Maximum Subarray Problem in 1D and 2D via Weighted Paths in Directed Acyclic Graphs

Yann Barsamian

Maximum Subarray Problem in 1D and 2D via Weighted Paths in Directed Acyclic Graphs

Yann Barsamian[1]

Université de Strasbourg, 300 boulevard Sébastien Brant,
CS 10413, F-67412 Illkirch Cedex, France
ybarsamian@unistra.fr

Abstract. The Maximum Subarray Problem was encountered by Ulf Grenander in [8] for maximum likelihood estimation in pattern analysis. We are given a vector (or matrix) of numbers, and we have to find the contiguous sub-vector (or sub-matrix) which has the maximum sum of numbers in it. Apart from the original application, the problem also arises for example in biological sequence analysis ([9]).

We present here a linear-time algorithm in one dimension which is different from the one known due to Kadane ([4]), and present a way of extending it to two dimensions. To achieve the latter, we provide a new technique, the red-blue graphs, which encodes all the contiguous sub-matrices of an $m \times n$ matrix in size $\mathcal{O}(m \times n)$.

1 Introduction

The Maximum Subarray Problem (MSP) has been studied since 1977 (see for example [2, Chapter 1] for a review). Grenander devised an $\mathcal{O}(n^3)$ algorithm to solve the MSP in one dimension (1D). In [4], Jon Bentley presented two $\mathcal{O}(n^2)$ algorithms, an $\mathcal{O}(n \log n)$ one and an optimal $\mathcal{O}(n)$ algorithm due to Jay Kadane.

Here, we present an alternative $\mathcal{O}(n)$ algorithm for the MSP 1D. This algorithm is a reduction to a single-source maximum weighted path problem in a Directed Acyclic Graph (DAG). In [5, Footnote 1], the authors cite a personal communication which seems to imply this reduction, but to the best of our knowledge, this – simple but useful – reduction has not been published yet. We believe that this technique should be available in a written context.

Following Jon Bentley's orders (*Readers who feel that the linear-time algorithm for the one-dimensional problem is "obvious" are therefore urged to find an "obvious" algorithm for [the two-dimensional problem] !*), we have designed a similar technique for the MSP in two dimensions (2D). To scan all the contiguous sub-matrices of a given matrix, we introduce a new type of graphs : the red-blue graphs. With this data structure, we could nevertheless only come up with a cubic algorithm, as is the case when we extend Kadane's algorithm in two dimensions. Algorithms exist with better complexity (Takaoka, in [10], reduced this problem to $(\min, +)$ Matrix Multiplication for which we have a

sub-cubic-time algorithm – there is an $\mathcal{O}(n^3 \log^3 \log n / \log^2 n)$ algorithm for All Pairs Shortest Path by Chan in [6] which shares the same asymptotical time complexity, cf. [1, pp. 211–212] – leading to an $\mathcal{O}(m^2 n \log^3 \log m / \log^2 m)$ algorithm for the MSP 2D), but we hope that this new data structure will lead to an algorithm for finding maximum weighted paths with less complexity in the future.

The main contributions of this paper are the presentations of :

1. a reduction of the MSP 1D to a graph problem with optimal complexity
2. a new data structure to scan all the contiguous sub-matrices of a given matrix

The remainder of this paper is organized as follows : Section 2 presents the problem in 1D and 2D. Section 3 presents a reduction from the MSP 1D to a single-source maximum weighted path problem in a DAG. Section 4 presents a reduction from the MSP 2D to a single-source maximum weighted path problem in a new type of graphs.

2 Presentation of the Problem

2.1 MSP 1D

In one dimension, we are given a vector of n numbers, and we have to find the contiguous sub-vector which has the maximum sum of numbers in it. If the vector only has non-negative (≥ 0) numbers in it, a trivial solution is the vector itself, and the maximum sum is the total sum. Alternatively if the vector only has non-positive (≤ 0) numbers, a trivial solution is the empty sub-vector, and the maximum sum is 0. So an interesting problem only arises when we look at a vector which contains both positive (> 0) and negative (< 0) numbers. Let us take the example from [4], and look at the vector :

$$V = (\begin{array}{cccccccc} 31 & -41 & 59 & 26 & -53 & 58 & 97 & -93 \end{array} \; -23 \;\; 84)$$

75 is the sum of elements in $V[0..3] = (31\ -41\ 59\ 26)$, -19 is the sum of elements in $V[6..8] = (97\ -93\ -23)$, etc. The maximum sum here is 187, and lies inside $V[2..6] = (59\ 26\ -53\ 58\ 97)$.

