Lecture 19: Complexity

Decision Problems

- **Decision problem**: assignment of inputs to YES (1) or NO (0)
- Inputs are either **NO inputs** or **YES inputs**

Problem	Decision
s-t Shortest Path	Does a given G contain a path from s to t with weight at most d?
Negative Cycle	Does a given G contain a negative weight cycle?
Longest Simple Path	Does a given G contain a **simple** path with weight at least d?
Subset Sum	Does a given set of integers A contain a subset with sum S?
Tetris	Can you survive a given sequence of pieces in given board?
Chess	Can a player force a win from a given board?
Halting problem	Does a given computer program terminate for a given input?

- **Algorithm/Program**: constant-length code (working on a word-RAM with $\Omega(\log n)$-bit words) to solve a problem, i.e., it produces correct output for every input and the length of the code is independent of the instance size
- Problem is **decidable** if there exists a program to solve the problem in finite time

Decidability

- Program is finite (constant) string of bits, i.e., a nonnegative integer $\in \mathbb{N}$.
  Problem is function $p : \mathbb{N} \to \{0, 1\}$, i.e., infinite string of bits.
- (# of programs $|\mathbb{N}|$, countably infinite) $\ll$ (# of problems $|\mathbb{R}|$, uncountably infinite)
- (Proof by Cantor's diagonalization argument, probably covered in 6.042)
- Proves that most decision problems not solvable by any program (undecidable)
- E.g., the Halting problem is undecidable (many awesome proofs in 6.045)
- Fortunately most problems we think of are algorithmic in structure and are decidable

Decidable Decision Problems

R	problems decidable in finite time	('R' comes from recursive languages)
EXP	problems decidable in exponential time $2^{n^{O(1)}}$	(most problems we think of are here)
P	problems decidable in polynomial time $n^{O(1)}$	(efficient algorithms, the focus of this class)

- These sets are distinct, i.e., **P** $\subsetneq$ **EXP** $\subsetneq$ **R** (via time hierarchy theorems, see 6.045)
- E.g., Chess is in **EXP** $\setminus$ **P**

Nondeterministic Polynomial Time (NP)

- **P** is the set of decision problems for which there is an algorithm A such that, for every input I of size n, A on I runs in poly(n) time and solves I correctly
- **NP** is the set of decision problems for which there is a **verification** algorithm V that takes as input an input I of the problem and a **certificate** bit string of length polynomial in the size of I, so that:
  - V always runs in time polynomial in the size of I;
  - if I is a YES input, then there is some certificate c so that V outputs YES on input (I, c); and
  - if I is a NO input, then no matter what certificate c we choose, V always output NO on input (I, c).
- You can think of the certificate as a **proof** that I is a YES input.
  If I is actually a NO input, then no proof should work.

Problem	Certificate	Verifier
s-t Shortest Path	A path P from s to t	Adds the weights on P and checks whether $\leq d$
Negative Cycle	A cycle C	Adds the weights on C and checks whether < 0
Longest Simple Path	A path P	Checks whether P is a **simple** path with weight $\geq d$
Subset Sum	A set of items A'	Checks whether $A' \in A$ has sum S
Tetris	Sequence of moves	Checks that the moves allow survival

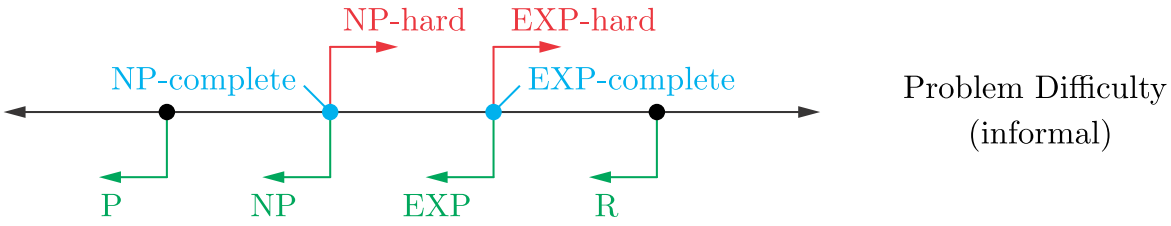
- **P** $\subseteq$ **NP**: The verifier V just solves the instance ignoring any certificate
- **NP** $\subseteq$ **EXP**: Try all possible certificates! At most $2^{n^{O(1)}}$ of them, run verifier V on all
- **Open:** Does **P** $=$ **NP**? **NP** $=$ **EXP**?
- Most people think **P** $\subsetneq$ **NP** ($\subsetneq$ **EXP**), i.e., generating solutions harder than checking
- If you prove either way, people will give you lots of money ($1M Millennium Prize)
- Why do we care? If can show a problem is hardest problem in **NP**,
  then problem cannot be solved in polynomial time if **P** $\neq$ **NP**
- How do we relate difficulty of problems? Reductions!

Reductions

- Suppose you want to solve problem A
- One way to solve is to convert A into a problem B you know how to solve
- Solve using an algorithm for B and use it to compute solution to A
- This is called a **reduction** from problem A to problem B ($A \to B$)
- Because B can be used to solve A, B is **at least as hard** as A ($A \leq B$)
- General algorithmic strategy: reduce to a problem you know how to solve

A	Conversion	B
Unweighted Shortest Path	Give equal weights	Weighted Shortest Path
Integer-weighted Shortest Path	Subdivide edges	Unweighted Shortest Path
Longest Path	Negate weights	Shortest Path

- Problem A is **NP-hard** if every problem in **NP** is polynomially reducible to A
- i.e., A is at least as hard as (can be used to solve) every problem in **NP** ($X \leq A$ for $X \in$ **NP**)
- **NP-complete = NP** $\cap$ **NP-hard**
- All **NP-complete** problems are equivalent, i.e., reducible to each other
- First **NP-complete** problem? Every decision problem reducible to satisfying a logical circuit, a problem called "Circuit SAT".
- Longest Simple Path and Tetris are **NP-complete**, so if any problem is in **NP** $\setminus$ **P**, these are
- Chess is **EXP-complete**: in **EXP** and reducible from every problem in **EXP** (so $\notin$ **P**)



Examples of NP-complete Problems

- Subset Sum from L18 ("weakly NP-complete" which is what allows a pseudopolynomial-time algorithm, but no polynomial algorithm unless **P** = **NP**)
- 3-Partition: given n integers, can you divide them into triples of equal sum? ("strongly NP-complete": no pseudopolynomial-time algorithm unless **P** = **NP**)
- Rectangle Packing: given n rectangles and a target rectangle whose area is the sum of the n rectangle areas, pack without overlap
  - Reduction from 3-Partition to Rectangle Packing: transform integer a_i into $1 \times a_i$ rectangle; set target rectangle to $n/3 \times \left(\sum_i a_i\right)/3$
- Jigsaw puzzles: given n pieces with possibly ambiguous tabs/pockets, fit the pieces together
  - Reduction from Rectangle Packing: use uniquely matching tabs/pockets to force building rectangles and rectangular boundary; use one ambiguous tab/pocket for all other boundaries
- Longest common subsequence of n strings
- Longest simple path in a graph
- Traveling Salesman Problem: shortest path that visits all vertices of a given graph (or decision version: is minimum weight $\leq d$)
- Shortest path amidst obstacles in 3D
- 3-coloring given graph (but 2-coloring $\in$ **P**)
- Largest clique in a given graph
- SAT: given a Boolean formula (made with AND, OR, NOT), is it every true?
  E.g., x AND NOT x is a NO input
- Minesweeper, Sudoku, and most puzzles
- Super Mario Bros., Legend of Zelda, Pokémon, and most video games are **NP-hard** (many are harder)

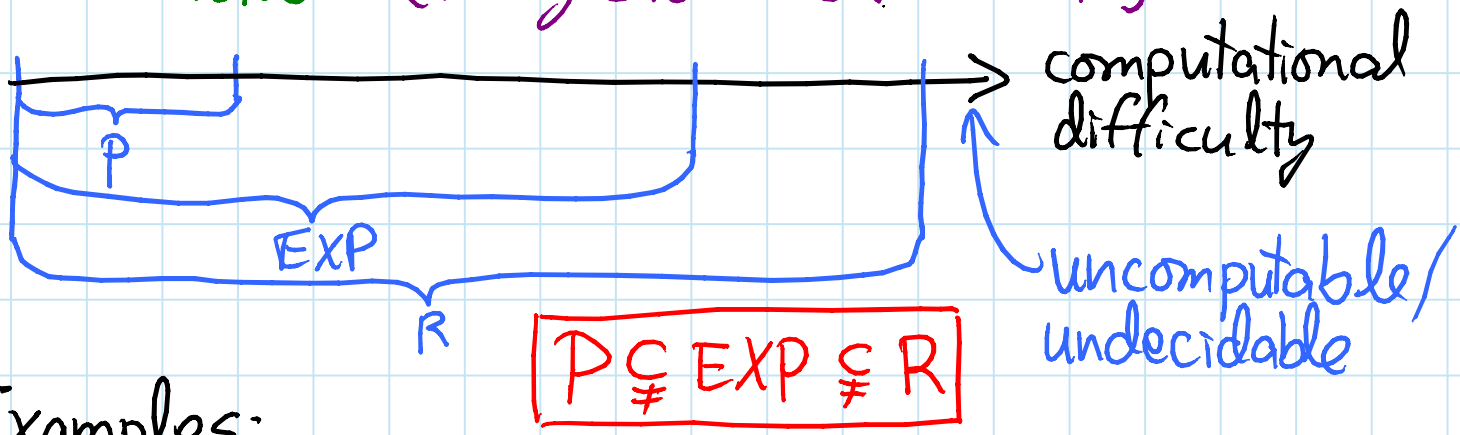
6.006	Lecture 19	Apr. 30, 2020

TODAY: Computational Complexity
- P, NP, EXP, R
- most problems are uncomputable ($\notin R$)
- hardness & completeness
- reductions

