Models of Parallel Computation

Complexity Theory

Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic

Lukáš Charvát

This material was created with the support of the Czech Ministry of Education, Youth and Sports (project FRVŠ 166/2013/G1).

Parallel Computation

- Raises new questions from the point of view of complexity:
  - Given a task, can it be efficiently parallelised?
  - Is there a connection between the number of processes used and the required run time?
- We will investigate two models of parallel computation:
  - PRAM: Parallel Random Access Machine,
  - Boolean Circuits.
- A new complexity class **NC** containing algorithms that can be easily executed in parallel will be defined and described as well.

A Quick Revision of RAMs

- Random Access Machine (RAM):
  - An (infinite) array of registers $R = (r_0, r_1, \ldots)$.
  - Each register is capable of containing an arbitrarily large integer (possibly negative).
  - The possibility to directly access an arbitrary register.
  - Register 0 serves as an accumulator.
  - Program counter κ.
- A RAM program $\Pi = (\pi_1, \ldots, \pi_m)$ is a finite sequence of instructions.
- The input is placed in a finite array of input registers $I = (i_1, \ldots, i_n)$.
- Three addressing modes: j, $\uparrow j$, and $= j$.

Parallel Random Access machine (PRAM)

- A parallel extension of RAM.
- A PRAM program is a (possibly infinite) sequence of RAM programs: $P = (\Pi_0, \Pi_1, \Pi_2, \ldots)$ (one for each RAM).
- An infinite array of shared registers: $C = (c_0, c_1, c_2, \ldots)$
- Each RAM has its own array of local registers.
- New instructions for accessing shared registers are introduced.

Read and Write Conflicts

- What happens when several RAMs want to access the same shared registers at the same time?
- Three policies:
  - EREW: exclusive read and exclusive write,
  - CREW: concurrent read and exclusive write,
  - CRCW: simultaneous read and write allowed.
- We will assume the CREW policy further.

PRAM – A Note About Conventions

- Let $I = (i_1, \ldots, i_n)$ be the input placed into shared registers c_1, ..., c_n.
- Only a finite number of machines is activated to perform the computation:
  1. The RAM with the program Π_0 is activated first.
  2. Based on the number of integers in the input I, and its total length $len(I)$, Π_0 determines the number q of RAMs needed for the computation.
  3. Additional RAMs are activated.
- After all RAMs halt, the output $O = (o_1, \ldots, o_k)$ can be read from registers $c_1, \ldots, c_k$.

Function Computed by PRAM

Definition

Let P be a program, $D \subseteq \mathbb{Z}^*$ be a set of finite sequences of integers, and ϕ, t, p be functions s.t. $\phi : D \to D$, $t : \mathbb{N} \to \mathbb{N}$, $p : \mathbb{N} \to \mathbb{N}$. P computes F in parallel time t with p processors iff $\forall I = (i_1, \ldots, i_n) \in D$:

- *the PRAM running P activates less than $p(len(I))$ RAMs,*
- *all RAMs stop after at most $t(len(I))$ steps,*
- *$\phi(i_1, \ldots, i_n) = (c_1, \ldots, c_k)$.*

Proposition

A language L is decided in parallel time $n^{O(1)}$ with $n^{O(1)}$ processors iff L is decided in sequential time $n^{O(1)}$.

Corollary

The class of parallelly solvable problems is equivalent to the class **P**.

Boolean Circuits

Definition

A Boolean circuit C over the set of variables X is a finite directed acyclic graph with labeled nodes:

- *the input nodes are labeled with a variable $x \in X$ or with a constant 0 or 1,*
- *the gate nodes have one or more incoming edges and they are labelled with one of the Boolean functions $\{\wedge, \vee, \neg\}$[a],*
- *the output node has no outgoing edge.*

[a] the no. of incoming edges for $\wedge$ and $\vee$ is greater than one, and one for $\neg$

Definition

We denote by $size(C)$ the number of gates of C and by $depth(C)$ the maximum distance from an input to the output of C.

Deciding Languages with Circuits

- A single Boolean circuit cannot be used to decide languages with strings of arbitrary length.
- Problem can be solved by definition of a family $\{C_i\}$ of Boolean circuits.
- Each member of the family is then dedicated for deciding strings of a certain length.

Definition

A language $L \subseteq \{0,1\}^*$ is decided by a family of circuits $\{C_i\}$, where C_n takes n input variables, if $\forall x \in L\colon C_{|x|}(x) = 1 \iff x \in L$.

