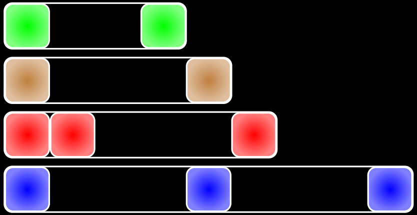
On Solving the Sparse Matrix Compression Problem Greedily

Dominik Köppl[1] Vincent Limouzy[2] Andrea Marino[3] Giulia Punzi[4]
Takeaki Uno[5]

[1]: University of Yamanashi
[2]: University Clermont Auvergne
[3]: University of Florence
[4]: University of Pisa
[5]: National Institute of Informatics

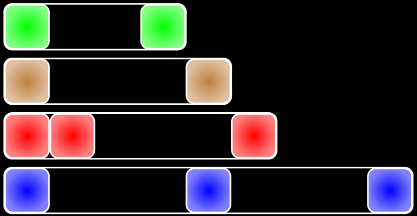


LSD & LAW for Costas

problem setting

given

- n 1-dimensional tiles
- a tile consists of blocks and gaps



task

- combine all n tiles to a single tile, called placement
- can fill gaps but blocks must not overlap
- goal: construct shortest placement

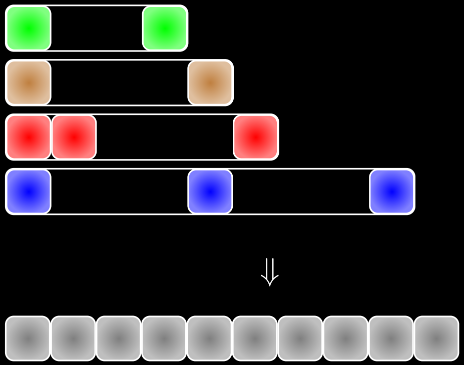
problem setting

Lemma

a computed placement with no gaps is a solution

Proof.

because blocks cannot overlap □



decision problems

MINLENGTH can you combine all tiles to a placement of length k?

MAXSHIFT if the first block of each tile is on the first column, can you form a placement with a maximum shift to the right of at most k?

turns out that MAXSHIFT has already been studied under the name Sparse Matrix Compression (SMC) problem

- Garey+'79 showed that SMC is $\mathcal{NP}$-hard for $k \geq 2$
- Bannai+'24 showed that both problems are $\mathcal{NP}$-hard even for widths in $\Omega(\lg n)$

Problem (SMC, [Garey+'79, Chapter A4.2, Problem SR13])

given:
- $n \times \ell$ *matrix* $A[1..n][1..\ell]$ *with* n *rows and* ℓ *columns and entries* $A[i][j] \in \{0,1\}$ *for all* $i \in [1..n]$, $j \in [1..\ell]$
- *integer* $k \in [0..\ell \cdot (n-1)]$

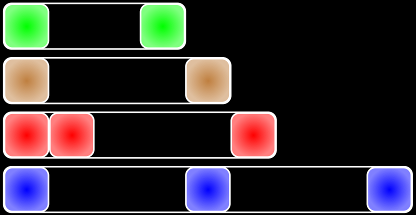
goal: check whether the following two can exist:

- *an integer array* $C[1..\ell + k]$ *with* $C[i] \in [0..n]$ *for every* $i \in [1..\ell + k]$, *and*
- *a shift function* $s : [1..n] \to [0..k]$ *such that* $A[i][j] = 1 \Leftrightarrow C[s(i) + j] = i \ \forall\ i \in [1..n],\ \forall\ j \in [1..\ell]$
- *assume* $A[0][j] = 0\ \forall\ j$ *to allow setting* $C[i] = 0$ *for some* i, *modelling that this entry is unassigned*

applications:

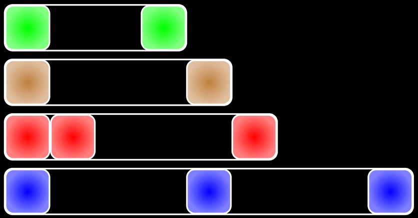
- matrix compression [Ziegler'77]
- search trie implementations [Tarjan,Yao'79]
- compilers [Aho+'86]
- Bloom filters [Chang,Wu'91]

from tiles to matrix



$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

from tiles to matrix



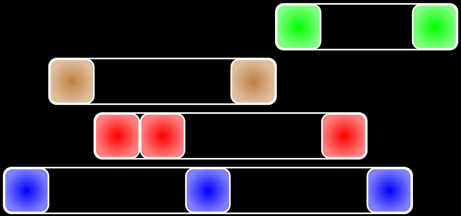
$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

solution:

- $s = [6, 1, 2, 0]$
- $C = [4, 2, 3, 3, 4, 2, 1, 3, 4, 1]$

$$B = \begin{pmatrix} & & & & & & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & & & & & \\ & & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & & & \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & & & & & & \end{pmatrix}$$

from tiles to matrix



$$A = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

solution:

- $s = [6, 1, 2, 0]$
- $C = [4, 2, 3, 3, 4, 2, 1, 3, 4, 1]$

$$B = \begin{pmatrix} & & & & & & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & & & & & \\ & & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & & & \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & & & & & & \end{pmatrix}$$

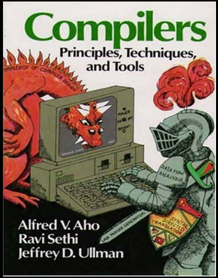
approximation algorithm

Ziegler'77: greedy algorithm: first fits first

- place first tile at first position
- for each subsequent tile: put it at the leftmost fitting position
- repeat

used in the classic textbook "Compilers: Principles, Techniques, and Tools", Section 3.9.8

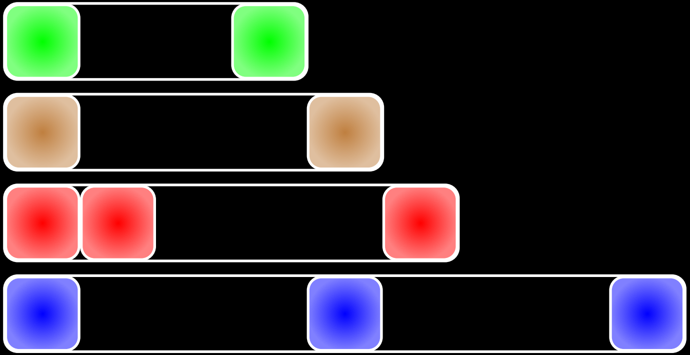
While we may not be able to choose *base* values so that no *next-check* entries remain unused, experience has shown that the simple strategy of assigning *base* values to states in turn, and assigning each *base*[s] value the lowest integer so that the special entries for state s are not previously occupied utilizes little more space than the minimum possible.



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

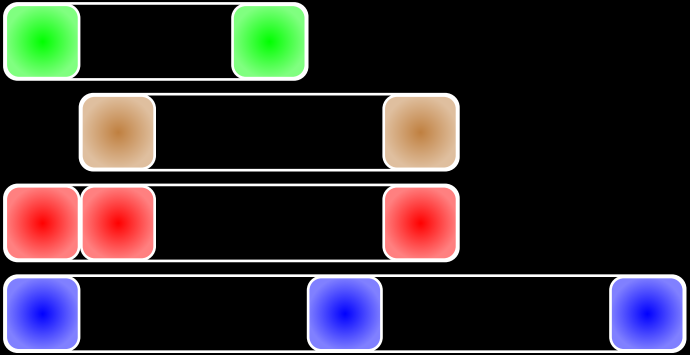
- place first tile at first position
- for each subsequent tile: put it at the leftmost fitting position
- repeat