2.2 MSP 2D

In two dimensions, we are given a matrix of $m \times n$ numbers, and we have to find the contiguous sub-matrix which has the maximum sum of numbers in it. The previous considerations about positive and negative numbers of course apply. Let us take the example from [2, Chapter 1], and look at the matrix :

$$M = \begin{pmatrix} -1 & 2 & -3 & 5 & -4 & -8 & 3 & -3 \\ 2 & -4 & -6 & -8 & 2 & -5 & 4 & 1 \\ 3 & -2 & 9 & -9 & -1 & 10 & -5 & 2 \\ 1 & -3 & 5 & -7 & 8 & -2 & 2 & -6 \end{pmatrix}$$

6 is the sum of elements in $M[1..3][0..0] = \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$, -14 is the sum of elements in $M[0..1][1..3] = \begin{pmatrix} 2 & -3 & 5 \\ -4 & -6 & -8 \end{pmatrix}$, etc. The maximum sum here is 15, and lies inside $M[2..3][4..5] = \begin{pmatrix} -1 & 10 \\ 8 & -2 \end{pmatrix}$.

3 Reduction of the MSP 1D to Maximum Weighted Path in Directed Acyclic Graphs

3.1 An Example

Let us look at the vector $W = \big(31\ {-41}\ 59\ 26\ {-53}\ 58\ 97\big)$. We put the values of this vector as weights of arcs in a graph. Because we chain the arcs in the same order as the values in W, the sum of a contiguous sub-vector in W will be the weight of the corresponding path in the graph, as shown in Fig. 1.

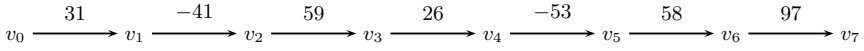

Fig. 1.

To simplify matters, we add a source vertex s and a sink vertex p, so that we only have to care about the length of paths between s and p. A contiguous sub-vector of W is characterized by the location of the first and the last value in W. From s we simply add 0-weighted arcs to the beginning of each of these arcs, and to p we add 0-weighted arcs from the end of each of those arcs. In the end, we have the graph $\mathcal{G}$ shown in Fig. 2.

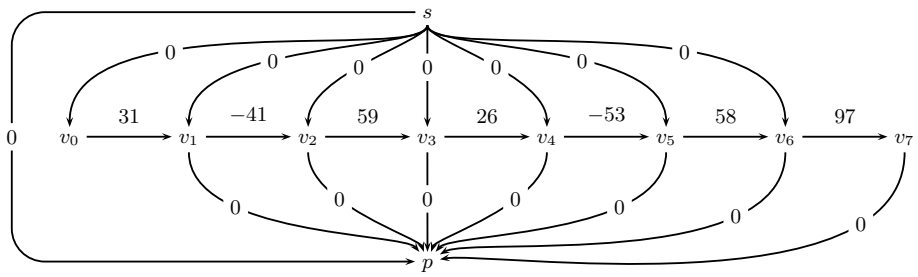

Fig. 2.

In this graph, $(s, v_2, v_3, v_4, v_5, v_6, v_7, p)$ is the path of maximum weight, and corresponds to the sub-vector $W[2..6] = \big(59\ 26\ {-53}\ 58\ 97\big)$.

This example highlighted the reason why looking at the path of maximum weight between s and p gives the contiguous sub-vector of maximum sum. We will show in Sect. 3.2 that it is the case, and why doing so takes linear time.

3.2 Proof of Equivalence

In the general case, to a vector $V[0..n-1]$ of size n, we associate the DAG $\mathcal{G} = (S, A)$ defined as follows :

$$\begin{cases} S = \{s, p\} \cup \left(\bigsqcup_{0 \leq i \leq n} \{v_i\} \right) \\ A = \{(s,p)\} \cup \left(\bigsqcup_{0 \leq i \leq n-1} \{(v_i, v_{i+1})\} \right) \cup \left(\bigsqcup_{0 \leq i \leq n-1} \{(s, v_0)\} \right) \cup \left(\bigsqcup_{1 \leq i \leq n} \{(v_i, p)\} \right) \end{cases}$$

With the weights : $\forall\ 0 \leq i \leq n-1$, $w((v_i, v_{i+1})) = V[i]$ and for every other arc a, $w(a) = 0$.

Property. This graph has a linear ($\mathcal{O}(n)$) size.