Definition

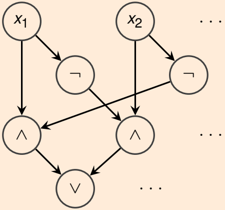
Let $d, s : \mathbb{N} \to \mathbb{N}$ be functions. We say that a family $\{C_i\}$ has depth d and size s iff $\forall n \in \mathbb{N}\colon depth(C_n) \leq d(n) \wedge size(C_n) \leq s(n)$.

Families of Boolean Circuits – Examples

Example

$L_{op} = \{x \in \{0,1\}^* \mid x \text{ has an odd number of 1s}\}$

- *each $\oplus$-gate is of the depth 3*
- *logarithmic depth*



Example

$L_{uhalt} = \{1^n \mid n \text{ encodes tuple } (M, x) \text{ such that TM } M \text{ halts on } x\}$

- *For each n such that $1^n \in L_{uhalt}$, the circuit C_n is a tree of $\wedge$-gates.*
- *Otherwise C_n is the constant-0 circuit.*
- *L_{uhalt} is clearly undecidable yet can be decided by a family of circuits of linear size.*

Uniform Boolean Circuits

- The description of the circuit family for L_{uhalt} presented in the last example is not computable.

Definition

A family of polynomially-sized circuits $\{C_i\}$ is logspace-uniform if there exists a logspace deterministic TM M which for every n computes the transformation $1^n \mapsto \overline{C}_n$ where $\overline{C}_n$ denotes the description of C_n.

Example

The circuit family for L_{op} is logspace-uniform.

Simulation between PRAMs and Boolean Circuits

Proposition

A function $f : \{0,1\}^* \to \{0,1\}^*$ *can be computed by a uniform family of circuits* $\{C_i\}$ *with depth* $d(n) = \log^{O(1)} n$ *and size* $s(n) = n^{O(1)}$ *iff* f *can be computed by a PRAM in parallel time* $t(n) = \log^{O(1)} n$ *with* $p(n) = n^{O(1)}$ *processors.*

Note that the notation $\log^k x$ is an abbreviation for $(\log x)^k$

Proof (Idea)

"$\Rightarrow$":

- *Based on* $|n|$*, compute the description of* C_n*.*
- *One circuit node* $\longleftrightarrow$ *one processor.*
- *Each processor computes its output and sends it (via shared registers) to all other processors that need it.*

Simulation between PRAMs and Boolean Circuits

Proof (Idea)

"$\Leftarrow$":

- *Configuration of the simulated PRAM is a bit-vector containing the value of the program counter and registers for each RAM.*
- *We must face three major problems for all RAMs:*
  1. *The type of instruction must be determined.*
  2. *Operand must be fetched (by examining all register-value pairs).*
  3. *Write conflicts must be resolved.*
- *The first two problems can be solved by redundancy (computing all possible instructions with all operands at once in parallel).*
- *The last problem involves recording of all writes in current step with subsequent conflict resolution.*

Nick's Class **NC**

Definition

The class **NC** *is the class of languages decidable in parallel time* $\log^{O(1)} n$ *with* $n^{O(1)}$ *processors.*

Lemma

$$\mathbf{NC} \subseteq \mathbf{P}$$

- Describes an efficient parallel computation:
  - polynomially many processors,
  - polylogarithmic time.
- **NC** is robust w.r.t. different PRAM models (and circuits).
- Includes problems such as list-ranking, matrix multiplication, sum of prefixes.

NC-reduction and P-completeness

- Similarly to the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question, we do not know whether the inclusion $\mathbf{NC} \subseteq \mathbf{P}$ is proper, or not.

Definition

A language L_1 is **NC***-reducible to language L_2 (denoted $L_1 \leq_{\mathbf{NC}} L_2$) iff there exist an* **NC***-computable function $r : \Sigma^* \to \Sigma^*$ such that*

$$x \in L_1 \iff r(x) \in L_2$$

Definition

A language L is **P***-hard if $\forall L' \in \mathbf{P} : L' \leq_{\mathbf{NC}} L$.*

Definition

A language L is **P***-complete if L is* **P***-hard and $L \in \mathbf{P}$.*

P-completeness of the Circuit Value Problem (CVP)

Definition

Let C be a circuit and x its input. A pair $(C, x) \in CVP \iff C(x) = 1$.