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

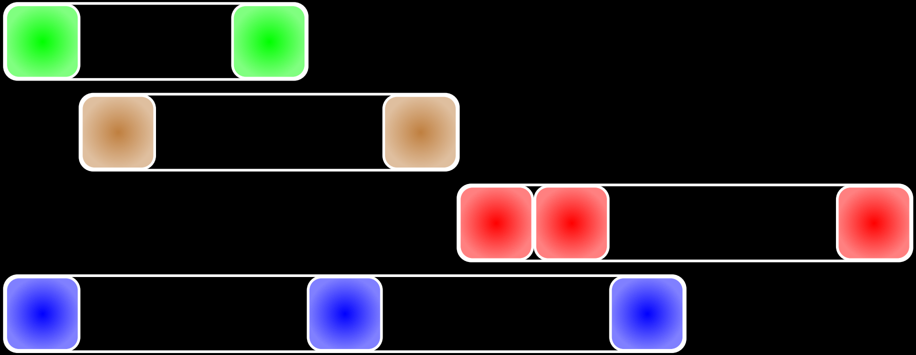
- place first tile at first position
- for each subsequent tile: put it at the leftmost fitting position
- repeat



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

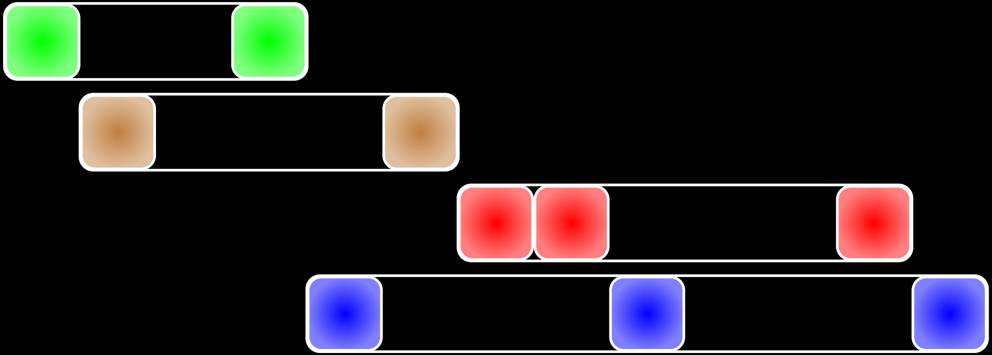
- place first tile at first position
- for each subsequent tile: put it at the leftmost fitting position
- repeat



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

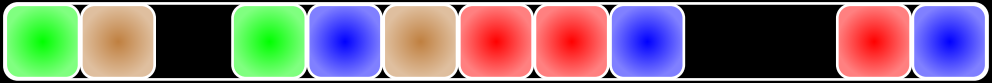
- place first tile at first position
- for each subsequent tile: put it at the leftmost fitting position
- repeat



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

- place first tile at first position
- for each subsequent tile: put it at the leftmost fitting position
- repeat



- approximation ratio really so small?
- answer: NO, in fact: $\Theta(\sqrt{m})$, where m is the optimal value!

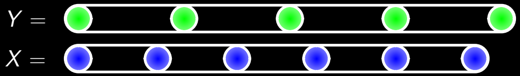
lower bound: $\Omega(\sqrt{m})$ approximation ratio

- two different tiles: X and Y, $X = (1 \cdot 0^{k-2})^k$, $Y = (1 \cdot 0^{k-1})^k$
- $\#X$ tiles: $k-2$, $\#Y$ tiles: $k-1$
- tiles are given in order $Y, X, Y, X, Y, \ldots$
- each placement adds length at least $k^2 - k$ to the solution, so total length is $\Omega(k^3)$
- contrarily all X and Y's can be combined within themselves to solid blocks of length $\Theta(k^2)$ (optimal value)

