FACTA UNIVERSITATIS (NIŠ)
SER. MATH. INFORM. Vol. 29, No 2 (2014), 189–208

TRIANGULATION OF CONVEX POLYGON WITH STORAGE SUPPORT *

Predrag Krtolica, Predrag Stanimirović, Milan Tasić, Selver Pepić

Abstract. Unlike the algorithms for convex polygon triangulation which make the triangulation of an n-gon from the scratch, we propose the algorithm making the triangulation of an $(n+1)$-gon on the base of the already found triangulations of an n-gon. For such a purpose we must maintain suitable file storage to store previously derived triangulations and later use them to generate the triangulations of polygon with one more vertex. The file storage is partially exploited for the elimination of the duplicates our algorithm produces. Yet, the generation and elimination of duplicates do not critically decrease our algorithm performances for smaller values of n.

1. Introduction and Preliminaries

The convex polygon triangulation problem represents finding all possible polygon splittings on triangles by its diagonals without gaps and overlaps of these splittings. This is a classical problem that has so far been solved in several ways.

Triangulation of a convex polygon with n vertices requires $n-3$ nonintersecting internal. For convex polygons, all diagonals are internal diagonals. In this case, the number of triangulations of a convex n-gon is independent of shape, and therefore, it can uniquely be characterized by the number of vertices n. It is well known that the number of triangulations of a convex n-gon is equal to the $(n-2)$-th Catalan number [3, 4], i.e.

$$C_{n-2} = \frac{1}{n-1}\binom{2n-4}{n-2},\ n \geq 3.$$

This number grows rapidly. In the following section, we describe our attempt to employ file storage to make this task less tedious and to eliminate the repetition of calculations.

In this paper we try to get the $(n+1)$-gon triangulations on the base of the n-gon triangulations. A similar approach was used by Hurtado and Noy in [2] where

Received March 17, 2014.; Accepted June 9, 2014.
2010 *Mathematics Subject Classification.* Primary 68P20; Secondary 68W30,32B25
*The authors were supported in part by the Research Project 174013 of the Serbian Ministry of Science.

the tree of the convex polygon triangulations is presented. Also, the algorithm for generating the triangulations of $(n+1)$-gon on the base of n-gon triangulations is proposed. We will use this distinguished algorithm for comparison purposes later in the paper. Because of this, in Algorithm 1.1 we present the Hurtado's algorithm in an appropriate algorithmical form.

Algorithm 1.1 Hurtado's algorithm from [2].

Require: Positive integer n and the triangulations of an n-gon. Each triangulation is described as a structure containing $2n-3$ vertex pairs presenting n-gon diagonals (here *diagonals* means both internal diagonals and outer face edges).

1: Check the structure containing $2n-3$ vertex pairs looking for pairs (i_k, n), $i_k \in \{1, 2, \ldots, n-1\}, 2 \leqslant k \leqslant n-1$, i.e. diagonals incident to vertex n. The positions of these indices i_k within structure describing a triangulation should be stored in the array.

2: For every i_k perform the transformation $(i_l, n) \rightarrow (i_l, n+1)$, $i_l < i_k$, $0 \leqslant l \leqslant n-2$.

3: Insert new pairs $(i_k, n+1)$ and $(n, n+1)$.

4: Take next i_k, if any, and go to Step (2).

5: Continue the above procedure with next n-gon triangulation (i.e. structure with $2n-3$ vertex pairs) if any. Otherwise halt.

The implementation of Hurtado's algorithm in several languages is described in [7].

The paper is organized as follows. In the second section we present our algorithm for the polygon triangulation with file storage support. The application details are given in the third section. Numerical experience is presented in the fourth section. Algorithm complexity is described in the fifth section. Implementation details are described in the Appendix.

2. The Algorithm

As the algorithm presented in [5] is recursive, finding one particular triangulation is equal to passing via one particular path of the corresponding expression tree. It means that much of the computational effort is repeated.

We have an idea to employ file storage and use the already completed work without redundant repetitions. Particulary, if we have found all triangulations for a n-gon, we make only the necessary actions to get all triangulations of a $(n+1)$-gon.

We use a set of internal diagonals (i.e. an $(n-3) \times 2$ matrix where every row contains ending vertices of the diagonal) as a data structure presenting one triangulation. The procedure is based on transforming the outer face edge into the internal diagonal which is equal to transformation of this edge into the triangle.

We are aware that we will get some duplicates, which should be eliminated. Storage support is used for this purpose, too.

Algorithm 2.1 Algorithm with only diagonals stored.

Require: Positive integer $n \geq 3$.

1: For every triangulation of a n-gon store the collection of diagonals sets $E_d(n) = \{E_d^1(n), \ldots, E_d^{C_{n-2}}(n)\}$, where each $E_d^i(n)$ contains diagonals describing one particular triangulation. For the triangle $E_d(3) = \emptyset$. Also, generate the set of outer edges for a n-gon

$$E_o(n) = \{(i, (i+1) \bmod n) \mid i = 1, \ldots, n\}.$$

What we really need in the further steps is the set

$$E_b(n) = E_o(n) \setminus \{(1,2)\}.$$

2: Denote by $T_d(n)$ the set of different elements in $E_d(n)$. To make all triangulations (or collection of sets $E_d(n+1)$) of the $(n+1)$-gon from every set $T_d^i(n)$ make additional $n-1$ sets $E_d^{i_k}(n+1)$, $k = 1, \ldots, n-1$ of $(n+1)$-gon diagonals by including one of the members of the set $E_b(n)$. Therefore,

$$E_d^{i_k}(n+1) = T_d^i(n) \cup \{E_b(n)[k]\},\ i \in \{1, \ldots, C_{n-2}\}, k = 1, \ldots, n-1. \tag{2.1}$$

In the general case, we have

$$E_d^i(n+1) = T_d^{\lfloor i/n \rfloor + 1}(n) \cup \{E_b(n)[\langle i, n\rangle]\},\ i = 1, \ldots, (n-1)C_{n-2}, \tag{2.2}$$

where $\lfloor \cdot \rfloor$ denotes the floor function and

$$\langle i, n\rangle = \begin{cases} i \bmod (n-1), & i \bmod (n-1) \neq 0, \\ n-1, & i \bmod (n-1) = 0. \end{cases}$$

3: If we include a pair $(k, k+1)$, then increase by 1 every pair member (from the set $E_d^i(n+1)$) greater than or equal to $k+1$, including the newly added. Let us observe that when the pairs of the form $(1, n)$ are included, the increment is not needed.

