The complexity of non-uniform problems

The analysis made till now was directed towards software solutions:

the algorithm we have derived is thought of working for an
input of arbitrary size.

The situation for hardware is different, since there the (maximum)
size of the input is fixed (e.g. think of a multiplex in the
arithmetic unit of a CPU).

The corresponding computational model is the circuit:

A circuit for inputs of length m:

- has n boolean variables \( x_1, \ldots, x_n \) and the constants 0 and 1
  as inputs

- can be described as a sequence \( G_0, \ldots, G_m \) of gates

  - each gate \( G_i \) has two inputs \( E_{i,1}, E_{i,2} \) (that are among the
    previous gates \( G_0, \ldots, G_{i-1} \) and the inputs)

- A circuit cannot have cycles.

- \( G_i \) applies a binary operation \( \circ_i \) to its two inputs

  - the gate \( G_i \) is realized by a function

    \[
    f_i(x) = f_{i,1}(x) \circ_i f_{i,2}(x)
    \]

    \( f_{i,1}, f_{i,2} \) realize the inputs for \( G_i \)

- overall the whole circuit realizes a function

  \[
  f = (f_0, \ldots, f_m) : \{0,1\}^m \to \{0,1\}^m
  \]

  for some \( m \geq 1 \)

  of each component function \( f_i \) in realized by an input or a gate.
We restrict the attention on symmetric operators only:

AND, OR, XOR, ... plus NOT

= we don't care about the order of the arguments

Example: circuit for addition on three bits:

The circuit is 5 gates of the circuit
(hence NOT gates do not count)

Is a measure of
- the hardware costs
- the sequential computation time

e.g.: adders have size 5

However, circuits are inherently parallel computation devices:

(cf. example) => we consider a different measure

2) The (circuit) depth:

- depth of a gate, is the length of the longest path from
  an input to the gate
- all gates of depth 0 can be evaluated simultaneously at step 1 in time
- circuit depth = maximum gate depth of all gates in the circuit
  e.g.: adder has depth 3
We consider mostly circuits that compute a boolean function:
\[ f : \{0,1\}^n \rightarrow \{0,1\} \]

i.e. circuits with a single output (corresponding to decision problems).

For an asymptotic analysis we cannot consider a single circuit, because we need to consider inputs of increasing length.

\[ \Rightarrow \] we consider a circuit family \( C = \{ C_n \} \) \((C_n) \) stands for \( \{ C_1, C_2, \ldots, C_n, \ldots \} \)

A family \( C = \{ C_n \} \) computes a family \( f = \{ f_n \} \) of boolean functions if each \( f_n \) is computed by \( C_n \).

Relationship between decision problems \( L \subset \{0,1\}^* \) and sequences of boolean functions \( f = \{ f_n \} \) where \( f_n : \{0,1\}^n \rightarrow \{0,1\} \):

- for each language \( L \), there is a sequence of functions
  \[ f^L = (f^L_n) \text{ with } \] \( f^L_n(w) = 1 \implies w \in L \)

- for each family \( f = \{ f_n \} \) of boolean functions, we can define the language \( L_f \) with:
  \[ w \in L_f \iff f_{|2^n|}(w) = 1 \]

Hence, we consider only problems on input alphabet \( \{0,1\} \)

For a sequence \( f = \{ f_n \} \) of boolean functions, we analyze the complexity measures of size and depth:

- \( S_f(n) \) ... minimal size of a circuit computing \( f_n \)
- \( D_f(n) \) ... depth

What is the relationship between time complexity of \( L \) and the circuit size of \( f \)?
For other complexity measures, like time and space, there are languages of arbitrarily high complexity. Instead, the raise complexity of a problem/language is always at most exponential.

**Definition:** \( L \in \text{SIZE}(O(n)) \) if \( L \) is solved by a family \( (c_n) \) of circuits where \( c_n \) has size \( \leq O(n) \).

i.e. \( \sum_{i=1}^{n} s_{c_i}(x) \leq O(n) \)

**Theorem:** For every language \( L \), \( L \in \text{SIZE}(O(2^n)) \)

**Proof:** We need to show that for every \( n \)-output function \( f_n : \{0,1\}^n \to \{0,1\} \), we can construct a circuit computing it of size \( \leq O(2^n) \).

