POLYGON MATCH OPTIMAZATION WITH VOTING TREES

Author: Zarin Zuo

Date: 2021-10-23

1.Introduction

The optimation of the matches between 2 polygons' vertexes is a basic edition for Correspondence Problem which is of vital importance in Computer Graphics. Here we will try to use Voting Trees to optimaze this problem.

We are given 2 polygons A and B, of which we must find an optimal correspondence for the vertexes in them. The optimal correspondence is a set of pairs of integers $(m_1, n_1), \ldots, (m_n, n_n)$, for instance, (m_n, n_n) representing a match between the $m_n - th$ vertex of polygon A and the $n_n - th$ vertex of polygon B, and it is needed that $m_1 < \cdots < m_n$, $n_1 < \cdots < n_n$. Vertexs of the 2 polygons are given in input with a specific order.

There are no standard answers, as the problem is only for an optimation. However, one should get an algorithm fast enough to solve a match between $[3, 100]$ vertexs and somewhat an acceptable result of correspondences.

2.Algorithm Specification

a.Voting Tree, or A-star with Voting

Input: 2 polygons A and B, among which the smaller vertex count is N, bigger M

Output: N pairs of integers $(m_1, n_1), \ldots (m_N, n_N)$, representing a match between the vertexes of the 2 polygons

Main Idea:

1. Find out an function $F(x)$ of a match as a criterion of optimal matches. Here my $F(x)$ is defined as:

For every 3 consecutive vertex matches, consider them as a triangle. adding up the ratios($>= 1$) of the lengths of the connection edges except one already added up in the last consecutive matches and the ratio of the area of the triangle. $F(x)$ is just adding all these ratios up.

In consider of programming and optimazation, actually the first connection edge is added twice in realization.

2. Use Depth First Search to find a Voting Tree. The process of building the tree, however, is ignored because we can use return values to calculate the Votes without the help of a existing Voting Tree structure occupying a lot of space.

3. When the Search has finished reached the $N - th$ level, give out an vote to every pair of points appeared in the Tree Path (Search Route). These votes are sumed up in a Voting Table. We consider the match of points in A one by one, and choose the pair of points which receives the most votes as the best match.

4. Consider the nature of $F(x)$ to cut up some search routes. There may be a process similar to the A-star algorithm to estimate whether the search route can get some Votes. We can use Binary Search to estimate a least value of $F(x)$ that can get effective votes. And as the times of Search process may become too high, We set a exit limit(in my code, 10^5 times) of Search, and when the times of Search has reached the exit limit, it will be terminated and return a approximate Voting Table.

Pseudo Code:

1. The startup of Voting process

For each a from 1 to N

use Binary Search to get a suitable limit of $F(x)$

For each j from match[i-1] to M

vote[j]:=Search(|i, j|,0)

if(vote[j]>vote[match[i]]) match[i]:=j

2. The Search Process (Recursive)

int Search(|pa, pb|,excl)

if extl>1e5 then return 0

if $pa = N$ and excl+F(last match, |pa, pb|, |1, match[1]|)<limit then return 1

if $pa = N$ and excl+F(last match, |pa, pb|, |1, match[1]|)>=limit then return 0

ret:=0

for each j from $pb + 1$ to $M + 1 - (N - pa)$

delta:=F(last match, |pa, pb|, |pa+1, j|)

if(excl+delta+3*(N-pa)<limit) then ret+=Search(pa+1,j,excl+delta)

return ret

b.Voting Tree without Binary Search process

Input: 2 polygons A and B, among which the smaller vertex count is N, bigger M

Output: N pairs of integers $(m_1, n_1), \ldots (m_N, n_N)$, representing a match between the vertexes of the 2 polygons

Main Idea:

1. Find out an function $F(x)$ of a match as a criterion of optimal matches. Here my $F(x)$ is defined as:

For every 3 consecutive vertex matches, consider them as a triangle. adding up the ratios($>= 1$) of the lengths of the connection edges except one already added up in the last consecutive matches and the ratio of the area of the triangle. $F(x)$ is just adding all these ratios up.

In consider of programming and optimazation, actually the first connection edge is added twice in realization.

