



AYBEYAN SELIMI   
SAMEDIN KRRABAJ  
MUZAFER SARAČEVIĆ   
SELVER PEPIĆ

## MEMOIZATION METHOD FOR STORING MINIMUM-WEIGHT TRIANGULATION OF CONVEX POLYGON

**Abstract** *This study presents a practical view of dynamic programming, specifically in the context of the application of finding optimal solutions for the polygon triangulation problem. The problem of the optimal triangulation of a polygon is considered to be a recursive substructure. The basic idea of the constructed method lies in finding an adequate method for the rapid generation of optimal triangulations and storing them in as small a memory space as possible. Our method is based on a memoization technique, and its emphasis is in storing the results of the calculated values and returning the cached result when the same values occur again. The significance of the method is in the generation of the optimal triangulation for a large number of  $n$ . All of the calculated weights in the triangulation process are stored and performed in the same table. The processing of the results and implementation of the method were carried out in the Java environment, and the experimental results were compared with the square matrix and Hurtado-Noy method.*

**Keywords** Catalan number, data storage, memoization, recursion, minimum-weight triangulation.

**Citation** Computer Science 20(2) 2019: 195–211

## 1. Introduction

Triangulation is a term that appears in various disciplines of mathematics as well as in computer science. Triangulation is defined as the decomposition of a particular region into smaller pieces (called triangulation elements – usually triangles) and that are easy to handle. An application of triangulation can be seen in different areas where there is a fixed set of points, such as algebra, topology, volume calculations, and meshing [20].

The triangulation problem is generally based on determining a set of all triangulations for a given geometric figure (construction of triangular meshes) or in finding the optimal triangulation of a given set of points in a plane or in space. Triangular meshes represent the dominant discretization of surfaces in computer graphics, for which there is a significant amount of research. Many triangulation applications in geometry rely on an orthogonal primal/dual network structure. The use of such dual structures of triangulation depends on the type of application used in physical simulation [4], parametrization [10, 15], and architecture modeling [9, 13], for example. Most of the obtained research results are based on planar triangular meshes.

In many applications of computational geometry, determining the boundary of a surface occurring by connecting to one or more closed geometric figures is necessary in the problem of triangulation. Such surfaces are encountered when filling the holes of an incomplete mesh, while this is dealt with by introducing more identified boundaries such as the outer boundary and several inner islands in the complex holes problem [1]. In contour interpolation, 2D curves from neighboring flat parts must be connected with one or more surfaces. In practice (and in the case of arbitrary planar points), an initial mesh for the surface treatment is first created; this is usually a triangulation involving only the vertices of the polygon and then refining this starting surface in order to achieve better smoothness or mesh quality. The initial triangulation of one polygon can be calculated by using a simple dynamic programming algorithm [2, 3].

The important feature of the proposed algorithm is the production of an optimal triangulation that minimizes the sum of a certain “weight”, whether it is the size that can be measured individually for each triangle or for each pair of adjacent triangles. Optimization of the triangulation is important for the successful construction of a final mesh and depend on the quality of the initial mesh.

In this paper, we present an algorithm in order to find the optimal triangulation of a convex polygon with memoization. We suggest a formula that avoids the calculation of the same values several times and does not endanger the optimality of the obtained results. More precisely, given the number of calculations required in determining the gravity of the triangles in triangulation, our algorithm finds the optimal triangulation value by minimizing the optimal function that is defined as the sum of all of the individual triangulation weights in the triangulation. Practically, this paper also presents the optimal triangulation algorithm of a convex polygon based on the principles of dynamic programming, significantly reducing the computational cost when compared to the algorithm constructed in this paper [17].

## 2. Related works

The triangulation of a planar polygon is one of the basic problems of computational geometry. It is applied in obtaining the process of three-dimensional representations of objects from a given set of points. The main purposes of the triangulation process is obtaining the speediest polygon triangulation of all possible variants. The speed in this process is important, as the number of these different triangulations increases drastically with increases in the number of polygon vertices.

There are many efficient algorithms that generate the triangulation of different geometric figures [6,19], where the optimality of the constructed triangulation is most often not guaranteed. In papers [8,12], the algorithms are given for the calculation of minimum weight polygon triangulation into the plane; these are also known as algorithms that minimize the sum of the total lengths of the edges and have time complexity  $O(n^3)$ . Algorithms improved on the principle of dynamic programming are aimed at constructing the optimal triangulation of larger domains and use the previously calculated optimal triangulation of the smaller sub-domains.

Paper [3] presents an algorithm that finds the optimal polygon triangulation by minimizing the triangle weight in the triangulation process with the same complexity. There are other approaches for polygon triangulation, but they do not guarantee optimality and are often limited to specific class inputs. In cases where the polygon is sufficiently planar, there are algorithms that use polygon projection in the best-fitting plane and ultimately get the required triangulation after the triangulation of this projected polygon [16].

However, this algorithm cannot be applied to the triangulation of the polygon curve; in many cases, these polygons do not have planar projections. In addition to the concept of triangulation, there are also methods that construct quadrangulations of the polygon by interpolating sketches with flow lines [5]. For triangulation of the 3D polygon in [11], extracts of near-developable surfaces using convex hulls are presented.

A calculation of the optimal triangulation of a convex polygon can be constructed on the basis of the block method for generating triangulations that have been expounded in [14]. The presented/recommended method in this paper reflects on the fact that the recorded values are used in order to find the optimal triangulation and directly record the values of the vertices and their weights.