Proof. It has $n+3$ vertices and $3n+1$ arcs. □

Property. This graph is acyclic.

Proof. There is no cycle containing s because there is no arc entering s. There is no cycle containing p because there is no arc leaving p. There is no cycle containing only v_is because there is no arc going from v_j to v_i if $j \geq i$. □

We can derive the topological ordering ([7, Chapter 22]) of that graph from this proof : $s < v_0 < \cdots < v_n < p$. We can then apply Bellman's equations ([3]) to find the path of maximum weight between s and p, leading to a linear-time algorithm for the MSP 1D shown in Fig. 3. Bellman's equations lead to an $\mathcal{O}(|S|+|A|)$ algorithm on DAGs, here $\mathcal{O}(n)$ because $|S| = n+3$ and $|A| = 3n+1$.

Bellman's algorithm.

```
1 d[s] ← 0
2 For each x in {v_0, ..., v_n, p}
                          in that order, do
3     d[x] ← −∞
4     For each predecessor y of x, do
5         d[x] ← max(d[x], d[y] + w(y, x))
6     End for
7 End for
8 Return d[p]
```

Kadane's algorithm ([4]).

```
1 MaxSoFar ← 0
2 MaxEndingHere ← 0
3 For i from 0 to n − 1, do
4     MaxEndingHere ← max(0,
              MaxEndingHere + V[i])
5     MaxSoFar ← max(MaxSoFar,
              MaxEndingHere)
6 End for
7 Return MaxSoFar
```

Fig. 3.

We can improve this algorithm to have the same comparisons than Kadane's ones. At each vertex v_i we make two comparisons because there are two ongoing

arcs : (s, v_i) and (v_{i-1}, v_i). But (s, v_i) weights 0, so we can initialize d with 0 and avoid looking at that arc in the loop. We will end up with one comparison, as in Kadane's algorithm to update `MaxEndingHere`. In Kadane's algorithm, we update `MaxSoFar` at each step. Here, we compute this at the end (when we scan the predecessors of p), doing the same comparisons.

Theorem. The two algorithms return the same value on every input.

Proof. We will show that to every sub-vector $V[i..j]_{i \leq j}$ of sum $\sigma = \sum_{i \leq k \leq j} V[k]$ corresponds a path in $\mathcal{G}$ of weight σ. Then we will show that there is no other path of non-zero weight by a counting argument.

To the empty sub-vector corresponds the path (s, v) of weight 0. To each contiguous sub-vector $V[i..j]_{i \leq j}$ corresponds the path $(s, v_i, v_{i+1}, \ldots, v_j, v_{j+1}, p)$. The weight of this path is $w((s, v_i)) + w((v_i, v_{i+1})) + \cdots + w((v_j, v_{j+1})) + w((v_{j+1}, p)) = 0 + V[i] + \cdots + V[j] + 0 = \sum_{i \leq k \leq j} V[k]$. Thus :

Each contiguous sub-vector sums up to the weight of a path. (*)

By construction, there is exactly one path from v_n to p, and for every $i \in \{1, \ldots, n-1\}$ the number of paths from v_i to p is equal to the number of paths from v_{i+1} to p plus one (because there are only two arcs from v_i : the one to v_{i+1}, and the one to p). And the number of paths from v_0 to p is equal to the number of paths from v_1 to p (because there is only one arc from v_0 : the one to v_1). It leads to $(2 + \cdots + n) + n$ paths from $\{v_0, \ldots, v_{n-1}\}$ to p. Because s has arcs to each of those vertices plus one extra arc to p, it leads to a total of $(2+\cdots+n)+n+1$ paths from s to p. In those paths, we have the $1+(1+\cdots+n)$ paths which correspond to contiguous sub-vectors of V (for $1 \leq i \leq n$ there are $n-i+1$ contiguous sub-vectors of length i in V, plus one empty sub-vector), to which we add the $n-1$ paths $(s, v_i, p)_{1 \leq i \leq n-1}$, each of weight 0. Thus :

Each path weights either 0 either the sum of a contiguous sub-vector. (**)

Taken together, (*) and (**) imply that the maximum weight in $\mathcal{G}$ is equal to the maximum sum in V. □

