Computational Geometry Algorithms for Clustering, Obstacle Avoidance and Optimal Path Planning

Dr. Marina Gavrilova

Associate Professor, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada.
Talk Overview

• Research Interests
• Clustering and CRYSTAL
• Clearance-based optimal path
• Weighted terrain and resulting system
• Conclusions
Research Activities and Interests
Activities in Brief

Founder and Co-Director:

- Biometric Technologies Laboratory, CFI
- SPARCS Laboratory for Spatial Analysis in Computational Sciences, GEOIDE

Editor-in-Chief:

- International Journal LNCS Transactions on Computational Sciences, Springer-Verlag

Guest Editor:

- Int. Journal of Computational geometry and Applications
- IEEE Robotics and Automation Magazine RAM
Activities in Brief

Chair and Co-Founder:
- International Conference on Computational Sciences and Applications since 2003
- International Workshop on Computational Geometry and Applications (since 2001)
- Chair, the 3rd International Symposium on Voronoi Diagrams and Applications 06

Author of new books:
S. Yanushkevich, M. Gavrilova, P. Wang and S. Srihari

- M. Gavrilova “Computational Intelligence: A Geometry-Based Approach,” Series on Studies in Computational Intelligence, Springer-Verlag, 2008 (upcoming)
Areas of Research

- Topological properties of data sets
- Terrain rendering and surface triangulation
- Path planning and obstacle avoidance
- Robotics and Navigation
- Autocorrelation analysis
- Spatial-temporal models
- Nearest neighbor properties
- Terrain reconstruction and triangulation

- Biological systems modeling
- Molecular systems representation and analysis
- Geographical Information Systems
- Biometric analysis and synthesis
- Granular-type materials simulation
Areas of Interests

- Biometric research
- Porous materials
- Terrain modeling
- Coral models (with J. Kaandorp)
- Lipid bi-layers and molecular modeling (with N.N. Medvedev)
- Dynamic data structures (with I. Kolingerova)
Voronoi Diagrams – A Brief Overview
Voronoi diagrams in selected applications (ISVD 2006)

M. Moriguchi and K. Sugihara, Japan

T. Taylor and I. Vaisman, USA

L. Wang et. al. China

A. Mukhopadhyay, S. Das, Canada

James Dean Palmer, USA

Tetsuo Asano, Japan

Deok-Soo Kim, Korea

C. Gold and M. Dakowicz, UK

P. Bhattacharya and M. Gavrilova, Canada
Voronoi diagrams in tiling (ISVD 2006)

Craig S. Kaplan, University of Waterloo, Canada

Jos Leys, Belgium
Voronoi diagram and Delaunay Tessellation

Voronoi diagram is one of the fundamental computational geometry data structures that stored proximity information for a set of objects. It’s dual structure, often used in computer graphics, is Delaunay Tessellation.

A generalized Voronoi diagram (GVD) for a set of objects in space is the set of generalized Voronoi regions

\[ \text{Vor}(P) = \{x \mid d(x, P) \leq d(x, Q), \forall Q \in S \setminus \{P\}\} \]

where \(d(x,P)\) is a distance function between a point \(x\) and a site \(P\) in the \(d\)-dimensional space.
Delaunay Tessellation

A generalized Delaunay tessellation *(triangulation in 2d)* is the dual of the generalized Voronoi diagram obtained by joining all pairs of sites whose Voronoi regions share a common Voronoi edge according to some specific rule.
Delaunay simplex (tetrahedron in 3D) defines a simplicial configuration of spheres and a void (empty space) between spheres.
Voronoi diagram and Delaunay triangulation in 2D
Main properties of the Voronoi Diagram

Under assumptions that no four sites from the object (generator) set $S$ are cocircular:

- **Voronoi vertex** is the intersection of 3 Voronoi edges and a common point of 3 Voronoi regions
- **Voronoi vertex** is equidistant from 3 sites. It lies in the center of a circle inscribed between 3 cites
- **Empty circle property** This inscribed circle is empty, i.e. it does not contain any other sites
- **Nearest-neighbor property** If $Q$ is the nearest neighbor of $P$ then their Voronoi regions share an edge (to find a nearest neighbor it is sufficient to check only neighbors in the VD)
Main properties of the Delaunay Triangulation

Under assumptions that no three sites from the set S (generator) lie on the same straight line:

- The straight-line dual of the Voronoi diagram is a triangulation of S
- The circumcircle of any Delaunay triangle does not contain any points of S in its interior
- If each triangle of a triangulation of the convex hull of S satisfies the empty circle property, then this triangulation is the Delaunay triangulation of S.
- If Q is the nearest neighbor of P then their Voronoi regions share an edge (to find a nearest neighbor it is sufficient to check only neighbors in the VD).
The Optimal Path Planning Problem
The Problem

- Given two locations $A$ and $B$ and the geographical features of the underlying terrain, what is the optimal route for a mobile agent between these two locations?
- When the nature of the terrain is allowed to vary, the robot has to make a decision which path is the most suitable based on the set of priorities.
- We concentrate on marine applications, where robot is conceived as a ship sailing from one port to another. The decision making process can become very complex and combines AI, uncertainty theory and multi-varied logic with temporal-spatial data representation.
- The problem arises in such areas as Robotics, Risk Planning, Route Scheduling, Navigation, and Processes Modeling.

The risk areas are defined by cluster analysis performed on the incident database of the Maritime Activity and Risk Investigation System (MARIS).
The Geometry-based approach

- Design of a new Delaunay triangulation based clustering method to identify complicated cluster arrangements.
- Development of an efficient Delaunay triangulation and visibility graph based method for determining clearance-based shortest path between source and destination in the presence of simple, disjoint, polygonal obstacles.
- Introduction of a new method for determining optimal path in a weighted planar subdivision representing a varied terrain.
- This research was carried out at the SPARCS Laboratory, University of Calgary, by graduate students Russel Apu, Priyadarshi Bhattachariya and Mahmudul Hasan.
Marine Risk Analysis

- Identification of high-risk areas in the sea based on incident and traffic data from the Maritime Activity and Risk Investigation System (MARIS), maintained primarily by the University of Halifax.
Clustering: the CRYSTAL Algorithm
Definition of Clustering

- Clustering is the unsupervised classification of patterns (observations, data items or feature vectors) into groups (clusters).

  — A.K. Jain, M. N. Murty, P. J. Flynn, Data Clustering: A Review
Clustering – desired properties

• Linear increase in processing time with increase in size of dataset (scalability).

• Ability to detect clusters of different shapes and densities.

• Minimal number of input parameters.

• Robust with regard to noise.

• Insensitive to data input order.

• Portable to higher dimensions.

Approaches to clustering

- Hierarchical clustering (Chameleon, 1999)
- Density-based clustering (DBScan, 1996)
- Grid-based clustering (Clique, 1998)
- Model-based clustering (Vladimir, Poss, 1996)
- Partition-based clustering (Greedy Elimination Method, 2004)
- Graph-based clustering (Autoclust, 2000)

Our algorithm falls in this category
Hierarchical Clustering

- Creates a tree structure to determine the clusters in a dataset (top-down or bottom-up). Bottom-up: consider each data element as a separate cluster and then progressively merge clusters based on similarity until some termination condition is reached (agglomerative). Top-down: consider all data elements as a single cluster and then progressively divides a cluster into parts (divisive).

- Hierarchical clustering does not scale well and the computational complexity is very high (CHAMELION). The termination point for division or merging for divisive and agglomerative clustering respectively is extremely difficult to determine accurately.
Density-based clustering

• In density-based clustering, regions with sufficiently high data densities are considered as clusters. It is fast but it is difficult to define parameters such as epsilon-neighborhood or minimum number of points in such neighborhoods to be considered a cluster.

• These values are directly related to the resolution of the data. If we simply increase the resolution (i.e. scale up the data), the same parameters no longer produce the desired result.

• Advanced methods such as TURN consider the optimal resolution out of a number of resolutions and are able to detect complicated cluster arrangements but at the cost of increased processing time.
Grid-based clustering

- Grid-based clustering performs clustering on cells that discretize the cluster space. Because of this discretization, clustering errors necessarily creep in.
- The clustering results are heavily dependent on the grid resolution. Determining an appropriate grid resolution for a dataset is not a trivial task. If the grid is coarse, the run-time is lower but the accuracy of the result is questionable. If the grid is too fine, the run-time increases dramatically. Overall, the method is unsuitable for spatial datasets.
Model-based clustering

- In model-based clustering, the assumption is that a mixture of underlying probability distributions generates the data and each component represents a different cluster. It tries to optimize the fit between the data and the model. Traditional approaches involve obtaining (iteratively) a maximum likelihood estimate of the parameter vectors of the component densities. Underfitting (not enough groups to represent the data) and overfitting (too many groups in parts of the data) are common problems, in addition to excessive computational requirements.

