The complexity of non-uniform problems

The analysis made till now was directed towards software solutions: the algorithm we have devised or thought of works 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 multiplier in the arithmetic unit of a CPU).

The corresponding computational model is the circuit.

A circuit for inputs of length n:

- has n boolean variables $x_1, \dots, x_n$ and the constants 0 and 1 as inputs
- can be described as a sequence $G_1, \dots, G_s$ of gates
  - each gate G_i has two inputs $E_{i,1}, E_{i,2}$ (that are among the previous gates $G_1, \dots, G_{i-1}$ and the inputs)
    - $\Rightarrow$ A circuit cannot have cycles.
  - G_i applies a binary operation op_i to its two inputs
    the gate G_i is realized by a function:

$$g_i(\vec{a}) = g_{i,1}(\vec{a}) \; op_i \; g_{i,2}(\vec{a})$$

($g_{i,1}, g_{i,2}$ realize the inputs for G_i)

- overall, the whole circuit realizes a function

$$f = (f_1, \dots, f_m) : \{0,1\}^n \to \{0,1\}^m \quad \text{, for some } m \geq 1$$

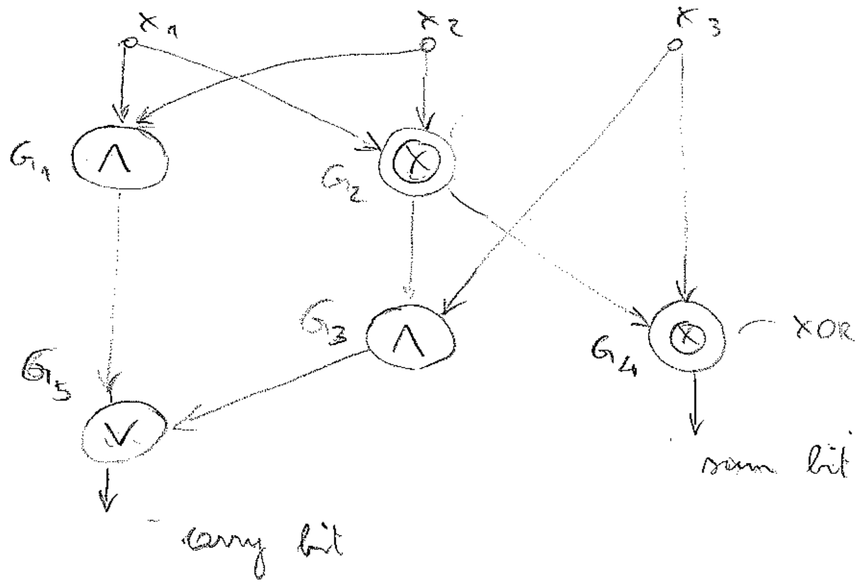
if each component function f_i is realized by an input or a gate.

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:



2 measures for the evaluation of the efficiency of circuits:

1) The (circuit) size = number of binary gates of the circuit (hence NOT gates do not count)

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

e.g. adder has 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 d can be evaluated simultaneously at step d in time
- circuit depth = maximal gate depth of all gates in the circuit

e.g. adder has depth 3

15/1/2008

We consider mostly circuits that compute a boolean function:

$$f : \{0,1\}^n \rightarrow \{0,1\}$$

i.e. circuits with a single output (correspond 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 on $\{0,1\}^*$ and sequences of boolean functions $f = (f_n)$ with $f_n : \{0,1\}^n \rightarrow \{0,1\}$:

- for each language L, there is a sequence of functions
  $f^L = (f_n^L)$ with $f_n(w) = 1 \Leftrightarrow w \in L$

- for each family $f = (f_n)$ of boolean functions, we can define the language L_f with: $w \in L_f \Leftrightarrow f_{|w|}(w) = 1$

Hence, we consider only problems over 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_L?

For other complexity measures, like time and space, there are languages of arbitrarily high complexity.

Instead, the size complexity of a problem / language is always at most exponential.

