

Finding Maximum and Minimum Size Matrices: The Algorithmic Complexity of Coding Challenges

A. Abdelmonsef
Swarthmore College

X. Dong
University of Pennsylvania

D. Průša
Czech Technical
University

M. Wehar
Bryn Mawr College

C. Xu
University at Buffalo

Fun with Algorithms (FUN 2026)

- 1 Introduction & Prior Works
- 2 Positive Border Problems
- 3 Bounded Sum Problems
- 4 Conclusion

Background

- We focus on combinatorial pattern matching in two dimensions.
- **Input:** A matrix over a fixed alphabet (sometimes called a picture).
- **Output:** A rectangle within the input matrix (sometimes called a subblock or subpicture) that matches the pattern.
- We are specifically interested in largest and smallest subblocks in terms of area or perimeter.
- The computational complexity of such a problem appears to fluctuate depending on the specified pattern.

Prior Works: Classic Problems

Consider LeetCode “85. Maximal Rectangle” and “221. Maximal Square”:

- Find a largest rectangle within a Boolean matrix \mathbf{M} that contains only 1 entries.
- These are classic problems that appear in algorithms textbooks.
- **Claim:** If \mathbf{M} has m rows and n cols, then it's solvable in $O(mn)$ time.
- *Hint:* First solve LeetCode “84: Largest Rectangle in Histogram”.
- Square variant can be solved with a similar approach.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We previously generalized this algorithm in TCS 2021 to find a max / min area match for any border-unaware local picture language. E.g. checkerboard pattern with 0's and 1's.

Prior Works: The Four Corners Problem

Consider GeeksforGeeks “Submatrix with corners as 1”:

- Find a largest / smallest rectangle within a Boolean matrix \mathbf{M} whose four corners are 1's.
- In other words, the upper-left, upper-right, lower-left, and lower-right corners are all 1's.
- We use area to measure size and matches must satisfy $m, n \geq 2$.

1	0	1	0	1	1
0	1	0	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
0	1	0	0	0	0
0	0	0	1	0	0

Theorem: We previously showed in DLT 2020 that the max-variant is solvable in $O(mn \cdot (\min\{m, n\})^{0.5302})$ time by a reduction to minimum witness for Boolean matrix multiplication. Also, the min-variant is solvable in $\tilde{O}(m \cdot n)$ time by applying an approximation algorithm repeatedly.

- 1 Introduction & Prior Works
- 2 Positive Border Problems
- 3 Bounded Sum Problems
- 4 Conclusion

Max / Min Rectangles with Positive Border

In this work, we consider border patterns, instead of our previous corner and interior patterns. Consider problems “Fort Moo” from USACO 2016 and “Max Sub-Rectangle” from KPI-OPEN 2017:

- Find a largest / smallest rectangle within a Boolean matrix \mathbf{M} whose border entries are all 1's.
- The USACO problem uses area and the KPI-OPEN problem uses perimeter to measure size.
- Matches must satisfy $m, n \geq 2$.

1	0	0	0	1	1	1
0	1	0	0	1	0	1
1	1	1	1	1	1	1
1	0	0	1	0	0	0
1	1	1	1	0	0	1
1	0	0	1	1	1	1
1	1	1	1	0	0	1

The published solutions online for these problems have cubic runtime. We present an $O(m \cdot n \cdot \log(\min\{m, n\}))$ time divide and conquer solution.

(Solution) Max / Min Rectangles with Positive Border

- Consider a Boolean matrix M and now split it in half horizontally.
- Either the largest rectangle is within the top half, the bottom half, or spans across the split between the top and bottom halves.
- If the largest rectangle lives fully within the top or bottom half, then we further perform a vertical split on the top and bottom halves.
- If $t(n)$ represents the time it takes to find a largest rectangle in an n by n matrix and supposing that we can handle the “split case” in $O(n^2)$ time, then we have the recurrence relation $t(n) = 4 \cdot t(\frac{n}{2}) + O(n^2)$.
- Solving this recurrence leads to $t(n) = O(n^2 \log(n))$. Although we oversimplified things by making the matrices square, with a bit of work, we can similarly get $O(m \cdot n \cdot \log(\min\{m, n\}))$ time for rectangular matrices.

What is a Staple?

A staple in top half:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

A flipped staple in bottom half:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The Split Case

- It remains to show how the split case works.
- For the top half, we compute a “map” m_{top} that assigns to each pair of column indexes (c_1, c_2) the smallest row index i such that there is a “staple” of 1’s with left at column c_1 , right at column c_2 , and top at row i .
- For the bottom half, we compute m_{bottom} that assigns to each pair of column indexes (c_1, c_2) the largest row index i such that there is a flipped staple of 1’s with left at column c_1 , right at column c_2 , and bottom at row i .
- Once we have computed both maps, we can simply consider each pair of columns with the top row from m_{top} and the bottom row from m_{bottom} .
- Whichever resulting rectangle is largest / smallest, will be the largest / smallest that spans across the split.