- Deriving optimal partitions from these models is very difficult. Also, fitting a static model to the data often fails to capture a cluster's inherent characteristics. These algorithms break down when the data contains clusters of diverse shapes, densities and sizes.
Partition based clustering

1. Place K points into the space represented by the data points that are being clustered. These points represent initial group centroids.

2. Partition the data points such that each data point is assigned to the centroid closest to it.

3. When all data points have been assigned, recalculate the positions of the K centroids.

4. Repeat Steps 2 and 3 until the centroids no longer move.
Disadvantages

• Number of clusters have to be prespecified.

• Clustering result is sensitive to initial positioning of cluster centroids.

• Able to detect clusters of convex shape only.

• Clustering result is markedly different from human perception of clusters.

Greedy Elimination Method (K = 5)
Recent Papers

Z.S.H. Chan, N. Kasabov:  
“Efficient global clustering using the Greedy Elimination Method”,  

Aristidis Likas, Nikos Vlassis, Jakob J. Verbeek:  
“The global k-means clustering algorithm”,  
Pattern Recognition, 36(2), 2003.

Global K-Means (K = 5)
Summary of shortcomings of existing methods

• Able to detect only convex clusters.
• Require prior information about dataset.
• Too many parameters to tune.
• Inability to detect elongated clusters or complicated arrangements such as cluster within cluster, clusters connected by bridges or sparse clusters in presence of dense ones.
• Not robust in presence of noise.
• Not practical on large datasets.
Graph-based clustering – a chosen approach

• The graph-based clustering algorithms based on the triangulation approach have proved to be successful. However, most algorithms based on the triangulation derive clusters by removing edges from the triangulation that are longer than a threshold (Eldershaw, Kang, Imiya). But distance alone cannot be used in separating clusters. The technique succeeds only when the intra-cluster distance is sufficiently high. It fails in case of closely lying high density clusters or clusters connected by bridges.

• We utilize a number of unique properties of Delaunay triangulation, as it is an ideal data structure for preserving proximity information and effectively eradicates the problem of representing spatial adjacency using the traditional line-intersection model. It can be constructed in O(nlogn) time and has only O(n) edges.
Triangulation based clustering

- Construct the Delaunay triangulation of the dataset.
- Remove edges based on certain criteria with connected components eventually forming clusters.

Disadvantages

- The use of global density measures such as mean edge length is misleading and often precludes the identification of sparse clusters in presence of dense ones.

- The decision of whether to remove an edge or not is usually a costly operation. Also, deletion of edges may result in loss of topology information required for identification of later clusters. As a result, algorithms often have to recuperate some of the lost edges later on.
CRYSTAL - Description

• **Initialization phase:** Generates the Voronoi diagram of the data points and sorts them in increasing order of the area of their Voronoi cells. This ensures that clustering starts with the densest clusters.

• **Grow cluster phase:** Scans the sorted vertex list $L$ and for each vertex $V_i \in L$ not yet visited, attempts to grow a cluster. The Delaunay triangulation is utilized as the underlying graph on which a breadth-first search is carried out. The cluster growth stops at a point identified as a boundary point but continues from other non-boundary points. Several criteria are employed to effectively determine the cluster boundary.

• **Noise removal phase:** Identifies noise as sparse clusters or clusters that have very few elements. They are removed at this stage.
Merits of CRYSTAL

• The growth model adopted for cluster growth allows spontaneous detection of elongated and complicated cluster shapes.

• The algorithm avoids the use of global parameters and makes no assumptions about the data.

• The clusters fail to grow from noise points or outliers. Thus noise can be easily eliminated without any additional processing overhead.

• The algorithm works very fast in practice as the growth model ensures that identification of different cases like cluster within cluster or clusters connected by bridges do not require any additional processing logic and are handled spontaneously.

• It requires no input parameter from user and the clustering output closely resembles human perception of clusters.
CRYSTAL – Geometric Algorithms

- **Triangulation phase:**
  Forms the Delaunay Triangulation of the data points and sorts the vertices in the order of increasing average length of incident edges. This ensures that, in general, denser clusters are identified before sparser ones.