P = {problems solvable in polynomial time}
$n^{O(1)}$ time where n = INPUT SIZE
(what this class is all about)

EXP = {problems solvable in exponential time}
$2^{n^{O(1)}}$

R = {problems solvable in finite time}
"recursive" [Turing 1936; Church 1941]



$P \subsetneq EXP \subsetneq R$

Examples:
- negative-weight cycle detection $\in P$
- $n \times n$ Chess $\in EXP$ but $\notin P$
  - who wins from given board config.?
- Tetris $\in EXP$ but don't know whether $\in P$
  - survive given pieces from given board

Halting problem: given a computer program, does it ever halt (stop)?

- uncomputable ($\notin R$): no algorithm solves it (correctly in finite time on all inputs)
  [see 6.045 for a proof]
- decision problem: answer is YES or NO [L18]

Most decision problems are uncomputable:

- program $\approx$ binary string $\approx$ nonneg. integer $\in \mathbb{N}$
- decision problem = a function from binary strings ($\approx$ nonneg. integers) to $\{YES, NO\}$ ($\approx \{0, 1\}$)

$\approx$ infinite sequence of bits $\approx$ real number $\in \mathbb{R}$

- $|\mathbb{N}| \ll |\mathbb{R}|$: no assignment of unique nonneg. integers to real numbers ($\mathbb{R}$ uncountable)
  (proof by "diagonalization" argument [6.042])

$\Rightarrow$ not nearly enough programs for all problems

- each program solves only one problem

$\Rightarrow$ almost all problems cannot be solved ☹

- luckily, many (most?) problems we care about can be solved

NP = {decision problems solvable in poly. time via a "lucky" algorithm}

↳ can make lucky guesses, always "right", without trying all options (cf. DP)

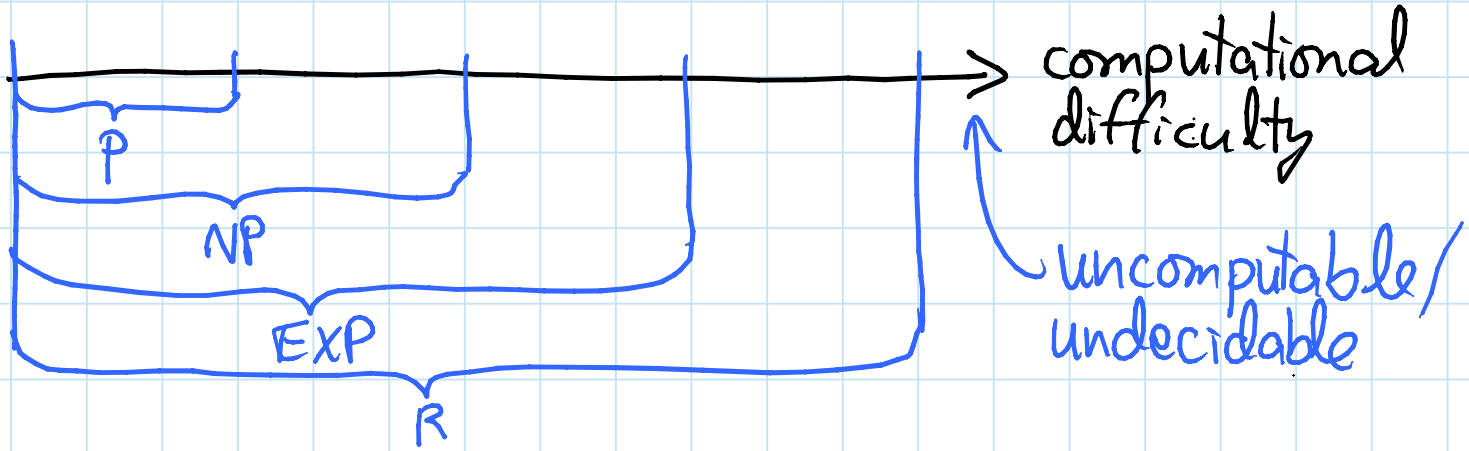
- nondeterministic model: algorithm makes (binary) guesses & then says YES or NO
- guesses guaranteed to lead to YES outcome if possible (NO otherwise)

= {decision problems with solutions that can be "checked" in polynomial time}

- given problem input + some "certificate," polynomial-time verification algorithm

∃ proofs → - ∀ YES input: ∃ certificate: verifier says YES

no false proofs → - ∀ NO input: ∀ certificate: verifier says NO



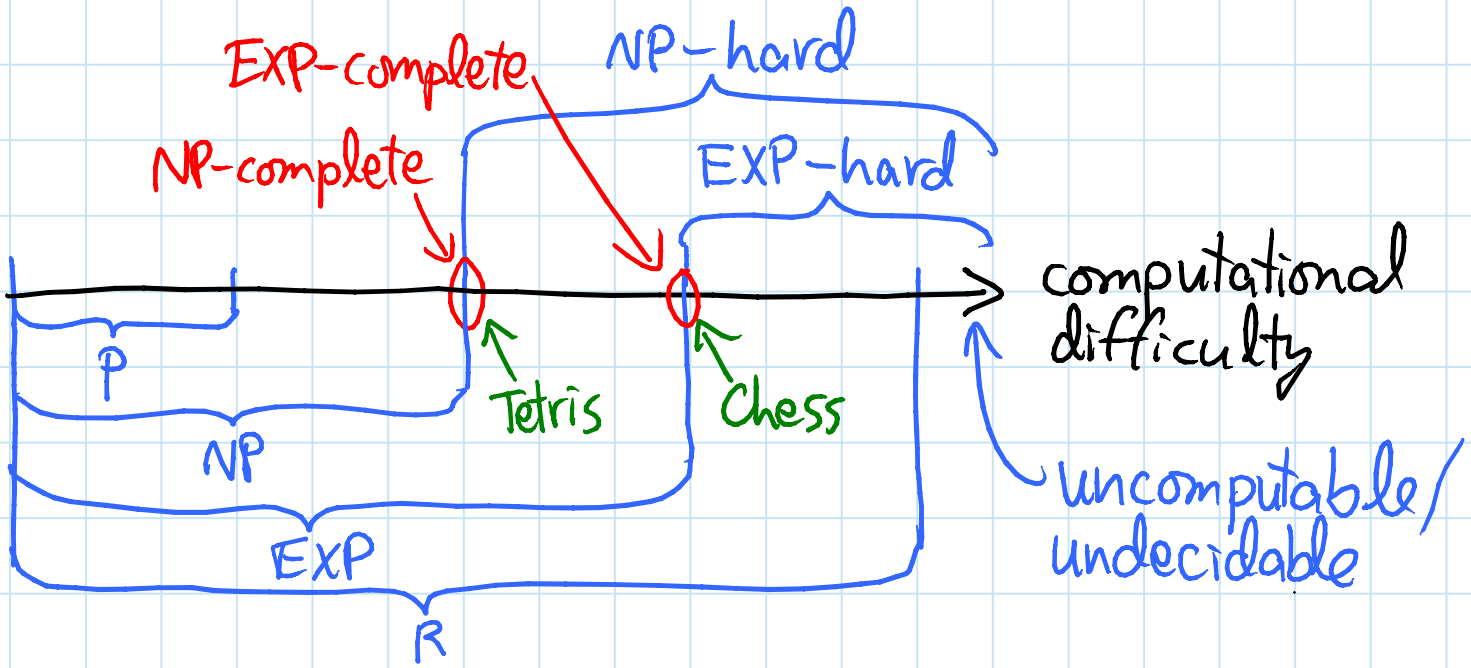
Example: Tetris ∈ NP

- nondeterministic alg: - guess each move
  - did I survive?
- proof of YES: list what moves to make (rules of Tetris are easy)