We can construct a circuit for \( f_n \) recursively, by using the identity:

\[
f_n(x_1, \ldots, x_n) = (x_n \land f_n(x_1, \ldots, x_{n-1}, 1)) \lor (x_n \land f_n(x_1, \ldots, x_{n-1}, 0))
\]
We get the following recurrence relation for the size $s(n)$ of the circuit: 

$$s(n) = 3 + 2 \cdot s(n-1)$$

base case: $s(1) = 1$

The solution is $s(n) = 2^2 - 3 = O(2^n)$.

The exponential bound is almost tight.

**Theorem**: There are languages $L$ such that $L \in \text{SIZE}(o(2^{n/m})).

In particular, for every $n > 11$, there is $f: \{0,1\}^n \rightarrow \{0,1\}$

that cannot be computed by a circuit of size $2^{n/4}$. 

**Proof**: is based on a counting argument: there are

1. $2^{2^n}$ functions $f: \{0,1\}^n \rightarrow \{0,1\}$
2. at most $O(n \cdot \log n)$ circuits of size $n$ (assuming $n > 11$).

We show (2) by creating a compact binary encoding of such circuits: Let the gates be numbered: $1, 2, \ldots, n$. For each gate we need to specify:

1) where the two inputs come from: $2 \cdot \log_2 (m+n)$ bits
2) whether they are complemented or not: $2$ bits
3) the type of the gate: $1$ bit

$\Rightarrow$ total number of bits required for the circuit is

$$N = (2 \cdot \log_2 (m+n) + 3) \leq N(2 \cdot \log_2 2n + 3) =$$

$$= N(2 \cdot \log_2 n + 5)$$

We get that the number of circuits of size $s$ is at most:

$$2^{2 \cdot \log_2 n + 5n}$$

which is not sufficient to compute all possible functions if $2 \cdot \log_2 n + 5n < 2^n$.

This is satisfied, if $n \leq \frac{2^n}{4n}$ and $n \geq 11$. \[\square\]
Similarly, we can show that we can give an upper bound on the depth of a boolean function with $n$ inputs.

**Definition:** \( L \in \text{DEPTH}(d(n)) \) if \( L \) is solved by a family \( (C_n) \) of circuits where \( C_n \) has depth \( \leq d(n) \)

i.e., \( D_{C_n} (\bar{a}) \leq d(n) \)

**Theorem:** For every language \( L \), \( L \in \text{DEPTH}(O(n)) \)

**Proof:** Follows directly from the construction for \( f_n \) used to show the \( \text{SIZE}(O(2^n)) \) bound

Note that the \( \text{SIZE} \) and \( \text{DEPTH} \) upper bounds hold also for sequences of circuits/functions \( (C_n) \) or \( (f_n) \) for which \( L \) is not even computable.

There are even non-computable languages \( L \) such that for each \( n \)

- \( L \cap \{0,1\}^n = \{0,1\}^n \) \( \Rightarrow \) \( L \) contains all \( w \) with \( \text{len}(w) = n \)

- \( L \cap \{0,1\}^n = \emptyset \) \( \Rightarrow \) \( L \) contains no \( w \) \( \Rightarrow \) 

Then \( f_n \) is a constant function for each \( n \) and has size 0.

**Difference between SW solutions (algorithms) and HW solutions (circuit families):**

- **Algorithm:**
  - With an algorithm for inputs of arbitrary length \( n \), we have
    - an algorithm for each particular \( n \)
  - an algorithm has a finite description, which is a uniform description of a solution procedure for all input lengths

- **Circuit:**
  - For fixed \( n \), we have a specific circuit family \( C_n \)
  - The depth of \( C_n \) is bounded by \( O(n) \)

- For \( L \in \text{HEIGHT}(O(n)) \), we have a circuit family \( C_n \)
  - For each \( n \), \( C_n \) has depth \( O(n) \)
- Family of circuits: \( C = (C_n) \)
  - we need the entire family to process inputs of arbitrary length
  - for a non-computable language \( L \), the sequence of circuits described above is not even computable
  - \( (C_n) \) only leads to a uniform description of a solution procedure if we have an algorithm to compute \( C_n \) from \( \).

**Definition:** A circuit family \( C = (C_n) \), where \( C_n \) has size \( s(n) \)

is called uniform if \( C_n \) can be computed from \( n \) in \( O(\log s(n)) \) space.

However, it is often simpler to show that \( C_n \) can be computed in polynomial time in \( s(n) \). Then to show the log \( s(n) \)

space bound is in general easy but tedious

Every language \( L \subseteq \{0,1\}^* \) has a non-uniform variant consisting of the sequence \( f^L = (f^L_n) \) of boolean functions.

