Effective Tilings

Florent Becker

September 3, 2015

Local rules are not enough

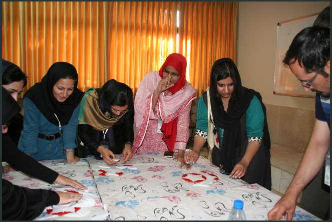
- You saw in Thomas' lecture that one can enforce some Cut-and-project tilings through local rules.

Local rules are not enough

- You saw in Thomas' lecture that one can enforce some Cut-and-project tilings through local rules.
- Still, if you turn these local rules into a puzzle, it is hard to assemble.

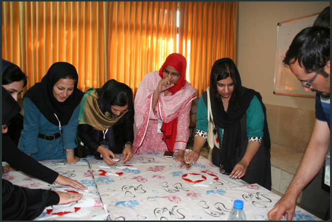
Local rules are not enough

- You saw in Thomas' lecture that one can enforce some Cut-and-project tilings through local rules.
- Still, if you turn these local rules into a puzzle, it is hard to assemble. I have pictures to prove this.



Local rules are not enough

- You saw in Thomas' lecture that one can enforce some Cut-and-project tilings through local rules.
- Still, if you turn these local rules into a puzzle, it is hard to assemble. I have pictures to prove this.
- A Tiling Algorithm is harder to construct than a Tiling.



The Emil Makovicky Objection

Imagine you are building a tomb in Maragha

- then you are only interested in finite patterns

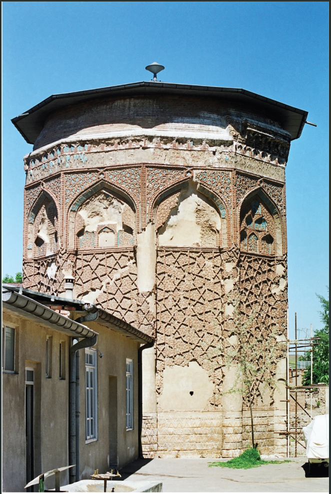


Fig. 1

The Emil Makovicky Objection

Imagine you are building a tomb in Maragha

- then you are only interested in finite patterns
- so everything is decidable

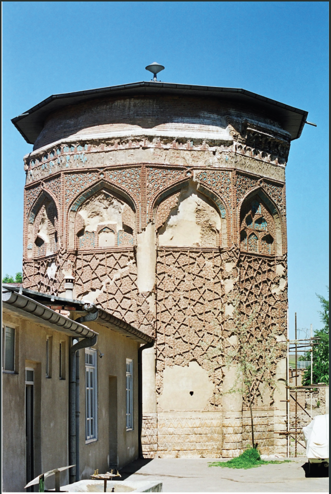


Fig. 1

The Emil Makovicky Objection

Imagine you are building a tomb in Maragha

- then you are only interested in finite patterns
- so everything is decidable
- But: what is the algorithm?

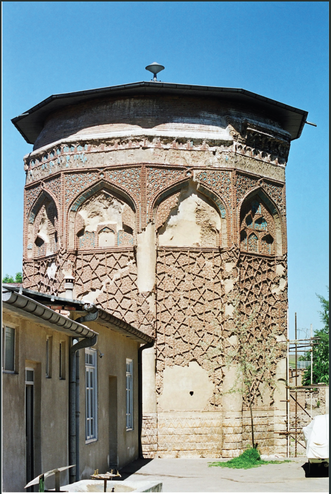


Fig. 1

The Emil Makovicky Objection

Imagine you are building a tomb in Maragha

- then you are only interested in finite patterns
- so everything is decidable
- But: what is the algorithm? Build the tomb and destroy it again and again until the Tiling is good

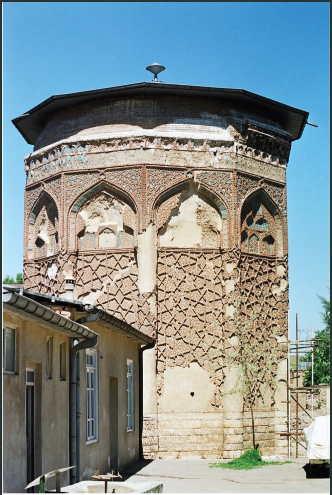


Fig. 1

The Emil Makovicky Objection

Imagine you are building a tomb in Maragha

- then you are only interested in finite patterns
- so everything is decidable
- But: what is the algorithm? Build the tomb and destroy it again and again until the Tiling is good
- I don't think they had enough resources, so they had to be clever.