P≠NP: big conjecture (worth $1,000,000)
≈ can't engineer luck
≈ generating proofs/solutions is sometimes harder than checking them

Claim: if P≠NP, then Tetris ∈ NP∖P
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2004]

Why? Tetris is **NP-hard**
= "as hard as" every problem ∈ NP
– in fact **NP-complete** = NP ∩ NP-hard



Similarly: Chess is **EXP-complete**
= EXP ∩ EXP-hard
as hard as every problem in EXP
⇒ if NP≠EXP, then Chess ∉ EXP∖NP
also open, but less famous/"important"
– know P≠EXP ⇒ Chess ∉ P

Reductions: convert your problem into a problem you already know how to solve (instead of solving from scratch)

- most common algorithm design technique (and big part of 6.006)
- unweighted shortest path → weighted
  (set weights = 1)
- integer-weighted shortest path → unweighted
  (subdivide edges)
- longest path → shortest path
  (negate weights)

A→B reduction: A input → B input

B solution → A solution

⇒ A is at least as easy as B
i.e. B is at least as hard as A

- NP-complete problems are all interreducible using polynomial-time reductions (same difficulty)

⇒ can use reductions to prove NP-hardness
e.g. 3-Partition → Tetris [see slides]

Examples of NP-complete problems:

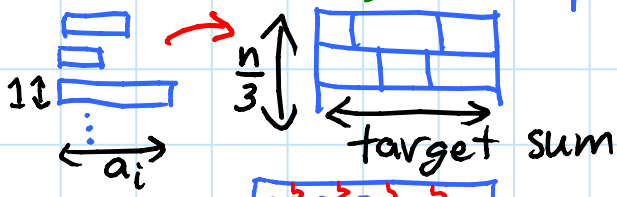
[L18]

- Subset Sum (pseudopoly, not poly) "weakly NP-complete"
- 3-Partition: given n integers, can you divide them into triples of equal sum? "strongly NP-complete"

(not even pseudopoly. possible if P≠NP)

REDUCTIONS:

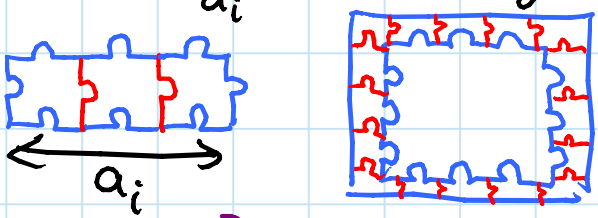
→ rectangle packing



→ jigsaw puzzles

ambiguous

unique



[Demaine & Demaine 2007]

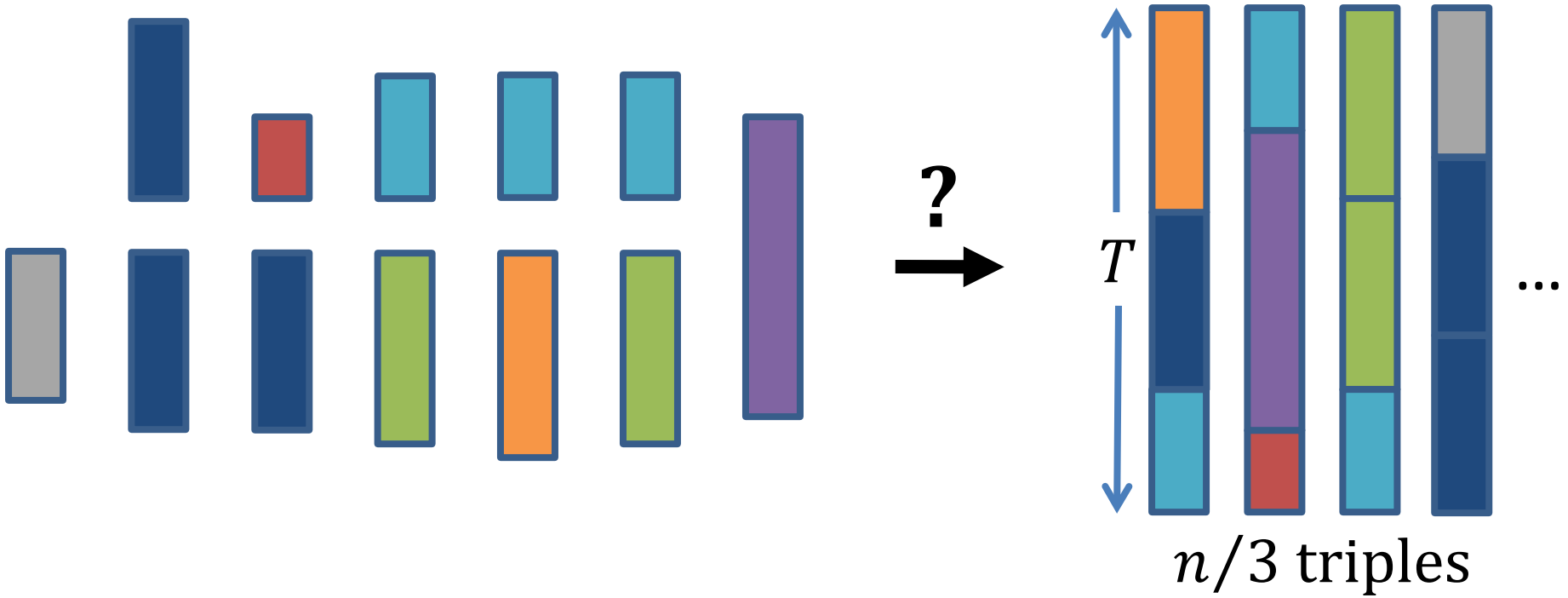
- longest common subsequence of n strings
- longest simple path in a graph
  - → deciding whether you've already won in Settler's of Catan is NP-complete!
- Traveling Salesman Problem: shortest path that visits all vertices of a given graph
  - decision version: is min weight ≤ x?
- shortest paths amidst obstacles in 3D
- 3-coloring a given graph (but 2-coloring ∈ P)
- find largest clique in a given graph
- SAT: given a Boolean formula (and, or, not), is it ever true? x and not x → NO
- Minesweeper, Sudoku, & most puzzles
- Super Mario Bros., Legend of Zelda, Pokémon, ... are NP-hard [Aloupis, Demaine, Guo 2012]

(probably not ∈ NP)

Classic NP-complete Problem: *3-Partition*

[Garey & Johnson 1975]

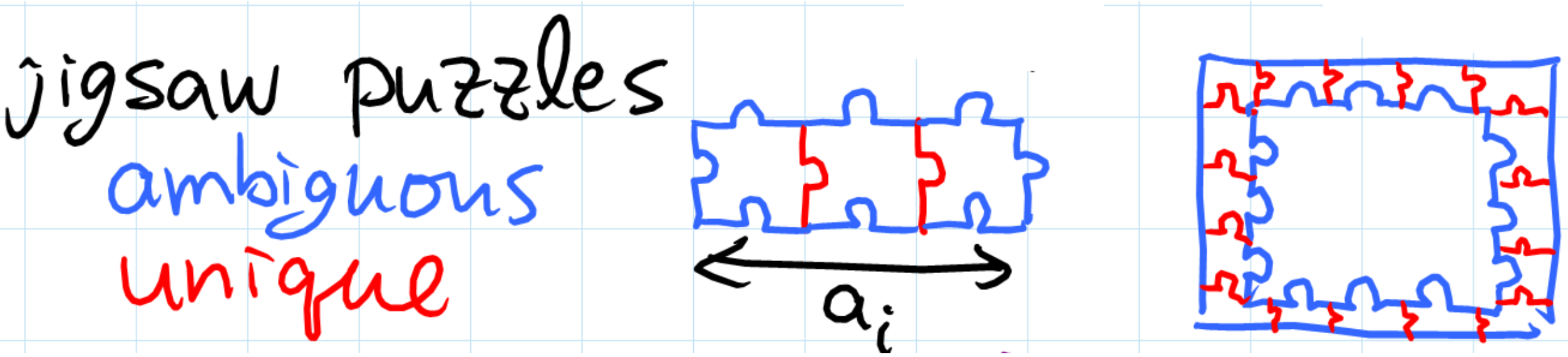
- Given n integers $a_1, a_2, \ldots, a_n$ between 0 and n, can you partition them into $n/3$ triples with the same sum? $\left[T = \frac{(\sum_i a_i)}{n/3}\right]$