4 Reduction of the MSP 2D to Maximum Weighted Path in Red-Blue Graphs

4.1 General Idea

The main idea is the same as in 1D : put arcs with weights equal to the matrix elements (thick plain line on next figures). The chosen chaining of the arcs ensures that we scan the columns contiguously : we chain the elements of the 0-th column, then of the 1-st, etc. The blue (normal plain line on next figures) and red (dotted line on next figures) arcs ensure that the rows are contiguous. These arcs have weight 0 like in 1D, but have additional labels for row constraints. A blue

arc labelled x^- means that the path corresponds to a contiguous sub-matrix for which the 0-th (upper when we write the matrix) row is the x-th row in the full matrix. A red arc labelled y^+ means that the path corresponds to a contiguous sub-matrix for which the last (lower when we write the matrix) row is the y-th row in the full matrix. To be **valid**, all blue arcs of a path have to be labelled with the same number, and the same goes for the red arcs. As in 1D, we will see that a **valid** path corresponds to a contiguous sub-matrix, and its weight is the sum of the elements of the sub-matrix.

Remark : Of course there is a dual approach for the construction of the graph : chaining the elements of the 0-th row, then of the 1-st, etc. to ensure that the rows are scanned contiguously, and adding extra constraint arcs (the blue and red arcs) to ensure that the columns are contiguous.

4.2 An Example

Let us take as example the matrix $N = \begin{pmatrix} 1 & 2 & -1 & -4 \\ -8 & -3 & 4 & -2 \\ 3 & 8 & 10 & 1 \end{pmatrix}$. Its associated red-blue graph is shown in Fig. 4.

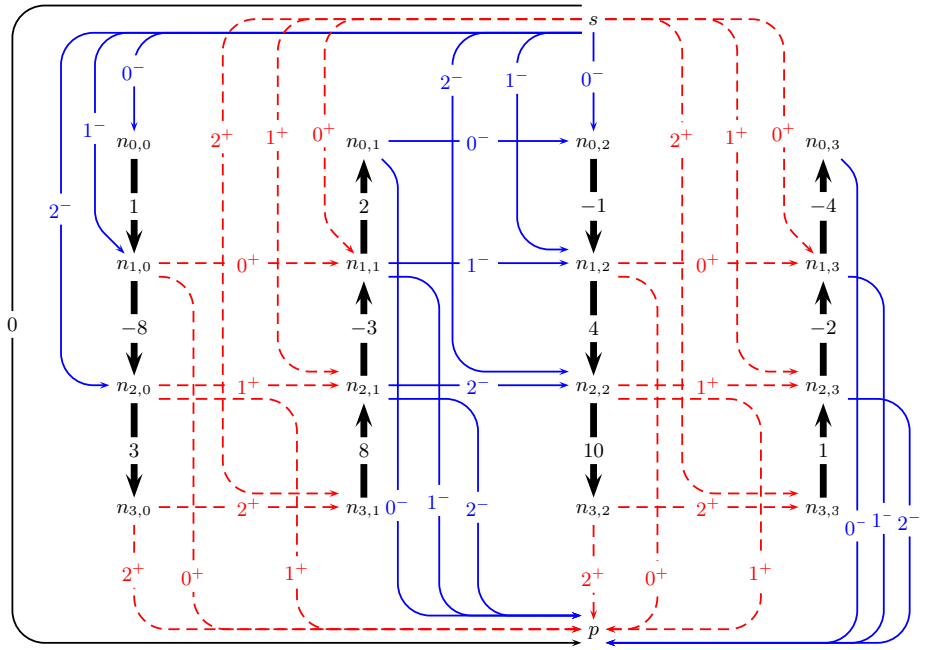

Fig. 4.

Let us take the path $(s, n_{1,2}, n_{2,2}, n_{3,2}, n_{3,3}, n_{2,3}, n_{1,3}, p)$ shown in Fig. 5, which corresponds to the sub-matrix $N[1..2][2..3] = \begin{pmatrix} 4 & -2 \\ 10 & 1 \end{pmatrix}$.

$\star$ This path leaves the vertex s with a blue arc labelled 1^-. It means that the 0-th row of the contiguous sub-matrix is the 1-st row of N.

$\star$ Then it takes the black arcs weighted with 10 and 1. It means that this sub-matrix will contain these values, located in $N[1][2]$ and $N[2][2]$.

$\star$ Then it takes the red arc labelled 2^+. It means that the last row of this sub-matrix is the 2-nd row of N.

$\star$ In the next column, the black arcs mean that the sub-matrix contains both $N[2][3]$ and $N[1][3]$. The blue arc still indicates that the 0-th row is 1.

The black part of the graph (thick plain line) shows the values inside the matrix, as in the 1D problem. We will explain in Sect. 4.3 why we put the arcs alternatively down and up.

