

On Solving the Sparse Matrix Compression Problem

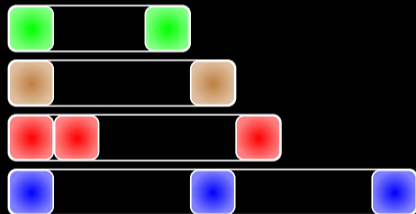
Hideo Bannai
Keisuke Goto
Jesper Jansson
Vincent Jugé
Shunsuke Kanda
Akitoshi Kawamura
Dominik Köppl
Vincent Limouzy
Andrea Marino
Giulia Punzi
Jannik Olbrich
Takeaki Uno



game



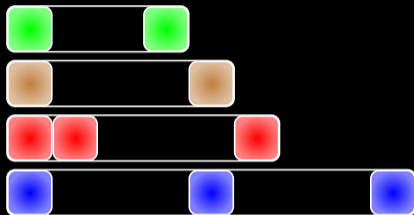
slides



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

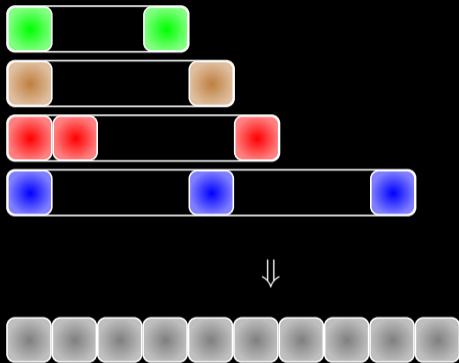
problem setting

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 ?

MINMAXSHIFT 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 **MINMAXSHIFT** has already been studied under the name **Sparse Matrix Compression (SMC) problem**

- ▀ Garey, Johnson '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 $\Omega(\lg n)$

If all tiles have the same length, then **MINLENGTH** = **MINMAXSHIFT**

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

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 $k \in [0..\ell \cdot (n - 1)]$

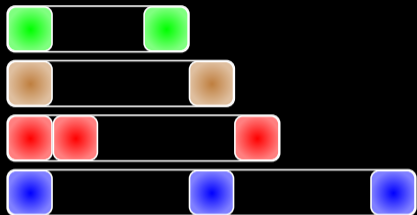
goal: check whether the following two can exist:

- an integer array $C[1..\ell + k]$ with $C[i] \in [0..n]$ for every $i \in [1..\ell + k]$, and
- a shift function $s : [1..n] \rightarrow [0..k]$ 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



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

constant-time access query

sparse matrix representation of A

- ▮ $s = [6, 1, 2, 0]$

- ▮ $C = [4, 2, 3, 3, 4, 2, 1, 3, 4, 1]$

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

- ▮ access query $A[r][c]$ (row r , column c)
- ▮ $s[r]$ is the shift of row r
- ▮ so $A[r][c] = 1$ if and only if $C[c + s[r]] = r$

Example

- ▮ access query $A[3][2]$ (3rd row, 2nd column)
- ▮ $s[3] = 2$, so 3rd row shifted by 2
- ▮ $C[2 + s[3]] = C[4] = 3$, so $A[3][2] = 1$

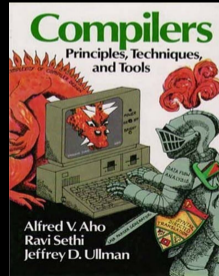
approximation algorithm

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

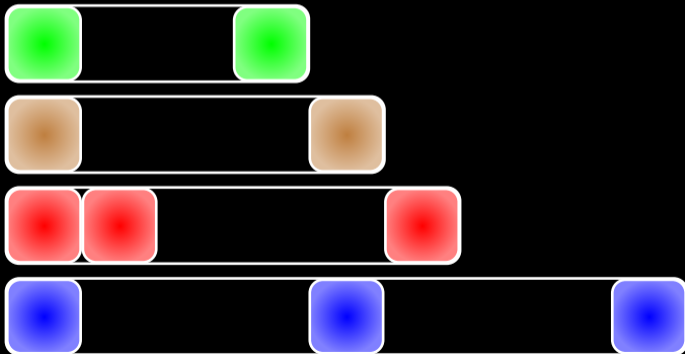
While we may not be able to choose *base* values so that no *next-check* entries remain unused, experience has shown that the simple strategy of assigning *base* values to states in turn, and assigning each $base[s]$ value the lowest integer so that the special entries for state s are not previously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