- **Grow cluster phase:**
  Scans the sorted vertex list and grows clusters from the vertices in that order, first encompassing first order neighbors, then second order neighbors and so on. The growth stops when the boundary of the cluster is determined. A sweep operation adds any vertices to the cluster that may have been left out.

- **Noise removal phase:**
  The algorithm identifies noise as sparse clusters. They can be easily eliminated by removing clusters which are very small in size or which have a very low spatial density.
Average cluster edge length

= Average of the length of edges incident on the vertex

= Edge Length between the two vertices (d)

(d1 + d2) / 2

In general, (Sum of edge lengths) / (Number of elements in cluster – 1)
Detecting a cluster boundary

Mean of incident edge lengths of $V_i > 1.6 \times \text{Average cluster edge length}$?
Grow cluster phase

Enqueue( Q, v_k )
C ← v_k
tot_dist ← 0

avg ← AvgadjEdgeLen( v_k )

\[ d ← \text{Head}( Q ) \]
\[ L ← 1\text{st-order neighbors of } d \text{ sorted by edge length} \]
For all \( L_i \in L \) do
   C ← C U L_i
   tot_dist ← tot_dist + EdgeLen( d, L_i )
   avg ← tot_dist / (Size( C ) - 1)
   If Not a boundary vertex and Not connected to a boundary vertex then
      Enqueue( Q, L_i )
   End If
End For

Dequeue( Q )

Q = \Phi ?

Perform \textit{sweep} and consider next vertex
Sweep

- The average cluster edge length towards the end of Grow Cluster phase better represents the local density than at the starting phase of cluster growth.

- This operation ensures that any data points that were left out at the initial stages of the cluster growth are put back into the cluster.

**Description:**

- Scan the vertices added to cluster.

- For each vertex, inspect if there are any 1\textsuperscript{st} order neighbors for which \textit{edge length} < 1.6 * average cluster edge length. If so, add that vertex to the cluster and update the average cluster edge length.
# Boundary detection

A vertex is considered to be a boundary vertex of the cluster if any one of the following is true:

- Voronoi cell area of the vertex in the Voronoi diagram is $> \text{Th} \times \{\text{average Voronoi cell area of cluster}\}$

- The maximum of the Voronoi cell areas of the neighbors of the vertex (including itself) $> \text{Th} \times \{\text{average Voronoi cell area of cluster}\}$

- The vertex is connected to another vertex in the Delaunay Triangulation which is already present in the cluster so that the edge length is $> \text{Th} \times \{\text{average cluster edge length}\}$

The value of Th is empirically determined.
Example Processing
Sample output

Original dataset (n = 8,000)

CRYSTAL output (Th = 2.4)
Comparison

### Clustering results on t7.10k dataset


<table>
<thead>
<tr>
<th></th>
<th>Clustering Method</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>K-Means</td>
<td>$k = 9$</td>
</tr>
<tr>
<td>B</td>
<td>CURE</td>
<td>$k = 9, \alpha = 0.3$ and 10 representative points per cluster</td>
</tr>
<tr>
<td>C</td>
<td>ROCK</td>
<td>$\theta = 0.975$ and $k = 1000$</td>
</tr>
<tr>
<td>D</td>
<td>CHAMELEON</td>
<td>K-NN = 10, MinSize = 2.5%, $k = 9$</td>
</tr>
<tr>
<td>E</td>
<td>DBSCAN</td>
<td>$\epsilon = 5.9$, MinPts = 4</td>
</tr>
<tr>
<td>F</td>
<td>DBSCAN</td>
<td>$\epsilon = 5.5$, MinPts = 4</td>
</tr>
<tr>
<td>G</td>
<td>WaveCluster</td>
<td>Resolution = 5, $\Gamma = 1.5$</td>
</tr>
<tr>
<td>H</td>
<td>WaveCluster</td>
<td>Resolution = 5, $\Gamma = 1.999397$</td>
</tr>
</tbody>
</table>
CRYSTAL output

t7.10k dataset (9 visible clusters, n = 10,000)
CRYSTAL output

CRYSTAL on t7.10k dataset ($\Gamma = 1.8$)
More examples

Original dataset

Crystal output (Th = 2.5)