Proposition

The Circuit Value Problem (CVP) is **P***-complete.*

Proof

Assume TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f)$ that decides L in time $T(n)$ where $Q = \{q_0, \ldots, q_s\}$ and $\Gamma = \{a_1, \ldots, a_m\}$. We will design an **NC***-algorithm that given x computes a circuit $r(x)$ such that $x \in L$ iff $r(x) \in CVP$. The layered circuit $r(x)$ computes the following functions:*

$$\begin{aligned}
h(i, t) = 1 &\iff \text{the head of } M \text{ is on the } i\text{-th cell in the } t\text{-th step,} \\
c(i, j, t) = 1 &\iff \text{the contents of the } i\text{-th cell is } a_j \text{ in the } t\text{-th step,} \\
s(k, t) = 1 &\iff M \text{ is in the state } q_k \text{ in the } t\text{-th step.}
\end{aligned}$$

P-completeness of the Circuit Value Problem (CVP)

$$
\begin{aligned}
h(i,t) = 1 &\iff \text{the head of } M \text{ is on the } i\text{-th cell in the } t\text{-th step,}\\
c(i,j,t) = 1 &\iff \text{the contents of the } i\text{-th cell is } a_j \text{ in the } t\text{-th step,}\\
s(k,t) = 1 &\iff M \text{ is in the state } q_k \text{ in the } t\text{-th step.}
\end{aligned}
$$

Proof

We set the initial head position $h(1,0) = 1$ *and* $h(i,0) = 0$ *for all* $i > 1$*.*
Let $I_d = \{(k',j') \mid \delta(q_{k'}, a_{j'}) = (q_k, d)\}$ *for* $d \in \{L, R\}$ *and*
$I_\Gamma = \{(k',j') \mid \delta(q_{k'}, a_{j'}) = (q_k, a_j)\}$ *for* $a_j \in \Gamma$*. Then, for* $t > 0$*,*

$$
\begin{aligned}
h(i,t) = &\Big(h(i-1,t-1) \wedge \bigvee_{(k',j')\in I_R} c(i-1,j',t-1) \wedge s(k',t-1)\Big) \vee\\
&\Big(h(i+1,t-1) \wedge \bigvee_{(k',j')\in I_L} c(i+1,j',t-1) \wedge s(k',t-1)\Big) \vee\\
&\Big(h(i,t-1) \wedge \bigvee_{(k',j')\in I_\Gamma} c(i,j',t-1) \wedge s(k',t-1)\Big)
\end{aligned}
$$

$h(i,t)$ *can be evaluated by a* $\{\wedge, \vee\}$*-circuit of constant size* $O(|Q| \cdot |\Gamma|)$ *and can be generated in constant time using* $O(T^2(n))$ *processors.*

P-completeness of the Circuit Value Problem (CVP)

$$\begin{aligned} h(i,t) = 1 &\iff \text{the head of } M \text{ is on the } i\text{-th cell in the } t\text{-th step,} \\ c(i,j,t) = 1 &\iff \text{the contents of the } i\text{-th cell is } a_j \text{ in the } t\text{-th step,} \\ s(k,t) = 1 &\iff M \text{ is in the state } q_k \text{ in the } t\text{-th step.} \end{aligned}$$

Proof

We set the initial tape contents $c(i,j,0) = 1$ iff the i-th cell contains a_j. Let $W_j = \{(k',j') \mid \delta(q_{k'}, a_{j'}) = (q_k, a_j)\}$ where $a_j \in \Gamma$. Then, for $t > 0$,

$$c(i,j,t) = (\neg h(i,t-1) \ \wedge \ c(i,j,t-1)) \vee \left(h(i,t-1) \quad \wedge \ \left(\bigwedge_{(k',j') \in W_j} c(i,j',t-1) \wedge s(k',t-1) \right) \right)$$

Similarly, $c(i,j,t)$ can be evaluated by a $\{\wedge, \vee, \neg\}$-circuit of constant size and can be generated in constant time using $O(T^2(n))$ processors.