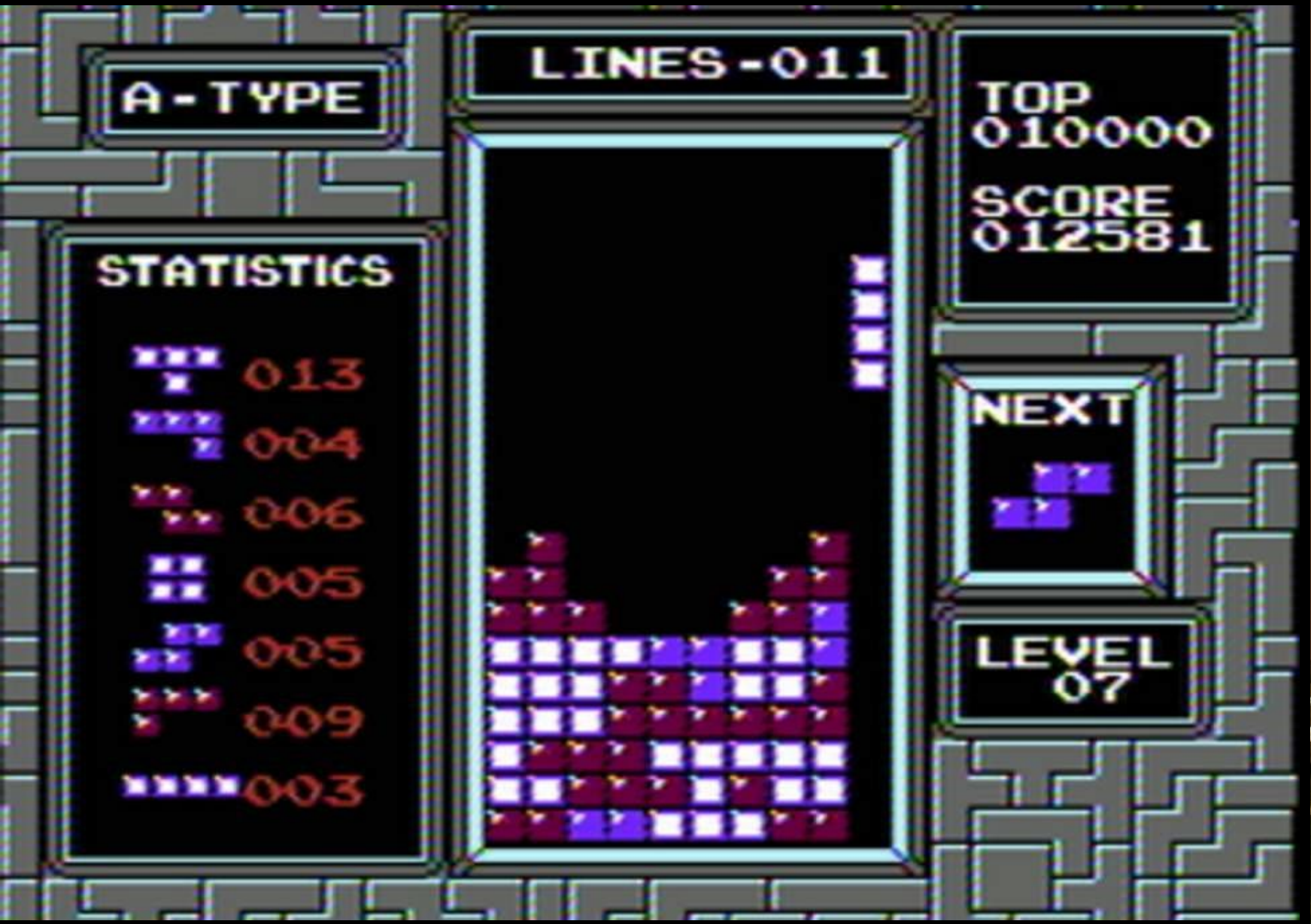


Reduction from 3-Partition to Jigsaw Puzzles

[Demaine & Demaine 2007]

- Given n integers $a_1, a_2, \ldots, a_n$ between 0 and n, can you partition them into $n/3$ triples with the same sum? $\left[t = \frac{(\sum_i a_i)}{n/3}\right]$

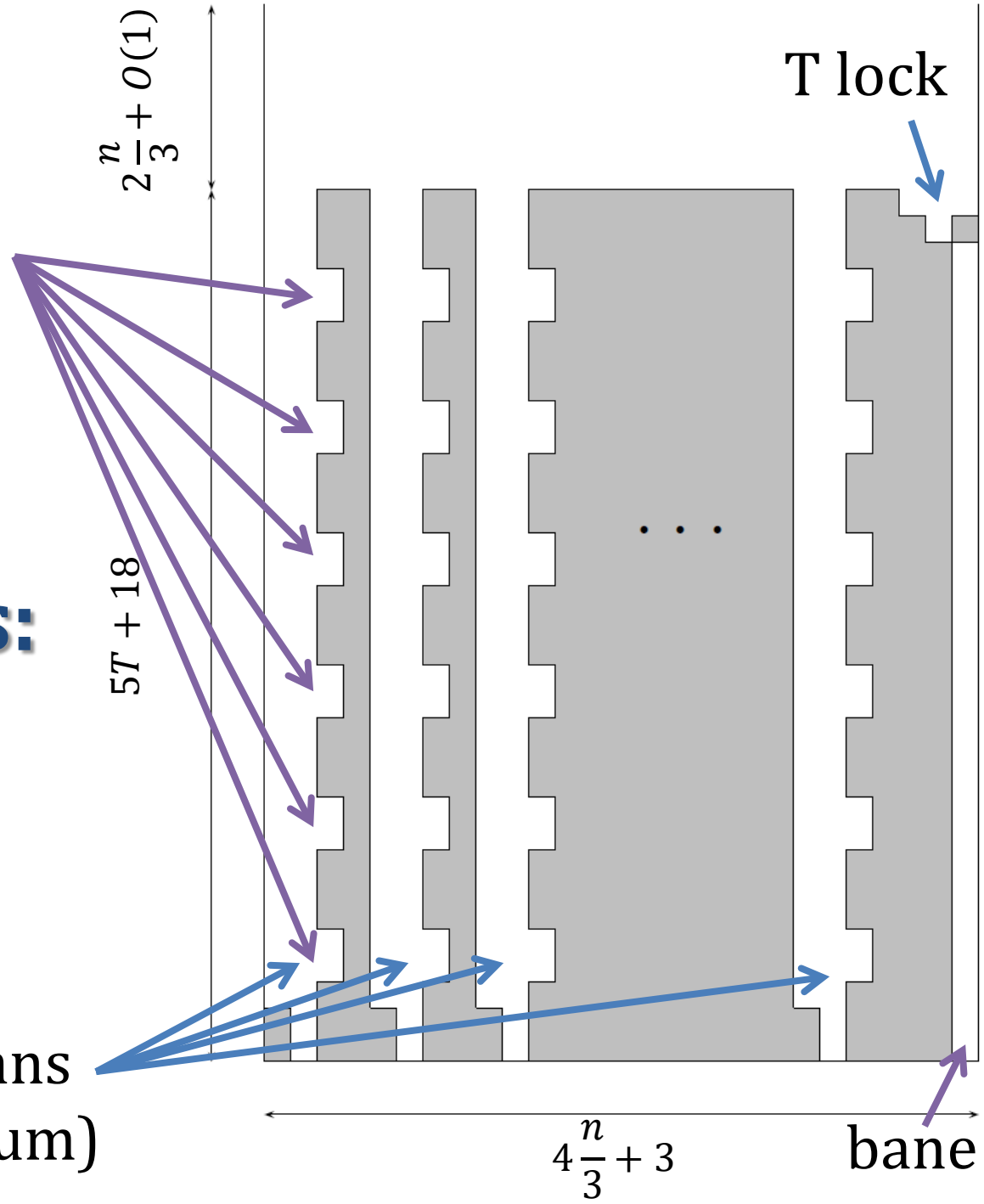




MIT Green Bldg.
April 2012

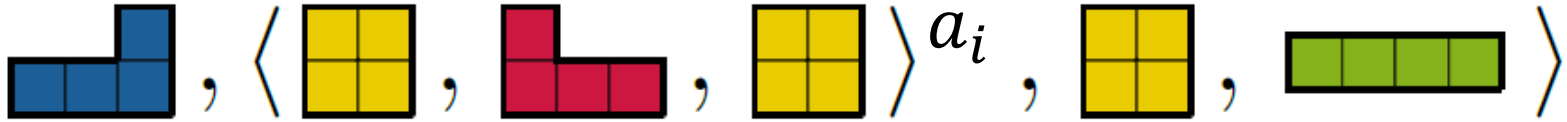
Reduction from 3-Partition to Tetris: Initial Board

