Computational Complexity Theory

Lecture 4: Cook-Levin theorem (contd.); More NP-complete problems

Department of Computer Science,
Indian Institute of Science

Recap: A natural NP-complete problem

- Definition. A Boolean formula is in *Conjunctive Normal Form* (*CNF*) if it is an AND of OR of literals.

  e.g. $\phi = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_2)$

- Definition. Let SAT be the language consisting of all *satisfiable CNF formulae*.

- Theorem. *(Cook 1971, Levin 1973)* SAT is NP-complete.

  Easy to see that SAT is in NP.

  Need to show that SAT is NP-hard.

Recap: Cook-Levin theorem: Proof

- Main idea: Computation is ***local***; i.e., every step of computation *looks at* and *changes* only constantly many bits; and this step can be implemented by a small CNF formula.

- Let $L \in NP$. We intend to come up with a polynomial-time computable function $f: x \longmapsto \phi_x$ s.t.,
  - $x \in L \iff \phi_x \in SAT$

  - Notation: $|\phi_x| :=$ size of ϕ_x

    $=$ number of $\vee$ or $\wedge$ in ϕ_x

Recap: Cook-Levin theorem: Proof

- Language L has a poly-time verifier M such that

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

- Idea: For any fixed x, we can capture the computation of M(x, ..) by a CNF ϕ_x such that

$$\exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1 \iff \phi_x \text{ is satisfiable}$$

- For any fixed x, M(x, ..) is a deterministic TM that takes u as input and runs in time polynomial in |u|.

Recap: Cook-Levin theorem: Proof

- Main Theorem. Let N be a deterministic TM that runs in time T(n) on every input u of length n, and outputs 0/1. Then, (think of N = M(x, ..) for a fixed x.)
  1. There's a CNF ϕ(u, "*auxiliary variables*") of size *poly*(T(n)) such that for every u, ϕ(u, "*auxiliary variables*") is satisfiable as a function of the "*auxiliary variables*" if and only if N(u) = 1.
  2. ϕ is computable in time *poly(T(n))* from N, T & n.

- ϕ(u, "*auxiliary variables*") is satisfiable as a function of **all** the variables if and only if ∃u s.t N(u) = 1.

Recap: Main theorem: Proof

- **Step 1.** Let N be a deterministic TM that runs in time T(n) on every input u of length n, and outputs 0/1. Then,
    1. There's a Boolean circuit ψ of size *poly*(T(n)) such that $\boxed{\psi(u) = 1 \text{ if and only if } N(u) = 1.}$
    2. ψ is computable in time *poly(T(n))* from N, T & n.

The key insight: ψ "encodes" N.

- Step 2. "Convert" circuit ψ to a CNF ϕ efficiently by introducing <u>auxiliary variables</u>.