The number of duplicates generated by Algorithm 2.1 is equal to

$$(n-1)\text{size}(T_d(n)) - \text{size}(T_d(n+1)) = (n-1)C_{n-2} - C_{n-1} = \frac{(n-2)(n-3)}{n} C_{n-2},$$

where the operator size$(\cdot)$ denotes the number of elements in a set.

Example 2.1. Let us start from $T_d(3) = \emptyset$ and

$$E_b(3) = E_o(3) \setminus \{(1,2)\} = \{(2,3),(3,1)\}.$$

Applying (2.2), we get the collection $E_d(4) = \{E_d^1(4), E_d^2(4)\}$ corresponding to the square triangulations:

$$\begin{aligned} E_d^1(4) &= T_d(3) \cup \{E_b(3)[1]\} = \{(2,3)\} \to \{(2,4)\} \\ E_d^2(4) &= T_d(3) \cup \{E_b(3)[2]\} = \{(3,1)\}. \end{aligned}$$

The set $E_b(3)$ has only two elements so we could get only two sets $E_d^i(4)$. Note that we have had to update elements in $E_d^1(4)$ according to Step (3) of Algorithm 2.1.

Now, we will make triangulations of a pentagon on the base of stored sets $E_d^i(4)$ and easily generated set $E_b(4) = \{(2,3),(3,4),(4,1)\}$. According to Algorithm 2.1, it is easy to verify

$$E_d(5) = \{E_d^i(5),\ i = 1,\ldots,2*3\} = \{E_d^i(5),\ i = 1,\ldots,C_2*3\}.$$

Therefore, since $C_3 = 5$, one triangulation will be excessive.

$$\begin{aligned} E_d^1(5) &= T_d^1(4) \cup \{E_b(4)[1]\} = \{(2,4),(2,3)\} \to \{(2,5),(2,4)\} \\ E_d^2(5) &= T_d^1(4) \cup \{E_b(4)[2]\} = \{(2,4),(3,4)\} \to \{(2,5),(3,5)\} \\ E_d^3(5) &= T_d^1(4) \cup \{E_b(4)[3]\} = \{(2,4),(4,1)\} \\ E_d^4(5) &= T_d^2(4) \cup \{E_b(4)[1]\} = \{(3,1),(2,3)\} \to \{(4,1),(2,4)\} \text{ (the same as } E_d^3(5)) \\ E_d^5(5) &= T_d^2(4) \cup \{E_b(4)[2]\} = \{(3,1),(3,4)\} \to \{(3,1),(3,5)\} \\ E_d^6(5) &= T_d^2(4) \cup \{E_b(4)[3]\} = \{(3,1),(4,1)\} \end{aligned}$$

3. Application and Implementation Details

Our application is presented by client/server model and it is implemented using the PHP/MySQL environment for the following reasons [1, 9].

PHP is a specialized script language originally aimed for producing Web pages. It is a simple language with C-like syntax with facilities for dynamic memory allocation, high performances, possibilities to be integrated with many DBMS's and good portability.

MySQL is an open source software with high performances and reliability. Its usage is simple but functionality is powerful. Free technical support is easily obtained. It works on different platforms and in multiuser environments.

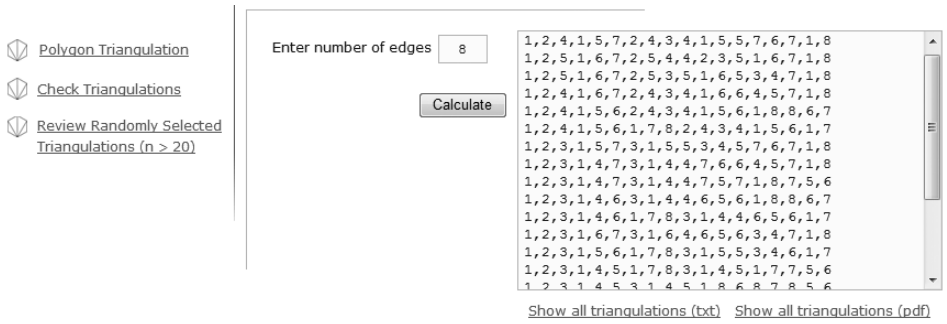

FIG. 3.1: Web interface of application for triangulations display.

The implementation of the web database application is based on the three-tier architecture. The database tier consists of the database management system and presents the basis of the three-tier architecture [9]. A complex middle tier, which contains most of the application logic and communicates data between the other tiers, is at the top of the database tier. Usually web browser software interacts with the application and it is placed at the top of the client tier. Most of three-tier web database systems contain the application logic in the middle tier. The client tier presents data to the user and collects data from the user.

The application is the extension of the application from [8] and it is presented by combinations of *PHP*/*MySQL* elements and implemented by special database operations, which support the SQL-aware implementation of a wide range of classification algorithms.