2. Use Depth First Search to find a Voting Tree. The process of building the tree, however, is ignored because we can use return values to calculate the Votes without the help of a existing Voting Tree structure occupying a lot of space.

3. When the Search has finished reached the $N - th$ level, give out an vote to every pair of points appeared in the Tree Path (Search Route). These votes are sumed up in a Voting Table. We consider the match of points in A one by one, and choose the pair of points which receives the most votes as the best match.

4. Consider the nature of $F(x)$ to cut up some search routes. There may be a process similar to the A-star algorithm to estimate whether the search route can get some Votes. We will estimate a least value of $F(x)$(in my code, $4.1N$) that can get effective votes. And as the times of Search process may become too high, We set a exit limit(in my code, 10^5 times) of Search, and when the

times of Search has reached the exit limit, it will be terminated and return a approximate Voting Table.

Pseudo Code:

1. The startup of Voting process

For each a from 1 to N

limit:=$4.1 * N$

For each j from match[i-1] to M

vote[j]:=Search(|i, j|,0)

if(vote[j]>vote[match[i]]) match[i]:=j

2. The Search Process (Recursive)

int Search(|pa, pb|,excl)

if extl>1e5 then return 0

if $pa = N$ and excl+F(last match, |pa, pb|, |1, match[1]|)<limit then return 1

if $pa = N$ and excl+F(last match, |pa, pb|, |1, match[1]|)>=limit then return 0

ret:=0

for each j from $pb + 1$ to $M + 1 - (N - pa)$

delta:=F(last match, |pa, pb|, |pa+1, j|)

if(excl+delta+3*(N-pa)<limit) then ret+=Search(pa+1,j,excl+delta)

return ret

c. A scratch of the main program

Pseudo Code:

1. Input Function

GetInput()

input N,M

if N>M then

swaflag:=1

swap(N,M)

input $(b_1, \ldots, b_M)$

input $(a_1, \ldots, a_N)$

else

input $(a_1, \ldots, a_N)$

input $(b_1, \ldots, b_M)$

2. main()

GetInput()

Process()

for each i from 1 to N

output (i, match[i])

3.Testing Results

Some generated inputs are in the in1,...,in10.txt, and their outputs are separated into outa1,...,outa10.txt and outb1,...,outb10.txt. These results are specifically defined on my personal computer.

Test Case	Purpose	Time used(a)	Time used(b)	Output Comparison
1	Sample	0.000	0.000	Both Correct
2	Sample, A and B swapped	0.000	0.000	Both Correct
3	A 3 points, B 3 points, random in[0,4]	0.000	Failed	(a) finds a not precise solution, (b) cannot give out a solution
4	A and B 3 same points, in[0,4]	0.000	0.000	(a) and (b) both find the only solution
5	A 3 points, B 3 random points, with overlapped vertexes	Failed	Failed	Both Correct
6	A 3 points, B 10 points, in[0,4]	0.000	0.000	(a) more precise
7	A 10 points, B 50 points, random	0.003	0.156	(a) more precise and quicker
8	A 10 points, B 100 points, random	0.025	0.187	(a) more precise and quicker
9	A 50 points, B 100 points, random	1.187	1.968	(a) more precise and quicker
10	A 100 points, B 100 points, random	0.039	Failed	(a) find a not precise enough answer, while (b) give no solution

From these resulys, we can see that, algorithm (a) can quickly find a best solution in all solutions (even not precise at all), however (b) will terminate if the matches were not so precise. As (b) has a bound of search, it will in most time cost more time and may be wrong because of a lot of votes from not so precise solutions.

4.Analysis and Comments

a. Voting Tree, or A-star with Voting

Time Complexity Analysis: $O(NM^{N+1})$ (worst). The Binary Search process will cost $O(NlogNM^N)$ at most. As we need to expand the tree from each pair of points ($O(NM)$) in A and B, and each expand process is a search process with M choices and N steps. However, because there exists a limit of $F(x)$, the times of recursion will be dramatically reduced, however in theory it's hard to calculate how many time could be saved, so that's still $O(NM^{n+1})$.