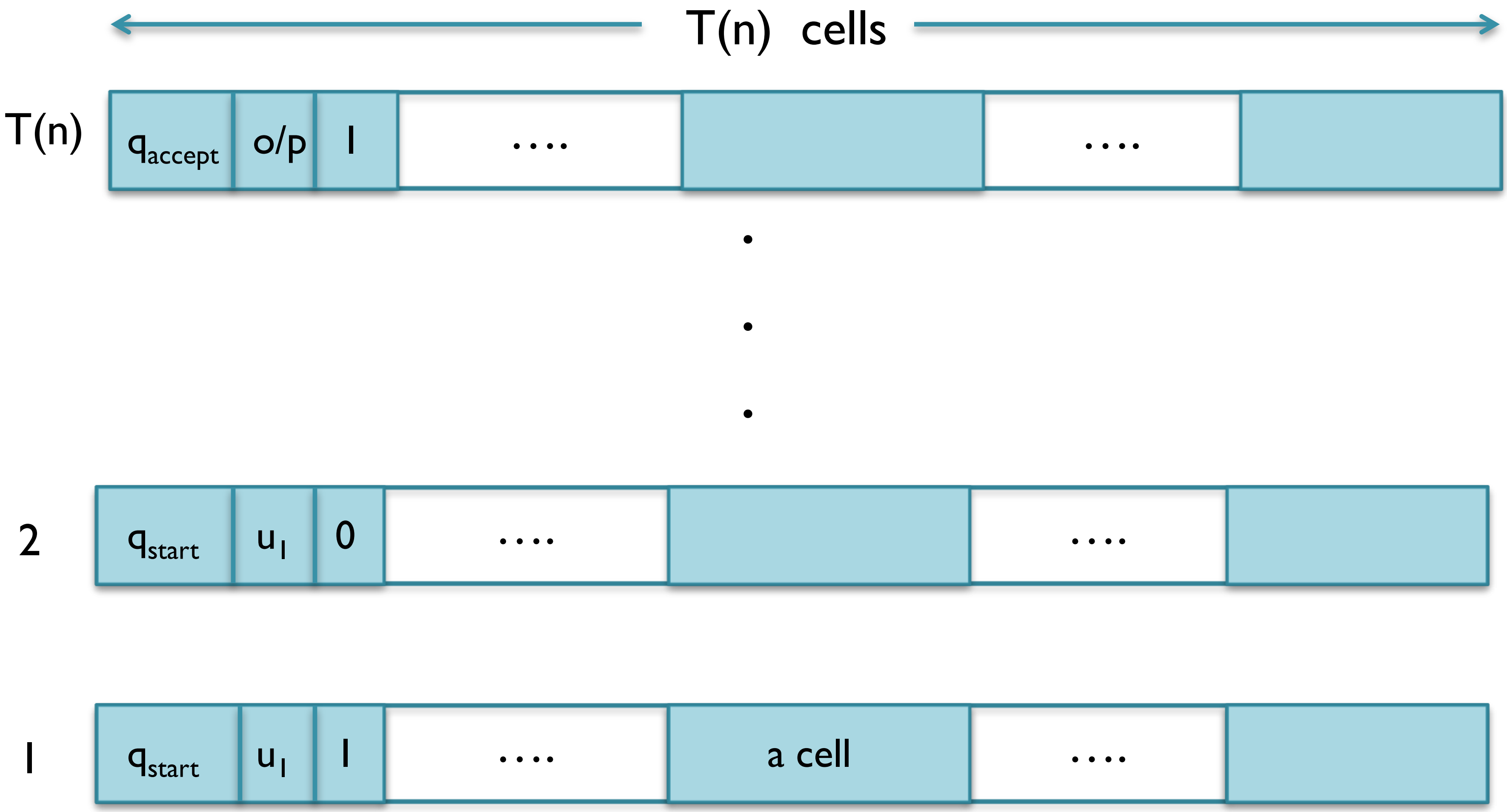
Recap: Main theorem: Step 1

- Assume (w.l.o.g) that N has a single tape and it writes its output on the first cell at the end of computation.

- A step of computation of N consists of
  - Changing the content of the current cell
  - Changing state
  - Changing head position

- Think of a ‘<u>compound</u>’ tape: Every cell stores the current state, a bit content and head indicator.