Another approach of convex polygon triangulation is presented in [18,21]. In these papers, authors developed an algorithm for convex polygon triangulation that uses previously constructed triangulations.

## 3. Minimum-weight triangulation method based on memoization

Dynamic programming is a technique that analyzes the solution of problems with methods based on the principles of optimization. The optimal solution of a problem is independent of the initial position and is obtained as a succession of optimal solutions of subproblems. The variables that correspond to the system in dynamic programming

are determined consecutively. The method developed in this paper is based on the principle storage of the best solution by memoization.

In solving the optimization problem with the dynamic programming technique, the optimal substructure and overlapping property of the subproblems are two key elements that are examined. Our method for finding and storing the optimal triangulations of a convex polygon use this overlapping property of the subproblems. The first step in solving an optimization problem by dynamic programming is to characterize the structure of an optimal solution [7].

The idea for the construction of the method for obtaining the optimal triangulation of a convex polygon with memoization starts from the possibility that gives Java as a programming language in the implementation of the weight storage results that are determined with the advanced application of packages in the work with table cells. Memoization is a set of dynamic programming and recursion. The technique is in the top-down direction, where all solutions are stored in the memory. This technique does not solve the same problem many times; it uses the previously calculated solutions to find the solution to the general problem. The dynamic matrix initially denoted with 0.

The *DefaultTableCellRenderer.Java* package takes the main role in storing the triangulation's weight. This package enables the storage of multiple values within a single table cell (which provide efficient cell division into multiple columns and rows) [17]. The calculations of the constructed method are based on the principles of dynamic programming techniques. Except for the possibility of the fast calculation of optimal triangulations, the memoization technique provides an opportunity to save memory space and time during the calculation of polygonal triangulations.

Let  $P = \langle v_0, v_1, \dots, v_{n-1} \rangle$  represent a convex polygon by listing its vertices in counterclockwise order. Given two non-adjacent vertices  $v_i$  and  $v_j$ ,  $\delta_{ij}$  is a diagonal of the polygon. A triangulation of a polygon is a set  $T$  of the diagonals of the polygon that divide the polygon into triangles with nonintersecting diagonals. In the optimal polygon triangulation problem, weight function  $w$  is defined on the triangles formed by the edges and diagonals of  $P$ . The problem is to find a triangulation with a minimal sum of the weights of the triangles in the decomposition of a convex polygon. Weight  $w$  of triangle  $v_i v_j v_k$  is calculated as follows:

$$w(i, j, k) = |v_i v_j| + |v_j v_k| + |v_k v_i| \quad (1)$$

where  $|v_i v_j|$  is the length of the edge with vertices  $v_i$  and  $v_j$  of the triangle.

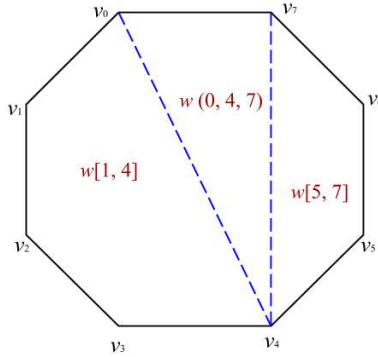
We have developed a method that uses a correspondence between polygon triangulation, Catalan numbers, and balanced parenthesis. The edges of a convex polygon with  $n$  vertices are denoted as matrices  $A_i, i = 1, 2, \dots, n - 1$ . The balanced parenthesized product of  $n - 1$  matrices correspond to a polygon with  $n$  vertices. Each matrix  $A_i$  from this product corresponds to edge  $v_i v_{i+1}$  of the polygon, and diagonal  $\delta_{ij}, (i < j)$  corresponds to matrix  $A_{i+1, \dots, j}$  that is obtained during the matrix product calculation. From the fact that the matrix chain is a special case of the

optimal triangulation problem for matrix product  $A_1 A_2 \cdots A_n$ , we consider the triangulation of a convex polygon with  $(n + 1)$  vertices. For matrix  $A_i$  with dimension  $p_{i-1} \times p_i$ , the weight function of the triangulation is  $w(i, j, k) = p_i p_j p_k$ .

By replacing matrix dimensions  $p_0, p_1, \dots, p_n$  in the matrix chain with vertices  $v_0, v_1, \dots, v_n$  of a convex polygon, we have the following weight function of triangulation:

$$w = m[i, k] + m[k + 1, j] + w(i, j, k) \tag{2}$$

The weight of the optimal triangulation is the sum of the weights of sub-polygons  $\langle v_0, v_1, \dots, v_k \rangle$  and  $\langle v_k, v_{k+1}, \dots, v_n \rangle$  and triangle  $\Delta v_0 v_k v_n$  in the triangulation (see Figure 1).