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

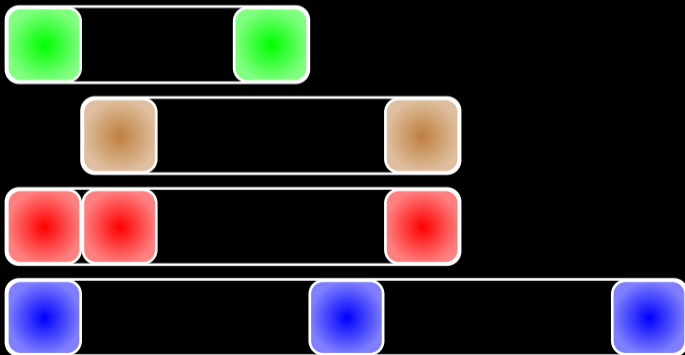
- ▮ place first tile at first position
- ▮ for each subsequent tile: put it at the leftmost fitting position
- ▮ repeat



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

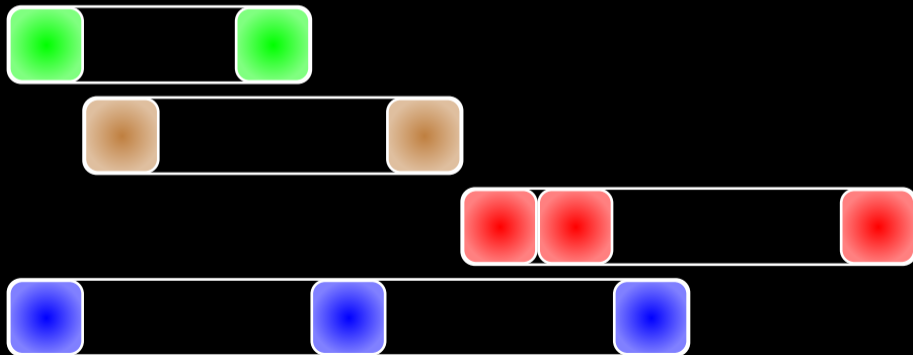
- ▣ place first tile at first position
- ▣ for each subsequent tile: put it at the leftmost fitting position
- ▣ repeat



approximation algorithm

Ziegler'77: greedy algorithm: first fits first

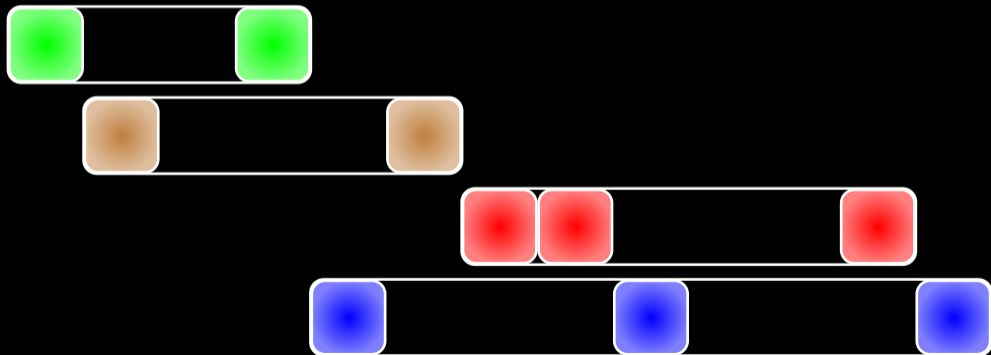
- ▮ place first tile at first position
- ▮ for each subsequent tile: put it at the leftmost fitting position
- ▮ repeat



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! Köppl+'25

Does it matter?