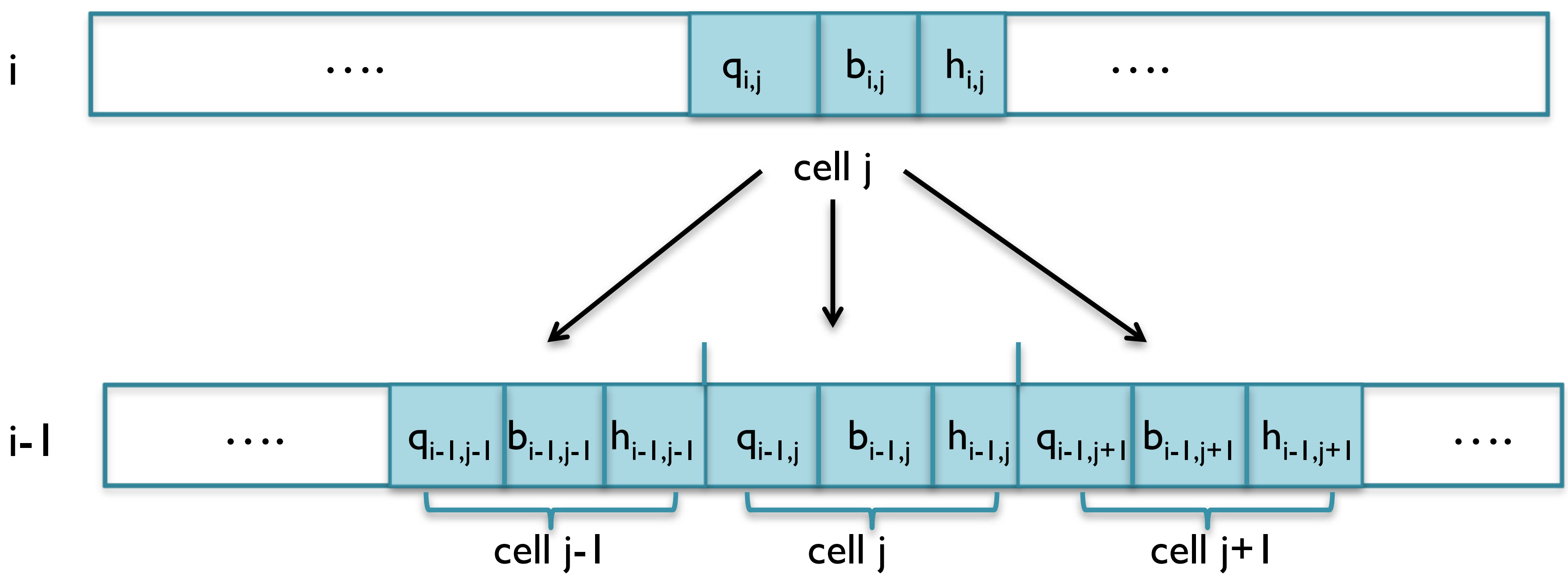
Recap: Main theorem: Step 1



A compound tape

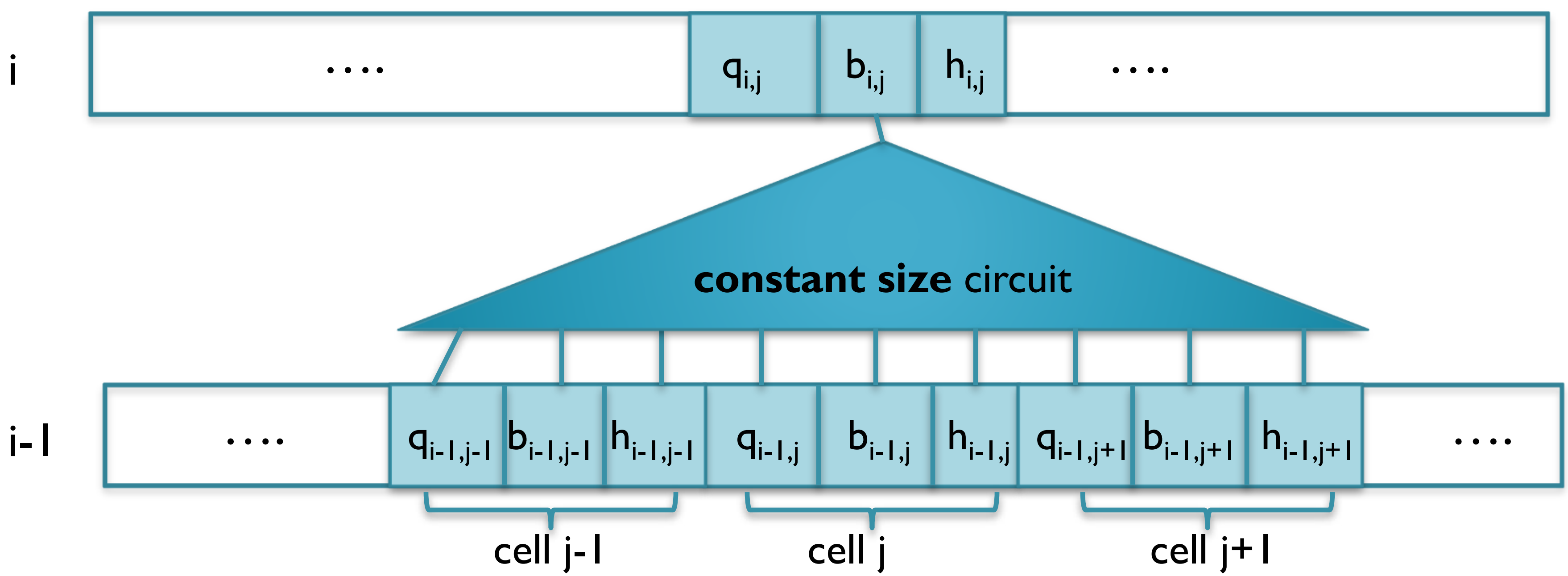
Recap: Main theorem: Step 1

- **Locality of computation:** The bits in $h_{i,j}$, $b_{i,j}$ and $q_{i,j}$ depend ***only on*** the bits in
  - $h_{i-1,j-1}$, $b_{i-1,j-1}$, $q_{i-1,j-1}$,
  - $h_{i-1,j}$, $b_{i-1,j}$, $q_{i-1,j}$,
  - $h_{i-1,j+1}$, $b_{i-1,j+1}$, $q_{i-1,j+1}$