For this approach, a suitable Web interface, shown in figures 3.1–3.3, has been developed.

In the left part of the web page presented in Figure 3.1 there is a navigator, where the user selects the link for the calculation of all possible triangulations for a given n-gon. The page with a field to enter the polygon edges opens by selecting a link.

This page presents the possibility to show the obtained solutions in the form of `pdf` or `txt` file. All previously generated results are memorized in the database. If for some n, the triangulations are already generated, they will be displayed from the database. Otherwise, the triangulation algorithm is engaged.

Application could check user obtained triangulation. The user enters the elements describing the triangulation in the text area, chooses the element separator and clicking on <*Calculate*> button gets the message about entered triangulation validity. This option is illustrated in Figure 3.2.

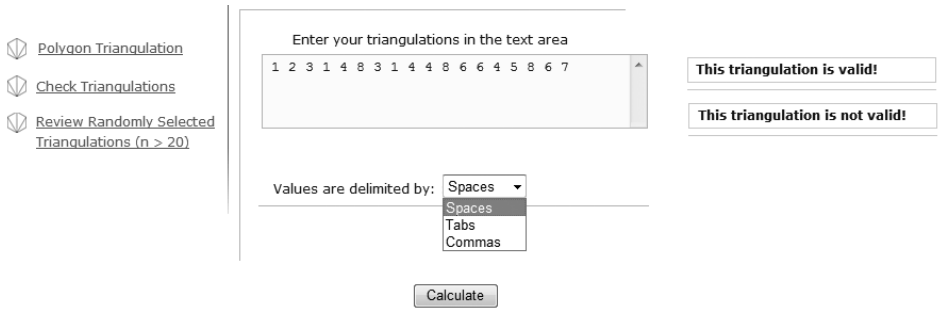

FIG. 3.2: Page that allows the user to determine the validity of triangulation.

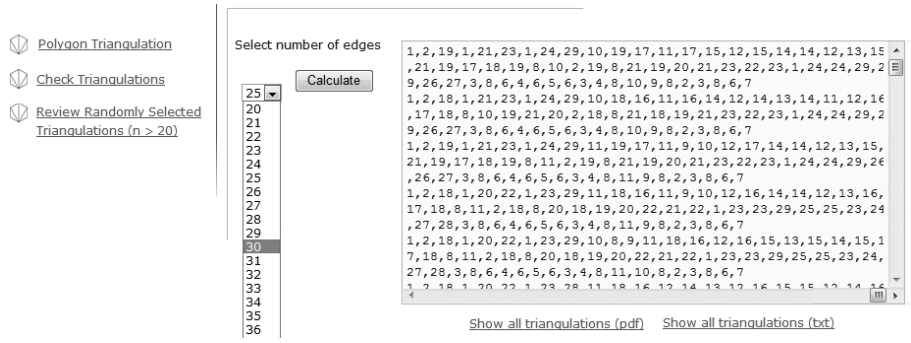

FIG. 3.3: Page that allows the review of the randomly selected triangulation.

In Figure 3.3 the facility for listing randomly chosen triangulations when $n > 20$ is displayed. The user chooses the number of polygon edges from the dropdown list and gets a display in the text area.

In our implementation we use three different strategies, called the *Application Logic* (*AL*), *Application Logic+Database* (*ALD*) and *Application Logic+Database+ Files* (*ALDF*).

Approach *AL* uses only the arrays of strings, where each string represents a triangulation. The main properties of this approach are denoted by **AL1** - **AL3**.

AL1: The main disadvantage is that the generated triangulations are not stored for further usage.

AL2: Our algorithms generate some duplicate triangulations which have to be removed. This removal will be payed by a significant computational cost (operating memory consumption could be a problem too). Built-in PHP function called `array_unique()` takes an input array and returns a new array without duplicate values. Function `array_unique()` firstly sorts the values treated as strings, keeping the first key encountered for every value, and ignores all following keys.

AL3: The number of triangulations of n-gon grows rapidly with n. The *AL* approach uses array to store triangulations, which is the reason of relatively quick memory exhaustion. The maximum size of an array in PHP depends on `memory_limit` directive in the `php.ini` configuration file. There is not a limit on the size of an array, but there is a limit on the size of the memory allocated to PHP script.

Consequently, we need to use secondary memory facilities to store triangulations, which implies the necessity of the *ALD* approach. This approach implies DBMS support to the application logic. Instead of storing every new triangulation $E_d^i(n+1)$ in the database immediately upon its generation, we generate data in blocks size up to 5 million triangulations and then record them in a database. In this way we reduce the storage time.

The properties of the *ALD* approach, denoted by **ALD1** - **ALD4**, should be pointed out.

ALD1: Firstly, it solves the problem of the memory exhaustion.

ALD2: Moreover, in *ALD* approach, duplicates removal could be automatized with unique key usage for every triangulation (i.e. set of diagonals in the case of Algorithm 2.1). If we have already stored particular triangulation in the database, recording of the duplicate one will be denied.

ALD3: Also, the triangulation correctness could be checked upon a simple query on database table.

ALD4: The drawback of the *ALD* approach is frequent reading and writing data, which is time consuming.

The approach *ALDF* solves the disadvantage ALD4. Database approach is necessary to avoid re-computations and the memory exhaustion as well as to store generated triangulations. On the other hand, in order to avoid a permanent triangulations retrieval and recording, we decide to involve a text file support. Because of their simplicity, text files are commonly used for storage of information that are structured as a sequence of lines. In our case this is a logical choice because we present triangulations as strings. *AL* approach is the fastest in removing duplicates, while the duplicates elimination is the slowest in text files.