The blue part of the graph (normal plain line) shows the 0-th row of the sub-matrix considered. To be **valid**, all the blue arcs in a path have to be labelled with the same number. It means that every column starts at the same row.

For example, Fig. 6 shows the path $(s, n_{1,0}, n_{2,0}, n_{2,1}, n_{1,1}, n_{0,1}, p)$ which is not **valid** because the blue arc $(s, n_{1,0})$ is labelled 1^- but the blue arc $(n_{0,1}, p)$ is labelled 0^-. Of course, $\begin{pmatrix} & 2 \\ -8 & -3 \end{pmatrix}$ is not a sub-matrix !

The red part of the graph (dotted line) shows the last row of the sub-matrix considered. To be **valid**, all the red arcs in a path have to be labelled with the same number. It means that every column ends at the same row.

For example, Fig. 7 shows the path $(s, n_{0,0}, n_{1,0}, n_{1,1}, n_{0,1}, n_{0,2}, n_{1,2}, n_{2,2}, p)$ which is not **valid** because the red arc $(n_{1,0}, n_{1,1})$ is labelled 0^+ but the red arc $(n_{2,2}, p)$ is labelled 1^+. Of course, $\begin{pmatrix} 1 & 2 & -1 \\ & & 4 \end{pmatrix}$ is not a sub-matrix !

This example highlighted the reason why looking at the **valid** path of maximum weight between s and p gives the contiguous sub-matrix of maximum sum. We will show in Sect. 4.3 that it is the case.

Remark : Because blue and red arcs bear additional information (they have additional labels to ensure row contiguity), we cannot use standard graph algorithms to compute the maximum weighted path. If we forget these constraints and just take the maximum weighted path in the underlying weighted graph (recall that the weight of those arcs is 0), the resulting maximum weight is an upper bound for the maximum sum of contiguous sub-matrices.

In the example, the maximum weighted path (if we abstract from the row constraints) is depicted on Fig. 8. But $\begin{pmatrix} 2 & -1 & \\ -3 & 4 & \\ 3 & 8 & 10 & 1 \end{pmatrix}$ is not a sub-matrix.

Bellman's algorithm, which would take quadratic time, only gives us the upper bound 24, the maximum among contiguous sub-matrices being 22 in the sub-matrix $\begin{pmatrix} 3 & 8 & 10 & 1 \end{pmatrix}$.

In the general case, we have no more than an upper bound. It can even happen that the maximum weighted path (if we abstract from the row constraints) is disjoint from the sub-matrix of maximum sum. In Sect. 4.3 we will then only focus on valid paths, and show that finding the sub-matrix of maximum sum and finding the valid path of maximum weight is the same problem.

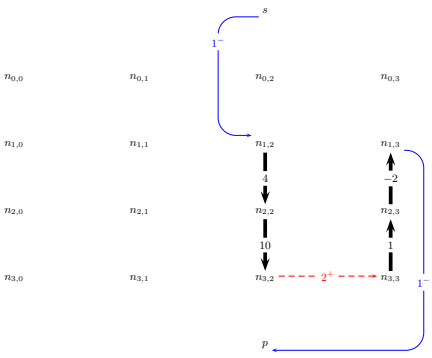

Fig. 5.

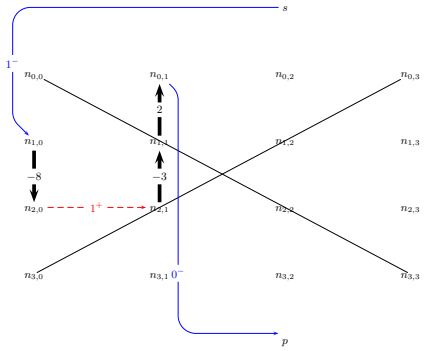

Fig. 6.

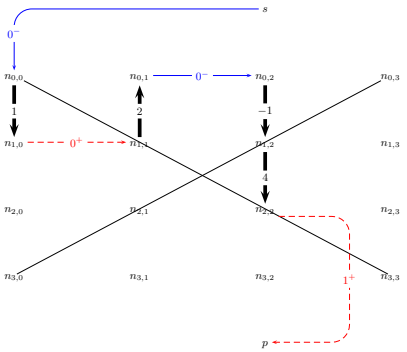

Fig. 7.