P-completeness of the Circuit Value Problem (CVP)

$$
\begin{aligned}
h(i,t) = 1 &\iff \text{the head of } M \text{ is on the } i\text{-th cell in the } t\text{-th step,}\\
c(i,j,t) = 1 &\iff \text{the contents of the } i\text{-th cell is } a_j \text{ in the } t\text{-th step,}\\
s(k,t) = 1 &\iff M \text{ is in the state } q_k \text{ in the } t\text{-th step.}
\end{aligned}
$$

Proof

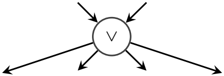
Finally, we set the initial state $s(0,0) = 1$ *and* $s(k,0) = 0$ *for all* $k > 0$*. Let* $S_k = \{(k', j') \mid \delta(q_{k'}, a_{j'}) = (q_k, x)\}$ *where* $x \in \Gamma \cup \{L, R\}$*. For* $t > 0$*,*

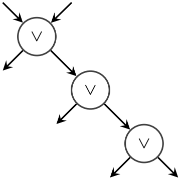
$$s(k,t) = \bigvee_{1 \le i \le T(n), (k',j') \in S_k} h(i, t-1) \wedge c(i, j', t-1) \wedge s(k', t-1)$$

$s(k,t)$ *can evaluated as a* $\{\wedge, \vee\}$*-circuit of the size* $O(T(n))$ *and can be generated in the time* $O(\log n)$ *using* $O(T^2(n))$ *processors.*
If we assume that M *writes* $1 \in \Gamma$ *to the first cell when it accepts, then the node* $c(1, u, T(n))$ *where* $a_u = 1$ *is the output of the circuit* $r(x)$*.*

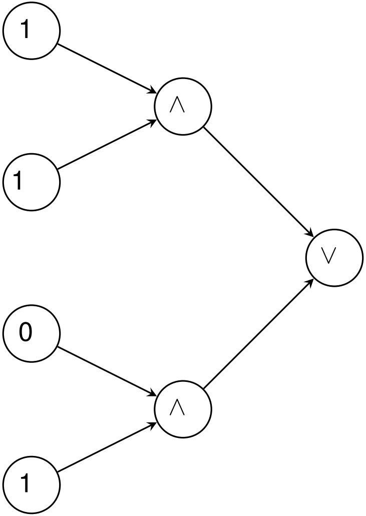
Some Modifications of CVP

- Following modifications of CVP are **P**-complete as well:
  - MCVP – The problem of the value of a monotone circuit, i.e. a circuit that contains no $\neg$-gates.
  - MCVP2 – MCVP where the number of outgoing edges of a node is restricted to at most 2 and the output is a $\vee$-gate.

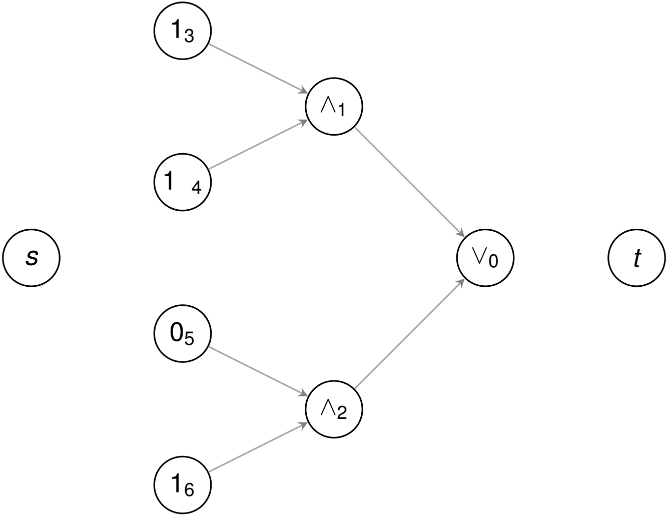




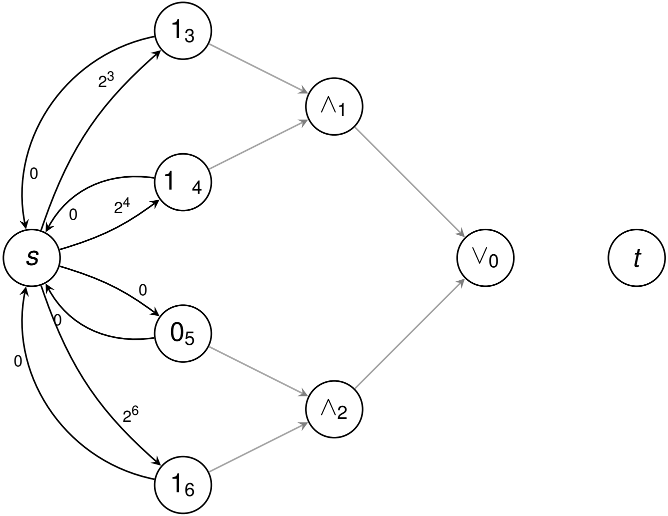
Example of **NC**-reduction from MCVP2 to MAXFLOW