**Figure 1.** Optimal triangulation of convex octagon with associated weights

Similar to the computation of minimum cost  $m[i, j]$  in the matrix chain, for  $1 \leq i < j \leq n$ ,  $w[i, j]$  is defined to be the weight of the optimal triangulation of sub-polygon  $v_{i-1}, v_i, \dots, v_j$ , and the optimal triangulation of the polygon is  $w[1, n]$ . Then, the optimal triangulation has the following recursive formulation:

$$w[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \leq k < j} w[i, k] + w[k + 1, j] + w(i, j, k) & \text{if } i < j \end{cases} \tag{3}$$

The method for storing triangulation weights is based on a table with  $(n \times n)$  dimension ( $i = j = n$ ), where  $i$  represents a row,  $j$  represents a column, and  $n$  is the number of vertices of the polygon. Table  $(n \times n)$  is used for the calculation of the weights of the triangles in the triangulation. The table is divided diagonally into two parts. The diagonal divides ( $i = j$ ) the table into left and right parts. The left side of the table is filled with the edges of the triangles whose lengths are represent by matrices  $A_1, A_2, \dots, A_{n-1}$  in the triangulation. The elements in the diagonal of the

table represent the vertices of the polygon and are filled with zero because they have no weights. From the correspondence between the  $n$  balanced parenthesized product and the  $(n + 1)$ -vertex polygon, the elements of the  $n$ -th column of the table were not needed in the matrix chain calculation and are filled with  $X$ .

For  $k = 1$ , we have the polygon edge that is determined by adjacent vertices  $v_1$  and  $v_2$  (see Figure 2). Similarly, for  $k = 2, 3, \dots, n - 1$ , we get the others edges of the polygon that are determined from adjacent vertices  $v_i$  and  $v_j$  for  $i = 2, 3, \dots, n - 1$  and  $j = i + 1$ .

$i$	1	2	3	...	$n-2$	$n-1$	$n$
1	0	$w[1, 2]$	$w[1, 3]$	...	$w[1, n-2]$	<i>Opt w</i>	$X$
2	$ v_1 v_2 $	0	$w[2, 3]$	...	$w[2, n-2]$	$w[2, n-1]$	$X$
3	$ v_3 v_1 $	$ v_3 v_2 $	0	...	$w[3, n-2]$	$w[3, n-1]$	$X$
	⋮	⋮	⋮		⋮	⋮	⋮
$n-2$	$ v_{n-2} v_1 $	$ v_{n-2} v_2 $	...	...	0	$w[n-2, n-1]$	$X$
$n-1$	$ v_{n-1} v_1 $	$ v_{n-1} v_2 $	$ v_{n-2} v_3 $	...	$ v_{n-1} v_{n-2} $	0	$X$
$n$	$ v_n v_1 $	$ v_n v_2 $	$ v_n v_3 $	...	$ v_n v_{n-2} $	$ v_n v_{n-1} $	0

**Figure 2.** General scheme of labeling of cells in table

By increasing the number of rows in the table, we obtain the edges of the triangles that are the elements of triangulation; i.e., the edges that are obtained from the non-adjacent vertices of the polygon. For example, for  $k = 1, 2$ , we have the weight of the edge between vertices  $v_1$  and  $v_3$ ,  $k = 2, 3$ , the weight of the edge between vertices  $v_2$  and  $v_4$ , and so on. Similarly, in the end for  $k = 1, 2, \dots, n - 1$ , we obtain the weight of the edge between vertices  $v_1$  and  $v_n$ , which represents the total weight of the triangulation.

Below, we present a  $5 \times 5$  table for pentagon triangulation according to the filling general scheme given in Figure 2.

We present Algorithm 3.1 for finding and storing the optimal triangulation. At the beginning, the algorithm expects the number of rows and columns of table  $n$  that

correspond to the  $n$ -vertex polygon. All vertices of the polygon match the diagonal elements of the table.

The  $n$ -th column of the table consists of those elements that do not have a role in the calculation of the weights in the triangulation process; due to this, they are labeled with  $X$ , which has the meaning of a cell that does not have a weight.

---

**Algorithm 3.1** Storing of optimal triangulation with memoization

---

**Require:**  $n$ , Table ( $n \times n$ ).

- 1: Create a new ( $n \times n$ ) Table for ( $i = 1; i \leq n; i++$ )  
 Label the field  $(i, j) = 0$ , where is  $i = j$  and  $(i, j) = adj$ , where is  $j - i = 1$
  - 2: Filling the other fields for ( $i = 1; i \leq n; i++$ ) for ( $j = 2; j \leq n; j++$ )  $j = i + 1$
  - 3: Calculate the Matrix Chain Product for elements of Table  
 if  $(w[i, j] < \infty)$  return  $w[i, j]$   
     if  $(i == j)$  then  $w[i, j] = 0$  else  $k = i$ ;  
          $Opt\ w = w[i, k] + w[k + 1, j] + w(i, j, k)$   
         for  $(k = i + 1; k < j; k++)$   
             if  $(Opt\ w < w[i, j])$  then  $Optw = w[i, j]$   
**return**  $w[i, j]$
- 

Based on correspondence  $P(n + 1) = C_n$  for pentagon ( $n = 5$ ), we obtain five different triangulations. The edges of the pentagon are labeled with matrices  $A_i$ ,  $i = 1, 2, \dots, 4$ . The matrices with an even number of indexes have a dimension of  $5 \times 1$ , and the matrices with an odd number of indexes have a dimension of  $1 \times 5$ . This example shows the recursion tree of the memorized matrix chain for the pentagon.