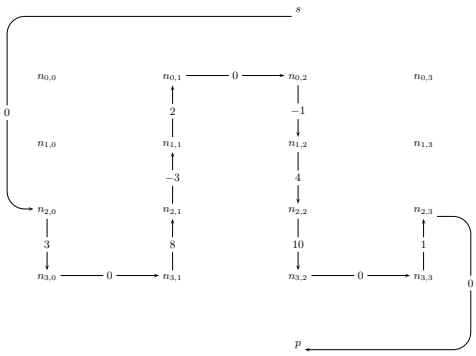

Fig. 8.

4.3 Proof of Equivalence

In the general case, to a matrix $M[0..m-1][0..n-1]$ of size $m \times n$, we associate the red-blue graph $\mathcal{H} = (T, B)$ defined as follows :

$$
\begin{cases}
T = \{s,p\} \cup \left(\bigsqcup_{\substack{0\le i\le m\\ 0\le j\le n-1}} \{v_{i,j}\} \right) \\
B = \{(s,p)\} \cup \left(\bigsqcup_{\substack{0\le i\le m-1\\ 0\le 2k\le n-1}} \{(v_{i,2k}, v_{i+1,2k})\} \right) \cup \left(\bigsqcup_{\substack{0\le i\le m-1\\ 1\le 2k+1\le n-1}} \{(v_{i+1,2k+1}, v_{i,2k+1})\} \right) \cup \\
\left(\bigsqcup_{\substack{0\le i\le m-1\\ 0\le 2k\le n-1}} \{(s, v_{i,2k})\} \right) \cup \left(\bigsqcup_{\substack{0\le i\le m-1\\ 1\le 2k+1\le n-1}} \{(v_{i,2k+1}, p)\} \right) \cup \left(\bigsqcup_{\substack{0\le i\le m-1\\ 2\le 2k+2\le n-1}} \{(v_{i,2k+1}, v_{i,2k+2})\} \right) \\
\cup \left(\bigsqcup_{\substack{1\le i\le m\\ 1\le 2k+1\le n-1}} \{(s, v_{i,2k+1})\} \right) \cup \left(\bigsqcup_{\substack{1\le i\le m\\ 0\le 2k\le n-1}} \{(v_{i,2k}, p)\} \right) \cup \left(\bigsqcup_{\substack{1\le i\le m\\ 1\le 2k+1\le n-1}} \{(v_{i,2k}, v_{i,2k+1})\} \right)
\end{cases}
$$

With the weights : $\forall\ 0 \le i \le m-1, \forall\ 0 \le 2k \le n-1$, $w((v_{i,2k}, v_{i+1,2k})) = M[i][2k]$ and $w((v_{i+1,2k+1}, v_{i,2k+1})) = M[i][2k+1]$, and for every other arc a, $w(a) = 0$.

And with additional labels for blue and red arcs, that denote the 0-th and last row of the sub-matrix associated with a path containing these arcs : blue arcs that enter or leave a $v_{i,j}$ vertex are labelled i^- and red arcs that enter or leave a $v_{i,j}$ vertex are labelled $(i-1)^+$.

Property. This graph has a quadratic ($\mathcal{O}(m \times n)$) size.

Proof. It has $(m+1)n + 2$ vertices and $3mn + m(n-1) + 1$ arcs. □

Remark : The black arcs in the graph go down in the first column, then up in the second one, then down, etc. There are of course other ways of constructing a similar graph that would allow paths corresponding to sub-matrices : having all the arcs going down for example. But in that case, we would need an arc from each vertex of one column to each vertex which is upper than it in the next column. It would create $(m-1) + (m-2) + \cdots + 1$ arcs per column hence a cubic ($\mathcal{O}(m^2 n)$) number of arcs. If we want a less-than-cubic-time algorithm, we cannot afford this.

Property. This graph is acyclic.

Proof. There is no cycle containing s because there is no arc going to s. There is no cycle containing p because there is no arc going from p. There is no cycle containing only $v_{i,j}$s because there is no arc going from $v_{i,2k}$ to $v_{j,2k}$ if $j \ge i$, there is no arc going from $v_{i,2k+1}$ to $v_{j,2k+1}$ if $j \le i$ and there is no arc from $v_{i,k}$ to $v_{j,l}$ if $l < k$. □

We can derive the topological ordering of that graph from this proof :

- ⋆ $s < v_{0,0} < \cdots < v_{m,0} < v_{m,1} < \cdots < v_{0,1} < \cdots < v_{0,n-1} < \cdots < v_{m,n-1} < p$ if n is odd and

⋆ $s < v_{0,0} < \cdots < v_{m,0} < v_{m,1} < \cdots < v_{0,1} < \cdots < v_{m,n-1} < \cdots < v_{0,n-1} < p$ if n is even.