Map Construction

- We originally used Binary Heaps to construct the maps in $O(n^2 \log(n))$ time. We could improve to $O(n^2 \log \log(n))$ using Van Emde Boas Trees.
- An anonymous reviewer suggested that we should use a special case of union-find where the unions and finds are predetermined. This enabled us to achieve map construction in $O(n^2)$ time.
- Map construction works as follows. WLOG let's focus on m_{top} . Consider any right column c_2 , we will compute what (c_1, c_2) maps to for all possible $c_1 < c_2$ in $O(n)$ time where some helpful precomputation of the matrix has already been done.
- Consider row indexes where there is a vertical sequence of 1's in column c_2 down to the bottom. We put each of these row indexes into a bucket based on how far to the left there is a sequence of 1's starting from column c_2 .
- We go through the buckets from largest length to smallest. With each bucket, we are at a column c_1 . We compute the smallest row index i such that there is a vertical sequence of 1's down to the bottom. Finally, we query the smallest row index we have seen so far that is greater than or equal to i .

All Rectangles with Positive Border and Empty Interior

We implemented a solution to an easier problem in JavaScript (see our code on [GitHub](#)):

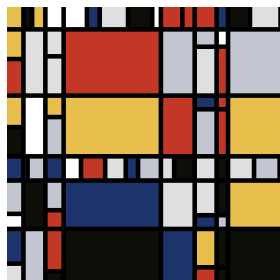
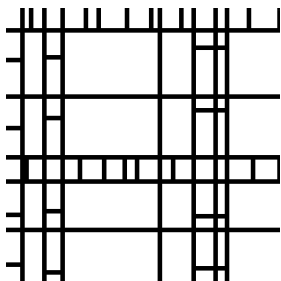
- Find all rectangles within a Boolean matrix \mathbf{M} whose border entries are 1's and interior entries are 0's.
- Our solution runs in $O(m \cdot n)$ time using $O(\min\{m, n\})$ space.
- We further applied our algorithm to a computer art project. :)

1	0	0	0	1	1	1
0	1	0	0	1	0	1
1	1	1	1	1	1	1
1	0	0	1	0	0	0
1	1	1	1	0	0	1
1	0	0	1	1	1	1
1	1	1	1	0	0	1

Remark: For many patterns, finding all matches would take more than $O(m \cdot n)$ time because there are $O(m^2 \cdot n^2)$ possible subblocks. However, for this pattern there cannot be more than $O(m \cdot n)$ matches.

FUN with Computer Art!

It was a bit unnecessary, but we used our algorithm to find and fill rectangles in images! The resulting image (right) resembles works by artist Piet Mondrian.



Ultimately, it was a useful way to validate our algorithm, but we suggest that more traditional flood fill algorithms could have been used instead.

- 1 Introduction & Prior Works
- 2 Positive Border Problems
- 3 Bounded Sum Problems
- 4 Conclusion

Max / Min Squares with Bounded Sums

Consider “Maximum size of square such that all submatrices of that size have sum less than k ” from GeeksforGeeks:

- Given a value k , find the largest side length ℓ s.t. all square ℓ by ℓ subblocks have sum at most k .
- We no longer just consider individual matches and the entries can be any **non-negative** integers.
- The online solution runs in $O(m \cdot n \cdot \log(m + n))$ time. We improve this to $O(m \cdot n)$ time.

$$\begin{bmatrix} 3 & 0 & 2 & 1 & 3 \\ 1 & 3 & 2 & 4 & 1 \\ 2 & 1 & 3 & 2 & 0 \\ 1 & 1 & 3 & 1 & 3 \\ 2 & 3 & 0 & 4 & 1 \end{bmatrix}$$

Example: If $k = 20$, then the largest side length value we get is $\ell = 3$.

(Solution) Max / Min Squares with Bounded Sums

- We apply a range minimum query approach: we preprocess so that for any subblock with upper left corner (i_1, j_1) and bottom right corner (i_2, j_2) , we can compute the sum across all entries, denoted by $s(i_1, j_1, i_2, j_2)$, in $O(1)$ time.
- We precompute and store all of the $s(\cdot, \cdot, m, n)$ values so that $s(i_1, j_1, i_2, j_2) = s(i_1, j_1, m, n) - s(i_2 + 1, j_1, m, n) - s(i_1, j_2 + 1, m, n) + s(i_2 + 1, j_2 + 1, m, n)$.
- Now, we start with $\ell = \min\{m, n\}$. We scan across the matrix considering every possible entry as an upper left corner. If the ℓ by ℓ sum is at most k , then continue to the next entry. Otherwise, decrement ℓ and try again at the current entry.
- Since the entries are non-negative, this solution takes $O(m \cdot n)$ time because we don't need to backtrack when decrementing ℓ .
- **Question:** How efficiently can we solve this problem when entries are allowed to be both positive and negative? Can we achieve $O(m \cdot n)$ time?

- 1 Introduction & Prior Works
- 2 Positive Border Problems
- 3 Bounded Sum Problems
- 4 Conclusion

Conclusion

- **Results:** We presented $O(m \cdot n \cdot \log(\min\{m, n\}))$ time algorithms for the *Positive Border Problems*. We also presented an $O(m \cdot n)$ time algorithm for a *Bounded Sums Problem*.
- **Insights:** Valuable algorithmic improvements can be made by studying coding challenge problems. There are many more patterns that we are still working on. We also would like to consider problems for counting all matches.
- **Invitation:** If anyone would like to join me in working on a few recreational coding challenge problems this week, please let me know!

Thank You!

Questions or Comments?