<u>Definition:</u> $L \in SIZE(s(n))$ if L is solved by a family (C_n) of circuits where C_i has size $\leq s(i)$.

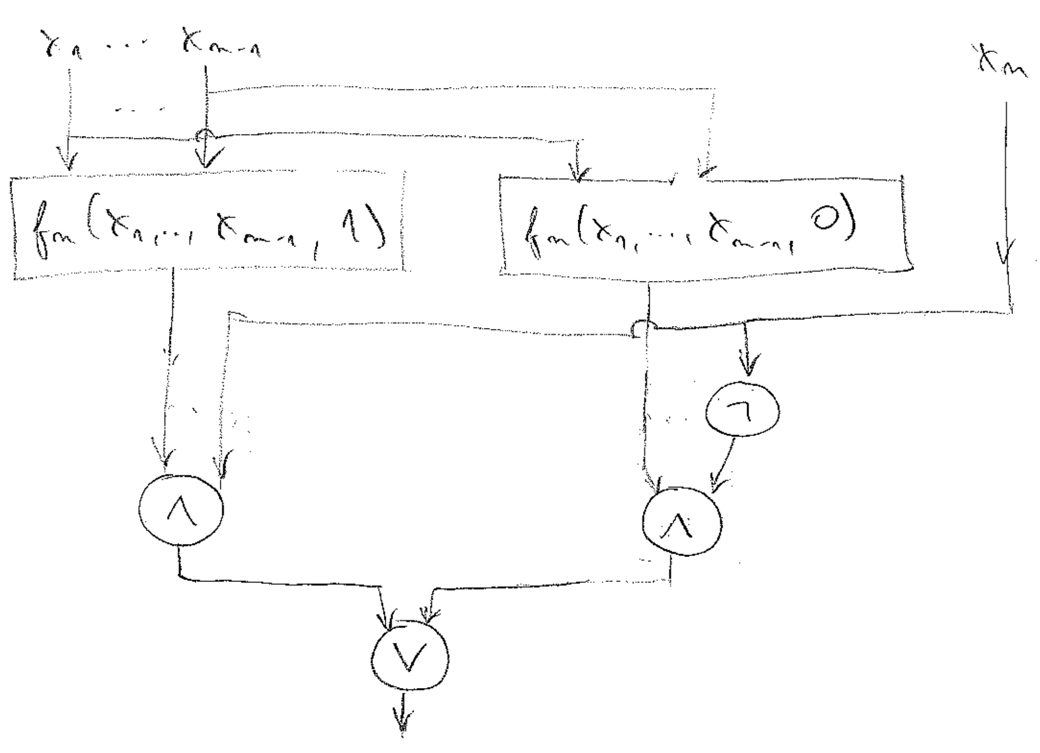
i.e. $S_{f_L}(i) \leq s(i)$

<u>Theorem:</u> For every language L, $L \in SIZE(O(2^n))$

<u>Proof:</u> We need to show that for every 1-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:

$$f_n(x_1, \dots, x_n) = (x_n \wedge f_n(x_1, \dots, x_{n-1}, 1)) \vee (\bar{x}_n \wedge f_n(x_1, \dots, x_{n-1}, 0))$$



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 $s(n) = 2 \cdot 2^n - 3 = O(2^n)$. □

The exponential bound is almost tight

<u>Theorem</u>: There are languages L such that $L \notin SIZE(o(2^n/n))$.
In particular, for every $n \geq 11$, there is $f: \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/4n$.

<u>Proof</u>: is based on a counting argument: there are

(1) 2^{2^n} functions $f: \{0,1\}^n \to \{0,1\}$

(2) at most $2^{O(s \cdot \log s)}$ circuits of size s (assuming $s \geq n$).