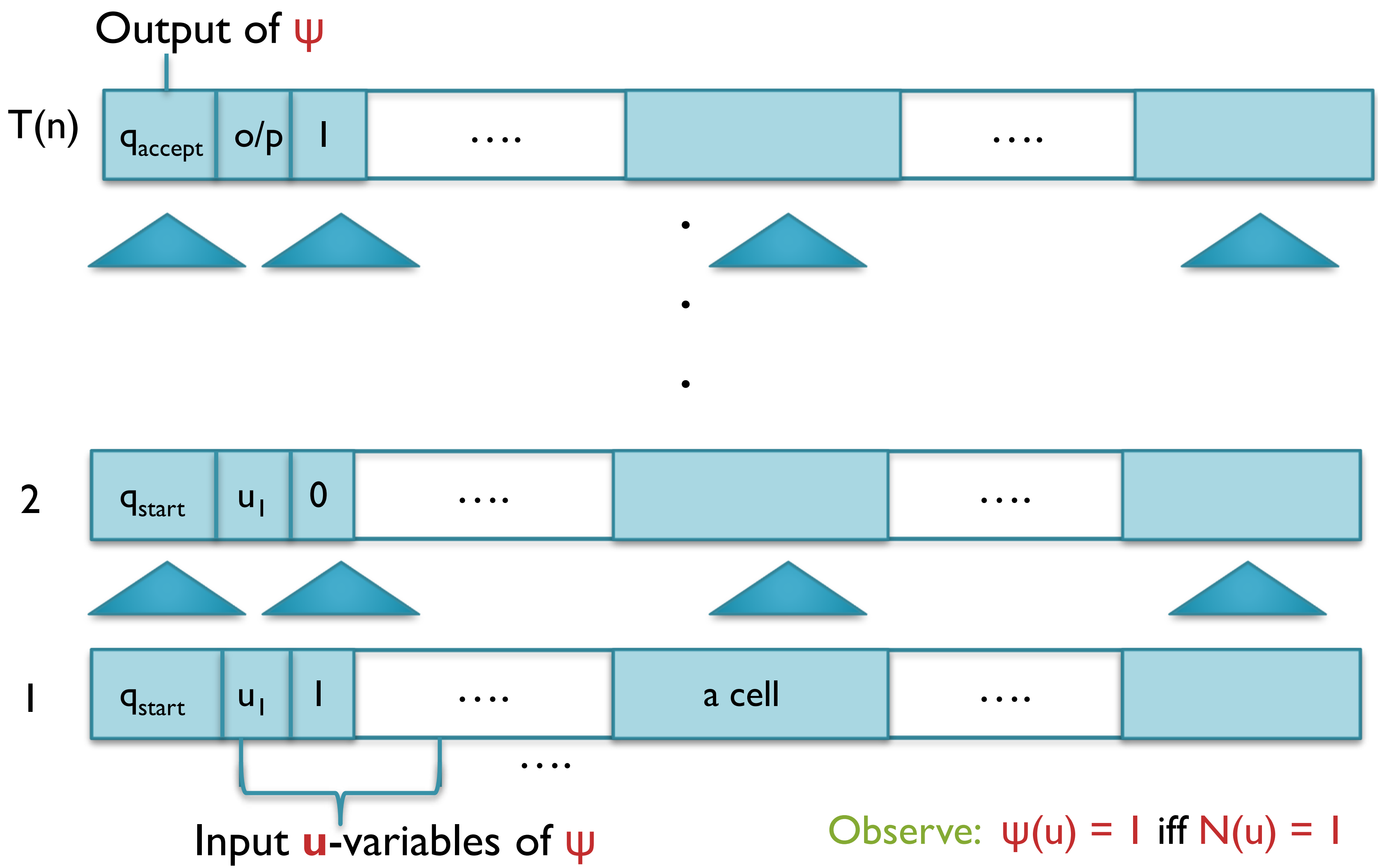


Recap: Main theorem: Step 1

- **Locality of computation:** The bits in $h_{i,j}$, $b_{i,j}$ and $q_{i,j}$ depend ***only on*** the bits in
  - $h_{i-1,j-1}$, $b_{i-1,j-1}$, $q_{i-1,j-1}$,
  - $h_{i-1,j}$, $b_{i-1,j}$, $q_{i-1,j}$,
  - $h_{i-1,j+1}$, $b_{i-1,j+1}$, $q_{i-1,j+1}$



Recap: Main theorem: Step 1



Observe: $\psi(u) = 1$ iff $N(u) = 1$

Recall Steps 1 and 2

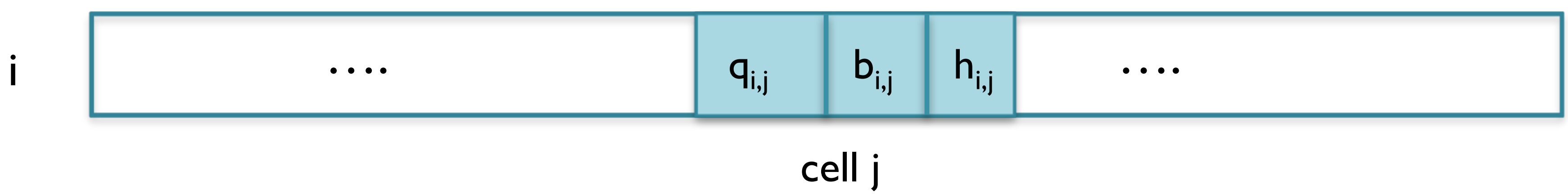
- Step 1. Let N be a deterministic TM that runs in time T(n) on every input u of length n, and outputs 0/1. Then,
  1. There's a Boolean circuit ψ of size *poly*(T(n)) such that ψ(u) = 1 if and only if N(u) = 1.
  2. ψ is computable in time *poly(T(n))* from N, T & n.

- **Step 2.** "Convert" circuit ψ to a CNF ϕ efficiently by introducing auxiliary variables.