The proposed algorithm has three steps:

1. We form table  $i \times j$  and fill the cells along the diagonal with 0 where  $(i = j)$ . Then, we fill the edges of the polygon at position  $(i, j), i = 2, 3, \dots, n$  and  $j = 1, 2, \dots, n - 1$ . They are the adjacent vertices of a polygon that can be used to form a  $(i, j, k)$  triangle in the triangulation process. Their weights are filled in the cells of the table where  $(i, j), i = 1, 2, \dots, n - 1, j = i + 1$ ; this means that there is no value  $k$  for them between  $(i, j)$ .
2. We calculate all triangulation weights  $w[i, j]$  based on memoization, bearing in mind that the calculations do not calculate multiple times the same weight for the rows and columns of the table where  $j - i > 1$  and  $i = 2, 3, \dots, n - 1$ . According to Eq. 3, we will go to the next step. To the assigned  $k$  values in position  $(i, j)$ , we add their correspondent weights on position  $(j, i)$ .
3. In cell  $1 \times (n - 1)$  of the table, we obtain the weight of the optimal triangulation.

There is a  $(5 \times 5)$  table for all five triangulations containing the optimal triangulation that corresponds to the minimum weight in the triangulation process of the pentagon (see Table 1). The table for the pentagon is given in the following example according to the labeling of the general scheme from Figure 2.

We describe the acquisition of the optimal triangulation for the irregular convex pentagon with the following steps:

- **Step 1:** In this step, the  $5 \times 5$  table is formed, the cells in the diagonal are filled with zeroes, and the cells below the diagonal on the left side are represented by the adjacent vertices (that is, the vertices of the polygon). Thereafter, the values for all  $(i, j, k)$  triangles are set up. The values in parentheses represent the number of vertices that are part of the triangulation; for example, 2 is between Vertices 1 and 3 and is in cell  $(3,1)$  of the table. After these steps, the table takes the following form.

**Table 1**  
Optimal triangulation table of pentagon

0	25	10	<b>35</b>	X
(2,1)	0	5	10	X
2	(3,2)	0	25	X
2,3	3	(4,3)	0	X
2,3,4	3,4	4	(5,4)	0

- **Step 2:** We find the weights of the triangles that are part of the triangulation procedure for the  $C_3$  rows that are given in the  $5 \times 5$  table. The table for the generated triangulations is created on the basis of Algorithm 3.1. The sums of all of the triangle weights are saved in the bottom-up technique in memoization (see Figure 3). Below, the table (Tab. 2) of weights is given that correspond to the diagonals in the possible triangulation of the pentagon (see Figure 4).

**Table 2**  
Extended table for triangulation of pentagon

Triangulations	Diagonals in triangulation	Optimal triangulation
1	$\delta_{13}, \delta_{35}$	71
2	$\delta_{14}, \delta_{24}$	61
3	$\delta_{24}, \delta_{25}$	<b>35</b>
4	$\delta_{25}, \delta_{35}$	51
5	$\delta_{14}, \delta_{13}$	76

- **Step 3:** In this step,  $Opt w$  is calculated according to the following condition:

$$Opt w < w [i, j] \tag{4}$$

where is  $1 \leq i \leq 5$ . Based on Equation 4, optimal triangulation  $Optw = 35$  is obtained.