We show (2) by creating a compact binary encoding of such circuits: Let the gates be numbered: $1, 2, \ldots, s$.
For each gate we need to specify:
1) where the two inputs come from: $2 \cdot \log_2(n+s)$ bits
2) whether they are complemented or not: 2 bits
3) the type of the gate: 1 bit

⇒ total number of bits required for the circuit is

$$s \cdot (2 \cdot \log(n+s) + 3) \leq s \cdot (2 \cdot \log_2 2s + 3) =$$
$$= s \cdot (2 \log_2 s + 5)$$

We get that the number of circuits of size s is at most: $2^{2 \cdot s \log_2 s + 5s}$, which is not sufficient to compute all possible functions if $2s \cdot \log_2 s + 5s < 2^n$.

This is satisfied, if $s \leq \frac{2^n}{4n}$ and $n \geq 11$

□

Similarly, we can show that we can give an upper bound on the depth of a boolean function with n inputs.

Definition: $L \in DEPTH(d(n))$ if L is solved by a family (C_n) of circuits where C_n has depth $\leq d(n)$
i.e., $D_{f_L}(n) \leq d(n)$

Theorem: For every language L, $L \in DEPTH(O(n))$

Proof: Follows directly from the construction for f_n used to show the $SIZE(O(2^n))$ bound

□

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

There are even non-computable languages L such that for each n
either $L \cap \{0,1\}^n = \{0,1\}^n$ ... L contains all w with $|w| = n$
or $L \cap \{0,1\}^n = \emptyset$ ... L contains no w — " —
Then f_n^L 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 also 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

- 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 n.

<u>Definition</u>: 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 $f^L = (f_n^L)$ of boolean functions.
Non-uniform complexity measures: SIZE
DEPTH
are useful, if we need to solve only instances of a problem for a specific input length.

e.g. a 64 bit divider

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

intuitively, we have SIZE $\leftrightarrow$ time, DEPTH $\leftrightarrow$ 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:

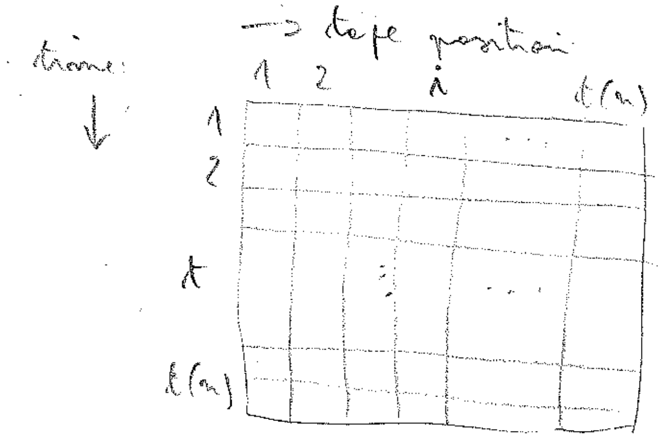
We show now that efficient computations can be simulated by small circuits

Theorem: If L is accepted by a (deterministic) TM with running time $t(n)$, then $L \in SIZE(O(t(n)^2))$

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

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

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



We have $t(n)$ cells

Each cell can be represented by a pair

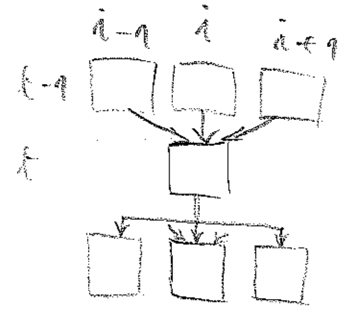
$(a, q) \in \Gamma \times (Q \cup \{\square\})$ ... k bits (where k is a constant not depending on n)

q: state of M (when the head is on the cell) or $\square$ (when the head is somewhere else)

a: tape symbol

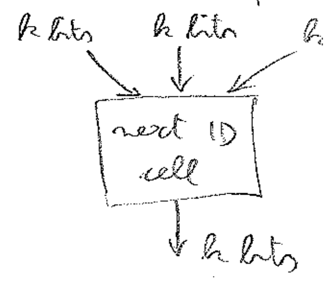
The content of each cell in position (i, t)

- is affected by the three cells above it
- affects the three cells below it



We can construct a fixed circuit (depending only on M, but not on n or input w)