Main theorem: Step 2

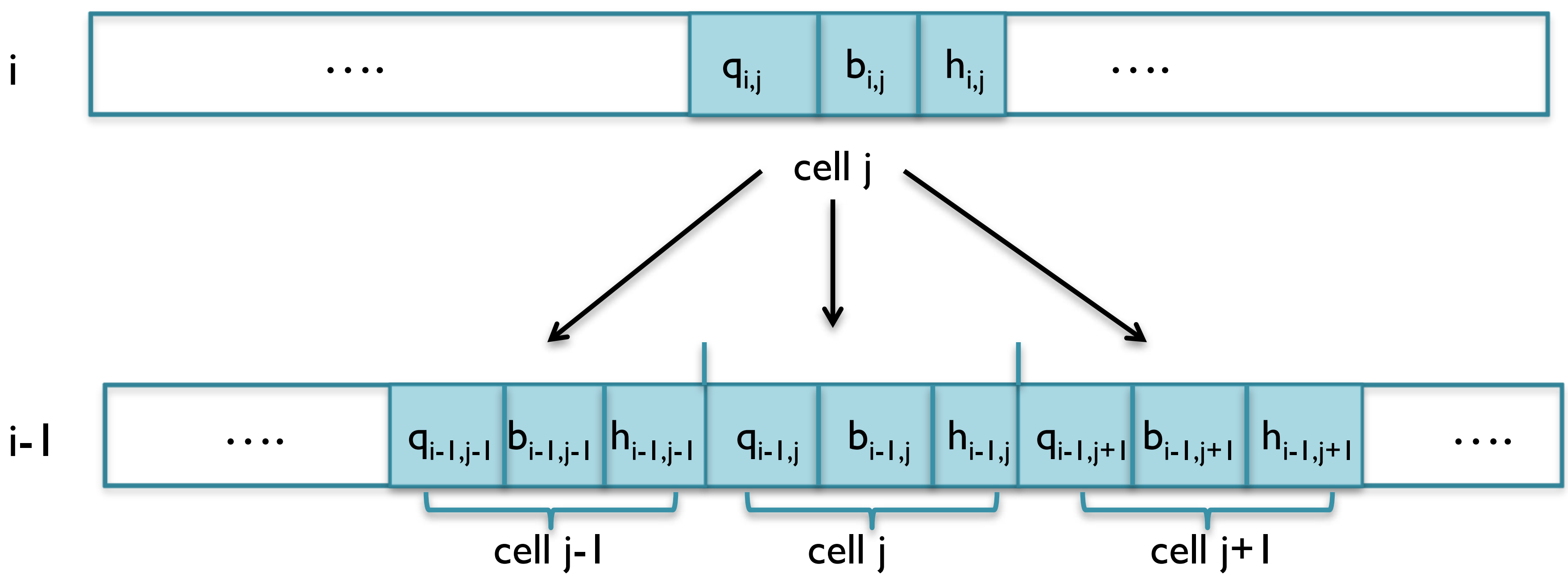
- Think of $h_{i,j}$, $b_{i,j}$ and the bits of $q_{i,j}$ as <u>formal Boolean variables</u>.

auxiliary variables



Main theorem: Step 2

- Locality of computation: The variables $h_{i,j}$, $b_{i,j}$ and $q_{i,j}$ depend only on the variables
  - $h_{i-1,j-1}$, $b_{i-1,j-1}$, $q_{i-1,j-1}$,
  - $h_{i-1,j}$, $b_{i-1,j}$, $q_{i-1,j}$, and
  - $h_{i-1,j+1}$, $b_{i-1,j+1}$, $q_{i-1,j+1}$



Main theorem: Step 2

- Hence,

  $b_{ij} = B_{ij}(h_{i-1,j-1}, b_{i-1,j-1}, q_{i-1,j-1}, h_{i-1,j}, b_{i-1,j}, q_{i-1,j}, h_{i-1,j+1}, b_{i-1,j+1}, q_{i-1,j+1})$

  = a fixed function of the arguments depending only on N's transition function δ.

- The above equality can be captured by a constant size CNF Ψ_{ij} . Also, Ψ_{ij} is easily computable from δ.

Main theorem: Step 2

- Hence,

  $b_{ij} = B_{ij}(h_{i-1,j-1}\,,\ b_{i-1,j-1}\,,\ q_{i-1,j-1}\,,\ h_{i-1,j}\,,\ b_{i-1,j}\,,\ q_{i-1,j}\,,\ h_{i-1,j+1}\,,\ b_{i-1,j+1}\,,\ q_{i-1,j+1})$

  = a fixed function of the arguments depending only on N's transition function δ.

- The above equality can be captured by a <u>constant size</u> CNF Ψ_{ij} . Also, Ψ_{ij} is easily computable from δ.

$x = y$ iff $(x \wedge y) \vee (\neg x \wedge \neg y) = 1.$

Main theorem: Step 2

- Similarly,

  $h_{ij} = H_{ij}(h_{i-1,j-1}\,,\ b_{i-1,j-1}\,,\ q_{i-1,j-1}\,,h_{i-1,j}\,,\ b_{i-1,j}\,,\ q_{i-1,j}\,,h_{i-1,j+1}\,,\ b_{i-1,j+1}\,,\ q_{i-1,j+1})$

  = a fixed function of the arguments depending only on N's transition function δ.

- The above equality can be captured by a <u>constant size</u> CNF Φ_{ij} . Also, Φ_{ij} is easily computable from δ.

Main theorem: Step 2

k-th bit of q_{ij} where $1 \leq k \leq \log |Q|$

- Similarly,

$$q_{ijk} = C_{ijk}(h_{i-1,j-1},\ b_{i-1,j-1},\ q_{i-1,j-1},\ h_{i-1,j},\ b_{i-1,j},\ q_{i-1,j},\ h_{i-1,j+1},\ b_{i-1,j+1},\ q_{i-1,j+1})$$

= a fixed function of the arguments depending only on N's transition function δ.

- The above equality can be captured by a <u>constant size</u> CNF θ_{ijk} . Also, θ_{ijk} is easily computable from δ.