heuristic allows pre-sorting tiles

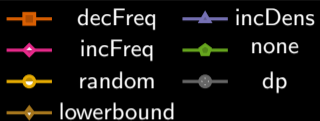
- ▀ none: no sorting
- ▀ incFreq: sort by increasing number of ones
- ▀ decFreq: sort by decreasing number of ones
- ▀ incDens: sort by increasing density $\frac{\text{number of ones}}{\text{length}}$
- ▀ decDens: sort by decreasing density
- ▀ random: 10 random shuffling, taking the shortest result

one bad input to rule them all

- ▀ only two tile types X and Y
- ▀ integer parameters c and g
- ▀ $X = (10^g)^c 1$, $Y = (10^{g-1})^c 10^c 1$
- ▀ $|X| = |Y| = c(g+1) + 1$
- ▀ $\#_1 X = \#_1 Y = c + 1$
- ▀ Request n tiles in alternating order X, Y, X, Y, \dots

$$\begin{array}{l} X = \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \\ Y = \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \\ (c = 5, g = 4) \end{array}$$

bad instance 1/2



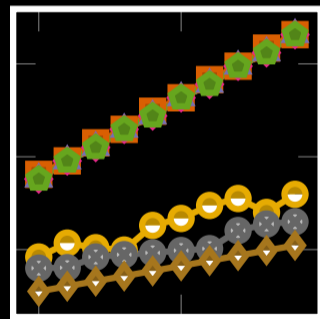
- lowerbound: total number of ones
- dp: optimal solution

why is random good?

- number of tile types to pick is too small
- best solution for this instance is to sort in order X, \dots, X, Y, \dots, Y

placement length

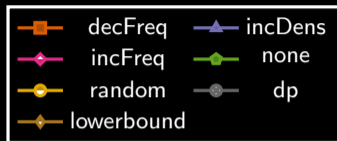
$$g = 3, c = 10$$



10 15

$n = \# \text{ rows}$

bad instance 2/2



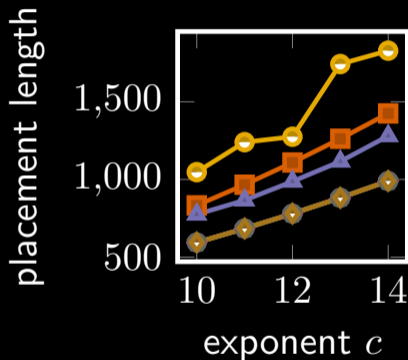
new tile type:

- ▮ $Z = (10^{g-2})^c 10^{2c} 1$
- ▮ order : X, Y, Z, X, Y, Z, \dots
- ▮ add some helper tiles to fill up all holes
 $\Rightarrow dp = \text{lowerbound}$

random performs worst!

hence: exact algorithms make sense to study

3 tile kinds, $g = 8, n = 48$



bird's-eye view

known results and open problems

Parameterized results for MINLENGTH and MINMAXSHIFT

	\mathcal{P} -time for	\mathcal{NP} -hardness even for
ℓ	$\ell \in O(\lg \lg n)$ or [Juge+'26] $\ell \in O(\lg n) \wedge \kappa \in O(\lg n)$	$\ell \in \Omega(\lg n)$ Bannai+'24
ζ	$\zeta = 1$ (trivial)	$\zeta = 2$ [Juge+'26] † $\zeta = 4$ [Juge+'26] ¶
κ	$\kappa \in O(\lg n) \wedge \ell \in O(\lg n)$	$\kappa = 1$ [Juge+'26]
m	$m = \ell$ (trivial) †	$m \leq \ell + 2$ Even+'77
ρ	$\rho = 0$ (trivial) ¶	$\rho \leq 2$ Even+'77 ¶