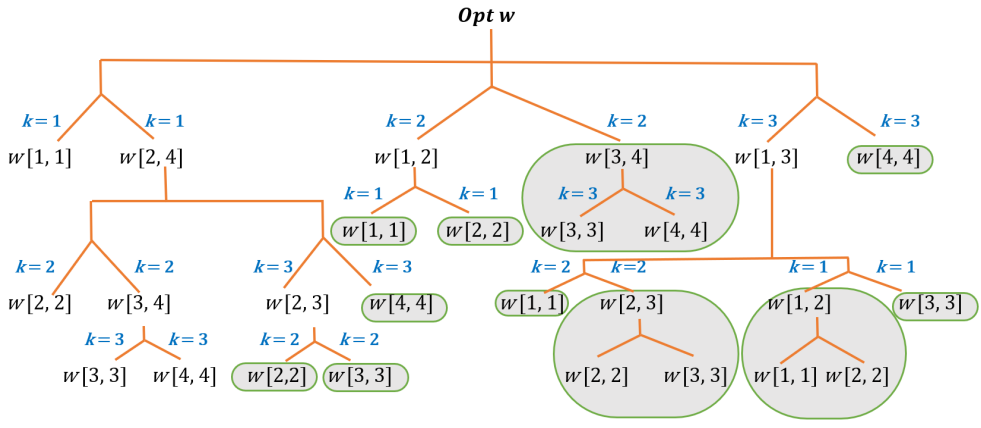


Figure 3. Recursion tree of memorized matrix chain of pentagon

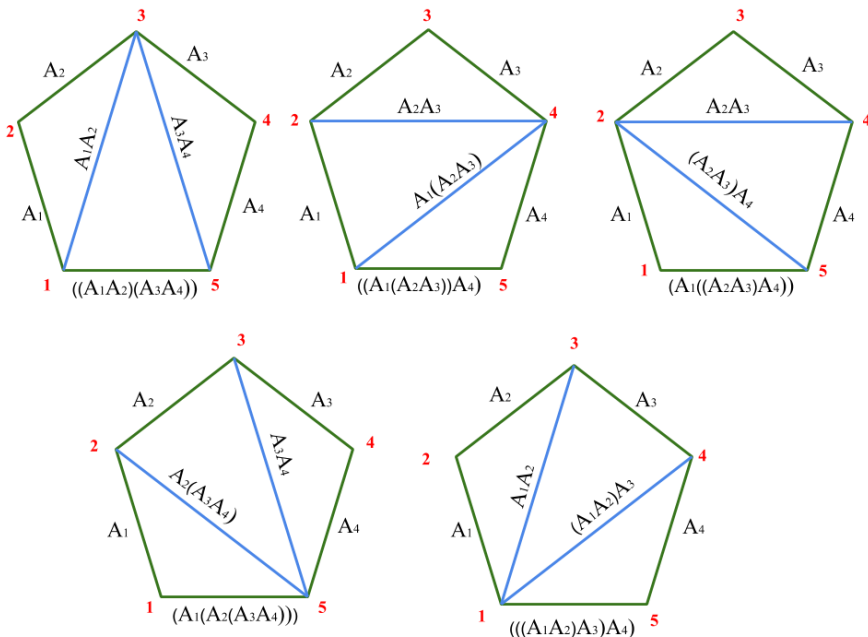


Figure 4. Triangulation of pentagon

After finding the optimal triangulation from the generated  $5 \times 5$  table, drawing the same triangulation of the pentagon based on non-crossing diagonals  $\delta_{ij}$  that correspond to the upper portion of the cells in the table; i.e., pair  $(i, j)$ :  $Optw = \{\delta_{14}, \delta_{24}\}$  (see Table 2).

## 4. Comparative analysis

For the purpose of analysis, we present the Hurtado–Noy algorithm for generating the polygon triangulation based on the predecessor’s results of triangulation; that is, triangulation  $P_n$  is obtained from  $P_{n-1}$ . This algorithm defines the triangulation tree of in the procedure of the  $n$ –vertex polygon triangulation. All of the  $T_n$  triangulations are arranged at level  $n$  of the tree and are used for calculating the weights in the process of storage. Each triangulation at level  $n$  has a “father” in  $T_{n-1}$  and two or more “sons” in  $T_{n+1}$ . The sons of the same father are “brothers”. There is an order among the children of a triangulation and, consequently, among all triangulations [11]. From the Hurtado–Noy algorithm, it follows that each triangulation at level  $n - 1$  is needed to perform  $2n - 5$  checks to find the diagonals that can be drawn from vertex  $n - 1$ . From this starting point for all possible triangulations, we see that  $(2n - 5)C_{n-3}$  is the number of checkings that can be taken. Further, we must go through the diagonals and copy some without transforming them, while some of them should be transformed, and two new diagonals should be inserted for each incident diagonal that has been found. In such a way, we make  $2n - 3$  pairs that describe one new triangulation.

The total number of incident diagonals is equal to  $C_{n-2}$ . All together, in the case of the Hurtado–Noy algorithm, the following number of weight calculations are given with Equation 5:

$$HN_w = (2n - 5)C_{n-3} + (2n - 3)C_{n-2} \quad (5)$$

The developed method of storing weights in this paper is compared with the Hurtado–Noy method and square matrix method [11].

In order to get a better overview for the developed method, we first need to analyze the calculations in the Hurtado–Noy and square matrix methods, which are needed to find the weight of each triangulation of the polygon separately. In [17], the square matrix method is presented for storing and finding the optimal triangulations of the polygon. In the square matrix method, the values of number  $k$  that are in the square matrix are determined in the beginning. According to the general scheme of this method in the filling of the matrix elements, number  $k$  increases as it goes further from the diagonal. In the first row of the matrix that is constructed by this method, there are no elements since they represent the so-called adjacent vertices; the second has one element, the third has two, and so on. The last diagonal row contains  $(n - 2)$ –vertices since the difference between the first and last  $n$ –vertex is always two, as the first and last vertices are always subtracted from the total [17]. The total number of stored  $k$ –vertices in an  $n$ –vertex polygon is labeled with  $V_n$ , which is given with the relationship in Equation 6.

$$V_n = \sum_{i=1}^{n-2} i(n - i - 1) \quad (6)$$

By using this relationship, the following results are obtained in this paper:  $V_5 = 10$ ,  $V_6 = 20$  and  $V_7 = 35$ . The number of calculated weights in the triangulation process is determined with  $SM_w = V_n$ .

Our method as a procedure is developed on the basis of the generic dynamic programming concepts that use the iteration technique through all subproblems in the calculation of the optimal triangulation. The iteration starts from the “smallest” and continues to the “biggest” subproblems. By using the previously computed optimal values of the smaller problems for each subproblem, the optimal value is found. The choices of recording the calculations of the weights allows us to obtain the optimal triangulation weight. At the case of pentagon triangulation, we have the value 35 as the optimal solution of the optimal triangulation problem. The calculations are made according to the relationship given in Equation 3. From requirement  $1 \leq j \leq i \leq n$  in relationship  $w[i, i + 1] = w[i, i] + w[i + 1, i + 1] + p_{i-1}p_i p_{i+1}$ , we have  $w[i, i] = 0$  and  $w[i, i + 1] = p_{i-1}p_i p_{i+1}$  for  $i = j$ . These values are the weights of the vertices and edges of the pentagon. The other calculations are made in the following order:

- **Step 1.**  $w[1, 2] = p_0 p_1 p_2 = 5 \cdot 1 \cdot 5 = 25, k = 1$   
 $w[2, 3] = p_1 p_2 p_3 = 1 \cdot 5 \cdot 1 = 5, k = 2$   
 $w[3, 4] = p_0 p_1 p_2 = 5 \cdot 1 \cdot 5 = 25, k = 3$
- **Step 2.**  $w[1, 3] = w[1, 1] + w[2, 3] + p_0 p_1 p_3 = 0 + 5 + 5 \cdot 1 \cdot 1 = 10, k = 1$   
 $w[1, 3] = w[1, 2] + w[3, 3] + p_0 p_2 p_3 = 25 + 0 + 5 \cdot 5 \cdot 1 = 50, k = 2$
- **Step 3.**  $w[2, 4] = w[2, 2] + w[3, 4] + p_1 p_2 p_4 = 0 + 25 + 1 \cdot 5 \cdot 5 = 50, k = 2$   
 $w[2, 4] = w[2, 3] + w[4, 4] + p_1 p_3 p_4 = 5 + 0 + 1 \cdot 1 \cdot 5 = 10, k = 3$
- **Step 4.**  $w[1, 4] = w[1, 1] + w[2, 4] + p_0 p_1 p_4 = 0 + 10 + 5 \cdot 1 \cdot 5 = 35, k = 1$   
 $w[1, 4] = w[1, 2] + w[3, 4] + p_0 p_2 p_4 = 25 + 25 + 5 \cdot 5 \cdot 5 = 175, k = 2$   
 $w[1, 4] = w[1, 3] + w[4, 4] + p_0 p_3 p_4 = 10 + 0 + 5 \cdot 1 \cdot 5 = 35, k = 3$

In the triangulation of the pentagon, there are ten diagonals with repetition; from these, only six are used in the calculation of the optimal triangulation with the memorization algorithm. Our method use memoized matrix chain product [7]; due to this, the rows in Steps 2 ( $k = 2$ ), 3 ( $k = 2$ ), and 4 ( $k = 2$  and  $k = 3$ ) are skipped. Our algorithm obtains the optimal triangulation from the upper portion of table  $n \times n$ . We notice that we have  $n - 2$  calculations in the first row of the table,  $n - 3$  calculations in the second, and  $n - (n - 1)$  calculations in the  $n - 1$ th. From here in the optimal triangulation of the  $n$  vertex polygon for the total number of saved calculations, we have

$$MM = n - 2 + n - 3 + \dots + n - (n - 1) = n - 2 + n - 3 + \dots + 2 + 1 = \frac{(n - 2)(n - 1)}{2} \quad (7)$$

different savings. In Table 3, a comparative analysis of our method with the square matrix and Hurtado–Noy methods is given according to Equation 7.

**Table 3**  
Comparative analysis: numbers of operations

$n$ -gon	HN	SM	MM	S(HN, MM)	S(SM,MM)	R(SM,MM)
7	45	35	15	30	20	2.33
8	161	56	21	140	35	2.67
9	588	84	28	560	56	3.00
10	2178	120	36	2142	84	3.33
11	8151	165	45	8106	120	3.67
12	30745	220	55	30690	165	4.00
13	116688	286	66	116622	220	4.33
14	445094	364	78	445016	286	4.67

Legends: *HN*(Hurtado–Noy method), *SM* (square matrix method), *MM* (memoization method), *S* (savings), *R* (ratio, speedup).

## 5. Experimental results

In our method, the optimal triangulation determination process is based on access/input to weight storing on filling the upper part of the table with subproblem solutions. While using the memoization technique in the table, the solutions of the subproblem are filled only once; the same results are not calculated several times. Each cell in the table has its own entry that shows that the weight of a particular triangle in the triangulation process is calculated the first time and then stored in the upper part of the table.

Each subsequent time when the weight of a triangle that is part of the triangulation is required, it is first checked whether this value is stored in any cell of the table. If this value is previously stored in the table, it is not calculated again but is used. If it is not stored, this value is computed for the first time and stored in the adjacent cell in the table.

In the upper part of the table constructed with our method are all values of  $w[i, j]$  that represent the solutions of the optimal triangulation problem of the convex polygon. Each cell in the table is initially assigned with infinity, which indicates that the input has yet to be filled in that cell of the table. In the calculation of  $Optw$ , if  $w[i, j] < \infty$ , then we return to the previous  $w[i, j]$ ; else, calculate  $w[i, j]$  and store it in field  $(i, j)$  of the table and return. In determining the value of  $Optw$  the return  $w[i, j]$  is calculated only if it first appears where the  $Optw$  is called with parameters  $i$  and  $j$ .

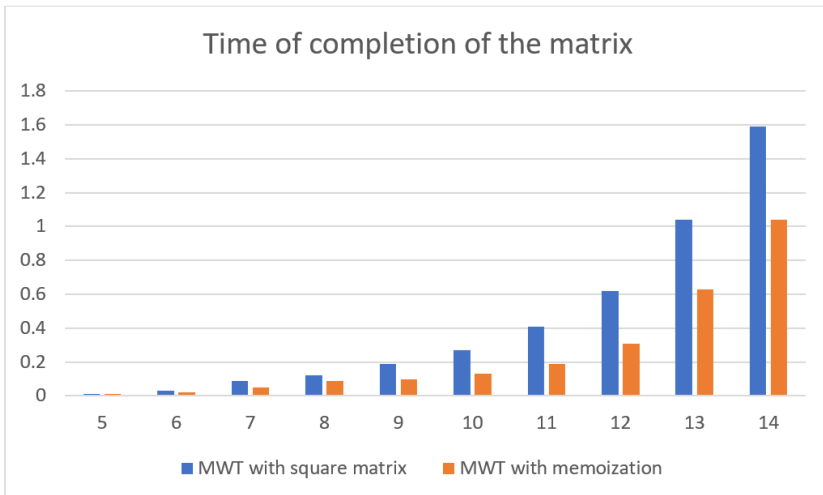
Table 4 presents the testing results for  $n = \{5, 6, \dots, 14\}$  and shows the calculation time for two methods: 1) the MWT method, which is based on the square matrix, and 2) the MWT method, which is based on memoization (A – Filling time of the matrix without the graphical generation of the minimum triangulation, and B – total execution time; i.e., Finding optimal triangulation with plotting, Ratio (R) for cases A and B) (see Figures 5, 6, 7).