$\Rightarrow$ approximation ratio is $\sqrt{m}$

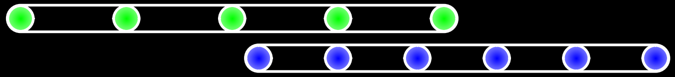
greedy algorithm

- start with Y and find first fitting place for X



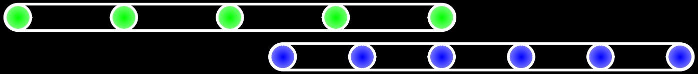
greedy algorithm

- start with Y and find first fitting place for X



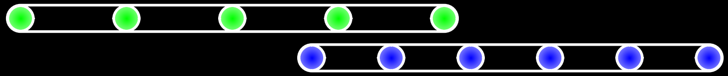
greedy algorithm

- start with Y and find first fitting place for X



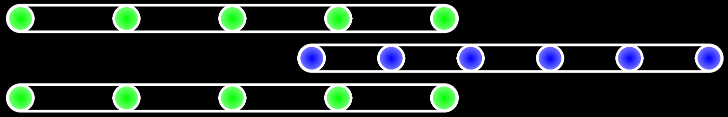
greedy algorithm

- start with Y and find first fitting place for X
- X fits visible at the last k entries of Y



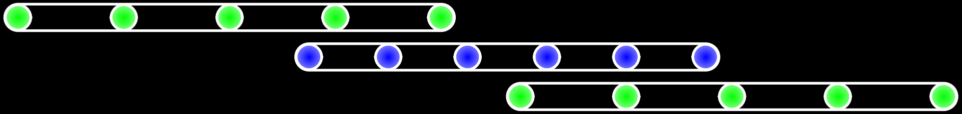
greedy algorithm

- start with Y and find first fitting place for X
- X fits visible at the last k entries of Y
- next Y conflicts with put Y and X



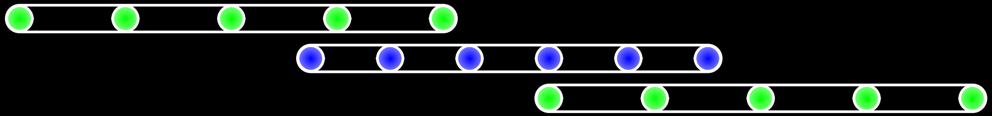
greedy algorithm

- start with Y and find first fitting place for X
- X fits visible at the last k entries of Y
- next Y conflicts with put Y and X



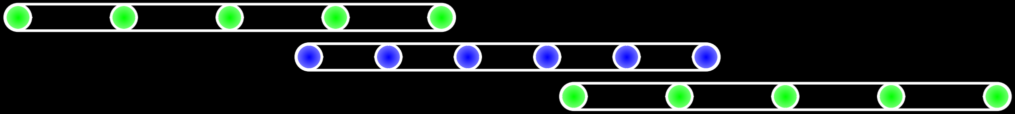
greedy algorithm

- start with Y and find first fitting place for X
- X fits visible at the last k entries of Y
- next Y conflicts with put Y and X



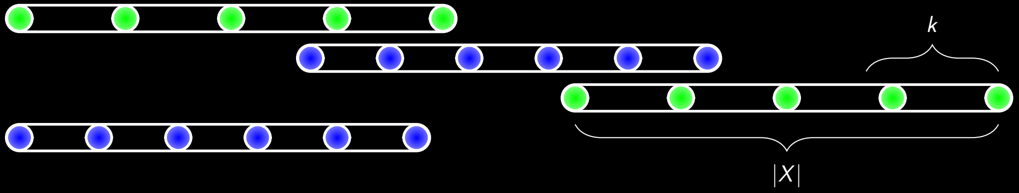
greedy algorithm

- start with Y and find first fitting place for X
- X fits visible at the last k entries of Y
- next Y conflicts with put Y and X
- fits only at the last k entries of X



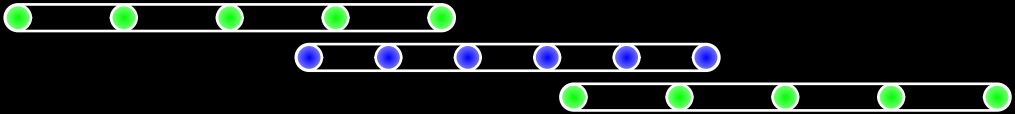
greedy algorithm

- start with Y and find first fitting place for X
- X fits visible at the last k entries of Y
- next Y conflicts with put Y and X
- fits only at the last k entries of X
- recurse
- placement enlarges by $|X| - k$ per put tile



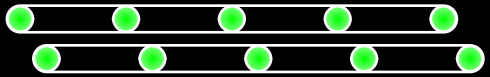
greedy algorithm: recap

- have tiles of types X and Y each $\Theta(k)$ times
- each tile has length $\Theta(k^2)$
- per tile: enlarge placement by at least $k^2 - k$
- total placement length: $\Omega(k^3)$
- what is a shortest placement?