▀ †: concerns only MINLENGTH

▀ ¶: concerns only MINMAXSHIFT

▀ ℓ : maximum length of a tile

▀ ζ : maximum number of numerals per tile

▀ κ : number of tile types

▀ m : optimal placement size

▀ ρ : maximum shift

from where to reduce? (negative side)

general hardness

- ▼ Even+'77: reduction from 3-coloring to MINMAXSHIFT [easy reduction]
- ▼ Bannai+'24: reduction from Hamilton path to MINMAXSHIFT and MINLENGTH, mimicking reduction to shortest superstring Gallant+'80 [very technical reduction]
- ▼ Kyoto-Group'25: reduction from 3-partition [unpublished, somewhat technical]

hard even for special input

- ▼ Juge+'26: reduction from clique [technical] (\rightarrow even one tile type is hard)
- ▼ reduction from multi Skolem words Nordh'07 [very easy] (\rightarrow one arbitrary tile and even having at most two 1s in a row is hard)
- ▼ Juge+'26: reduction from coupled scheduling problem [very easy] (\rightarrow even having at most two 1s in a row is hard)

algorithmic aspects (positive side)

- ▶ easy only for very special input (e.g., constant length)
- ▶ otherwise: dynamic programming

dynamic programming

- ▶ one tile type: $O(n^2 2^\ell \ell \log \ell)$,
 - ℓ : length of tile
 - fixed-parameter tractable in ℓ !
- ▶ general case: $O(n^\kappa \ell n 2^\ell n) \subset O(n^{2^\ell} \ell n 2^\ell n)$ time
- ▶ κ : number of tile types

DP for one tile type

naïve approach

- ▶ track order explicitly
- ▶ gives $\frac{n!}{c_1!c_2!\dots c_\kappa!}$ possibilities, where
 - c_i is the number of tiles of i -th tile kind
 - exponential in n even for small κ .

solution for one tile type $\kappa = 1$

- ▶ track number of taken tiles
- ▶ consider only $\Theta(\ell)$ length suffix of all possible solutions

DP table: $D[i][S]$

- ▶ i : how many tiles have been used?
- ▶ S : ℓ -length suffix
- ▶ value: shortest placement length with suffix S using i tiles

DP: general case

general solution: use Parikh vectors

- ▮ track count of each taken tile type with vector \vec{p}
- ▮ $\vec{p} = (p_1, p_2, \dots, p_\kappa)$ and $p_i \in [0..c_i]$
- ▮ do not track the order!

DP table: $D[\vec{p}][S]$

- ▮ \vec{p} : Parikh vector
(counts of each taken tile type)
- ▮ S : ℓ -length suffix
- ▮ value: shortest placement length

Inapproximability

- ▮ MINMAXSHIFT: there is a L-reduction from k -coloring, so there is no \mathcal{P} -time approximation with ratio $n^{1-\epsilon}$ for some $\epsilon > 0$ unless $\mathcal{NP} = \mathcal{P}$
- ▮ MINLENGTH: meta-proof via reduction from any NP-hard problem by adding a gadget: $2 - \epsilon$ approximation not possible unless $\mathcal{NP} = \mathcal{P}$

There is a big gap between both results.

recall

Ziegler's approximation algorithm has an approximation of

- ▮ $\Theta(\sqrt{m})$ for MINLENGTH, where m is the length of the optimal output
- ▮ $\Theta(\sqrt{\rho})$ for MINMAXSHIFT, where ρ is the allowed maximal shift of the optimal output

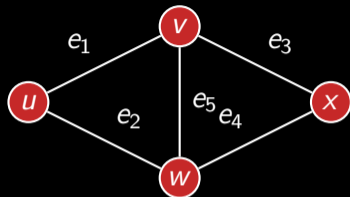
Problem

What is the best approximation we can achieve for MINLENGTH?