Main theorem: Step 2

- Let λ be the conjunction of Ψ_{ij} , Φ_{ij} and θ_{ijk} for all i, j, k.
    - $i \in [1, T(n)]$,
    - $j \in [1, T(n)]$, and
    - $k \in [1, \log |Q|]$
- λ is a CNF in the u-variables and the <u>auxiliary variables</u> $h_{i,j}$, $b_{i,j}$ and $q_{i,j,k}$. for all i,j,k. $|\lambda|$ is $O(T(n)^2)$.

Main theorem: Step 2

- Let λ be the conjunction of Ψ_{ij} , Φ_{ij} and θ_{ijk} for all i, j, k.
  - $i \in [1,T(n)]$,
  - $j \in [1,T(n)]$, and
  - $k \in [1, \log |Q|]$
- λ is a CNF in the u-variables and the <u>auxiliary variables</u> $h_{i,j}$, $b_{i,j}$ and $q_{i,j,k}$. for all i,j,k. $|\lambda|$ is $O(T(n)^2)$.
- Define $\phi = \lambda \wedge b_{T(n),1}$.

Main theorem: Step 2

- Observe: An assignment to u and the auxiliary variables satisfies λ if and only if it "captures" the computation of N on the assigned input u for T(n) steps.

Main theorem: Step 2

- Observe: An assignment to u and the auxiliary variables satisfies λ if and only if it "captures" the computation of N on the assigned input u for $T(n)$ steps.

- Hence, an assignment to u and the auxiliary variables satisfies ϕ if and only if $N(u) = 1$, i.e., for every u,

$$\phi(u, \text{“}\textit{auxiliary variables}\text{”}) \in SAT \iff N(u) = 1.$$

Recall the Main Theorem

- Main Theorem. Let N be a deterministic TM that runs in time $T(n)$ on every input u of length n, and outputs 0/1. Then,
  1. There's a CNF ϕ(u, "*auxiliary variables*") of size *poly*(T(n)) such that for every u, ϕ(u, "*auxiliary variables*") is satisfiable <u>as a function of the "*auxiliary variables*"</u> if and only if N(u) = 1.
  2. ϕ is computable in time *poly(T(n))* from N, T & n.

- ϕ(u, "*auxiliary variables*") is satisfiable <u>as a function of **all** the variables</u> if and only if $\exists$u s.t N(u) = 1.

Main theorem: Comments

- ϕ is a CNF of size $O(T(n)^2)$ and is also computable from N, T and n in $O(T(n)^2)$ time.

- Remark 1. With some more effort, size ϕ can be brought down to $O(T(n). \log T(n))$.

- Remark 2. The reduction from x to ϕ_x is not just a poly-time reduction, it is actually a <u>*log-space reduction*</u> (we'll define this later).

Main theorem: Comments

- ϕ is a function of u and some "auxiliary variables" (the b_{ij}, h_{ij} and q_{ijk} variables).

- Observe that once u is fixed <u>the values of the "auxiliary variables" are also determined</u> in any satisfying assignment for ϕ.

- **Each clause of ϕ has only <u>constantly</u> many literals!**

3SAT is NP-complete

- Definition. A CNF is a called a k-CNF if every clause has at most k literals.

  e.g. a 2-CNF $\phi = (x_1 \vee x_2) \wedge (x_3 \vee \neg x_2)$

- Definition. k-SAT is the language consisting of all *satisfiable k-CNFs.*

3SAT is NP-complete

- Definition. A CNF is a called a k-CNF if every clause has at most k literals.

  e.g. a 2-CNF $\phi = (x_1 \lor x_2) \land (x_3 \lor \neg x_2)$

- Definition. k-SAT is the language consisting of all *satisfiable k-CNFs.*

- Theorem. 3-SAT is NP-complete.

Proof sketch: $(x_1 \lor x_2 \lor x_3 \lor \neg x_4)$ is satisfiable iff $(x_1 \lor x_2 \lor z) \land (x_3 \lor \neg x_4 \lor \neg z)$ is satisfiable.

More NP-complete problems

NP complete problems: Examples

- Independent Set
- Clique
- Vertex cover
- 0/1 integer programming
- Max-Cut (NP-hard)

Karp 1972

- 3-coloring planar graphs *Stockmeyer 1973*
- 2-Diophantine solvability *Adleman & Manders 1975*

Ref: Garey & Johnson, "*Computers and Intractability*" 1979

NPC problems from number theory

- SqRootMod: Given natural numbers a, b and c, check if there exists a natural number $x \leq c$ such that

$$x^2 = a \pmod{b}.$$

- Theorem: SqRootMod is NP-complete.

Manders & Adleman 1976

NPC problems from number theory

- Variant_IntFact : Given natural numbers L, U and N, check if there exists a **natural number** $d \in [L, U]$ such that d divides N.

- Claim: Variant_IntFact is NP-hard under *randomized poly-time reduction*.

- Reference: *https://cstheory.stackexchange.com/questions/4769/an-np-complete-variant-of-factoring/4785*

A peculiar NP problem

- Minimum Circuit Size Problem (MCSP): Given the **truth table** of a Boolean function f and an integer s, check if there is a circuit of size $\leq$ s that computes f.

- Easy to see that MCSP is in NP.

- Is MCSP NP-complete? Not known!