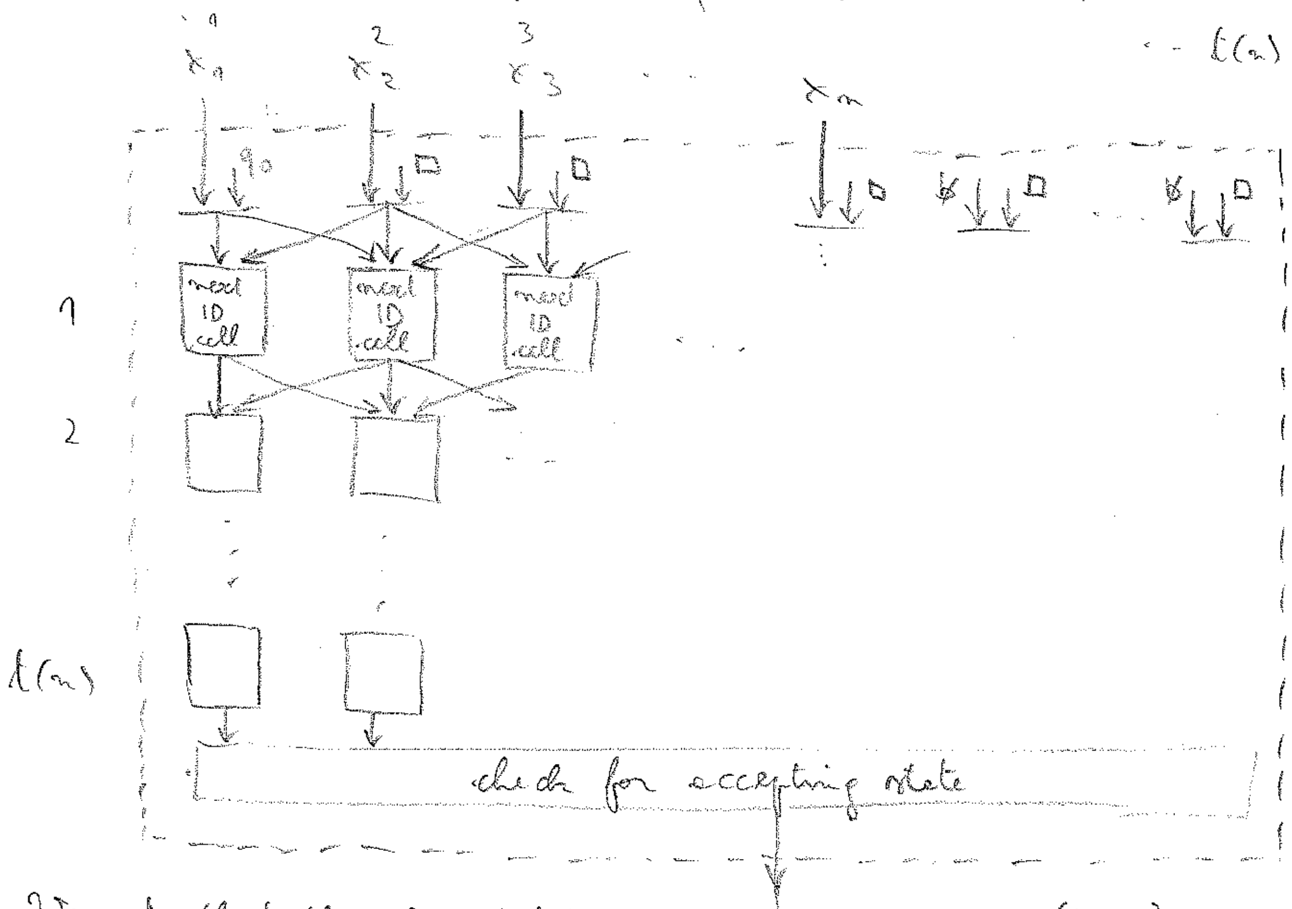
- with 3×k inputs and k outputs
- that computes the next configuration cell from the 3 cells above
- whose k outputs go to the three cells below it



The next ID cell circuit depends only on the transition function δ of M. ⇒ the size is constant in n

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 $O(t(n)^2)$ □

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

Moreover, the family of circuits $C^L = (C^L_n)$ 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 $t(n)$, then $L \in SIZE(O(t(n) \cdot \log t(n)))$

<u>Corollary</u>: $P \subseteq SIZE(n^{O(1)})$

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

We can also relate small space bounds with small DEPTH.