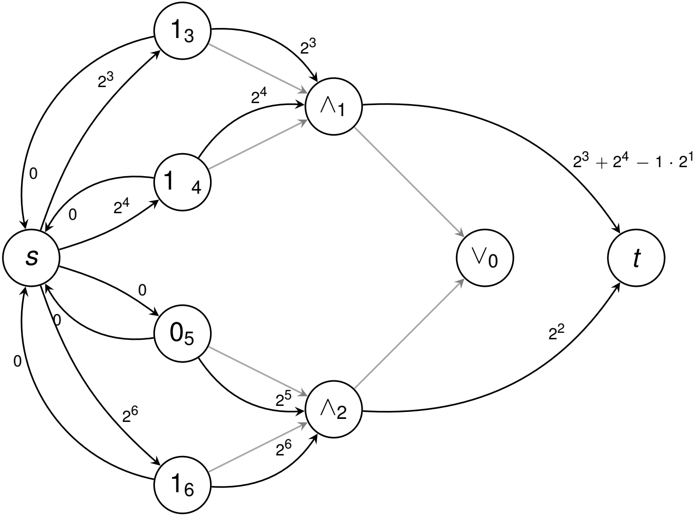
Example of **NC**-reduction from MCVP2 to MAXFLOW



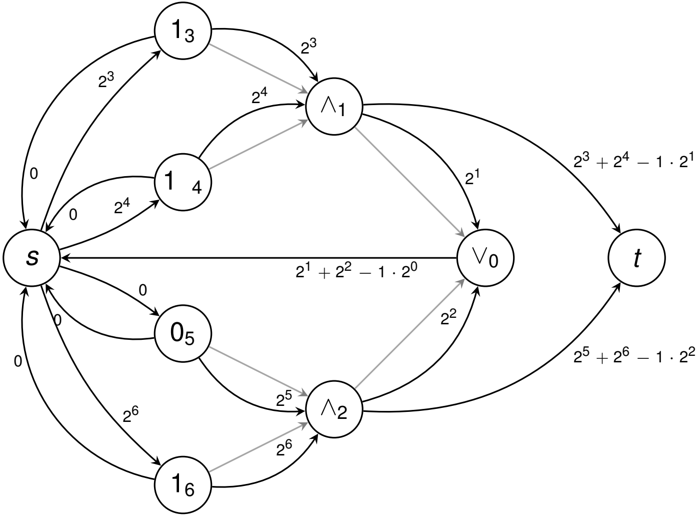
Example of **NC**-reduction from MCVP2 to MAXFLOW



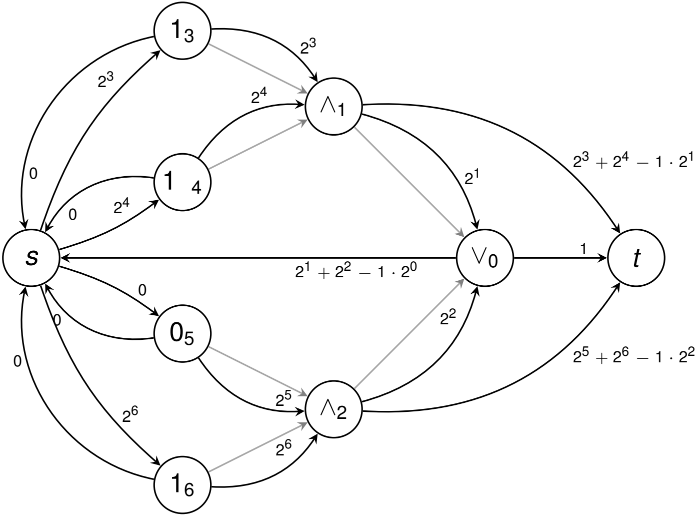
Example of **NC**-reduction from MCVP2 to MAXFLOW