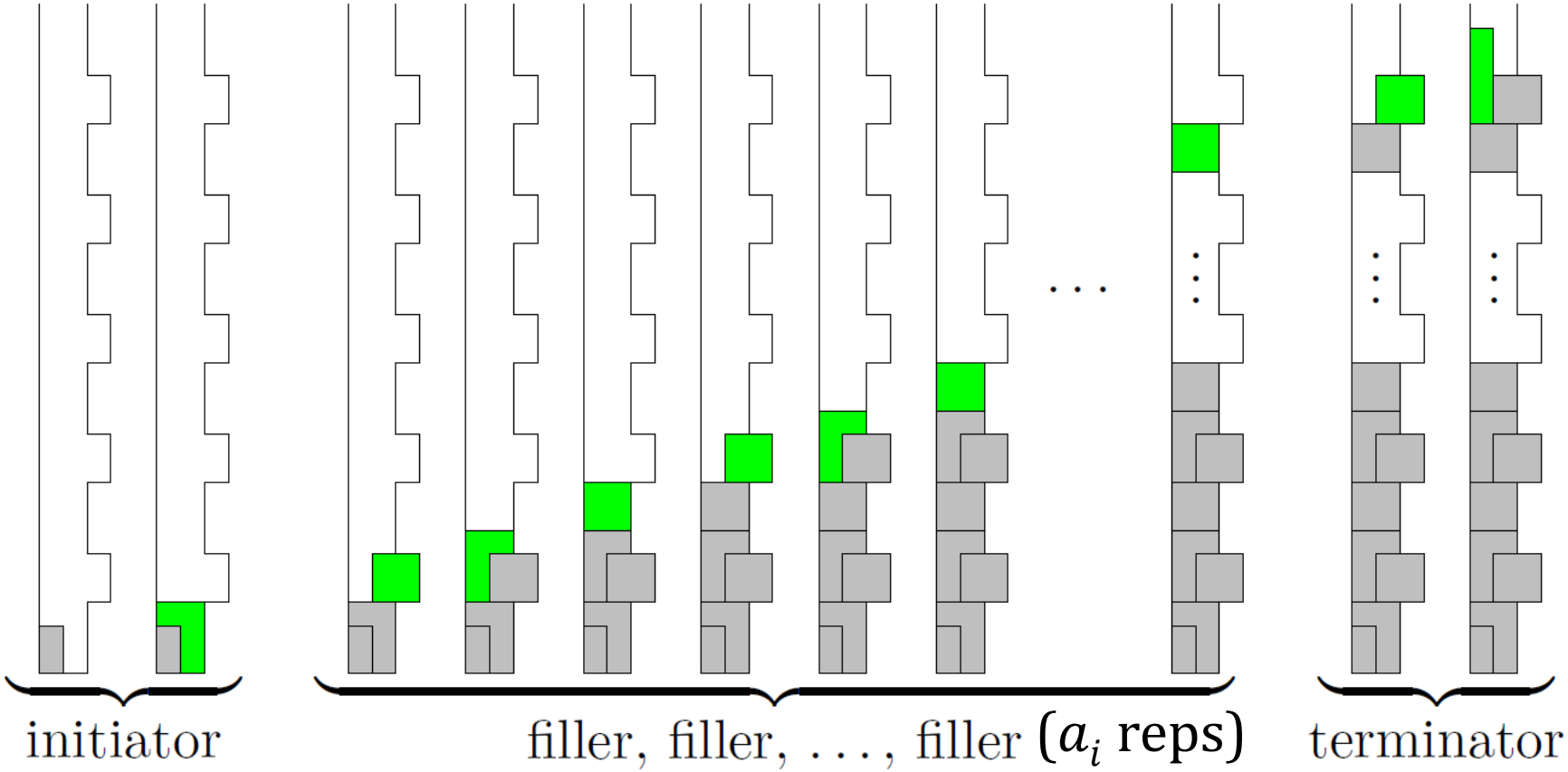
(it is possible to actually get here)



[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

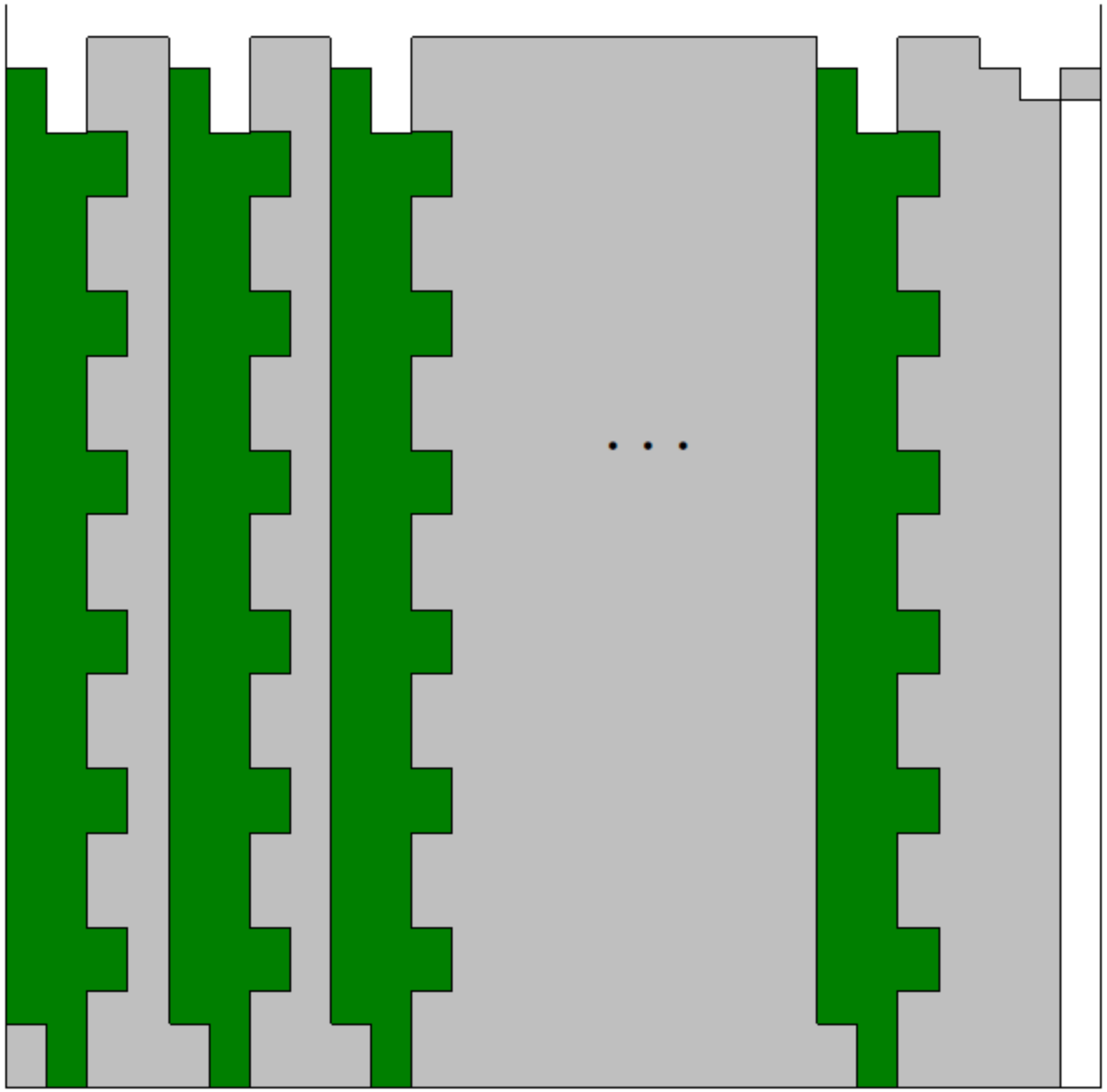
Reduction from 3-Partition to Tetris: Piece Sequence