Original dataset

Crystal output (Th = 2.4)
Time Comparison

Cluster size Vs CPU time in seconds
(550 MHz processor, 128 MB memory)

Vladimir Estivill-Castro, Ickjai Lee,

Cluster size (in 1000) Vs CPU time in milli-seconds
(3 GHz processor, 512 MB memory)
CRYSTAL
Demonstration

t7.10k dataset
Demonstration

Cluster within cluster

Sparse and dense clusters
Computational Geometry Algorithms for Clearance-based Path Planning
The problem

Plan an optimal collision-free path for a mobile agent moving on the plane amidst a set of convex, disjoint, polygonal obstacles \( \{P_1, \ldots, P_m\} \), given start and goal configurations \( s \) and \( g \).

By *optimal*, we mean the path should be:

- Short – not containing unnecessary long detours.
- Having some clearance – not getting too close to an obstacle.
- Smooth – not containing sharp turns.
Existing approaches

- Roadmap based techniques
- The potential field approach
- The cell decomposition method

Roadmap creates a map (partitioning) of the plane to navigate the robot.

Potential field approach fills the free area with a potential field in which the robot is attracted towards its goal position while being repelled from obstacles.

Cell-decomposition method utilized grid and computes its intersections with obstacles to compute the path.
Disadvantages of existing approaches

Potential field method: the robot may get stuck at a local minimum. The reported paths can be arbitrarily long.

Cell decomposition: path is not optimal because of the connectivity limitations in a grid, very difficult to correctly estimate the grid resolution.

Roadmap approaches (chosen approach):

- Probabilistic roadmap
- Visibility graph based
- Voronoi diagram based
Existing roadmap approaches

• **Probabilistic roadmap** is created by generating random points in the plane and connecting these points to the \( k \)-nearest neighbours taking care that the connecting edges do not cross any obstacle. The method is fast but the reported path is very often of poor quality because of the randomness inherent in the graph representing the free space connectivity. Also, a path may never be detected even if one exists.

• A **visibility graph** is a roadmap whose vertices are the vertices of the obstacles themselves and there is an edge between every pair of vertices that can see each other. A **visibility graph** is a graph of intervisible locations. However, the path planning based on querying visibility graph is very slow, and incorporating the clearance is very difficult.
Voronoi diagram based roadmap

(a) Shortest path from Voronoi diagram based roadmap (based on VD edges)
(b) Shortest path using developed algorithm (C_{min} = 0).
(c) Clearance based path using proposed algorithm (C_{min} = 2). Zoomed path on right.
Tools: Visibility graphs

- Used to plan shortest paths. Constructed in $O(n^2 \log n)$ time, where $n$ is the total number of obstacle vertices.

- Output-sensitive $O(n \log n + k)$ algorithm exists for construction where $k$ is the number of edges in visibility graph.

Resulting paths have no clearance.
Tools: Voronoi diagram

• Can be constructed in $O(n \log n)$ time where $n$ is the number of obstacle vertices.

• Computes a path that has maximum clearance from obstacles.
VD and visibility graph roadmaps
Clearance “hybrid” tool: the VV\(^{(c)}\)-Diagram

- The Visibility-Voronoi diagram ([Wein, 2005](#)) for clearance ‘c’ is a hybrid between the visibility graph and Voronoi diagram.

- Evolves from the visibility graph to the Voronoi diagram as ‘c’ increases.

Visibility-Voronoi diagram of a pair of obstacles.
The VV(ε)-Diagram – Cont’d

Pros:

• Generates smooth paths

• Paths maintain some amount of clearance from obstacles

Cons:

➢ In case obstacles are very close to one another, dilation of obstacles may lead to unstable situations because of overlapping.

➢ It takes O(n² log n) time to construct the visibility-Voronoi diagram and hence the method is impractical for large spatial datasets.

➢ The path is not the shortest possible for the required clearance value as length of path is compromised in lieu of smoothness.
Our approach

- We provide an algorithm based on Voronoi diagram to compute an optimal path between source and destination in the presence of simple disjoint polygonal obstacles.
- We evaluate the quality of the path based on clearance from obstacles, overall length and smoothness.
- We provide a detailed description of the algorithm for Voronoi diagram maintenance and dynamic updates.
- Experimental results demonstrate superior performance of the method in relation to other path planning algorithms.
Our approach: advantages

- Runs in just $O(n \log n)$ time where $n$ is the number of obstacle vertices.

