**Optimal Polygon Triangulation**:**A Dynamic Programming Based Approach** **for Triangulating Convex Polygon**

Partha Dip Sarkar†, Md.Ariful Islam\*, S.m. Muhaiminul Haque‡

2016-1-60-042, , 2016-1-60-019, 2016-1-60-025

†\*‡┼Department of Computer Science and Engineering, East West University, Dhaka-1212, Bangladesh.

Email: †partha.dip.1@gmail, \*Emai1@gmail/yahoo.com, ‡muhaiminul.anik@gmail.com

*Abstract*—This paper emphasizes on triangulation of convex polygon in a way that minimizes the cost of triangulation. The problem has similarities to Matrix Chain Multiplication problem and simple modification to the Algorithm for Matrix Chain Multiplication problem can be used to solve the problem of triangulation. Dynamic programming is applicable here due to existence of overlapping substructure and recurrence relationship.

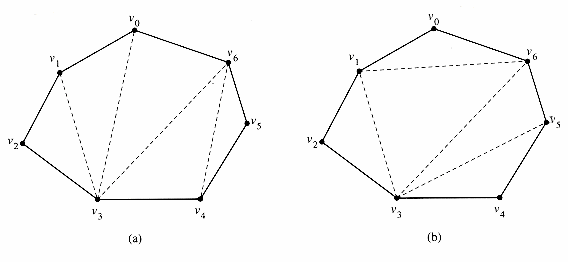
Keywords—Triangulation,Convex Polygon,Matrix Chain Multiplication,Dynamic,Programming,Overlapping Substructure, Recurrence Relationship.

# Introduction

Optimal polygon triangulation is a process of triangulating a polygon is such a way that the cost of triangulation is minimum. Triangulation of a polygon involves connecting different points in the polygon to form triangles .Cost of triangulation is the summation of the peremeters of all the triangles formed.We can triangulate a single convex polygon is many ways in brute force approach and then compare the cost of each triangulationto find the minimum.But doing so will not be efficient.For solving the problem efficiently we will use Dynamic Programming .

# Triangulation using Dynamic Programming

At first we define the optimal substructure.We can represent a polygon with n+1 points as a sequence of vertices like P = <v0, v1, ... , vn>, has n+1 sides, <v0,v1>, <v1,v2>, ... , <vn-1,vn>. <vn,v0>.

 Figure1:Same convex polygon triangulated differently

Careful observation makes the similarities between this problem and matrix chain multiplication problem clear[1].

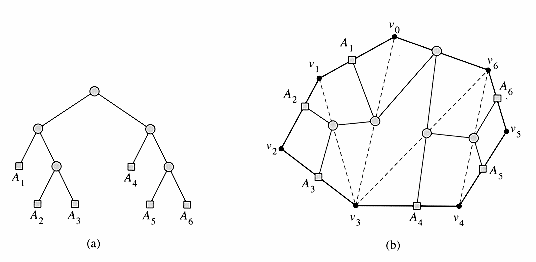


Figure 2:Triangulation represented as parse tree.

Figure2(a) shows the parse tree for full parenthesization of a matrix chain product ((*A*1(*A*2*A*3))(*A*4(*A*5*A*6))). Each leaf of the parse tree is one of the atomic elements(matrices).If a parent in this tree has left subtree of structure of Tl and right subtree of structure Tr then the parent can be represented as TlTr.A parse tree corresponds to a unique parenthesized expression.

Now if we design the the triangulation problem as figure2(b) we can represent it as parse tree too.So simple modification of matrix chain multiplication algorithm enables us to solve this problem of optimal triangulation [1] .

The idea is to divide the polygon into three parts: a single triangle, the sub-polygon to the left, and the sub-polygon to the right. We will try all possible combinations like this and find the one with minimum cost. We define the cost as follows,

Let *t*(*i*, *j*) be the cost of an optimal triangulation of the polygon <*vi*-1, *vi*, *vi*+1, ... , *vj*>. So   
*t*(*i*, *j*) = 0, if *i*=*j*  
 min*i*<=*k*<=*j*-1 { t(*i*,*k*) + t(*k*+1,*j*) + cost (<*vi*-1, *vk*, *vj*>)} if i < j

Here cost (<*vi*-1, *vk*, *vj*>) is the cost of triangle in the middle ,

cost (<vi, vj, vk>) = |vi,vj| + |vj, vk| + |vi,vk| .An algorithm based on this data only is possible but it will still be inefficient because multiple structures will generate similar substructe.So to efficiently solve this we will use an array that will store all calculated data an backtrack if necessary without calculating for same substructure again.We are representing that array as mem[][].

The algorithm can be written as,

MEMOIZED-OPT(*V*)

1 *n=* *Number*[*V*] - 1

2 **for** *i=* 1 **to** *n*

3 **do for** *j* *i* **to** *n*

4 **do** *m*[*i*, *j*] =infinity

5 **return** LOOKUP-MEM(*V*, 1, *n*)

LOOKUP-MEM(*V, i, j*)

1 **if** *m*[*i,j*] < infinity //Already calculated

2 **then return** *m*[*i, j*]

3 **if** *i = j //noncompatible vertices*

4 **then** *m*[*i, j*]*=*0

5 **else for** *k=* *i* **to** *j* - 1

6 **do** *q=*LOOKUP-MEM(*V, i, k*)+ LOOKUP-MEM(*V, k* + 1, *j*) +cost(ViVjVk)

7 **if** *q* < *m*[*i, j*]

8 **then** *m*[*i, j*]= *q*

9 **return** *m*[*i, j*] //at last m[1,n] will be returned

Like MATRIX-CHAIN-ORDER, the optimal triangulation procedure runs in time C:\Users\Partha Dip\Desktop\Intro to Algorithms  CHAPTER 16  DYNAMIC PROGRAMMING_files\bound.gif(*n*3) and uses C:\Users\Partha Dip\Desktop\Intro to Algorithms  CHAPTER 16  DYNAMIC PROGRAMMING_files\bound.gif(*n*2) space.

# Performance Analysis

|  |  |  |
| --- | --- | --- |
| Vertex in Polygon | Recursive | Dynamic |
| 10 | 33158 | 1000 |
| 100 | 1.61\*1057 | 1000000 |

The algorithm without a memorization array has the complexity of about C:\Users\Partha Dip\Desktop\Analysis of Algorithms  Lecture 12_files\omega.gif (4n / n3/2) [2]whereas one with memorization array has complexity of C:\Users\Partha Dip\Desktop\Intro to Algorithms  CHAPTER 16  DYNAMIC PROGRAMMING_files\bound.gif(*n*3).

So the difference between the running time is very clear even when we are calculating for 100 vertex polygon.

IV. CONCLUSION

Trangulation is used in Terrain mapping[3], GPS (complex triangulation)[4], Efficient Positioning of objects[5], Path finding, Voronoi Diagram applications[6] - Dual with Delaunay triangulation etc.So efficient solution of this problem is a necessity not ignorable.

##### References

1. http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap16.htm
2. http://www.cs.utexas.edu/users/djimenez/utsa/cs3343/lecture12.html
3. <https://en.wikipedia.org/wiki/Triangulated_irregular_network>
4. <http://www.mio.com/technology-trilateration.htm>
5. http://www-cgrl.cs.mcgill.ca/~godfried/teaching/cg-projects/97/Thierry/thierry507webprj/artgallery.html
6. https://www.ics.uci.edu/~eppstein/gina/scot.drysdale.html