Space Complexity Analysis: $O(N)$. Because that we choose the matches one by one, there are no need to store a full Voting Table. Only space for one line of table and original data is needed.

b.Voting Tree without Binary Search process

Time Complexity Analysis: $O(NM^{N+1})$ (worst). As we need to expand the tree from each pair of points ($O(NM)$) in A and B, and each expand process is a search process with M choices and N steps. However, because there exists a limit of $F(x)$, the times of recursion will be dramatically reduced, however in theory it's hard to calculate how many time could be saved, so that's still $O(NM^{n+1})$. And as the limit is not well decided, it may cost more recursion than (a), but the precision is well-limited.

Space Complexity Analysis: $O(N)$. Because that we choose the matches one by one, there are no need to store a full Voting Table. Only space for one line of table and original data is needed.

Appendix

The realizations of the (a) and (b) algorithm are put in pro.c and pro2.c. A simple generator (using full random) avaiable can be found in wrt.c.

These codes are for comparation with the pseudo codes. You may better read the original code (to get some comments).

How to use the generator: Input the size N and M you want, and it will give out a $N+M$ pairs of points representing the polygon A and B. The generation is full random, so don't except the output is truly a polygon.

Note: Both generator and solution use stdin and stdout.

Ratio Defination

```
double Ratio(double a,double b){
    if(a==0&&b==0) return 1;
    else if(a&&b) return fmax(a/b,b/a);
    else return lim;
}
```

Search Process

```
int Expand(int sp1,int p1,int sp2,int p2,double excl){
    if(++extl>1e5) return 0;
```

```
    if(p1==acnt){
        if(excl+GetWeight(pa[sp1],pa[p1],pa[1],pb[sp2],pb[p2],pb[mt[1]])
            <lim) return 1;
        else return 0;
    }
    int ret=0;
    for(int j=p2+1;j<=bcnt+1-(acnt-p1);++j){
        double delta=
            GetWeight(pa[sp1],pa[p1],pa[p1+1],pb[sp2],pb[p2],pb[j]);
        if(excl+delta+3*(acnt-p1)<lim)
            ret+=Expand(p1,p1+1,p2,j,excl+delta);
    }
    return ret;
}
```

Binary Search Functions

```
int PseudoExpand(int sp1,int p1,int sp2,int p2,double excl){
    if(++extl>1e5) return 0;
    if(p1==acnt){
        if(excl+GetWeight(pa[sp1],pa[p1],pa[1],pb[sp2],pb[p2],pb[mt[1]])
        <lim) return 1;
        else return 0;
    }
    for(int j=p2+1;j<=bcnt+1-(acnt-p1);++j){
        double delta=
            GetWeight(pa[sp1],pa[p1],pa[p1+1],pb[sp2],pb[p2],pb[j]);
        if(excl+delta+3*(acnt-p1)<lim&&
            PseudoExpand(p1,p1+1,p2,j,excl+delta)) return 1;
    }
    return 0;
}
int TryToExpand(int i,double x){
    lim=x;
    for(int j=preb;j<=bcnt;++j){
        if(i==1) mt[i]=j;
        if(PseudoExpand(i-1?i-1:i,i,mt[i-1]?mt[i-1]:j,j,0)) return 1;
    }
    return 0;
}
```

Main Program

```
signed main(signed argc,char** argv,char** env){
    GetInput();

    lop(i,acnt){
        ++preb,extl=0;
        double l=3,r=1e14;
        while(l+Eps<r){
            double mid=(l+r)/2;
            if(TryToExpand(i,mid)) r=mid;
            else l=mid;
        }
        lim=r,extl=0;
        for(int j=preb;j<=bcnt;++j){
            if(i==1) mt[i]=j;
            vote[j]=Expand(i-1?i-1:i,i,mt[i-1]?mt[i-1]:j,j,0);
            if(vote[preb]<vote[j]) preb=j;
        }
        if(vote[mt[i]=preb]==0){
            printf("Hard to find a solution close enough.\n");
            exit(0);
        }
    }
    lop(i,acnt){
        if(swaflag) printf("(%d, %d)\n",mt[i],i);
        else printf("(%d, %d)\n",i,mt[i]);
    }
}
```

Declaration

I hereby declare that all the work done in this project is of my independent effort.