THE GRAPH ACCESSIBILITY PROBLEM (GAP)

- GAP: Given a directed graph $G = (V, E)$ and two vertices $u, v \in V$, determine whether there exists a path from $u$ to $v$

  - GAP $\in$ NL:
    - Algorithm N-GAP($G = (V, E), u, v$) returns $\top/\bot$:
      1. $x \leftarrow u$
      2. while $x \neq v$ do
         1. nondeterministically guess a value $y \in V$
         2. if $(x, y) \notin E$ then return $\bot$
         3. $x \leftarrow y$
      3. return $\top$

  - GAP $\in$ DSPACE($\log^2 n$):
    - Algorithm D-GAP($G = (V, E), u, v$) returns $\top/\bot$:
      1. return PATH($G, u, v, |V|$)
    - Algorithm PATH($G = (V, E), i, j, k$) returns $\top/\bot$:
      1. if $k = 0$ then return $i = j$ else if $k = 1$ then return $(i, j) \in E$
      2. else return $\exists l \in V : \text{PATH}(i, l, \lfloor k/2 \rfloor) \wedge \text{PATH}(l, j, \lfloor k/2 \rfloor)$
      3. $O(\log n)$ recursion depth and $O(\log n)$ storage per level $= O(\log^2 n)$ space
      4. GAP can be solved in parallel in $O(\log^2 n)$ time (see hypercube algorithm)

SPACE-BOUNDED COMPUTATIONS

- A Turing machine $M$ is $s(n)$-space bounded, $s : \mathbb{N} \rightarrow \mathbb{N}$ if
  1. $M$ is a Turing machine with a read-only input tape, a write-only output tape, and a (read-write) work tape
  2. The output tape is initially empty and each time the machine writes on that tape it writes a symbol into the square immediately adjacent to the right of the last overwritten tape square
  3. A configuration of $M$ is a tuple $\{(q, w, uav, \alpha)\}$ where $q$ is the current state, $w$ is the (read only) input, $uav$ is the content of the work tape, and $\alpha$ is the output produced so far
  4. There is no configuration $(q, w, uav, \alpha)$ such that $(s, w, \epsilon, e) \vdash_M (q, w, uav, \alpha)$ and $|uav| > s(|w|)$

  - DSPACE($s(n)$) $/$ NSPACE($s(n)$) $\rightarrow$ the class of all the decision problems solved by $s(n)$-space bounded, deterministic/nondeterministic Turing machines

  - Shorthand: $L = \text{DSPACE}(\log n)$, $NL = \text{NSPACE}(\log n)$, $\text{POLYLOGSPACE} = \bigcup_{k \geq 1} \text{DSPACE}(\log^k n) = \text{DSPACE}(\log^{O(1)} n)$

  - Note in passing: $\text{DSPACE}(s(n)) = \text{DSPACE}(s(n)/c)$ for all $c \in \mathbb{N}$

  - $L \subseteq NL \subseteq \mathcal{P}$; widely believed (but not proven) that all the inclusions are strict

DETERMINISTIC VS NONDETERMINISTIC SPACE

**Theorem (Savitch’s theorem)**

$\text{NSPACE}(s(n)) \subseteq \text{DSPACE}(s(n)^2)$ for most useful functions $s(n) = \Omega(\log n)$, including polynomials and poly-logarithms (space-constructible functions)

- Let $M$ be an $s(n)$-space bounded Turing machine
- Size of configuration graph: $2^{O(s(n))}$ vertices
- Use GAP to determine whether the accepting configuration is accessible from the initial configuration $\rightarrow (\log 2^{O(s(n))})^2 = O(s(n)^2)$ space

**Corollary**

- $NL \subseteq \text{DSPACE}(O(\log^2 n))$
- $\text{NSPACE}(\log^{O(1)} n) = \text{DSPACE}(\log^{O(1)} n) = \text{POLYLOGSPACE}$
- $\text{DSPACE}(n^{O(1)}) = \text{NSPACE}(n^{O(1)}) = \text{PSPACE}$

- Known that $\mathcal{P} \neq \text{POLYLOGSPACE}$; conjectured that $\mathcal{P} \not\subseteq \text{POLYLOGSPACE}$ and $\text{POLYLOGSPACE} \not\subseteq \mathcal{P}$
**LOG-SPACE COMPLETENESS**

- A language $A$ is log-space reducible to language $B$ ($A \leq_{log} B$) if there exists a function $\tau$ computable in logarithmic space such that $x \in A$ if $\tau(x) \in B$.

- Let $C$ be a class of languages:
  - $B$ is log-space hard for $C$ if $A \leq_{log} B$ for all $A \in C$.
  - $B$ is log-space complete for $C$ if $B$ is log-space hard for $C$ and $B \in C$.
  - $P$-complete stands for "log-space complete for $P$".

- How can we conclude that if a problem is $P$-complete and also in POLYLOGSPACE then $P \subseteq \text{POLYLOGSPACE}$?
  - Naive approach: given input $x$ for some problem $A \in P$, use the log-space machine $M_\epsilon$ that computes the log-space reduction from $A$ to a $P$-complete problem $B$, then run the machine $M_\epsilon$ (that accepts $B$) on $M_\epsilon(x)$.
  - This approach fails (not enough space to store $M_\epsilon(x)$).
  - However, we can modify the Turing machine $M_\epsilon$ to obtain $M'_\epsilon$ such that $M'_\epsilon(x, i)$ is the $i$-th bit of $M_\epsilon(x)$.
  - Every transitions of $M_\epsilon$ depends on a single input bit.
  - So instead of computing all the input $M_\epsilon(x)$ in advance, we use $M'_\epsilon$ on demand to obtain the particular bit needed by the current transition of $M_\epsilon$.

**THE PARALLEL COMPUTATION THESIS**

Theorem (The parallel computation thesis)

*Time on any reasonable parallel model is polynomially equivalent to the space used by a sequential machine.*

- Technically a conjecture rather than theorem because of the presence of "reasonable".
  - A "reasonable" parallel machine usually features restrictions on word size, instruction set, and parallelism.
  - Powerful theoretical tool.

**Corollary**

*All $P$-complete problems are inherently sequential unless $P \subseteq \text{POLYLOGSPACE}$.*

- It is likely that no $P$-complete problem is in POLYLOGSPACE.
- Therefore according to the parallel computation thesis they cannot be solved in parallel in $O(\log^{O(1)} n)$ time.
- The only possibility remaining is that they can be solved in parallel in polynomial time $\to$ no better than solving them sequentially.

**THE PARALLEL COMPUTATION THESIS (CONT’D)**

**“REASONABLE” PARALLEL MODELS**

- Restrictions on the instruction set:
  - One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ space by a deterministic Turing machine, where $t(n)$ is the running time of the parallel machine.
  - One-time unit cost instructions should be computable in $O(t(n)^{O(1)})$ time by a deterministic Turing machine (stronger than the above).

- Restrictions on the number of processors:
  - Most people regard a parallel machine as feasible if the number of processors is $n^{O(1)}$ (small machine) and the running time is $\log^{O(1)} n$ (fast machine).
  - However, the parallel computation thesis holds even if the number of processors is $2^{O(t(n))}$ or even $2^{O(t(n)^{O(1)})}$.

- Restrictions on the word size:
  - Normally the word size is $t(n)^{O(1)}$ though in practice the tighter restriction of $O(\log n)$ size is used for simplicity.