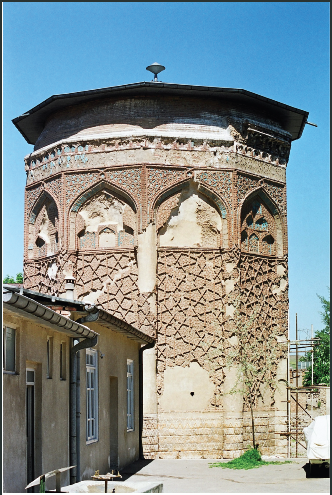


Fig. 1

The Emil Makovicky Objection

Imagine you are building a tomb in Maragha

- then you are only interested in finite patterns
- so everything is decidable
- But: what is the algorithm? Build the tomb and destroy it again and again until the Tiling is good
- I don't think they had enough resources, so they had to be clever.
- Is there *always* a clever solution?

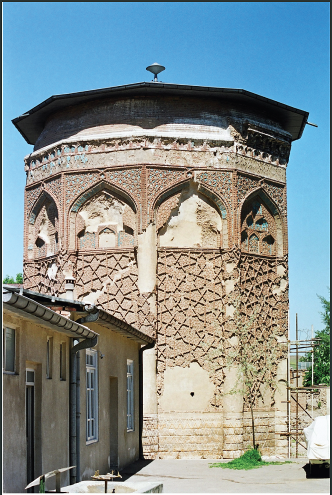


Fig. 1

Resources

- Time
- Working Memory (Space)
- What happens when they are restricted?
- A theory of *feasible* algorithms.

Resources

- Time
- Working Memory (Space)
- What happens when they are restricted?
- A theory of *feasible* algorithms.
- Tiling algorithms play a particular part in that theory.

Local algorithms

In Cellular Automata and Turing Machines, we assume a global synchronization mechanism:

- In Turing Machines, there is only one head

Local algorithms

In Cellular Automata and Turing Machines, we assume a global synchronization mechanism:

- In Turing Machines, there is only one head
- In Cellular Automata, all cells update synchronously

Local algorithms

In Cellular Automata and Turing Machines, we assume a global synchronization mechanism:

- In Turing Machines, there is only one head
- In Cellular Automata, all cells update synchronously
- Tomorrow, we will see a system for tiling algorithms *without* global synchronization: self-assembly.

Outline

Today efficient algorithms with Turing Machines:

- Time and Space complexity
- The classes P and NP
- NP-complete problems
- *this is not in the lecture notes (sorry)*

Tomorrow self-assembly

These are two mostly independent points of view.

Possible is not (necessarily) fast

Given an algorithm *i.e., a Turing Machine*,

- does it take long to compute
- does it eat all memory?

Time complexity

Definition

Let M be a Turing Machine, and x a word. The computation time $t_M(x)$ of M on x is the number of steps M takes to stop on input x.

The worst-case time complexity T_M of M is the function from $\mathbb{N} \to \bar{\mathbb{N}}$ defined by $T_M(n) = max\{t_M(x) | x \text{ of size n } \}$

Example

- There is a machine to check if a word contains a 0 with time complexity n.

Example

- There is a machine to check if a word contains a 0 with time complexity n.
- There is a machine to check if a word is a palindrome with time complexity $O(n^2)$.

Space complexity

Definition

Let M be a Turing Machine with k input tapes, and x a k-uple of words. The computation space $s_M(x)$ of M on x is the number of distrinct cells of the work-tape M writes on, when run on input x.
The worst-case space complexity S_M of M is the function from $\mathbb{N} \rightarrow \bar{\mathbb{N}}$ defined by $S_M(n) = max\{s_M(x) | x \text{ of size n }\}$.

A link between time and space

Theorem

Let M be a Turing Machine, $S_M \geq T_M$.

Proof.

To write on a tape-cell, one first needs to get there. □

Shrinking space

Theorem

Let M be Turing Machine, there is a Turing Machine M' with
$S_M = \lceil \frac{S_M}{2} \rceil$

Accelerating time

Theorem

Let M be Turing Machine, there is a Turing Machine M' with $T_M = \lceil \frac{T_M}{2} \rceil$

Using O notation

Because of the previous two theorems, time and space complexities are always given using the O notation.