Nevertheless, as noted in Sect. 4.2, we cannot apply standard graph algorithms to find the path of maximum weight between s and p, because we need to check that the paths are valid.

Theorem. The sum of elements in the contiguous sub-matrix of maximum sum is the same as the weight of the valid path of maximum weight in the associated red-blue graph.

Proof. We will show that to every sub-matrix $N[i..j][k..l]_{\substack{i\leq j\\k\leq l}}$ of sum $\sigma = \sum_{\substack{i\leq a\leq j\\k\leq b\leq l}}\sum N[a][b]$ corresponds a valid path in $\mathcal{H}$ of weight σ. Then we will show that there is no other valid path of non-zero weight.

To a contiguous sub-matrix $N[i..j][k..l]_{\substack{i\leq j\\k\leq l}}$, if both k and l are even, corresponds the path $(s, v_{i,k}, \ldots, v_{j+1,k}, v_{j+1,k+1}, \ldots, v_{i,k+1}, \ldots, v_{i,l}, \ldots v_{j+1,l}, p)$. The weight of this path is :

$$
\begin{aligned}
&w((s, v_{i,k})) + \big[w((v_{i,k}, v_{i+1,k})) + \cdots + w((v_{j,k}, v_{j+1,k}))\big] + \\
&w((v_{j+1,k}, v_{j+1,k+1})) + \big[w((v_{j+1,k+1}, v_{j,k+1})) + \cdots + w((v_{i+1,k+1}, v_{i,k+1}))\big] + \\
&\cdots + \big[w((v_{i,l}, v_{i+1,l})) + \cdots + w((v_{j,l}, v_{j+1,l}))\big] + w((v_{j+1,l}, p)) \\
&= 0 + \big(N[i][k] + \cdots + N[j][k]\big) + 0 + \big(N[j][k+1] + \cdots + N[i][k+1]\big) + \cdots + \\
&\big(N[i][l] + \cdots + N[j][l]\big) + 0 \\
&= \sum_{i\leq a\leq j} N[a][k] + \sum_{i\leq a\leq j} N[a][k+1] + \cdots + \sum_{i\leq a\leq j} N[a][l] = \sum_{\substack{i\leq a\leq j\\k\leq b\leq l}}\sum N[a][b].
\end{aligned}
$$

The computation is similar if k and/or l are odd, and to the empty sub-vector corresponds the path (s, v) of weight 0. Thus :

Each contiguous sub-matrix sums up to the weight of a valid path. (i)

Let us now take a valid path in the red-blue graph, different from (s, p) which weights 0. This path thus has at least one blue arc and one red arc (if it starts with a blue arc, entering an even column, the only arcs that leave this column are red ; if it starts with a red arc, entering an odd column, the only arcs that leave this column are blue). Let us call x^- the label of all blue arcs in the path and y^+ the label of all red arcs in the path. Blue arcs enter only even columns (or p) and red arcs enter only odd columns (or p). Blue arcs leave only odd columns (or s) and red arcs leave only even columns (or s). There are only two cases :

⋆ First case : $y = x - 1$. By construction, the x^--labelled arcs can only enter or leave vertices in $\{v_{x,j} \mid 0 \leq j \leq n-1\}$ and the y^+-labelled arcs can only enter of leave vertices in $\{v_{y+1,j} \mid 0 \leq j \leq n-1\}$ (apart from s and p). Because

$y = x - 1$, the only vertices apart from s and p are thus $v_{x,j}$ vertices. Thus the path cannot contain black arcs. Its weight is 0 (an example of that kind of paths is $(s, n_{1,0}, n_{1,1}, n_{1,2}, p)$).

$\star$ Second case : $y \geq x$. For the same reason as in the first case, the path enters the k-th even column via the vertex in $v_{x,2k}$ and leaves this column via the vertex $v_{y+1,2k}$. Thus the black arcs in that column are the arcs in $\{(v_{i,2k},\ v_{i+1,2k}) \mid x \leq i \leq y\}$ which have the weights $\{N[i][2k] \mid x \leq i \leq y\}$. And the same goes in the k-th odd column : the black arcs are $\{(v_{i+1,2k+1},\ v_{i,2k+1}) \mid y \geq i \geq x\}$ which have the weights $\{N[i][2k+1] \mid y \geq i \geq x\}$. All in all, the total weight of the arcs is $\sum_{\substack{x \leq a \leq y \\ z \leq b \leq t}} \sum N[a][b]$ for some $z \leq t$. This corresponds to the sum of elements in the contiguous sub-matrix $N[x..y][z..t]$.