The *ALDF* approach means that the elements of $T_d(n)$ are retrieved from the database and temporarily stored in the text file during the process of $E_d(n+1)$ generation. Each triangulation from $E_d(n+1)$ remains in the array until the maximal array index (the number of triangulations) is reached.

For the polygons with $n < 15$ vertices the whole set $E_d(n+1)$ (block) is placed in the text file. At this moment, duplicate triangulations in that block are removed using the *PHP* function *array_unique*(). Furthermore the contents of the text file is inserted into the database with a simple *SQL* statement `LOAD DATA LOCAL INFILE` (see Function 6).

For the polygons with $n \geq 15$ vertices, when the number of elements in the array reaches the maximum capacity of the block (in our implementation it is 5 million triangulations), duplicates within the block are removed firstly on the *AL* level, using the `array_unique()` function. After that, the whole block without duplicates is transferred to a text file. This procedure repeats with the next block until all triangulations are generated. In this way, in a text file duplicate triangulations from different blocks could be found. This is resolved during the database storage. More precisely, DBMS is used for efficient transformation of the set $E_d(n+1)$ into the set $T_d(n+1)$.

The best results are achieved by the *ALDF* approach, because we avoid recomputations and frequent storage. Besides, the user can see the results simply by querying the table. Also, we solved the problem of memory congestion and we provided a faster algorithm for data manipulations.

4. Experimental Results

The application is implemented obeying requirements of three-layer web architecture: client layer, application layer and database layer. The testing is done using application logic and using client/server model (using database).

Testing was done on the local machine and from a client in a wireless network. We had an access to the web server using the infrastructure mode wireless networking with an access point.

Testing was executed on the server machine with: Windows edition: *Windows Vista*$^{(TM)}$ *Ultimate*; Processor: *Intel*(*R*) *Pentium*(*R*) *Dual CPU T3200 @ 2.00GHz*; Memory (*RAM*) : 2940*MB*; System type: 32 – *bit Operating System*; Free Softwares: *PHP* 5.2.5, *MySQL* 5.0.45 and *phpMyAdmin* 2.11.2.1.

CPU times comparisons for Algorithm 2.1 and three different approaches, as well as for Hurtado's algorithm from [2], are given in Table 4.1, wherein the sign * means that operating memory exhausted.

As it is supposed, Table 4.1 illustrates that usage of *ALDF* approach for Algorithm 2.1 gives the best way to decrease the CPU time and memory consumption. Nevertheless Algorithm 2.1 produces the duplicates which must be eliminated, for $n \leqslant 11$ it is better even than Hurtado's algorithm, while for $n > 11$ its performances are not far away from the Hurtado's algorithm performances. This is a little bit surprising result and we try to explain this in Section 5.

For the input file with $C_7 = 429$ rows using the `LOAD DATA LOCAL INFILE` statement requires 0.066 seconds, and to select 1000 records from a table 0.000617 seconds are required. Time for data insertion in the database table as well as the time required to select data from the table do not depend on the total number of records in the table.

In order to shorten execution time for large n we have split the table with previously calculated triangulations into $k \geq 2$ parts. The parallelization process

Table 4.1: Times for Algorithm 2.1 with the database approach and file support vs. Hurtado's Algorithm.

n	$AL[n]$	$ALD[n]$	$ALDF[n]$	$Hurtado[n]$	$HurtadoALF[n]$
7	0. 007	0.086	0.006	0.009	0.0088
8	0. 019	0.189	0 .016	0.038	0.0285
9	0. 090	0.421	0.060	0.122	0.0967
10	0. 336	1.168	0.249	0.427	0.366
11	1. 301	3.562	1.035	1.978	1 .072
12	5. 955	10.276	4.620	5.626	4.002
13	26.909	40.634	18.166	23 .532	14.162
14	113.798	153.248	77.090	81.357	55.942
15	489.271	598.859	353.539	393.420	234.804
16	2103.860	2508.273	1798.729	1982.358	1540.989
17	*	10717 .120	9741.606	*	6443.415
18	*	44122.383	39564.870	*	27069.430
19	*	*very long time*	162452.920	*	115315.772

Table 4.2: Times for Algorithm 2.1 with the separated files support.

separate pieces	$n = 13$	$n = 14$	$n = 15$	$n = 16$
2	12.122	55.388	246.758	1342.577
3	12.568	55.463	259.880	1499.280
4	12.865	63.737	260.563	1530.782
5	14.643	64.237	287.844	1571.957

is simulated with more clients in the local network, where each of them generates triangulations from one part on a single server.

We get the best processing time in the case when the input file is divided in two parts processed by two PHP scripts. Table 4.2 contains CPU times for segmented files.

Further research could be directed to perform some computations in the network based on n hosts and m clients. The computation time would be obviously much shorter in that case.

In the case when we reach the maximum table capacity (for some n) we will observe the MySQL Cluster implementation. MySQL Cluster allows data sets larger than the capacity of a single machine. Data could be stored and accessed across multiple machines. MySQL Cluster is implemented through an additional storage engine available within MySQL called NDB or NDBCLUSTER ("NDB" stands for Network Database).

The application allows display of 100 randomly chosen triangulations for $n > 20$, which is presented in the third section. CPU times for this case are in the Table 4.3.