The main class `OptimalTriangulation` was developed in the Java NetBeans environment. This main class has the method computed that is responsible for calculating all of the weights in the table. The `DefaultTableCellRenderer` from the `Swing` package was used in working with the table cells. This class inherits the table class and allows manipulation over the table cells (in this case, it allows us to assign a series of independent values to one cell of the table). In addition to this class, `JTable`, `TableCellRenderer`, and `BasicTableUI` from the `Swing` package were also used. The obtained values can be recorded in the form of the table in the execution of the Java NetBeans application in the service of working with databases.

**Table 4**

Comparative analysis (in seconds): MWT square matrix (MwtS) vs. MWT memoization (MwtM)

$n$ -gon	T	MwtS(A)	MwtS(B)	MwtM(A)	MwtM(B)	R(A)	R(B)
5	5	0.01	1.7	0.01	0.8	1.00	2.13
6	14	0.03	2.4	0.02	1.4	1.50	1.71
7	42	0.09	2.8	0.05	2.1	1.80	1.33
8	132	0.12	3.7	0.09	3.0	1.33	1.23
9	429	0.19	5.2	0.10	4.7	1.90	1.11
10	1430	0.27	14.8	0.13	13.3	2.08	1.11
11	4862	0.41	27.4	0.19	24.9	2.16	1.10
12	16796	0.62	48.7	0.31	44.1	2.00	1.10
13	58786	1.04	64.6	0.63	59.3	1.65	1.09
14	208012	1.59	85.5	1.04	78.5	1.53	1.09



**Figure 5.** Matrix completion time

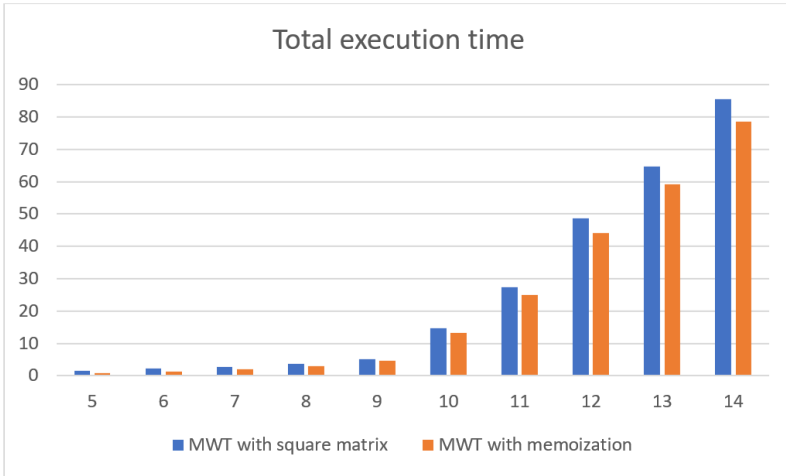


Figure 6. Total execution time



Figure 7. Ratio (speedup): MWT square vs MWT memoization (A – Filling time of matrix without graphical generation; B – total execution time)

In this case for testing the module of the "NetBeans" environment, the profiler for CPU testing was used. The testing was performed over a computer analysis with the following performance: CPU – Intel Core2 Duo, 2.40GHz, Cache 4MB, RAM: 2Gb, Graphics: NVIDIA GeForce 8600M GS.

The significance of this method of storage reflects the fact that the obtained values from the above method can effectively provide the drawing of the least weighted triangulation, provided that there is a method that will ensure the generation of all of the triangulations. In addition to the speed of searching for and plotting the optimal triangulation, it is important to emphasize that this saves memory.

## 6. Conclusion and further works

This paper describes an algorithm that finds the optimal triangulation of a convex polygon based on the dynamic programming technique. As a part of dynamic programming, memoization is the technique used in the algorithm. As an approach, memoization is very effective and suitable for solving certain types of complex problems. The basic idea in the method developed in this research is to avoid multiple calculations of the same value by using additional space in which the values are stored among the results.

The constructed algorithm is formulated using the rows and columns of an  $n \times n$  table. The algorithm finds the optimal triangulation on the principle of row/column pairings on different sides of a table diagonal. An optimization task is achieved via the memoization technique that corresponds to the values found in the upper and lower parts of the table. The complete cell fill strategy in the table is developed based on the calculation of the weights of the triangles that are parts of the polygon triangulation. The filling is done by finding the mutual matching of the values between the cell tables found on different sides of the diagonal.