$\star$ The case $y < x - 1$ is impossible : there are no arcs going up in even columns so a path cannot enter the k-th even column at vertex $v_{x,2k}$ and leave it at vertex $v_{y+1,2k}$ if $y + 1 < x$. The same goes for odd columns : there are no arcs going down in odd columns so a path cannot enter the k-th odd column at vertex $v_{y+1,2k+1}$ and leave it at vertex $v_{x,2k+1}$ if $y + 1 < x$. Thus :

Each valid path weights 0 or the sum of a contiguous sub-matrix. (ii)

Taken together, (i) and (ii) imply that the maximum weight of valid paths in $\mathcal{H}$ is equal to the maximum sum of contiguous sub-matrices in N. □

5 Conclusion

In this paper, we have presented a reduction from the maximum subarray problem in one dimension to a well known graph problem. We think that this reduction highlights the reason why we have a linear-time algorithm for this problem.

To extend this idea to two dimensions, we have designed a new data structure, red-blue graphs, that allows us to scan all the contiguous sub-matrices of a given matrix. By looking at the valid paths in this data structure, we obtain a reduction from the maximum subarray problem in two dimensions. Unfortunately, this reduction does not "obviously" lead to an optimal algorithm. Nevertheless, we think that this data structure may help in finding a better algorithm for the MSP 2D, and maybe for other matrix-related problems that need to scan contiguous sub-matrices.

A natural extension for the MSP is searching for the k maximum sums, instead of only the maximum sum. We can allow these sums to overlap (we have an optimal $\mathcal{O}(n+k)$ algorithm in one dimension : [5]) or we can instead want to have disjoint sums (we also have an optimal $\mathcal{O}(n)$ algorithm in one dimension : [9]). It's interesting to note, like Brodal and Jørgensen, that an alternative optimal algorithm when the overlap is allowed use the weighted graph presented.

If the 1D problem is essentially solved, there remains a lot of work in higher dimensions. We hope that in the future, part of the ideas presented in this paper will lead to better algorithms.

Last but not least, we would like to thank our advisor, Éric Violard, for his help on the writing of this article.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[2] Sung Eun Bae. "Sequential and parallel algorithms for the generalized maximum subarray problem". PhD thesis. University of Canterbury, 2007. URL: http://hdl.handle.net/10092/1202.

[3] Richard Bellman. *On a routing problem*. Tech. rep. 1956. URL: http://www.dtic.mil/dtic/tr/fulltext/u2/606258.pdf.

[4] Jon Bentley. "Programming Pearls: Algorithm Design Techniques". In: *Commun. ACM* 27.9 (Sept. 1984), pp. 865–873. URL: http://www.akira.ruc.dk/~keld/teaching/algoritmedesign_f07/Artikler/05/Bentley84.pdf.

[5] Gerth Stølting Brodal and Allan Grønlund Jørgensen. "A Linear Time Algorithm for the k Maximal Sums Problem". In: *Mathematical Foundations of Computer Science 2007*. Ed. by Luděk Kučera and Antonín Kučera. Vol. 4708. Springer Berlin Heidelberg, 2007, pp. 442–453. URL: https://users-cs.au.dk/gerth/papers/mfcs07sum.pdf.

[6] Timothy M. Chan. "More Algorithms for All-pairs Shortest Paths in Weighted Graphs". In: *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*. STOC'07. 2007, pp. 590–598. URL: https://cs.uwaterloo.ca/~tmchan/moreapsp.pdf.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 3rd ed. MIT Press and McGraw-Hill, 2009.

[8] Ulf Grenander. *Pattern analysis: Lectures in Pattern Theory 2*. New York: Springer, 1978.

[9] Walter L. Ruzzo and Martin Tompa. "A Linear Time Algorithm for Finding All Maximal Scoring Subsequences". In: *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, 1999, pp. 234–241. URL: http://homes.cs.washington.edu/~ruzzo/papers/maxseq.pdf.

[10] Tadao Takaoka. "Efficient Algorithms for the Maximum Subarray Problem by Distance Matrix Multiplication". In: *Electronic Notes in Theoretical Computer Science*. CATS'02, Computing: the Australasian Theory Symposium 61 (2002), pp. 191–200. URL: http://www.cosc.canterbury.ac.nz/tad.takaoka/cats02.pdf.