Table 4.3: Randomly selected triangulations for $n > 20$.

n	CPU time[s]
20	0.242
25	0.329
30	0.427
35	0.631
40	0.878
45	1.226
50	1.399

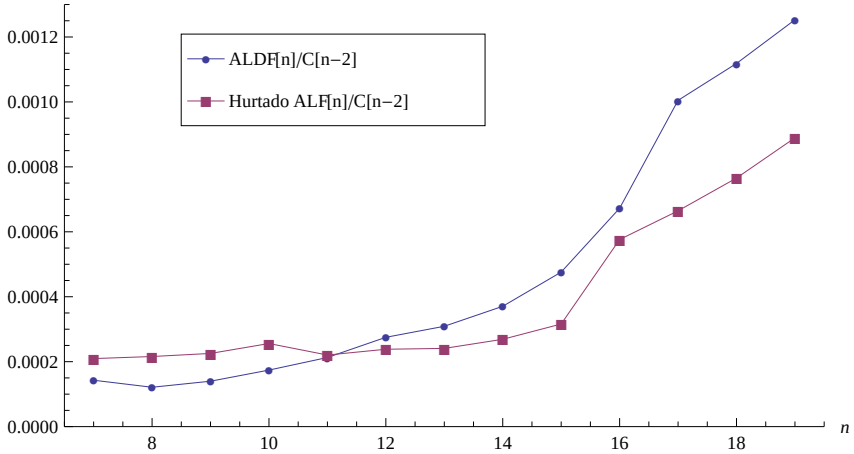

FIG. 4.1: Values for $ALDF[n]/C_{n-2}$ and $HurtadoALF[n]/C_{n-2}$

It is obvious that when we take a small set of data, our algorithm can reach a much larger n in a short time. If we take, for example, 100 randomly selected triangulations of $(n-1)$-gon, we get very quickly triangulations for polygons with a greater number of edges, as it is shown in Table 4.3.

4.1. Interpolation of data

We use the MATHEMATICA package to investigate the results arranged in the columns *ALDF* and *Hurtado ALF* of Table 4.1. For more information about the package see for example [10]. Using the standard function `Fit` we find a least–squares fit to the list of values for the quotients $ALDF[n]/C_{n-2}$ and $HurtadoALF[n]/C_{n-2}$. These quotients represent the CPU time needed for generating particular triangulations for a given n–gon. Graphical illustration for $n = 7, \ldots, 19$ is given in Figure 4.1.

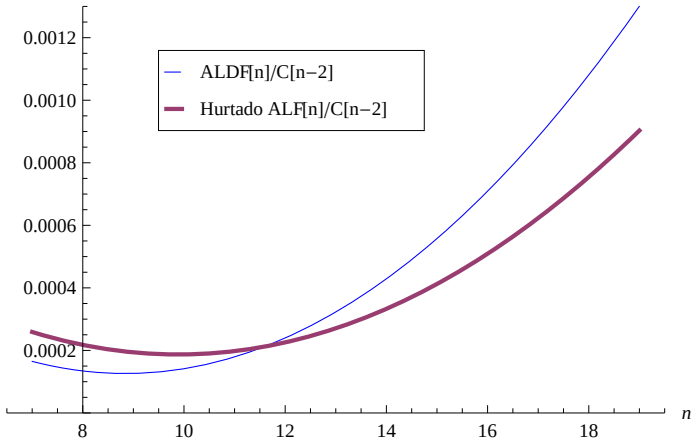

FIG. 4.2: Quadratic fits for $ALDF[n]/C_{n-2}$ and $HurtadoALF[n]/C_{n-2}$.

The quadratic fits to the sequences of values $ALDF[n]/C_{n-2}$ and $HurtadoALF[n]/C_{n-2}$ are equal to

$$0.0000113862x^2 - 0.000201402x + 0.00101689$$

and

$$8.622342551125387 \times 10^{-6}x^2 - 0.000170513x + 0.00103012,$$

respectively.

The graphical illustrations of these quadratic approximations are given in Figure 4.2.

5. Algorithm Complexity

Let us consider the complexity of producing a single triangulation $E_d^{i_k}(n+1)$, $1 \le i_k \le (n-1)C_{n-2}$ by Algorithm 2.1, applying (2.1).

Complexity of Step (1): To generate set $E_b(n)$ we need a loop of complexity $n-1$.

Complexity of Step (2): During the process of making the set $E_d^{i_k}(n+1)$ we need to include $E_b(n)[k]$ to a set $T_d^i(n)$. Denote by I the complexity of this inclusion.

Complexity of Step (3): The number of elements in $E_d^{i_k}(n+1)$ is $(n+1)-3 = n-2$. Therefore, the complexity of this step is $n-2$ operations on pairs.

The complexity of generating $E_d^{i_k}(n+1)$, defined by (2.1), is equal to $n-2+I$.

The complexity of generating all triangulations $E_d^{i_k}(n+1)$, $k = 1, \ldots, n-1$ is

$$(n-2+I)(n-1).$$

In total, since the number of elements in the set $T_d(n)$ is equal to C_{n-2}, we conclude that the complexity of generating $E_d(n+1)$ by Algorithm 2.1 is

$$Ef(n+1) = n - 1 + (n - 2 + I)(n - 1)C_{n-2}.$$

Note that the cardinality of $E_d(n+1)$ is $(n-1)C_{n-2}$ and the above complexity concerns the generation of the complete set $E_d(n+1)$.

Database storage time needed to store text file contents to database (and complete duplicate elimination) is denoted by $S(n+1)$ (complexity of transformation $E_d(n+1)$ into $T_d(n+1)$).

Therefore, overall complexity of Algorithm 2.1 is $Ef_o = Ef(n+1) + S(n+1)$.

Let us consider the complexity of Algorithm 1.1.

Complexity of Step (1): To check $2n-3$ pairs we need as many comparisons.

Complexity of Step (2): For every i_k we need l transformations. Maximal l is equal to $n-2$.

Complexity of Step (3): We need two insertions of pairs with complexity $2I$.

Complexity of Step (4): The maximal k is equal to $n-1$ and we need as many repetitions of Steps (2)-(3).