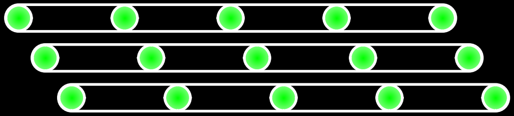
optimal solution

- first align all Y's



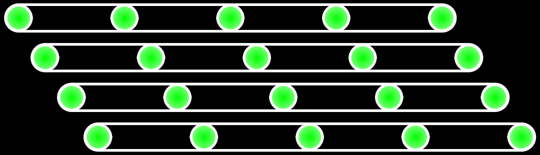
optimal solution

- first align all Y's



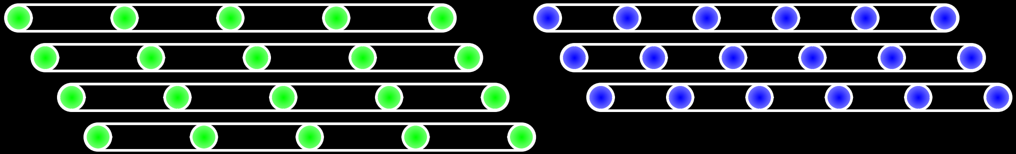
optimal solution

- first align all Y's
- all Y's fit perfectly



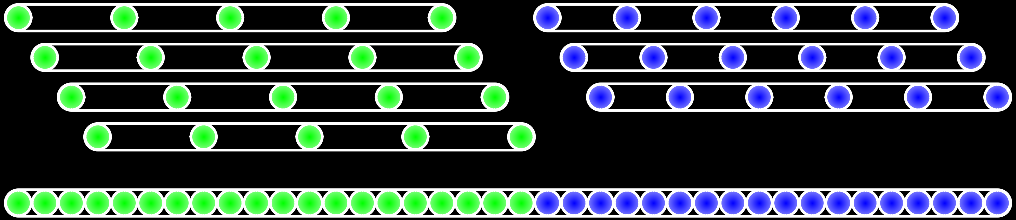
optimal solution

- first align all Y's
- all Y's fit perfectly
- same goes for all X



optimal solution

- first align all Y's
- all Y's fit perfectly
- same goes for all X
- solution is optimal since there are no gaps
- solution has length $(|X| + |Y|) + 2k \in \Theta(k^2)$



recap

- optimal solution length $m \in \Theta(k^2)$
- greedy algorithm solution length: $\Omega(k^3)$
- at least $\Omega(k)$ worse, where $k \in \sqrt{m}$!

we can also show:

- by pigeonhole principle, greedy cannot be worse that $\mathcal{O}(\sqrt{m})$

$\Rightarrow$ greedy has approximation ratio $\sqrt{m}$

- given an $n \times \ell$ matrix, we can solve both problems *exactly* in $\mathcal{O}(n^{2^\ell} \ell n 2^\ell n)$ time

$\Rightarrow$ For $\ell \in \mathcal{O}(\lg \lg n)$: problems are in $\mathcal{P}$

open problems

1. Lower bound of $\Omega(\sqrt{m})$ for any ordering?
2. Better approximation algorithms?
3. Is there an FPT algorithm parameterized by
   - number of tile types?
   - maximum number of blocks ('1') in a tile?
4. maximum length ℓ of tiles
   - $\Omega(\lg n) \Rightarrow \mathcal{NP}$-hard Bannai+'24
   - $\mathcal{O}(\lg \lg n) \Rightarrow \mathcal{P}$
   - $\omega(\lg \lg n) \cap o(\lg n) \Rightarrow$?