Non-uniform complexity measures: \( \text{SIZE} \)

\( \text{DEPTH} \)

are useful if we need to solve only instances of a problem for a specific input length:

E.g., 64 bit divider

We are interested in the relationship between uniform and non-uniform complexity measures:

Intuitively, we have \( \text{SIZE} \rightarrow \text{time}, \text{DEPTH} \rightarrow \text{space} \).

By analyzing non-uniform complexity measures, we can understand whether \( L \) is difficult because it requires large circuits or because it is not possible to efficiently compute small circuits.
Simulation of TMs by uniform circuits:

We show now that efficient computations can be simulated by small circuits.

Theorem: If \( L \) is accepted by a (deterministic) TM with running time \( t(n) \), then \( L \in \text{SIZE} \left( O(t(n)^2) \right) \).

Proof: Let \( M \) be a DTM s.t. \( L(M) = L \), and consider inputs of length \( m \). Then the run of \( M \) on such inputs takes time (and space) \( t(m) \).

We show how to simulate such a run by a circuit of size \( O(t(n)^2) \).

Consider \( w = a_1 a_2 \ldots a_n \) (i.e. \( |w| = n \)) and the sequence of states of the run of \( M \) on \( w \) (computation grid).

Each cell can be represented by a pair \( (i, q) \in \Gamma \times (\Delta \cup \{B\}) \)...

The content of each cell in position \( (i, t) \) is affected by the three cells above it

- effects the three cells below it
We can construct a field circuit (depending only on $M$)
but not on $n$ or input $w$)
- with $3 \times k$ inputs and $k$ outputs
- that computes the next configuration cell from the
  3 cells above
  
  whose $k$ outputs go to the three cells below it

The next TD cell circuit depends
only on the transition function
$S$ of $M$. => the size is constant
in $m$

Note: the size of the circuit is exponential in $k$.

The overall circuit simulating $M$ is as follows:

We get that the size of the overall circuit is $O(k(n)^2)$

Note: the depth of the circuit is $O(k(n))$, i.e. $L \in$ DEPTH $O(k(n))$.

Moreover, the family of circuits $C_n = (C_1)$ constructed by
simulating $M$ is uniform.
Note: with a more complicated proof, we can also show that if \( L = \mathcal{L}(M) \) for a TM with running time \( t(m) \), then \( L \in \text{SIZE}(O(t(m) \cdot \log t(m))) \).

**Corollary:** \( P \leq \text{SIZE}(n^{O(1)}) \)

*Note:* \( P \neq \text{SIZE}(n^{O(1)}) \), since \( \text{SIZE}(O(1)) \) contains undecidable languages.

We can also relate small space bounds with small DEPTH.

**Theorem:** If \( L \) is accepted by a DTM \( M \) with space bound \( s(m) \), then \( L \in \text{DEPTH}(O(s^2(m))) \), where \( s(m) = \max\{s(m), \lfloor \log m \rfloor \} \).

Moreover, the family of circuits simulating \( M \) is uniform.

**Proof idea:** based on simulating the configuration graph of \( M \).

Note that the above results relating TIME \( \leq \text{SIZE} \)

TIME \( \leq \text{DEPTH} \)

are mostly of interest for languages below \( \text{EXP} \) and \( \text{PSPACE} \)

respectively, since for every language \( L \), \( L \in \text{SIZE}(O(2^{\log m})) \) and \( L \in \text{DEPTH}(O(n)) \).

**Simulation of circuits by non-uniform TMs**

Circuit families \( C = (C_n) \) form a non-uniform computation model because we do not care how to obtain circuit \( C_n \) for input of length \( n \).

In a TM to simulate a circuit family, it must also have access to some information that depends only on \( n = |x_n| \), but not on \( x \) itself.
Definition: A non-uniform TM (NUTM) is a TM with two read-only input tapes:
- one input tape contains the input instance \( w \)
- a second input tape contains some helping (or advice) information \( h(1^{|w|}) \) that is identical for all inputs of the same length.

So the next, the non-uniform TM is as an ordinary TM.

Note: due to the advice \( h(1^{|w|}) \) on the second input tape, a NUTM with space bound \( s(n) \) has a number of configurations that is larger by a factor of \( h(n) \) than the number for a normal TM.

We can extend the results on simulating TMs by circuits also to NUTMs, simulated by non-uniform circuits:
\( h(n) \) represents for \( C_n \) a constant portion of the input.

We now discuss simulation results in the other direction:
- small circuit families can be simulated by fast NUTMs.
- shallow
  - small space requirements.