$a_i \mapsto \langle$  $\rangle^{a_i}$



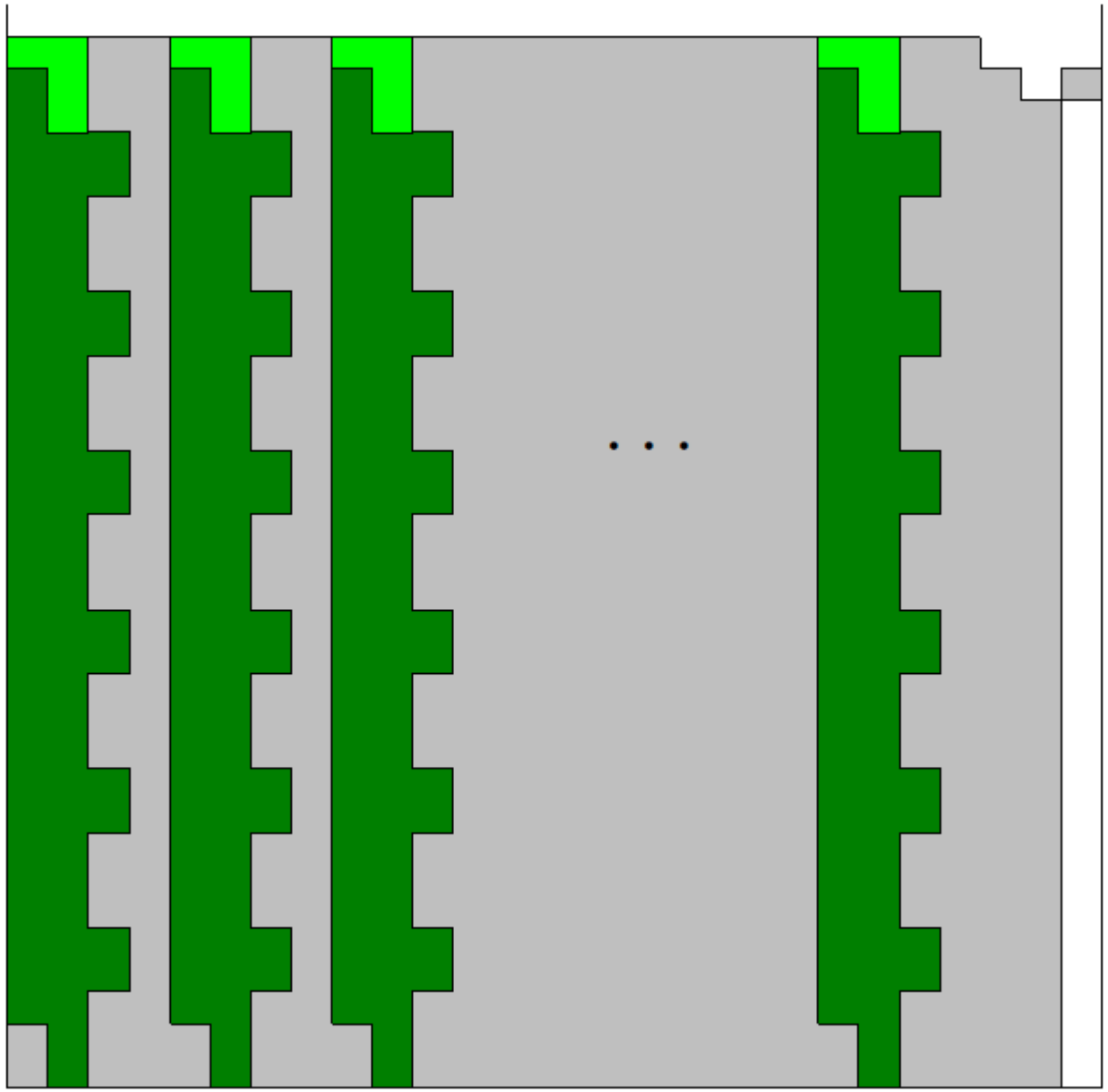
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

Finale Pieces



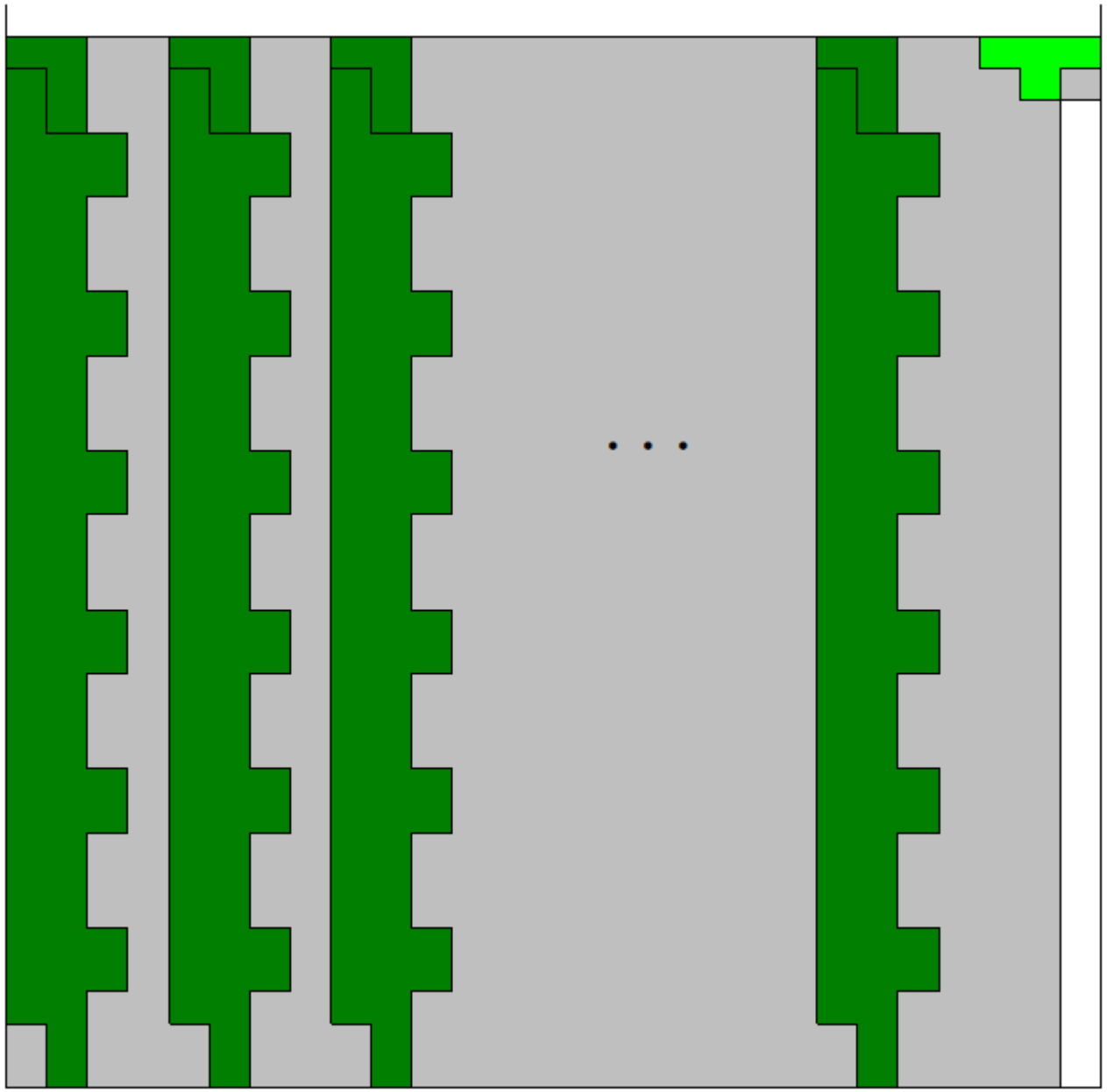
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

Finale Pieces



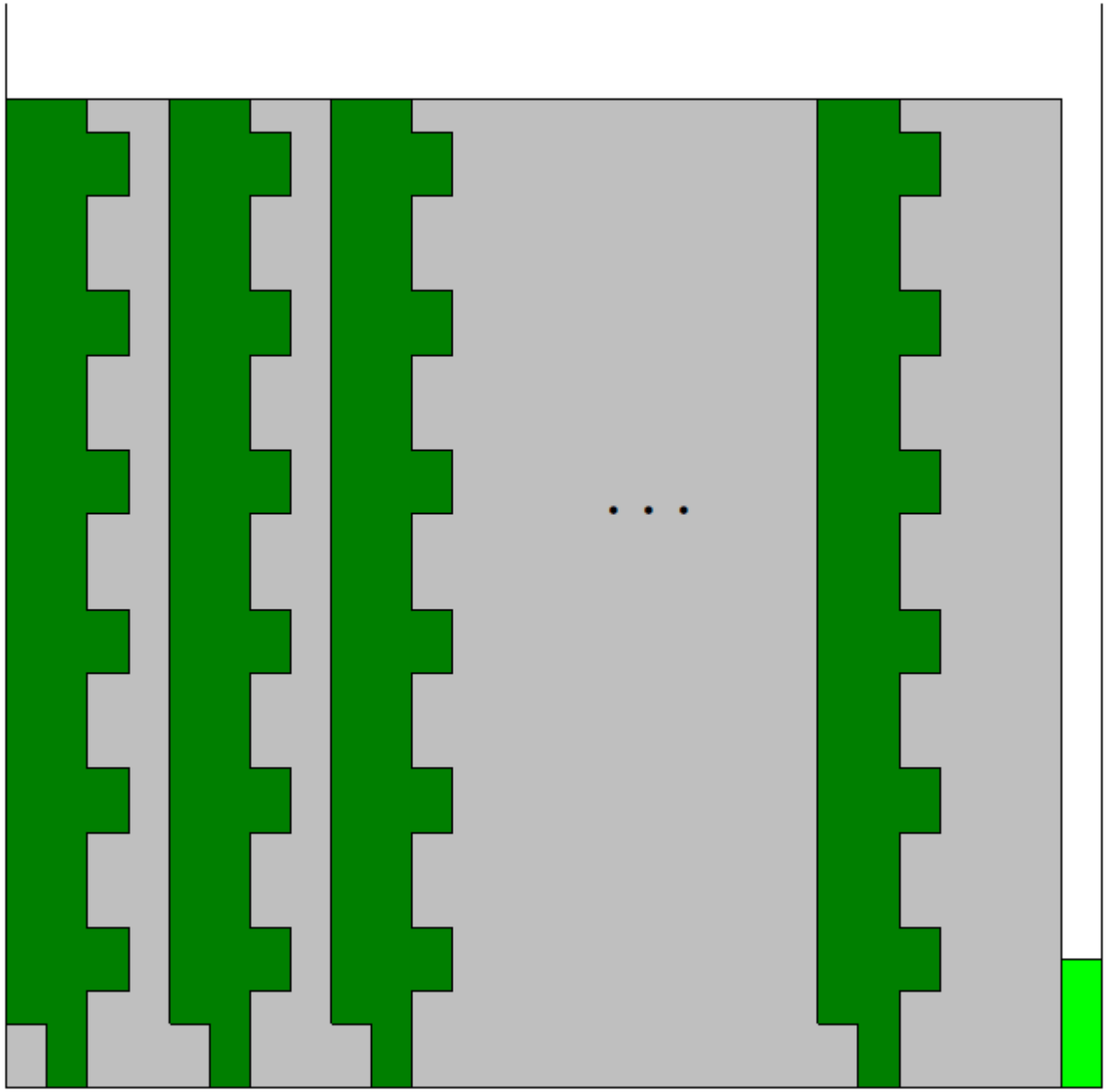
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