- Generates paths that are near-optimal with respect to the amount of clearance required. By optimal, we mean the path is the shortest possible while maintaining the necessary clearance.

- Since the refinement method is iterative, a tradeoff can be obtained between the optimality and processing time.
Key features

- Inserting the source and destination *dynamically* has two major advantages over simply connecting them to the nearest Voronoi vertex. There is no possibility of the connecting edges crossing an obstacle as they are contained inside the Voronoi cell. Also, multiple queries do not require diagram reconstruction.
- We next remove all those edges in the resulting diagram that have a clearance less than the minimum clearance required (Cmin, set by user). This guarantees we can report only paths with necessary clearance.
- We apply Dijkstra’s algorithm to determine shortest path in the roadmap and refine the path through removing unnecessary turns.
- We next utilize Steiner points along the edges of this path to perform corner-cutting to convert to an optimal path.
Flow diagram of the Voronoi diagram based algorithm
Voronoi diagram based path - illustration

Voronoi diagram

Roadmap extracted
Dynamic point removal in Voronoi diagram
Corner cutting using iterative Steiner point method

Steiner point S

Iterative refinement
Output by stages

Path from Voronoi diagram based roadmap

Path obtained after **RemoveRedundancy**

Path obtained after **corner-cutting**
Clearance-based path

Shortest path obtained from Voronoi diagram based roadmap

Shortest path after iterative refinement ($C_{\text{min}} = 0$)

Top-left corner: 81.435 degrees latitude and −90.405 degrees longitude. Bottom-right corner: 70.528 degrees latitude and −78.312 degrees longitude
Algorithm efficiency
Examples: Varying clearance

Clearance = 10 units
Clearance = 15 units
Clearance = 20 units
Clearance = 25 units
Clearance = 30 units
Clearance = 35 units
More examples

Clearance = 12

Clearance = 7

Clearance = 8

Clearance = 0

Clearance = 0

Clearance = 0
More examples

Value of minimum clearance required increases
Geraerts, 2004

More examples

$C_{\text{min}} = 12$

$C_{\text{min}} = 0$

Our approach
Clearance-based optimal path statistics

<table>
<thead>
<tr>
<th>$V$</th>
<th>$N$</th>
<th>$T_{Net}$</th>
<th>$T_{Query}$</th>
<th>$T_{Optm}$</th>
<th>$Clr_{Avg}$</th>
<th>$Clr_{Min}$</th>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>278</td>
<td>697</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>8.026</td>
<td>8.001</td>
<td>9.071</td>
<td>8.258</td>
</tr>
<tr>
<td>302</td>
<td>758</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>8.017</td>
<td>8.000</td>
<td>12.621</td>
<td>11.440</td>
</tr>
<tr>
<td>428</td>
<td>978</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>8.012</td>
<td>8.000</td>
<td>12.959</td>
<td>11.890</td>
</tr>
<tr>
<td>739</td>
<td>1890</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>8.018</td>
<td>8.000</td>
<td>27.110</td>
<td>24.314</td>
</tr>
<tr>
<td>1495</td>
<td>3918</td>
<td>23</td>
<td>6</td>
<td>5</td>
<td>8.005</td>
<td>8.000</td>
<td>50.693</td>
<td>48.184</td>
</tr>
<tr>
<td>2576</td>
<td>5834</td>
<td>27</td>
<td>8</td>
<td>3</td>
<td>8.004</td>
<td>8.000</td>
<td>76.078</td>
<td>67.647</td>
</tr>
<tr>
<td>4065</td>
<td>8720</td>
<td>34</td>
<td>14</td>
<td>2</td>
<td>8.007</td>
<td>8.000</td>
<td>77.097</td>
<td>64.747</td>
</tr>
<tr>
<td>6831</td>
<td>9208</td>
<td>43</td>
<td>23</td>
<td>5</td>
<td>8.007</td>
<td>8.000</td>
<td>67.499</td>
<td>62.866</td>
</tr>
</tbody>
</table>

Table 5.1: Experimental results on different shape files with $C_{min} = 8$. 
Processing time of developed method vs. visibility graph approach