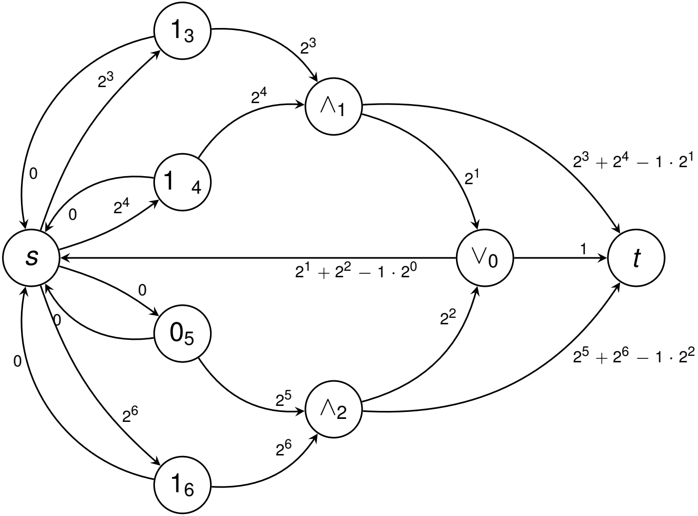
Example of **NC**-reduction from MCVP2 to MAXFLOW



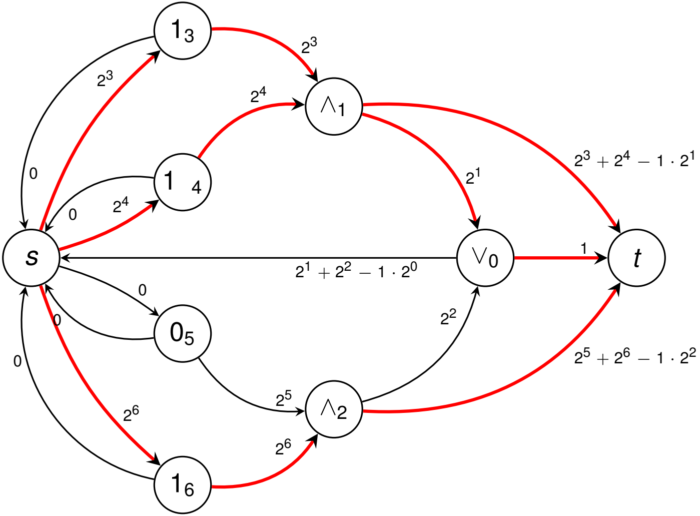
Example of **NC**-reduction from MCVP2 to MAXFLOW



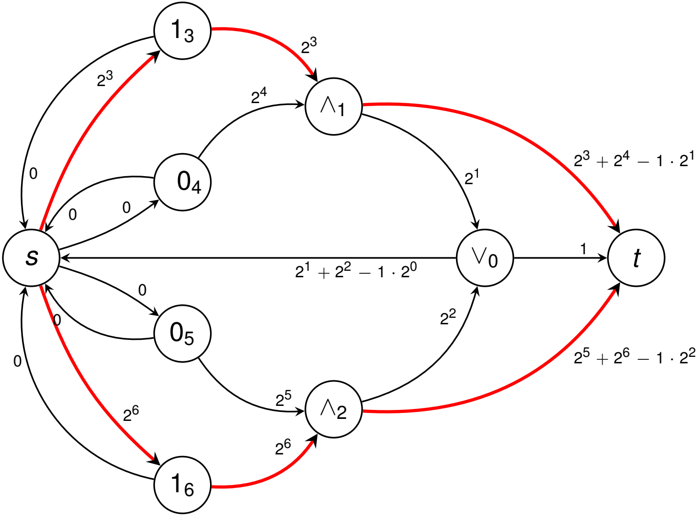
Example of **NC**-reduction from MCVP2 to MAXFLOW



Example of **NC**-reduction from MCVP2 to MAXFLOW



Example of **NC**-reduction from MCVP2 to MAXFLOW



P-completeness of MAXFLOW

Proposition

The MAXFLOW Problem is **P**-*complete.*

Proof

We will show a **NC**-*reduction of the MCVP2 problem to MAXFLOW. Let C be a circuit with the gate nodes $\{g_0, \ldots, g_n\}$ where g_0 is the output gate. We will construct a graph $G = (V, E)$ such that the maximal flow is odd iff $(C, x) \in MCVP2$.*

- $V = \{v_0, \ldots, v_n\} \cup \{s, t\}$ *where vertex v_i corresponds to g_i.*
- *For each input node g_i of C, we create the edge (s, v_i) with the capacity $c(s, v_i) = 2^i$ if $g_i = 1$ and $c(s, v_i) = 0$ otherwise. We include edges (v_i, s) with $c(v_i, s) = 0$ too.*