Overall efficiency for Algorithm 1.1 is

$$Ef_H(n+1) = [2n - 3 + (n - 2 + 2I)(n - 1)]\, C_{n-2}.$$

If we make a difference between the numbers of needed operations for these two algorithms we get

$$\begin{aligned} Ef_H(n+1) \; - \; & Ef(n+1) \\ = \; & [2n-3 + (n-2+2I)(n-1)]\, C_{n-2} - (n-1) - (n-2+I)(n-1)C_{n-2} \\ = \; & (2n-3)C_{n-2} + I(n-1)C_{n-2} - (n-1) \end{aligned}$$

which explains why our algorithm works faster than Hurtado's in some cases. Of course, our algorithm spends a certain amount of time for storage where elimination of duplicates is done. Hurtado's algorithm does not produce any duplicates and the time difference is reduced.

$$Ef_H(n+1) - Ef_o(n+1) = (2n-3)C_{n-2} + I(n-1)C_{n-2} - (n-1) - S(n+1)$$

It is difficult to estimate time needed to complete $S(n+1)$ operations of duplicate elimination and storage (it is DBMS dependent). But, for smaller n the number of duplicates is relatively small (one for $n=5$, 6 for $n=6$, and so on), the cost of its elimination is moderate and simplicity of Algorithm 2.1 prevails. For larger n the number of the generated duplicates is even larger than the number of "real" triangulations and cost of its elimination grows, decreasing Algorithm 2.1 performances.

Let us consider the complexity of producing a single triangulation $E_d^{i_k}(n+1)$, $1 \le i_k \le (n-1)C_{n-2}$ by Algorithm 2.1, applying (2.1).

Complexity of Step (1): To generate set $E_b(n)$ we need a loop of complexity $n-1$.

Complexity of Step (2): During the process of making the set $E_d^{i_k}(n+1)$ we need to include $E_b(n)[k]$ to a set $T_d^i(n)$. Denote by I the complexity of this inclusion.

Complexity of Step (3): The number of elements in $E_d^{i_k}(n+1)$ is $(n+1)-3 = n-2$. Therefore, the complexity of this step is $n-2$ operations on pairs.

The complexity of generating $E_d^{i_k}(n+1)$, defined by (2.1), is equal to $n-2+I$.

The complexity of generating all triangulations $E_d^{i_k}(n+1)$, $k = 1, \ldots, n-1$ is

$$(n-2+I)(n-1).$$

In total, since the number of elements in the set $T_d(n)$ is equal to C_{n-2}, we conclude that the complexity of generating $E_d(n+1)$ by Algorithm 2.1 is

$$Ef(n+1) = n-1+(n-2+I)(n-1)C_{n-2}.$$

Note that the cardinality of $E_d(n+1)$ is $(n-1)C_{n-2}$ and the above complexity concerns the generation of the complete set $E_d(n+1)$.

Database storage time needed to store text file contents to database (and complete duplicate elimination) is denoted by $S(n+1)$ (complexity of transformation $E_d(n+1)$ into $T_d(n+1)$).

Therefore, overall complexity of Algorithm 2.1 is $Ef_o = Ef(n+1) + S(n+1)$.

Let us consider the complexity of Algorithm 1.1.

Complexity of Step (1): To check $2n-3$ pairs we need as many comparisons.

Complexity of Step (2): For every i_k we need l transformations. Maximal l is equal to $n-2$.

Complexity of Step (3): We need two insertions of pairs with complexity $2I$.

Complexity of Step (4): The maximal k is equal to $n-1$ and we need as many repetitions of Steps (2)-(3).

Overall efficiency for Algorithm 1.1 is

$$Ef_H(n+1) = [2n-3+(n-2+2I)(n-1)]\,C_{n-2}.$$

If we make the difference between the numbers of needed operations for these two algorithms we get

$$\begin{aligned} Ef_H(n+1) \; - \; & Ef(n+1) \\ = \; & [2n-3+(n-2+2I)(n-1)]\,C_{n-2} - (n-1) - (n-2+I)(n-1)C_{n-2} \\ = \; & (2n-3)C_{n-2} + I(n-1)C_{n-2} - (n-1) \end{aligned}$$

which explains why our algorithm works faster than Hurtado's in some cases. Of course, our algorithm spends a certain amount of time for storage where elimination of duplicates is done. Hurtado's algorithm does not produce any duplicates and the time difference is reduced.

$$Ef_H(n+1) - Ef_o(n+1) = (2n-3)C_{n-2} + I(n-1)C_{n-2} - (n-1) - S(n+1)$$

It is difficult to estimate time needed to complete $S(n+1)$ operations of duplicate elimination and storage (it is DBMS dependent). However, for smaller n the number of duplicates is relatively small (one for $n = 5$, 6 for $n = 6$, and so on), the cost of its elimination is moderate and simplicity of Algorithm 2.1 prevails. For larger n the number of the generated duplicates is even larger than the number of "real" triangulations and cost of its elimination grows, decreasing Algorithm 2.1 performances.

6. Conclusion

We have suggested the algorithm for convex polygon triangulation based on triangulations already stored in database, corresponding to the polygon with one edge less. In this way, we do not repeat all work needed in the case of algorithm from [5].

Some computational and storage cost must be paid, but in return we could retrieve stored triangulations in very short time and use them for another purpose.

Used DBMS system also serves to prevent storing of duplicated triangulations generated by the algorithm.

This approach gives relatively good performance results in comparison with Hurtado's algorithm ([2]) which produces no duplicates at all.

The generation of different triangulations is independent and suitable for parallelization and execution on parallel machine.

Note that, for higher n, triangulations generation and storage in database need large amount of time (in days), because of very rapid growth of triangulations number. But, once we have filled database, retrieval time is very short. The reader should have in mind the nature of the problem: if someone wants to observe all triangulations of e.g. 18-gon, and he/she manages to check one triangulations per second, he/she still needs almost a year to complete the job. In the case of larger n we get quickly the situation that lifetime is not enough to complete browsing of all triangulations.