<table>
<thead>
<tr>
<th>$V$</th>
<th>$N$</th>
<th>$L_{\text{Approx}}$</th>
<th>$L_{\text{VG}}$</th>
<th>$T_{\text{Approx}}$</th>
<th>$T_{\text{VG}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>155</td>
<td>361</td>
<td>4.45</td>
<td>4.444</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>338</td>
<td>788</td>
<td>7.709</td>
<td>7.703</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>355</td>
<td>834</td>
<td>7.281</td>
<td>7.273</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>739</td>
<td>1873</td>
<td>26.025</td>
<td>26.014</td>
<td>8</td>
<td>28</td>
</tr>
<tr>
<td>939</td>
<td>2395</td>
<td>41.139</td>
<td>41.131</td>
<td>15</td>
<td>58</td>
</tr>
<tr>
<td>1866</td>
<td>3636</td>
<td>508.871</td>
<td>508.552</td>
<td>18</td>
<td>67</td>
</tr>
<tr>
<td>5365</td>
<td>10560</td>
<td>1590.943</td>
<td>1588.794</td>
<td>63</td>
<td>$&gt; 6\text{min}$</td>
</tr>
</tbody>
</table>

Table 5.2: Comparison of shortest path ($C_{\text{min}} = 0$) with that obtained from visibility graph algorithm.
Demonstration

Clearance-based path demo
Optimal Path in a weighted terrain
The more complex problem

Given start and goal configurations ‘s’ and ‘g’, the problem is to determine optimal path of a mobile agent in a plane subdivided into non-overlapping polygons, with the ‘cost per unit distance’ traveled by the agent being homogeneous and isotropic within each polygon.

An optimal path is defined as a path $P_i$ for which

$\Sigma (w_i \times |e_i|) \leq \Sigma (w_j \times |e_j|)$ for all $j \neq i$, where $w_i$ is weight of edge $e_i$ and $|e_i|$ is the Euclidean length of edge $e_i$ (Mitchell, Papadimitriou, 1991).
Existing approaches

- **Continuous Dijkstra method** – has very high computational complexity; difficult to implement
- **Grid based approach** – accuracy limited to connectivity of a grid; path is usually jagged and ugly (far from optimal)
- **Region graph approach** – obtained path may not be optimal as the underlying graph is based on region adjacency which may not have anything to do with path optimality
- **Building a pathnet graph** – computational complexity is $O(n^3)$ where $n$ is the number of region vertices. This is too high for spatial datasets; sensitive to numerical errors
- **Edge subdivision method** – computational complexity and accuracy depends on placement of Steiner points; Can generate high quality approximations

Our algorithm falls under this category
The approach

Concepts:

• Places Steiner points on region boundaries.
• Constructs a discrete graph with vertices that are either vertices of obstacles or Steiner points.

Original features:

➢ Uses the region space as it is without triangulating it. This implies lesser bending points for the path that lead to higher optimality.
➢ Applies a rotational sweep technique to generate the discrete graph.
➢ Uses grid-based refinement techniques to further optimize the path.
Flow chart

Overlay layers

Data layers

weighted planar subdivision

Add Steiner points on region boundaries

Construct Discrete graph (rotational sweep)

Assign weights to edges. Edges common to two regions are assigned minimum of the two weights.

Connect source and goal to discrete graph (rotational sweep)

Compute optimal path between source and goal

Optimize path using refinement techniques
Optimal path in weighted region

Low risk (avoids detour)

High risk (avoids area)
Path follows shipping lanes wherever possible.

Top-left corner: 72.37 degrees latitude and -102.69 degrees longitude.

Bottom-right corner: 69.63 degrees latitude and -96.17 degrees longitude.
More examples

Artificially generated dataset
Different start-goal pairs
Underlying hierarchical data structure
Demonstration

Path planning in a varied terrain
Application to Marine GIS
Goals of system utilization for marine GIS

- Finding intersections among ship routes.

- Identification of high-risk areas based on incident and intersection data.

- Finding an optimal minimum-risk path subject to various constraints like total distance traveled, environment and weather conditions.

- Analysis of incidents and route intersections to identify any correlation between the two.
Flow Diagram

Start

- Ship route information
  - Finding intersections
    - Intersection Points
    - Delaunay Triangulation
      - Proximity Information
      - Identification of high risk zones
      - Visibility Graph
        - Visibility edges
          - Shortest Path
            - Vertices on shortest path route avoiding obstacles
              - End
Steps performed

• Decide on the area under observation. This can be a simple bounding box.