Notation: for circuit families \( C = \{ C_n \} \)
\( \delta_C(n) \) denotes the size of \( C_n \) and \( \delta_C(n) = \max \{ \delta_C(n), m \} \)
\( \delta_C(n) \) - depth,
\( \delta_C(n) \) - max \{ \delta_C(n), \lceil \log m \rceil \}
Theorem: \( C = \{ C_n \} \) can be simulated by a NUTM with two work tapes (in addition to the two input tapes) in time \( O(\hat{S}_2(n)^2) \) and space \( O(\hat{S}_2(n)) \).

Proof:

We let the advice \( h(n) \) be a description of the circuit \( C_n \):
- consists of a list of all gates
  - for each gate in the list:
    - type of gate
      - for each of the two inputs:
        - the type (i.e., constant, input bit of \( C_n \), gate)
        - whether it is negated or not
        - its number (gate or input bit, depending on type)

\( h(n) \) has length \( O(\hat{S}_2(n) \log \hat{S}_2(n)) \).

The TM evaluates the gates in their natural order.

It uses two work tapes:
- tape 1 stores the values of already evaluated gates
- tape 2 stores a counter used to locate values on
  tape 1 on the input tape

To evaluate a gate, the TM has to
- retrieve the operator of the gate and whether the input should be negated or not: this is on the help tape
- retrieve the values of the inputs:
  1) for a constant, it is on the help tape
  2) for an input, it is on the input tape
  3) for a gate, it is on work tape 1

In cases (2) and (3), the TM takes the index from
the help tape, puts it on tape 2, and uses it to count to the right position on the input tape or work tape 1.
The cost of retrieving input \( i \) on gate \( i \) is
\[
O(1) = O(\hat{\lambda}(m))
\]
\[
(\max \{ \hat{\lambda}(m), m \})
\]

To process \( \lambda(m) \) gates, each with two inputs, the total time is
\[
O(\lambda(m) \cdot \hat{\lambda}(m)) = O(\lambda(m)^2).
\]

Space used by the TM:
- Work tape 1: not more than \( \hat{\lambda}(m) \) bits
- Work tape 2: \( \log \hat{\lambda}(m) \) bits

To obtain a NUTM with one work tape one can use the simulation of a 2-tape TM by a 1-tape TM
\[
\Rightarrow O(\hat{\lambda}(m)^2) \text{ time bound}
\]

We could also give a direct construction of a 1-tape NUTM with time bound \( O(\hat{\lambda}(m)^3 \cdot \log \hat{\lambda}(m)) \).

If we start from a uniform circuit family \( C = (c_n) \), we can
1) Compute \( c_n \) from an input of length \( n \)
2) Apply the construction above

We can also prove a tighter space bound for the simulation of \( C = (c_n) \) by a 1-TM.

**Theorem:** \( C = (c_n) \) can be simulated by a NUTM in space \( O(\hat{\lambda}(m)) \).

**Proof idea (details omitted):**
\[
\max \{ \hat{\lambda}(m), \log m \}
\]

It is based on unfolding the circuit from a DAG to a tree.
(Which does not increase its depth).

The advice for an input of length \( n \) is such a tree of depth \( n \) for \( c_n \).

If \( C = (c_n) \) is uniform, then the info on a gate can be computed in space \( O(\log 2^{\hat{\lambda}(m)}) = O(\hat{\lambda}(m)) \). So a uniform TM can get by with space \( O(\hat{\lambda}(m)) \).
Binary Decision Diagrams (BDDs):

BDDs are a non-uniform model of computation whose size characterizes the space used by a non-uniform TM asymptotically exactly.

BDDs are used not only in complexity theory, but also as state structures for boolean functions (see, e.g., symbolic model checking in formal verification).

Note: BDDs are also called branching programs.

Definition: A BDD for $n$ boolean variables $x_1, \ldots, x_n$ is a binary DAG (i.e., each internal node has two outgoing edges, one labeled 0 and one labeled 1)
- each internal node is labeled with a variable
- each leaf (or output node) is labeled with a value in $\{0, 1\}$

Each node $v$ in a BDD $G$ for variables $x_1, \ldots, x_n$ realizes a boolean function $f_G(x_1, \ldots, x_n)$ whose value can be computed as follows: To compute $f_G(c_1, \ldots, c_n)$
- we start from node $v$ and follow the edges of $G$ until we reach an output node
- whenever we are at a node labeled with variable $x_i$, we follow the edge labeled with $c_i$ (i.e., the value 0 or 1 of $x_i$)
- the value of $f_G(c_1, \ldots, c_n)$ is given by the label 0 or 1 of the reached output node.
Example BDD with two nodes realizing the sum and carry bits for the sum of three bits:

Complexity measure for a BDD:
- **length** = length of the longest path from a node to a leaf.
  This is a measure of the worst-case time required to evaluate the function.
- **size** = number of nodes
  Provides a measure of the space required to evaluate the function.