7. Appendix

7.1. Data Storage System

Our database application use single table database with no relations and foreign keys. Database contains one table called `triangulations`.

Table 7.1: Database table *triangulations*.

id	*diagonals*	*n*
1	2, 4, 2, 5	5
2	2, 5, 3, 5	5
3	2, 4, 4, 1	5
4	3, 1, 3, 5	5
5	3, 1, 4, 1	5

The unique field used in the tables is described as follows:

- `id`: identification number, defined as an autonumber;

- `diagonals`: string containing diagonals of polygon in the form of pairs, which are separated with the comma sign (sets $E_d(n)$).

- `n`: number of edges for the given polygon.

The field `diagonals` is declared as the primary key to prevent duplicates storage.

The structure of SQL code for generating table `triangulations` has the following form.

```
# Table structure for table ‘triangulations‘ CREATE TABLE
‘triangulations‘ (
    ‘id‘ int(11) NOT NULL auto_increment,
    ‘diagonals‘ varchar(180) NOT NULL,
    ‘n‘ int(3) NOT NULL,
     PRIMARY KEY  (‘diagonals‘),
) ENGINE=InnoDB DEFAULT CHARSET=latin1 MAX_ROWS=4294967295;
```

The example of the table for pentagon with stored diagonals (according to Algorithm 2.1) is presented in Table 7.1.

The effective maximum of the table size for MySQL databases is usually determined by the operating system constraints, not by MySQL internal limits [6]. We did not encounter a *full-table error* even though we used the InnoDB storage engine with the maximal number of rows equal to $2^{32} - 1$ and defined by $MAX\_ROWS = 4294967295$ rows. The main reason to limit the rows number in our table is to avoid limitations based on operating system file-size limits. The InnoDB storage engine maintains InnoDB tables within a tables space that can be created from several files. This enables a table to exceed the maximum individual file size [6].

7.2. Implementation Details of Algorithm 2.1

This algorithm gives the best results.

In the beginning, several auxiliary procedures used in our implementation are described.

Function 1. Increments every pair member in set of diagonals $E_d(n+1)$ greater than or equal to newly added element.

```
function increment($array,$last_num,$n){
    for($i=0;$i<2*($n-3);$i++){
        if($array[$i]>=$last_num)
            $array[$i]=$array[$i]+1;}
    return $array;
    }
```

Function 2. Creates set $E_b(n)$ from Algorithm 2.1.

```
function createEb($n){
    for($j=2,$z=0;$j<$n;$j++){
        $k=$j+1;
        $Eb[$z]="$j,$k";
        $z++;    }
    $Eb[$z]="$n,1";
    return $Eb;
    }
```

Function 3. Creates sets $E_d(n)$ without elements increment.

```
function setsEd($Ed_n-1,$Eb,$n){
    $Ed=array(array());
    for($j=0;$j<$n-2;$j++){
        $Ed[$j][0]=$Ed_n-1;
        $Ed[$j][1]=$Eb[$j];
        $Ed[$j]=$Ed[$j][0] . "," . $Ed[$j][1];    }
    return $array;
    }
```

Function 4. Lexicographic ordering by pairs of the set $E_d(n+1)$, generated by Algorithm 2.1.

```
function sort_string($array,$n){
    for($i=0,$z=0;$i<($n-3);$i++){
        for($j=0;$j<2;$j++){
            $array_2D[$i][$j]=$array[$z];
            $z++; }  }
    array_multisort($array_2D);
    for($i=0,$z=0;$i<($n-3);$i++){
        for($j=0;$j<2;$j++){
            $array_1D[$z]=$array_2D[$i][$j];
            $z++;  }   }
    $array_string=implode(",",$array_1D);
    return $array_string;
}
```

Function 5. Loads file into the database in order to prevent duplicate storage.

```
function file_mysql($i){ $file = $i . ".txt"; $query0 = ' LOAD
DATA LOCAL INFILE  $file'
    . ' INTO TABLE `triangulations`'
    . ' LINES TERMINATED BY \'\\n\''
    . ' FIELDS TERMINATED BY \';\''
    . ' (`diagonals`,`n`);';
$result0 = mysql_query($query0); }
```

Function 6. Selects data, without duplicates, from the database and place them into file.

```
function mysql_file($i){
    $file = $i . ".txt";
    unlink($link);
    $query1='SELECT `diagonals`,`n` INTO OUTFILE $file'
        . ' FIELDS TERMINATED BY \';\''
        . ' LINES TERMINATED BY \'\\n\''
        . ' FROM `triangulations`;';
    $result1 = mysql_query($query1);
}
```

Function 7. Computation of triangulations of a *n*-gon, and usage of database facilities to enter unique values.

```
function decomp($n,$link){
    $link_n = $n . ".txt";
    $fp = fopen($link, "r");
    $fp_n=fopen($link_n,"w+");
    $Eb = createEb($n-1);
    $array_out = array();
    while(!feof($fp)){
        $string_row=fgets($fp);
        $array_1D=array(array());
        $array_1D[$i]=setsEd(chop($string_row),$Eb,$n);
        for($j=0;$j<$n-2;$j++){
          $array_string=array(array());$array_row = array(array());
          $array_row=explode(",",$array_1D[$i][$j]);
          if(abs($array_row[2*($n-3)-2]-$array_row[2*($n-3)-1])==1){
              $max=max($array_row[2*($n-3)-2],$array_row[2*($n-3)-1]);
              $increment=increment($array_row,$max,$n);
              $array_out[]=sort_string($increment,$n);
          }
          elseif($array_row[2*($n-3)-2]==$n-1 and $array_row[2*($n-3)-1]==1)
          {
              $array_out[]=sort_string($array_row,$n);
        } }
        if(count($array_out)>=5000000){
            $array_out = array_unique($array_out);
            foreach($array_out as $key => $value){
                 fwrite($fp_n,$value."\n");
            }
            $array_out = array();
    }   }
    $array_out = array_unique($array_out);
    foreach($array_out as $key => $value){
       fwrite($fp_n,$value."\n");
    }
    return $array_out;
}
```