NP hardness

reduction from 3-coloring

3-COLORING to MINMAXSHIFT



- construct matrix A with n rows and $3m$ columns.
- assign each group of three columns to an edge, and each row to a vertex
- goal: solve MINMAXSHIFT on A !

$$A = \begin{array}{c|ccccc} & \underline{e_1} & \underline{e_2} & \underline{e_3} & \underline{e_4} & \underline{e_5} \\ \hline u & 100 & 100 & 000 & 000 & 000 \\ v & 100 & 000 & 100 & 000 & 0100 \\ w & 000 & 100 & 000 & 0100 & 100 \\ x & 000 & 000 & 100 & 0100 & 000 \end{array}$$

exact algorithm

dynamic programming on suffixes

idea for one tile type

dynamic programming

- ▶ use match with wildcards to find all possible combinations
- ▶ parameters :
 - number of used tiles i
 - ℓ -length suffix S of placement
- ▶ build DP table $D[i][S]$ incrementally

DP Table $D[i][S]$

- ▶ i : number of tiles used
- ▶ S : ℓ -length suffix
- ▶ value : shortest placement length

partial word matching

Definition (partial word)

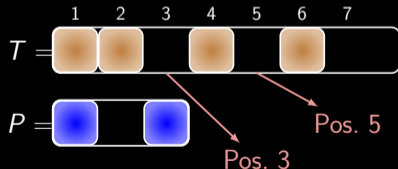
a string containing wildcard symbol 0 that matches any character

Lemma (P. Clifford, R. Clifford'07)

- given: text T and pattern P , both partial words
- task: find all text positions where pattern matches
- possible in $O(n \lg m)$ time

- give each tile a different color (= character)

- extend T with wildcards



merging operation

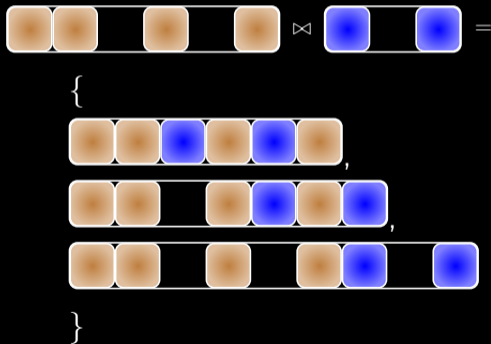
Definition

for two tiles A and B , merge $A \bowtie B$ produces all valid overlaps

valid merge requires:

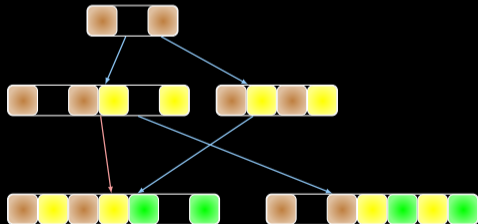
- no position where both have non-wildcard symbols
- combine symbols at each position

computable in $O(\ell \log \ell)$ time with
P. Clifford, R. Clifford'07 and
 $|A|, |B| \in O(\ell)$



DP algorithm: one distinct tile

- ▮ setting: all n tiles are identical, length $\ell = O(\lg n)$
- ▮ example: n tiles of type 101



$i \backslash S$	000	001	...	101	110	111
0	0	—	—	—	—	—
1	—	—	—	3	—	—
2	—	—	—	6	—	4
3	—	—	—	7	—	7
⋮						

algorithm complexity

Theorem

MINLENGTH with one distinct tile of length ℓ is solvable in $O(n^2 2^\ell \ell \log \ell)$ time.

analysis

- DP table size: $O(n \cdot 2^\ell)$ entries (for $i = [0..n]$ and $S \in \{0, 1\}^\ell$)
- for each entry: compute merges $A \bowtie B$ in $O(\ell \log \ell)$ time
- total time : $O(n \cdot 2^\ell \cdot \ell \log \ell)$

fixed-parameter tractable in ℓ and polynomial when $\ell = O(\lg n)$:
 $O(n \cdot 2^{O(\lg n)} \cdot O(\lg n) \cdot O(\lg \lg n)) = O(n \cdot n^{O(1)} \cdot \text{poly}(\lg n))$

general case: multiple tile types

input:

- ▀ κ different tile types T_1, \dots, T_κ
- ▀ c_i : count of tile type $i \in [1..\kappa]$, $\sum_{i=1}^{\kappa} c_i = n$

challenge: Order of tiles matters when they differ

naïve approach

- ▀ track order explicitly
- ▀ $\frac{n!}{c_1!c_2!\dots c_\kappa!}$ orderings
- ▀ exponential even for small κ

solution: Parikh vectors

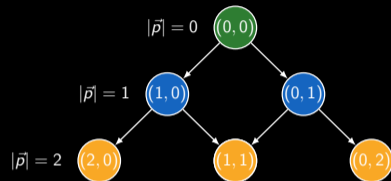
- ▀ track count of each taken tile type with vector \vec{p}
- ▀ $\vec{p} = (p_1, p_2, \dots, p_\kappa)$ and $p_i \in [0..c_i]$
- ▀ do not track the order!