P-completeness of MAXFLOW

Proof

- *For each* $\wedge$*-gate* $g_i = g_j \wedge g_k$ *we create edges* (v_j, v_i), (v_k, v_i), (v_i, t) *with capacities* $c(v_j, v_i) = 2^j$, $c(v_k, v_i) = 2^k$, *and* $c(v_i, t) = 2^j + 2^k - d \cdot 2^i$ *where* $d \leq 2$ *is the fan-out of* v_i.
- *For each* $\vee$*-gate* $g_i = g_j \vee g_k$ *we create edges* (v_j, v_i), (v_k, v_i), (v_i, s) *with capacities* $c(v_j, v_i) = 2^j$, $c(v_k, v_i) = 2^k$, *and* $c(v_i, s) = 2^j + 2^k - d \cdot 2^i$.
- *Finally, we add the edge* (v_0, t) *with* $c(v_0, t) = 1$.

The above described construction maps each gate of C *to at most three edges of* G*. Such a construction can be done with an* **NC***-algorithm. It remains to show that the reduction is correct.*

P-completeness of MAXFLOW

Proof

We define a function $f : E \to \mathbb{N}$ and show that f is a maximal flow in G.

- *For each input node g_i we set $f(s, v_i) = c(s, v_i)$*
- *For each edge $(v_i, v_j) \in E$ such that $v_i, v_j \notin \{s, t\}$ we set $f(v_i, v_j) = 2^i$ when g_i is a node evaluated to 1, and $f(v_i, v_j) = 0$ otherwise.*
- *For each $\wedge$-gate $g_i = g_j \wedge g_k$ we set $f(v_i, t) = c(v_i, v_t)$ when g_i is a node evaluated to 1, and $f(v_i, t) = f(v_j, v_i) + f(v_k, v_i)$ otherwise.*
- *For each $\vee$-gate $g_i = g_j \vee g_k$ we set $f(v_i, s) = f(v_j, v_i) + f(v_k, v_i) - d \cdot 2^i$ when g_i is a node evaluated to 1, and $f(v_i, s) = 0$ otherwise.*
- *Finally, we set $f(v_0, t) = 1$ if g_0 is evaluated to 1, and $f(v_0, t) = 0$ otherwise.*

P-completeness of MAXFLOW

Proof

Clearly, function f is a flow in G. We will show that f is always a maximal flow.

- *Assume the opposite. Then there must exist an auxiliary path Q for f in G.*
- *The path Q has to start with a "backward" edge, because capacities of all edges (s, v_i) are filled.*
- *Moreover, the path Q must end with a "forward" edge — the vertex t has no outgoing edges.*
- *Therefore, there must be three serially connected vertices $v_j, v_i, v_k \in V$ such that (v_j, v_i) is a backward edge and (v_i, v_k) is a forward edge.*
- *We will show that such a case cannot exist.*

P-completeness of MAXFLOW

Proof

To show this, one has to investigate three possibilities:

1. g_i *is an output node. Then* $v_j = s$ *which implies* $f(v_i, s) = 0$, *contradiction.*
2. g_i *is an* $\wedge$*-gate. Then the output of* g_j *is* g_i *and the fact that* $f(v_i, v_j) > 0$ *implies* $f(v_i, v_k) = c(v_i, v_k)$, *contradiction.*
3. g_i *is an* $\vee$*-gate. Then the flow* f *outgoing from* g_i *is either zero or the capacity of all edges is filled (with the possible exception of the edge* (v_i, s)*). Because* $v_k \neq s$ *and* $f(v_i, v_k) < c(v_i, v_k)$ *then* $f(v_i, v_j) = 0$ *which is in contraction with the fact that* (v_i, v_j) *is a backward edge.*

P-completeness of MAXFLOW

- We eliminated the possibility of the existence of an auxiliary path for f in G.
- The parity of MAXFLOW is derived only from the value of $f(v_0, t)$ (other edges have even values assigned).
- Therefore the output of the circuit C is 1 iff the result of MAXFLOW is odd.

Other **P**-complete Problems

- Word membership for context free grammars.
  - *For a given CFG G and a word w decide whether $w \in L(G)$.*
- Language emptiness for context free grammars.
  - *For a given CFG G decide whether $L(G) = \emptyset$.*
- Language finiteness for context free grammars.
  - *For a given CFG G decide whether $L(G)$ is finite.*