Complexity of a problem

Definition

Let f be a computable function, a function T is a time complexity lower bound for f if for any Turing Machine M implementing f, $T_M(n) = \Omega(f(n))$.

Theorem

The problem "palindrome" *has a time complexity lower bound of n^2 on one-tape Turing Machines.*

Finding a closed class

Evaluating precise complexity is complicated, in particular:

- it does not let us compose functions
- it is very sensitive to encoding
- it is sensitive to details of the definition of the machine

The class P

Definition

We say that a function f *is in class P* if there is a Turing Machine for computing f with time complexity $O(n^k)$ for some k.

This class corresponds to the generally admitted notion of *feasible*, but it also contains functions computable in n^{1234}.

Simplifying hypothesis

In order to determine if some problem is in P, we can assume the following:

- We work in a high-level like Python
- We count the number of steps of execution
- Arithmetic operations take logarithmic time
- Array access takes linear time

Generally, complexity results are not expressed

Checking versus deciding

Let wang-rectangle-tiling be the following problem:

Input
- 2 integers, n and m
- a list of 4-tuples of integers representing Wang Tiles

Output 1 if it is possible to tile an $n \times m$ rectangle with these Wang Tiles, with only the color 0 on the border, 0 otherwise.

- wang-rectangle-tiling is decidable; there is a Turing Machine for solving it; it has exponential complexity.
- we do not know of a better algorithm for this problem, except in particular cases, but we are looking at the worst case.

Certificate

On the other hand, we can easily convince someone that an instance of wang-rectangle-tiling is positive: just show them the solution.

Definition

A *polynomial certificate scheme* for a problem *prob* is a problem *prob*′ together with a polynomial c such that:

- P' is computable in polynomial time
- $\forall x, prob(x) \iff \exists y \in \{0,1\}^{\leq s(x)}, prob'(x,y)$

Decision Problems

The way we have phrased wang-rectangle-tiling is a bit peculiar, but it let us define a polynomial certificate scheme for this problem. Otherwise, we would have needed to state which tiling we want to give on each input.

Definition

A *decision problem* is a function with codomain $\{0, 1\}$.

For the rest of this lecture, all problems are decision problems.

The class NP

Definition

The class NP is the set of decision problems with a polynomial certificate scheme.

This corresponds to the phenomenon we have identified with wang-rectangle-tiling, where a solution is easy to verify, but not necessarily easy to find

Examples of problems in NP

- 3-coloring of a graph

Examples of problems in NP

- 3-coloring of a graph
- Fitting furniture into a truck

Examples of problems in NP

- 3-coloring of a graph
- Fitting furniture into a truck
- Every problem in P is also in NP

The million-dollar question

Is every problem in NP also in P?

Reduction

We want a way to compare the difficulty of problems without knowing the best algorithm for them.

Definition

A problem prob is *harder by reduction* than prob' if there is a function f *computable in polynomial time* from instances of prob' to instances of prob such that prob(x) if and only if prob'(x)

Examples of reduction

- palindrome is harder by reduction than square: reverse the second half of the word. . .

Examples of reduction

- palindrome is harder by reduction than square: reverse the second half of the word. . .
- . . . and vice-versa

Examples of reduction

- palindrome is harder by reduction than square: reverse the second half of the word. . .
- . . . and vice-versa
- Any problem in P reduces to the trivial problem of checking if the input is 0.

NP-completeness

Which are the tougher nuts to crack?

Definition

A problem P is *NP-hard* if it is harder by reduction than any problem in NP. If P is also in NP, then it is said to be *NP-complete*.

An NP-complete problem

Theorem

There exists an NP-complete problem.

This means that by studying this problem, we can get knowledge on all NP problems.

Tiling is NP-complete

Theorem

wang-rectangle-tiling *is NP-complete*

The SAT problem

Definition

SAT is the following problem: given a logical formula with variables $x_1, \ldots, x_k$, is there an assignment of its variables which makes it true.

Theorem

SAT is NP-complete

Proof.

By reduction from wang-rectangle-tiling: the tiling constraints can be represented by a logical formula, with variables representing the possible positions of the tiles. □