Parikh vector DP

modified DP table: $D[\vec{p}][S]$

- ▀ \vec{p} : Parikh vector
(counts of each taken tile type)
- ▀ S : ℓ -length suffix
- ▀ value: shortest placement length

Parikh vector growth for 2 types



algorithm

1. start: $D[\vec{p}_0][0] = 0$ where $\vec{p}_0 = (0, 0, \dots, 0)$
2. for each \vec{p} with $|\vec{p}|$ from 0 to $n - 1$ and suffix S in D :
 - for each tile type T_i with $p_i < c_i$:
 - compute new Parikh vector $\vec{p}' = \vec{p}$ incremented at position i
 - for each merge $Z \in S \bowtie T_i$ update $D[\vec{p}'][Z]$
3. return minimum in $D[\vec{p}_n][\cdot]$ where $\vec{p}_n = (c_1, \dots, c_\kappa)$

Complexity: General Case

Theorem

MINLENGTH with n tiles of length ℓ is solvable in $O(n^{2^\ell} \ell n^{2^\ell} n)$ time.

analysis

- ▶ number of tile types $\kappa \leq 2^\ell$
- ▶ number of Parikh vectors $\leq n^\kappa \leq n^{2^\ell}$
- ▶ for each Parikh vector: $O(2^\ell)$ suffixes
- ▶ per suffix: $O(\ell \log \ell)$ merge time

polynomial time

1. for $\ell = O(\lg \lg n)$: $O(n^{2^{O(\lg \lg n)}} \cdot \text{poly}(\lg n)) = O(n^{(\lg n)^{O(1)}} \cdot \text{poly}(\lg n))$
2. for $\ell = O(\lg n)$ and $\kappa = O(\lg n)$ distinct tiles: $O(n^{O(\lg n)} \cdot \text{poly}(n)) = \text{poly}(n)$

implementation details

space optimization

- do not store entire DP matrix
- use hash tables H_i for Parikh vectors with $|\vec{p}| = i$
- only keep two consecutive levels in memory
- space: $O(2^\ell)$ instead of $O(n \cdot 2^\ell)$ or $O(n^{2^\ell} \cdot 2^\ell)$

hash table structure

- key: (\vec{p}, S)
- value: shortest placement length
- compute H_{i+1} from H_i
- discard H_{i-1}

memory at step i :

H_{i-1} discarded

hash table H_i

compute

hash table H_{i+1}

only 2 levels in memory

Parameterized results for MINLENGTH and MINMAXSHIFT

	\mathcal{P} -time for	\mathcal{NP} -hardness even for
ℓ	$\ell \in O(\lg \lg n)$ or [Juge+'26] $\ell \in O(\lg n) \wedge \kappa \in O(\lg n)$	$\ell \in \Omega(\lg n)$ Bannai+'24
ζ	$\zeta = 1$ (trivial)	$\zeta = 2$ [Juge+'26] † $\zeta = 4$ [Juge+'26] ¶
κ	$\kappa \in O(\lg n) \wedge \ell \in O(\lg n)$	$\kappa = 1$ [Juge+'26]
m	$m = \ell$ (trivial) †	$m \leq \ell + 2$ Even+'77
ρ	$\rho = 0$ (trivial) ¶	$\rho \leq 2$ Even+'77 ¶

▀ †: concerns only MINLENGTH

▀ ¶: concerns only MINMAXSHIFT

▀ ℓ : maximum length of a tile

▀ ζ : maximum number of numerals per tile

▀ κ : number of tile types

▀ m : optimal placement size

▀ ρ : maximum shift