<u>Theorem</u>: If L is accepted by a DTM M with space bound $s(n)$, then $L \in DEPTH(O(\hat{s}(n)^2))$, where $\hat{s}(n) = \max\{s(n), \lceil \log n \rceil\}$.

Moreover, the family of circuits simulating M is uniform.

<u>Proof idea</u>: based on simulating the configuration graph of M.

Note that the above results relating time $\Leftrightarrow$ SIZE
space $\Leftrightarrow$ DEPTH
are mostly of interest for languages below EXPTIME and PSPACE respectively, since for every language L, $L \in SIZE(O(2^n))$ and $L \in DEPTH(O(n))$.

<u>Simulation of circuits by non-uniform TMs</u>

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.

For a TM to simulate a circuit family, it must also have access to some information that depends only on $n = |w|$, but not on w 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 $h(|w|)$ that is identical for all inputs of the same length

For the rest, the non-uniform TM is as an ordinary TM.

Note: due to the advice $h(|w|)$ 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.

(↑ $h(n)$: number of head positions on the advice tape)

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 — " — by NUTM with small space requirements

Notation: for circuit families $C = (C_n)$

$s_C(n)$ denotes the size of C_n and $\hat{s}_C(n) = \max\{s_C(n), n\}$

$d_C(n)$ — " — depth — " — $\hat{d}_C(n) = \max\{d_C(n), \lceil \log n \rceil\}$

Theorem: $C = (C_n)$ can be simulated by a NUTM with two work tapes (in addition to the two input tapes) in time $O(\hat{s}_C(n)^2)$ and space $O(\hat{s}_C(n))$

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 $O(s_C(n) \cdot \log \hat{s}_C(n))$

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 or 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
$O(i) = O(\hat{s}_C(n))$
$\max\{s_C(n), n\}$

To process $s_C(n)$ gates, each with two inputs, the total time is $O(s_C(n) \cdot \hat{s}_C(n)) = O(\hat{s}_C(n)^2)$.

Space used by the TM:
- work tape 1: not more than $s_C(n)$ bits
- work tape 2: $\log_2 \hat{s}_C(n)$ bits

□

17/1/2008

To obtain a NUTM with one work tape, we can use the simulation of a 2-tape TM by a 1-tape TM
$\Longrightarrow O(\hat{s}_C(n)^4)$ time bound

We could also give a direct construction of a 1-tape NUTM with time bound $O(\hat{s}_C(n)^2 \cdot \log \hat{s}_C(n))$.

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 NUTM.

Theorem: $C = (C_n)$ can be simulated by a NUTM in space $O(\hat{d}_C(n))$.
$\max\{d_C(n), \lceil \log n \rceil\}$

Proof idea (details omitted):
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 compute in space $O(\log 2^{d_C(n)}) = O(d(n))$. So a uniform TM can get by with space $O(\hat{d}_C(n))$.

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.