Example 1: Independent Set

- INDSET := {(G, k): G has independent set of size k}

- Goal: Design a poly-time reduction f s.t.

$$x \in 3SAT \iff f(x) \in INDSET$$

- Reduction from 3SAT: Recall, a reduction is just an efficient algorithm that takes input a 3CNF ϕ and outputs a (G, k) tuple s.t

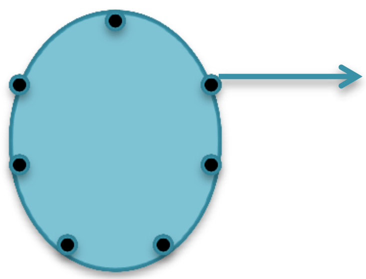
$$\phi \in 3SAT \iff (G, k) \in INDSET$$

Example 1: Independent Set

- Reduction: Let ϕ be a 3CNF with m clauses and n variables. Assume, every clause has exactly 3 literals.

Example 1: Independent Set

- Reduction: Let ϕ be a 3CNF with m clauses and n variables. Assume, every clause has exactly 3 literals.

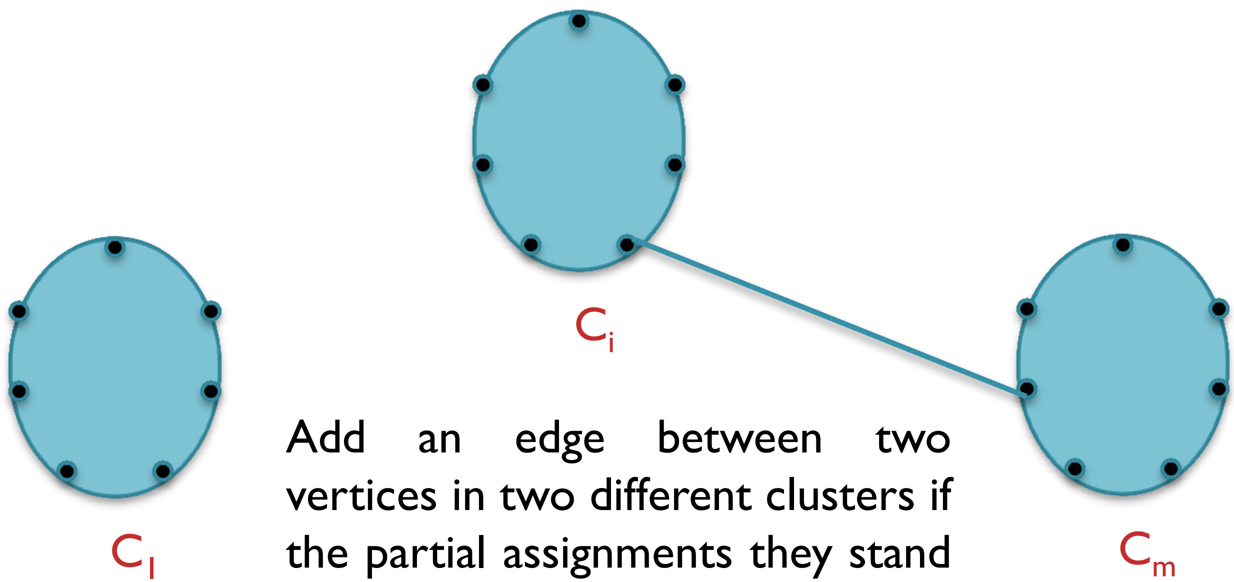


A vertex stands for a partial assignment of the variables in C_i that satisfies the clause

For every clause C_i form a complete graph (cluster) on 7 vertices

Example 1: Independent Set

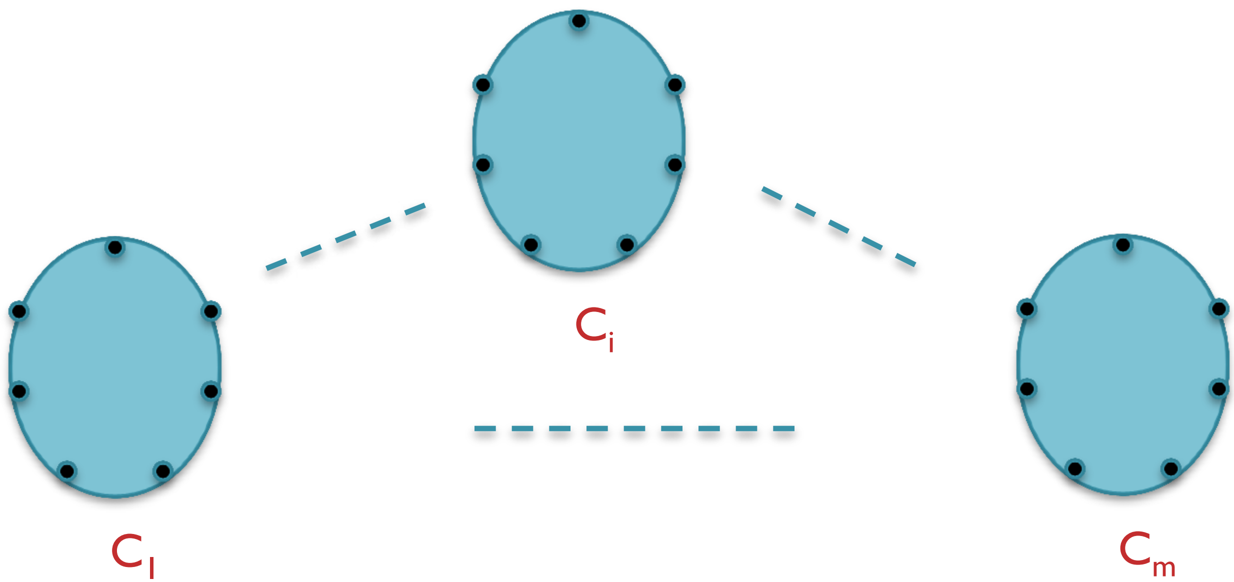
- Reduction: Let ϕ be a 3CNF with m clauses and n variables. Assume, every clause has exactly 3 literals.