Function 8. Displays the result, if it exists in the database, or calls the function to calculate the triangulations.

```
function decomposition($n){
    $query="select max(n) from triangulations";
```

```
    $result=mysql_query($query);
    $row=mysql_fetch_row($result);
    $max=$row[0];
    if($n<=$max){
        echo "Display pdf and txt file with triangulations";   }
    else{
        for($i=$max+1;$i<=$n;$i++){
            $link = $i-1.".txt";
            decomp($i,$link);
            file_mysql($i);
            //mysql_file($i);
}   }   }
```

7.3. Implementation Details for Hurtado's Algorithm

We present here some key parts of the code implementing Hurtado's algorithm.

Function 1. Transforms unidimensional array in the array of pairs.

```
function array1D_2Darray($array){
    for($j=0,$z=0;$j<count($array);$j=$j+2){
        $array_2D[$z]=$array[$j] . "," . $array[$j+1];
        $z++;
    }
    return $array_2D;
}
```

Function 2. Extracts pairs of the form (i, n).

```
function extract_n($niz,$k,$n){
    $s=0;
    for($i=0;$i<$k;$i++){
        $drugi = explode(",",$niz[$i]);
        if($drugi[1]==$n){
            $novi[$s]=$i . ";" . $niz[$i];
            $s++;
        }
    }
    return $novi;
}
```

Function 3. Makes diagonal "splitting" according to Hurtado's algorithm.

```
function hurtado($n, $link){
    $fp = fopen($link, "r");
    $link_n = "./hfiles1/" . $n+1 . ".txt";
    $fp_n=fopen($link_n,"w+");
    for($k=0;$k<numCatalan($n);$k++){
        $novi=array();
        $red = trim(fgets($fp));
        $red_a = array();
        $red_a = explode(",",$red);
        $novi = array1D_2Darray($red_a);
```

```
        $parovi_n = array();
        $parovi_n = extract_n($novi,count($novi),$n);
        for($i=0;$i<count($parovi_n);$i++){
            $red_b = array();
            $red_b = $red_a;
            $edim1 = explode(";",$parovi_n[$i]);
            $edim2 = explode(",",$edim1[1]);
            $t = $edim2[0];
            $parovi_ns = explode(",",$parovi_n[$i]);
            for($j=0;$j<count($parovi_n);$j++){
                $edim1 = explode(";",$parovi_n[$j]);
                $edim2 = explode(",",$edim1[1]);
                $t2 = $edim2[0];
                if($t2<$t){
                    $red_b[2*$edim1[0]+1]=$n+1;
                }
            }
            $red_b[2*(2*$n-3)] = $t;
            $red_b[] = $n+1;
            $red_b[] = $n;
            $red_b[] = $n+1;
            fwrite($fp_n,implode(",",$red_b)."\n");
        }
    }
    return $trian;
}
```

REFERENCES

1. J. Greenspan, B. Bulger, *MySQL/PHP Database Applications* M&T Books: An imprint of IDG Books Worldwide, Inc., 2001.
2. F. Hurtado, M. Noy, *Graph of Triangulations of a Convex Polygon and tree of triangulations*, Comput. Geom. **13** (1999), 179–188.
3. R. L. Graham, D. E. Knuth and O. Potsshnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, (1988).
4. H. W. Gould, *Research bibliography on two special number sequences*, Mathematica Monongaliae **12** (1971).
5. P. V. Krtolica, P. S. Stanimirović, R. Stanojević, *Reverse Polish Notation in Constructing the Algorithm for Polygon Triangulation*, FILOMAT **15** (2001), 25-33.
6. MySQL Developer Zone, *http://dev.mysql.com/doc/refman/5.0/en/full-table.html*
7. Saračević, M., Stanimirović, P., Mašović, S., Biševac, E., *Implementation of the convex polygon trangulation algorithm*, Facta Universitatis, Series Mathematics and Informatics, **27**(2) (2012), 213–228.
8. M. B. Tasić, P. S. Stanimirović, S. H. Pepić, *Computation of the generalized inverses using PHP/MySQL environment*, Int. J. Comput. Math. **88** (2011), 2429–2446.
9. H. Williams, D. Lane, *Web Database Applications with PHP & MySQL, 2nd Edition*, O'Reilly Media, Inc., Beijing, Cambridge, Famham, Köln, Paris, Sebastopol, Taipei, Tokyo, 2004.

10. S. Wolfram, *The Mathematica Book, 5th ed.*, Champaign: Wolfram Media, Inc., 2004.

Predrag Krtolica
Faculty of Science and Mathematics
Computer Science Department
Višegradska 33
18000 Niš, Serbia
krca@pmf.ni.ac.rs

Predrag Stanimirović
Faculty of Science and Mathematics
Computer Science Department
Višegradska 33
18000 Niš, Serbia
pecko@pmf.ni.ac.rs

Milan Tasić
Faculty of Science and Mathematics
Computer Science Department
Višegradska 33
18000 Niš, Serbia
milan12t@ptt.rs

Selver Pepić
Faculty of Science and Mathematics
Computer Science Department
Višegradska 33
18000 Niš, Serbia
p_selver@yahoo.com