theory

Lecture 11: The Space Complexity Analogue of P vs NP

Tom Gur

http://www.cl.cam.ac.uk/teaching/2324/Complexity

Summary: A Complexity Zoo

The key players:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

Summary: A Complexity Zoo

The key players:

$$\text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{NPSPACE} \subseteq \text{EXP} \subseteq \text{NEXP}$$

You should also know coNP, coNL, UP, R, RE, BQP (Quantum P)

Summary: A Complexity Zoo

The key players:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

You should also know coNP, coNL, UP, R, RE, BQP (Quantum P)

Bonus contemporary classes: IP, SZK, BPP, FP, FNP, PCP, QMA

Scaling up complexity results

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Given $x \in S$, we can emulate $\mathcal{A}(x01^{2^{|x|^k}})$ in polynomial space.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Given $x \in S$, we can emulate $\mathcal{A}(x01^{2^{|x|^k}})$ in polynomial space.

Thus $S \in \mathsf{PSPACE}$.

Padding arguments

We can scale up relations between complexity classes. For example:

$$\mathsf{L} = \mathsf{P} \implies \mathsf{PSPACE} = \mathsf{EXP}$$

Proof: Let $S \in \mathsf{EXP}$.

Then $S' = \{x01^{2^{|x|^k}} : x \in S\} \in \mathsf{P}$.

Hence, $S' \in \mathsf{L}$; denote the algorithm by $\mathcal{A}$.

Given $x \in S$, we can emulate $\mathcal{A}(x01^{2^{|x|^k}})$ in polynomial space.

Thus $S \in \mathsf{PSPACE}$.

A similar argument shows that if $\mathsf{P} = \mathsf{NP}$, then $\mathsf{EXP} = \mathsf{NEXP}$.

ST-Conn and NL

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.

Algorithm?

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.
Algorithm?

A simple search algorithm (BFS) solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.
Algorithm?

A simple search algorithm (BFS) solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.
Algorithm?

A simple search algorithm (BFS) solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.
Algorithm?

A simple search algorithm (BFS) solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

st-Connectivity

Recall the st-Connectivity problem: given a *directed* graph $G = (V, E)$ and two nodes $s, t \in V$, determine whether there is a path from s to t.
Algorithm?

A simple search algorithm (BFS) solves it:

1. mark node s, leaving other nodes unmarked, and initialise set S to $\{s\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if t is marked, accept else reject.

Complexity: $O(n^2)$ time, $O(n)$ space.

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:
    2.1 if $i = t$ then accept, else
    guess an index j ($\log n$ bits) and write it on the work space.

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:
   2.1 if $i = t$ then accept, else
   guess an index j ($\log n$ bits) and write it on the work space.
   2.2 if (i, j) is not an edge, reject, else replace i by j and return to (2).

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:
   2.1 if $i = t$ then accept, else
   guess an index j ($\log n$ bits) and write it on the work space.
   2.2 if (i, j) is not an edge, reject, else replace i by j and return to (2).

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:
   2.1 if $i = t$ then accept, else
   guess an index j ($\log n$ bits) and write it on the work space.
   2.2 if (i, j) is not an edge, reject, else replace i by j and return to (2).

When in vertex i, the algorithm tries all possible indices j in parallel.

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:
   2.1 if $i = t$ then accept, else
   guess an index j ($\log n$ bits) and write it on the work space.
   2.2 if (i, j) is not an edge, reject, else replace i by j and return to (2).

When in vertex i, the algorithm tries all possible indices j in parallel.

For edges (i,j), the computation can continue.

st-Conn is in NL

We can construct a (DFS-based) algorithm to show that the st-Conn is in NL:

1. write the index of node s in the work space;
2. for i, the index currently written on the work space:
   2.1 if $i = t$ then accept, else
   guess an index j ($\log n$ bits) and write it on the work space.
   2.2 if (i, j) is not an edge, reject, else replace i by j and return to (2).

When in vertex i, the algorithm tries all possible indices j in parallel.

For edges (i,j), the computation can continue.

If there is a path from s to s, there will be a computation that visits all the nodes on that path.

st-conn is NL-complete

The problem st-conn is in NL. Is it also NL-complete?

st-conn is NL-complete

The problem st-conn is in NL. Is it also NL-complete?

Definition (Logspace Reductions)
We write

$$A \leq_L B$$

if there is a reduction f of A to B that is computable by a deterministic Turing machine using $O(\log n)$ workspace

st-conn is NL-complete

The problem st-conn is in NL. Is it also NL-complete?

Definition (Logspace Reductions)
We write

$$A \leq_L B$$

if there is a reduction f of A to B that is computable by a deterministic Turing machine using $O(\log n)$ workspace

st-conn is NL-complete

The problem st-conn is in NL. Is it also NL-complete?