Finale Pieces



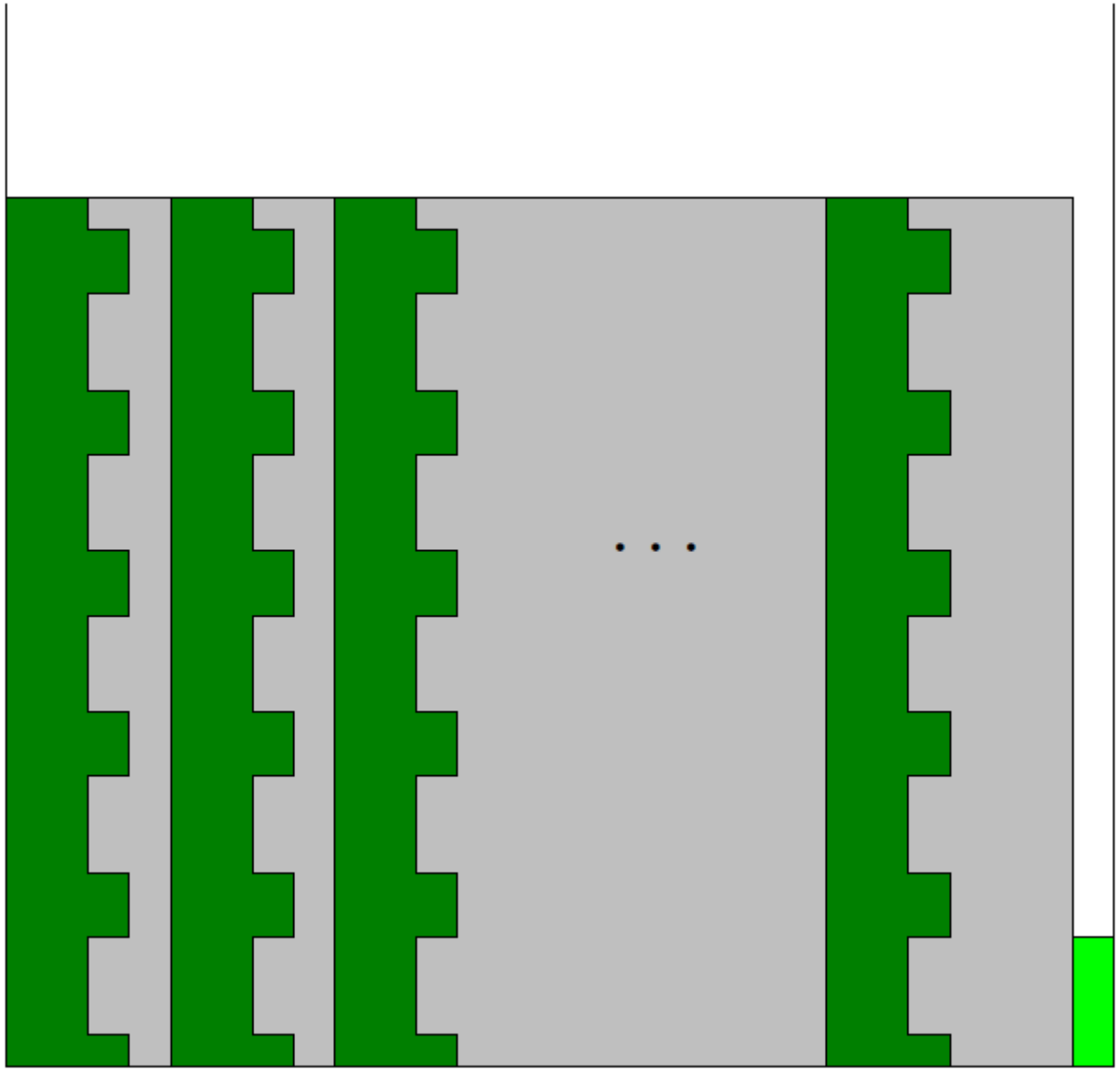
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

Finale Pieces



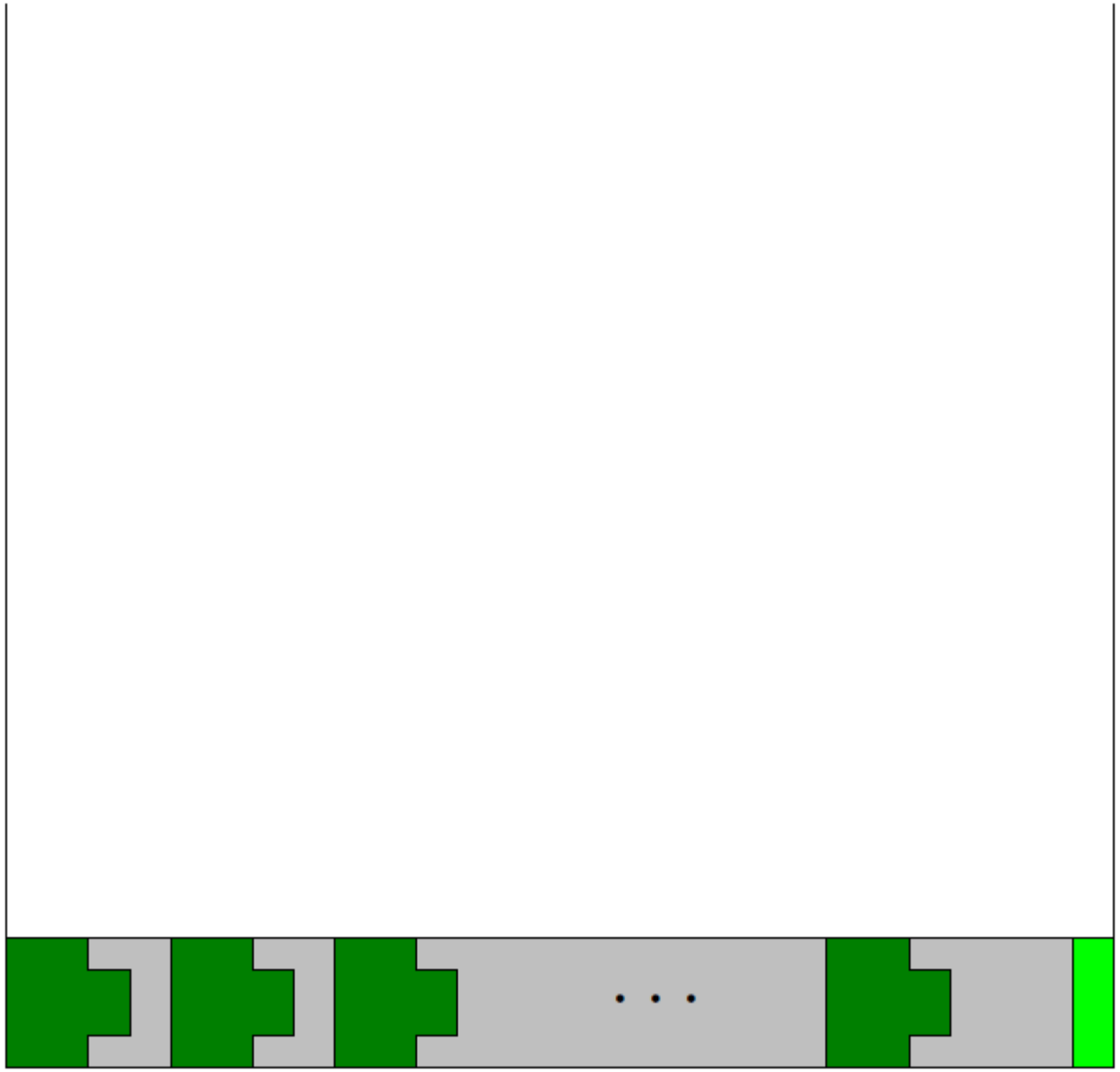
[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