Add an edge between two vertices in two different clusters if the partial assignments they stand for are <u>incompatible</u>.

Example 1: Independent Set

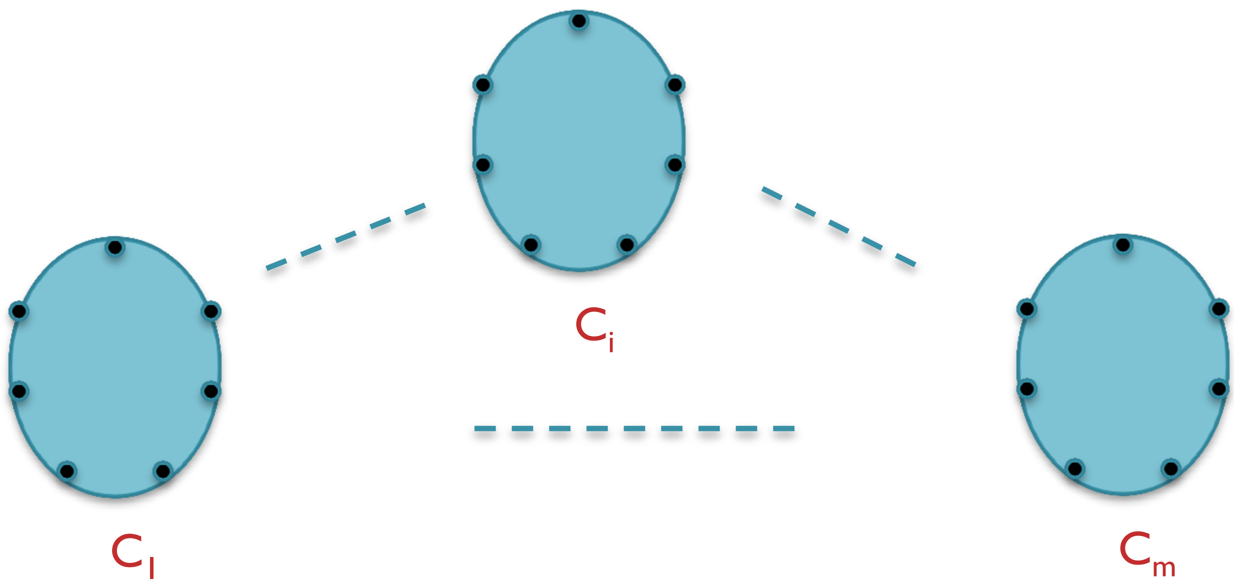
- Reduction: Let ϕ be a 3CNF with m clauses and n variables. Assume, every clause has exactly 3 literals.



Graph G on $7m$ vertices

Example 1: Independent Set

- Reduction: Let ϕ be a 3CNF with m clauses and n variables. Assume, every clause has exactly 3 literals.



- Obs: ϕ is satisfiable iff G has an ind. set of size m.

Example 2: Clique

- CLIQUE := {(H, k): H has a clique of size k}

- Goal: Design a poly-time reduction f s.t.

$$x \in \text{INDSET} \Leftrightarrow f(x) \in \text{CLIQUE}$$

- Reduction from INDSET: The reduction algorithm computes $\overline{G}$ from G

$$(G, k) \in \text{INDSET} \Leftrightarrow (\overline{G}, k) \in \text{CLIQUE}$$

Example 3: Vertex Cover

- VCover := {(H, k): H has a vertex cover of size k}

- Goal: Design a poly-time reduction f s.t.

$$x \in \text{INDSET} \iff f(x) \in \text{VCover}$$

- Reduction from INDSET: Let n be the number of vertices in G. The reduction algorithm maps (G, k) to (G, n-k).

$$(G, k) \in \text{INDSET} \iff (G, n-k) \in \text{VCover}$$

Example 4: 0/1 Integer Programming

- 0/1 IProg := Set of satisfiable 0/1 integer programs
- A <u>0/1 integer program</u> is a set of linear inequalities with rational coefficients and the variables are allowed to take only 0/1 values.

- Reduction from 3SAT: A clause is mapped to a linear inequality as follows

$$x_1 \vee \bar{x}_2 \vee x_3 \longrightarrow x_1 + (1 - x_2) + x_3 \geq 1$$