The presented algorithm provides a good basis for expanding this technique in the case of concave polygons and polyhedrons. The optimal triangulation process of a concave polygon would begin by clipping off the ears, which would end up with another closed polygon that would itself possess two ears to clip off. This procedure would be repeated until no more ears are left to clip off. The ear-clipping strategy would be developed using the memoization technique. By choosing an interior point and drawing the edges to the three vertices of the triangle that contains this point, our algorithm can be developed in the case of 3D shapes. The algorithms would be ended when all of the interior points of the polyhedron are finished.


## References

- [1] Attene M., Campen M., Kobbelt L.: Polygon Mesh Repairing: An Application Perspective, *ACM Computing Surveys*, vol. 45, pp. 1–33, 2013.
- [2] Barequet G., Dickerson M., Eppstein D.: On triangulating three-dimensional polygons. In: *Proceedings of the twelfth annual symposium on Computational geometry, SCG'96*, ACM, pp. 38–47, 1996.
- [3] Barequet G., Sharir M.: Filling gaps in the boundary of a polyhedron, *Computer Aided Geometry Design*, vol.12, pp. 207–229, 1995.
- [4] Batty C., Xenos S., Houston B.: Tetrahedral Embedded Boundary Methods for Accurate and Flexible Adaptive Fluids, *Computer Graphics Forum (Eurographics)*, vol. 29, pp. 695–704, 2010.
- [5] Bessmeltsev M., Wang C., Sheffer A., Singh K.: Design-Driven Quadrangulation of Closed 3d Curves, *VACM Transactions on Graphics, SigGraph Asia*, vol. 31(5), 2012.

- [6] Chazelle B.: Triangulating a Simple Polygon in Linear Time, *Discrete Computational Geometry*, vol. 6, pp. 485–524, 1991.
- [7] Cormen T.H., Leiserson C.E., Rivest R.L., Clifford S.: *Introduction to Algorithms*, Third Edition, MIT Press, 2009.
- [8] Gilbert P.D.: New results in planar triangulations, *Technical report, Urbana, Illinois: Coordinated Science Laboratory*, University of Illinois, 1979.
- [9] de Goes F., Breeden K., Ostromoukhov V., Desbrun M.: Blue Noise Through Optimal Transport, *ACM Transactions on Graphics*, vol. 31(6), pp. 1–11, 2012.
- [10] Jin M., Kim J., Luo F., Gu X.: Discrete surface ricci flow, *IEEE Transactions on Visualization and Computer Graphics*, vol. 14(5), pp. 1030–1043, 2008.
- [11] Kenneth R., Sheffer A., Wither J., Cani M.P., Thibert B.: Developable Surfaces from Arbitrary Sketched Boundaries, *Proceedings of Eurographics Symposium on Geometry Process*, 2007.
- [12] Klincsek G.T.: Minimal Triangulations of Polygonal Domains, *Combinatorics 79, Annals of Discrete Mathematics*, vol. 9, Elsevier, 1980.
- [13] Liu L., Bajaj C., Deasy J., Low D.A., Ju T.: Surface Reconstruction from Non-Parallel Curve Networks, *Computer Graphics Forum*, vol. 27(2), pp. 155–163, 2008.
- [14] Mašović S., Saračević M.: Finding optimal triangulation based on block method, *Southeast Europe Journal of Soft Computing*, vol. 3(2), pp. 14–18, 2014.
- [15] Mercat C.: Discrete Riemann surfaces and the Ising model, *Communications in Mathematical Physics*, vol. 218, pp. 177–216, 2001.
- [16] Roth G., Wibowoo E.: An Efficient Volumetric Method for Building Closed Triangular Meshes from 3-D Image and Point Data. In: *Proceedings of the Conference on Graphics Interface'97, Canadian Information Processing Society*, pp. 173–180, 1997.
- [17] Saračević M., Mašović S., Stanimirović P., Krtolica P.: Method for finding and storing optimal triangulations based on square matrix, *Applied Sciences Electronic Journal*, vol. 20, pp. 167–180, 2018.
- [18] Saračević M., Selimi A.: Convex polygon triangulation based on planted trivalent binary tree and ballot problem, *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 27(1), pp. 346–361, 2019.
- [19] Seidel R.: A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons, *Computational Geometry*, vol. 1, pp. 51–64, 1991.
- [20] Selimi A., Saračević M.: Computational geometry applications, *Southeast Europe Journal of Soft Computing*, vol. 7(2), pp. 8–15, 2018.
- [21] Stanimirović P.S., Krtolica P.V., Saračević M.H., Mašović S.H.: Block Method for Convex Polygon Triangulation, *Romanian Journal of Information Science and Technology*, vol. 15(4), pp. 344–354, 2012.



## Affiliations

**Aybeyan Selimi** 

International Vision University, Faculty of Informatics, Gostivar, aybeyan@vizyon.edu.mk,  
ORCID ID: <https://orcid.org/0000-0001-8285-2175>

**Samedin Krrabaj**

University of Prizren, Faculty of Computer Science, samedin.krrabaj@uni-prizren.com

**Muzafer Saračević** 

University of Novi Pazar, Department of Computer Science, muzafers@uninp.edu.rs,  
ORCID ID: <https://orcid.org/0000-0003-2577-7927>

**Selver Pepić**

Educons University, Department of Computer Science, Novi Sad, selverp@gmail.com

**Received:** 02.03.2019

**Revised:** 26.04.2019

**Accepted:** 26.04.2019