**Definition:** The branching program complexity $BP(f)$ of a boolean function $f$ is the minimal size of a BDD (or branching program) computing $f$. 
Let us analyze why there is a connection between the size of a BDD and the space required by non-uniform TMs.

1) From BDDs to NUTMs:
   To evaluate a function $f$ represented by a BDD, it is sufficient to remember the currently needed node.

2) From NUTMs to BDDs:
   A BDD can directly simulate the configuration graph of a space-bounded TM.

To formalize this, let $BP^*(f_m) = \max \{ BP(f_n), m \}$ and $k(m) = \max \{ s(m), \lceil \log s \rceil \}.

**Theorem:** The language $L_f$ corresponding to $f = (f_m)$ can be solved by a NUTM in space $O(\log BP^*(f_m))$.

**Proof:**
As advice on inputs of length $m$ we use a description of a BDD $G_m$ of minimal size for $f_m$:
- consists of a list of the modes of $G_m$ (starting with the mode $f(f_m)$)
- for each mode, the list element consists of:
  - the type (internal node or output node)
  - the number of the node
  - the internal information:
    - for an output node: the output value
    - for an inner node:
      - index of the variable
      - index of the 0-successor node
      - index of the 1-successor node
Hence, the description of each vertex has length $O(\log BP^*(f_n))$

The TM uses a work tape containing:
- the current node information
- a counter used to locate the next node and the input

These take $O(\log BP^*(f_n))$ space.

The TM repeatedly processes nodes, starting with node 1 (the root):
- if it copies the node information to the work tape;
- if it is an output node, the computation terminates;
- otherwise, let the current information be $(i, j_0, f_0)$:
  - the TM a) locates the value of $i$ on the input tape;
  - b) locates node $j_0$ on the work tape;
  - and continue with step 4.

\[\square\]

**Theorem:** An $O(n)$-space bounded TM can be simulated by a BDD of size $2 O(8(n))$

**Proof:**

The BDD has a node for each of the configurations that is reachable from the start configuration.

Since the TM has space-bound $O(n)$, the number of such configurations is bounded by $2 O(8(n))$ for an input of length $n$. 
The BDD is then constructed as follows:

- An accepting configuration is in a 1-output mode.
- A rejecting configuration is in a 0-output mode.
- An internal mode corresponding to configuration $K_i$ is labeled with the variable $x_i$ that is read from the input tape in configuration $K_i$.
- Has as $v$-child (for $x \in \{0, 1\}$) the configuration reached from $K_i$ when $x_i$ has value $v$.

Since we only consider TMs that always halt, the graph is acyclic, and hence in a BDD.

The boolean function representing the acceptance behavior of the TM on inputs of length $n$ is realized by the mode corresponding to the initial configuration.

What if the TM to simulate is a NUTM with help of length $h(m)$?

The number of configurations (and hence the size of the simulating BDD) grows by a factor of $h(m)$.

Since $h(m) \leq 2 \lceil \log h(m) \rceil$ is the number of different possible positions of the head on the read-only advice tape.

Hence, one adds $\lceil \log h(m) \rceil$ to the space used by a NUTM.

We would also define $\Phi(n) = \max \{ \rho(n), \log n, \lceil \log h(m) \rceil \}$ and state:

**Theorem:** An $\phi(n)$-space bounded NUTM can be simulated by a BDD of size $20(\Phi(n))$. 

We can summarize these results for the "normal" case where
\[ N(n) \geq \log n \]
\[ BP(f_n) \geq n \]
\[ h(n) \text{ is polynomially bounded} \]
by saying that space and the logarithm of the BDD size have the same order of magnitude.

Researchers are trying to exploit these results to address the open problems related to the relationships between
LOGSPACE, on one side, and
NP, P, NL, LOGSPACE on the other side.

In other words, are the following (obvious) inclusions strict?

\[
\text{LOGSPACE \subset \begin{cases} \text{NP} \\ \text{P} \\ \text{NL} \end{cases} \text{LOGSPACE}}
\]

Strictness would follow from proving a superpolynomial lower bound for the BDD size of the function \( f^L = (f^1)^L \) for some language \( L \in \text{NP/P/L}/\text{LOGSPACE} \).