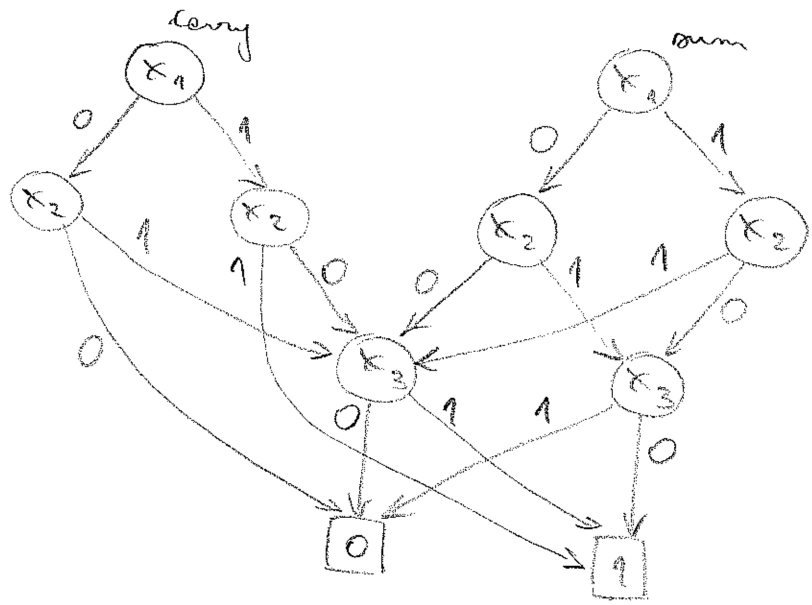
BDDs are used not only in complexity theory, but also as data structures for boolean functions (see, e.g., symbolic model checking in formal verification).

Note: BDDs are also called branching programs.

Definition: A BDD for n boolean variables $x_1, \dots, x_n$ 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 $c \in \{0,1\}$.

Each node v in a BDD G for variables $x_1, \dots, x_n$ realizes a boolean function $f_v(x_1, \dots, x_n)$, whose value can be computed as follows: To compute $f_v(a_1, \dots, a_n)$
- 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 a_i (i.e. the value 0 or 1 of x_i)
- the value of $f_v(a_1, \dots, a_n)$ 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. 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 analyse 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 reached node

2) From NUTMs to BDDs:

A BDD can directly simulate the configuration graph of a space-bounded TM

To formalize this, let $BP^*(f_n) = \max\{BP(f_n), n\}$

and $\hat{s}(n) = \max\{s(n), \lceil \log n \rceil\}$

<u>Theorem</u>: The language L_f corresponding to $f = (f_n)$ can be solved by a NUTM in space $O(\log BP^*(f_n))$

<u>Proof</u>:

As advice on inputs of length n we use a description of a BDD G_n of minimal size for f_n:

- consists of a list of the nodes of G_n (starting with the node for f_n)
- for each node, 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
      - " " 1- " "

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 one for f_n)
1) it copies the node information to the work tape
2) if it is an output node, the computation terminates
otherwise let the inner information be (i, j_0, j_1):
the TM a) locates the value v of x_i on the input tape,
b) locates node j_v on the help tape
and continue with step 1.

□

<u>Theorem</u>: An $s(n)$-space bounded TM can be simulated by a BDD of size $2^{O(\hat{s}(n))}$

<u>Proof</u>:

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

Since the TM has space-bound $s(n)$, the number of such configurations is bounded by $2^{O(\hat{s}(n))}$ for an input of length n.

The BDD is then constructed as follows:

- an accepting configuration is a 1-output node
- a rejecting — " — 0- — " —
- an internal node corresponding to configuration K
  - is labeled with the variable x_i that is read from the input tape in configuration K
  - has as v-child (for $v \in \{0,1\}$) the configuration reached from K when x_i has value v.

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

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

□

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

The number of configurations (and hence the size of the simulating BDD) grows by a factor of h(n).
(since $h(n) \leq 2^{\lceil \log h(n) \rceil}$ is the number of different positions of the head on the read-only advice tape).

Hence, one adds $\lceil \log h(n) \rceil$ to the space used by a NUTM.
We could also define $\hat{\hat{s}}(n) = \max\{s(n), \lceil \log n \rceil, \lceil \log h(n) \rceil\}$ and state:

<u>Theorem</u>: An s(n)-space bounded NUTM can be simulated by a BDD of size $2^{O(\hat{\hat{s}}(n))}$

We can summarize these results for the "normal" case where

$S(n) \geq \log n$

$BP(f_n) \geq n$

$h(n)$ 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, NLOGSPACE on the other side.

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

$$\text{LOGSPACE} \subseteq \begin{cases} \text{NP} \\ \text{P} \\ \text{NLOGSPACE} \end{cases}$$

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