Finale Pieces

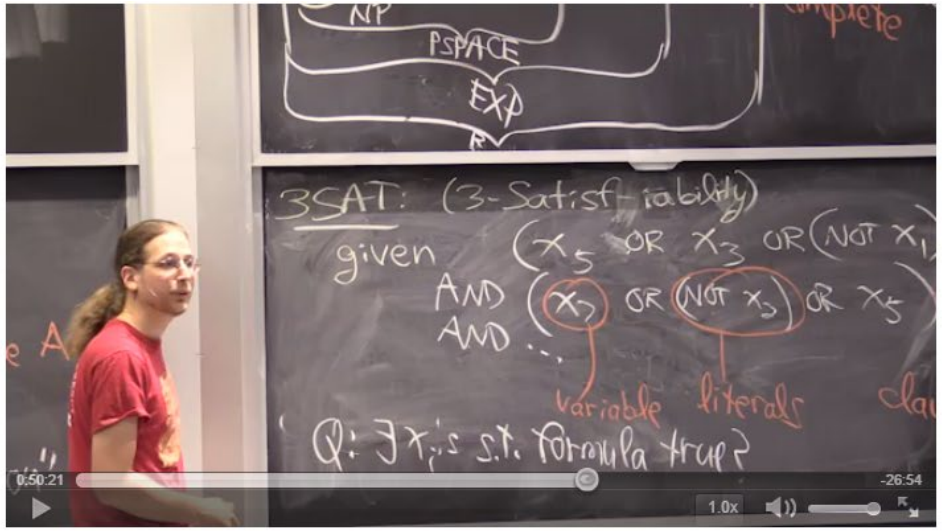


[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

Finale Pieces

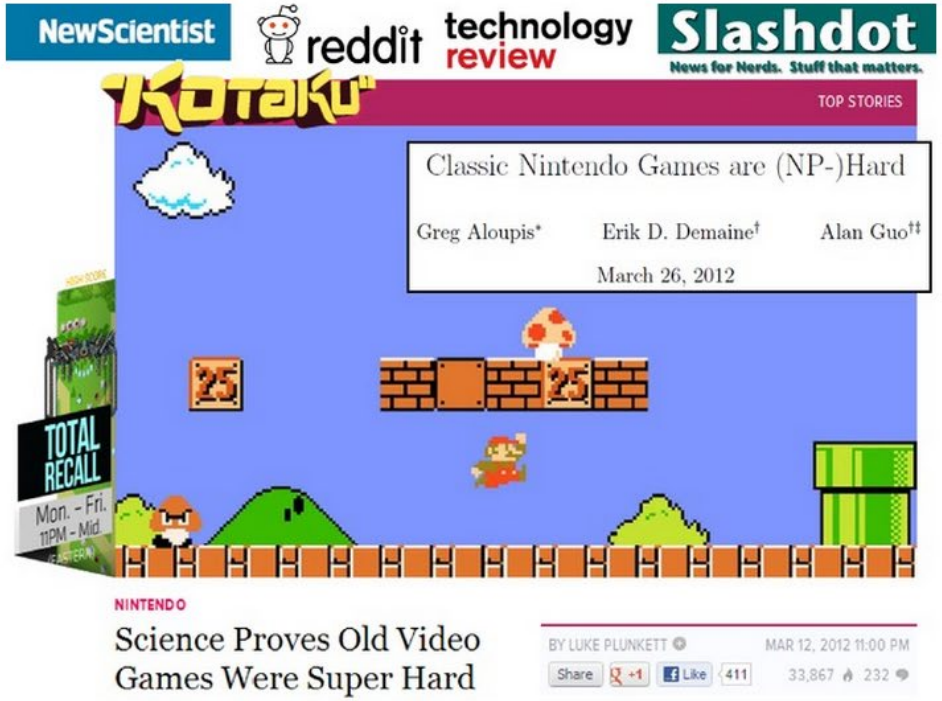


[Breukelaar, Demaine, Hohenberger, Hoogeboom, Kosters, Liben-Nowell 2003]

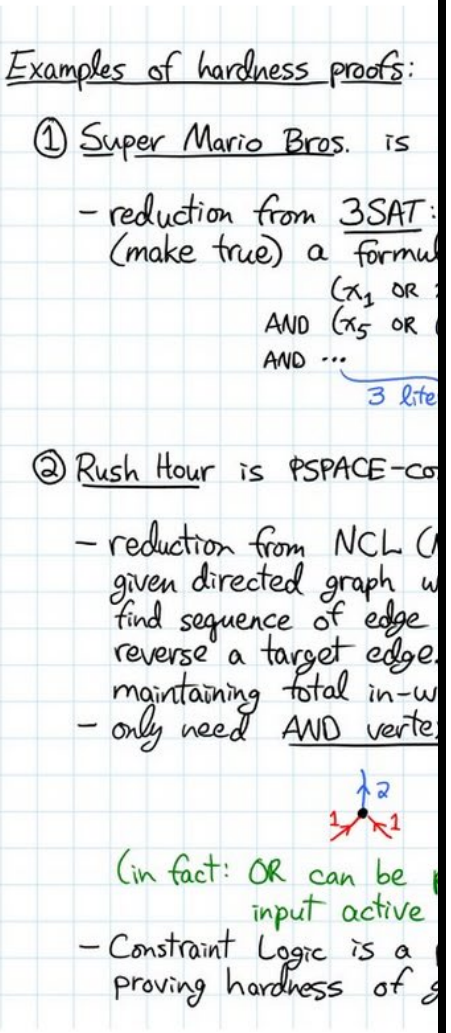


Download Video: 360p, 720p

Slides, page 5/21 • [previous page] • [next page] • [PDF]
Video times: • 47:27–50:56



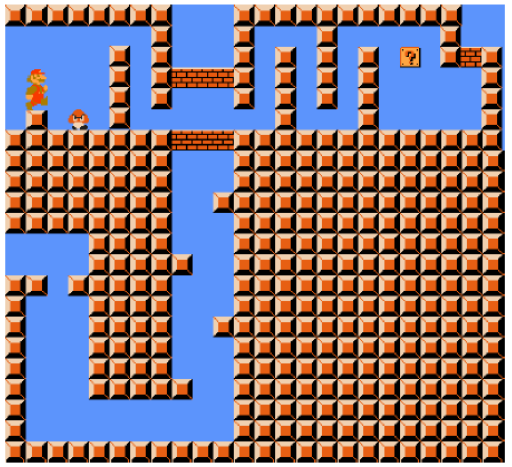
Handwritten notes, page 8/8 • [previous p
Video times: • 47:27-



Handwritten notes, page 8/8 • [previous p

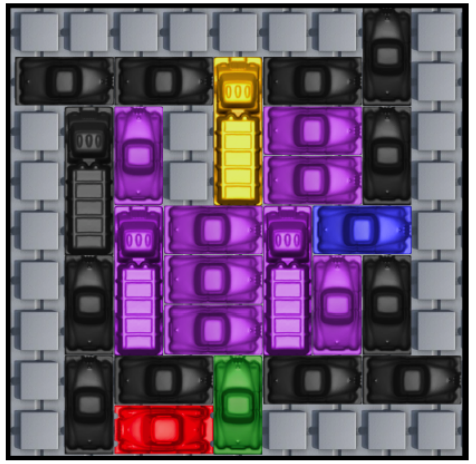
Algorithmic Lower Bounds: Fun with Hardness Proofs

Super Mario Bros.



Crossover gadget for NP-hardness

Rush Hour



AND gadget for PSPACE-hardness

Minesweeper

OR gadget for NP-hardness

Hardness Made Easy*

Learn **when to give up** the search for efficient algorithms; see **connections** between computational problems; **solve puzzles** to prove theorems, solve **open problems,** and write papers.

Topics: NP, PSPACE, EXPTIME, EXPTIME, EXPSPACE, 3SUM, approximation, fixed parameter, games & puzzles, 3SUM, key problems, gadgets, and proof styles.

6.890 taught by Professor Erik Demaine
Grad H, AUS, and Theoretical CS Concentration
Tuesday & Thursday 3:30-5:00pm in room 2-105
http://courses.csail.mit.edu/6.890/
sign up for our mailing list to join the class

Fall 2014
Spring 2019

** Easiness not guaranteed. Side effects such as open problems and a heightened sense of complexity may occur. Ask your advisor if 6.890 is right for you!*