Definition (Logspace Reductions)
We write

$$A \leq_L B$$

if there is a reduction f of A to B that is computable by a deterministic Turing machine using $O(\log n)$ workspace

We saw last lecture an outline for the proof that st-Conn is in NL:

- Start with an NL machine.

st-conn is NL-complete

The problem st-conn is in NL. Is it also NL-complete?

Definition (Logspace Reductions)
We write

$$A \leq_L B$$

if there is a reduction f of A to B that is computable by a deterministic Turing machine using $O(\log n)$ workspace

We saw last lecture an outline for the proof that st-Conn is in NL:

- Start with an NL machine.
- Construct its configuration graph.

st-conn is NL-complete

The problem st-conn is in NL. Is it also NL-complete?

Definition (Logspace Reductions)
We write

$$A \leq_L B$$

if there is a reduction f of A to B that is computable by a deterministic Turing machine using $O(\log n)$ workspace

We saw last lecture an outline for the proof that st-Conn is in NL:

- Start with an NL machine.
- Construct its configuration graph.
- Run an st-Conn algorithm and accept iff it accepted.

L vs NL

L vs NL

The problem st-Conn is NL-complete. Can we solve it deterministically?

The problem st-Conn is NL-complete. Can we solve it deterministically?

Theorem (Savitch's Theorem)
st-Conn can be solved by a deterministic algorithm in $O((\log n)^2)$ space.

L vs NL

The problem st-Conn is NL-complete. Can we solve it deterministically?

Theorem (Savitch's Theorem)
st-Conn can be solved by a deterministic algorithm in $O((\log n)^2)$ space.

L vs NL

The problem st-Conn is NL-complete. Can we solve it deterministically?

Theorem (Savitch's Theorem)
st-Conn can be solved by a deterministic algorithm in $O((\log n)^2)$ space.

Consider the following recursive algorithm for determining whether there is a path from a to b of length at most i.

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

<u>Path(a, b, i)</u>

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

else (if $i > 1$), for each vertex v, check:

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

<u>Path(a, b, i)</u>

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

else (if $i > 1$), for each vertex v, check:

1. Path($a, v, \lfloor i/2 \rfloor$)

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

else (if $i > 1$), for each vertex v, check:

1. Path$(a, v, \lfloor i/2 \rfloor)$
2. Path$(v, b, \lceil i/2 \rceil)$

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

else (if $i > 1$), for each vertex v, check:

1. Path$(a, v, \lfloor i/2 \rfloor)$
2. Path$(v, b, \lceil i/2 \rceil)$

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

Path(a, b, i)

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

else (if $i > 1$), for each vertex v, check:

1. Path$(a, v, \lfloor i/2 \rfloor)$
2. Path$(v, b, \lceil i/2 \rceil)$

if such an v is found, then accept, else reject.

Savitch's Theorem

An $O((\log n)^2)$ space st-Conn deterministic algorithm:

<u>Path(a, b, i)</u>

if $i = 1$:

1. if (a, b) is an edge or $a = b$ accept
2. else reject

else (if $i > 1$), for each vertex v, check:

1. Path$(a, v, \lfloor i/2 \rfloor)$
2. Path$(v, b, \lceil i/2 \rceil)$

if such an v is found, then accept, else reject.

The maximum depth of recursion is $\log n$, and the number of bits of information kept at each stage is $3 \log n$.

Savitch's Theorem

The space efficient algorithm for reachability used on the configuration graph of a nondeterministic machine shows:

$$\mathsf{NSPACE}(f) \subseteq \mathsf{SPACE}(f^2)$$

for $f(n) \geq \log n$.

Savitch's Theorem

The space efficient algorithm for reachability used on the configuration graph of a nondeterministic machine shows:

$$\mathrm{NSPACE}(f) \subseteq \mathrm{SPACE}(f^2)$$

for $f(n) \geq \log n$.

This yields

$$\mathrm{PSPACE} = \mathrm{NPSPACE} = \text{co-}\mathrm{NPSPACE}.$$

Complementation

A still more clever algorithm for Reachability has been used to show that nondeterministic space classes are closed under complementation:

Complementation

A still more clever algorithm for Reachability has been used to show that nondeterministic space classes are closed under complementation:

If $f(n) \geq \log n$, then

$$\mathrm{NSPACE}(f) = \text{co-}\mathrm{NSPACE}(f)$$

Complementation

A still more clever algorithm for Reachability has been used to show that nondeterministic space classes are closed under complementation:

If $f(n) \geq \log n$, then

$$\mathrm{NSPACE}(f) = \text{co-}\mathrm{NSPACE}(f)$$

In particular

$$\mathrm{NL} = \text{co-}\mathrm{NL}.$$

Bonus: Zero-Knowledge Proofs