• Determine the ship routes that cross that area.

• Find intersections between routes.

• Consider all incident locations in the area.

• Determine the Delaunay Triangulation of the intersections/incident locations.

• Using the proximity information in the Delaunay Triangulation determine the clusters (high-risk areas). The minimum cluster size can be controlled by user.
Steps performed

• Represent the clusters as Convex Polygonal regions.

• Determine the Reduced Visibility Graph for the set of convex polygonal regions.

• Accept the source and destination from user.

• Add to the Visibility Graph, the visibility edges for these two points.

• Apply Dijkstra's Algorithm on the edges in the Reduced Visibility Graph to find the shortest path between the source and destination.
CG Algorithms

Line Intersection Algorithm -

O ((n+k)*logn) where k is the number of intersections.

A vertical sweep has been used.
Processing at every event point is O (L) where L is the number of segments in the status structure at that point.

The algorithm takes care of all degenerate cases. In case of horizontal lines, the left end point has been taken as the upper endpoint. In case of collinear lines, intersection point is reported only once.

Delaunay Triangulation - O(nlogn)

The winged-edge data structure has been used for representation. The incremental method of construction has been used as it offers maximum flexibility and the advantages of local modification.
CG Algorithms

**Clustering algorithm** – Based on analysis of edge lengths in the Delaunay Triangulation.

**Convex Hull algorithm** - Graham's scan with a complexity of $O(n\log n)$.

**Visibility Graph** – $O(n^2\log n)$ algorithm using rotational sweep.

But are all visibility edges required to find the shortest path in case obstacles are convex polygons?

- The shortest path between any two points that avoids a set of polygonal obstacles is piecewise linear and has vertices which are either vertices of the obstacles or the start and end vertices.

- It can be proved that the shortest path will consist of only those visibility edges that are common tangents between a pair of simple disjoint polygons.
Finding the tangential common segments between a pair of convex polygons is done in $O(n_1 + n_2)$ where $n_1$ and $n_2$ are the number of vertices in the convex polygons.

Finding whether the common tangents cross any obstacle edge takes $O(n)$ time where $n$ is the number of obstacle edges.

Thus the overall time for constructing the Reduced Visibility Graph is $O(n^2)$.
How to find common tangents between a pair of convex polygons?

Check whether the vertex V2 is on same side of e1 and e2. If V2 is on same side, then the visibility edge V1V2 is not a common tangent with respect to convex hull C1. Finding on which side of a line a point is can be done using cross product.
**CG Algorithms**

**Dijkstra's Algorithm** - Takes $O(V^2 + E) = O(V^2)$ time where $V$ is the Number of vertices in the Reduced Visibility Graph. This can be reduced to $O(V \log V)$ by using a balanced binary tree in place of a list for storing the vertices.

Snapshots from program
Description of System

- The program has been implemented in Java using Eclipse SDK.

- The user is provided with the option of either reading the ship routes from a file or entering them on the screen by clicking on the left mouse button. Clicking on the right mouse button would indicate that the route has been entered and the user wants to enter the next route. An option has been provided to save the user-entered routes to a text file.
Snapshots
Snapshots of System
Snapshots of System
Summary

• A Delaunay triangulation based clustering algorithm has been developed which is able to detect complicated cluster arrangements and is robust in the presence of noise.

• The clearance-based optimal path finding algorithm has been experimentally observed to be successful at reporting high quality optimal paths.

• The geomery-based solution to the weighted region problem has been successfully applied in planning the route of a ship amidst landmass, sea-ice and high-risk areas.
Future Research Directions

- Investigation into how to completely automate the clustering process.
- Incorporation of learning and AI methods in path planning process
- Adding temporal analysis to spatial data
- Conducting user-studies on system features and interface
Publications


Thank you for your attention!
Only distance may not be enough

Inter-cluster distance not greater than intra-cluster distance:
Time Comparison

Cluster size Vs CPU time in seconds
(550 MHz processor, 128 MB memory)

Vladimir Estivill-Castro, Ickjai Lee,

Cluster size (in 1000) Vs CPU time in milli-seconds
(3 GHz processor, 512 MB memory)

CRYSTAL
Grow-cluster phase is $O(n)$
Resolution Vs. Number of vertices

Resolution Vs. Time consumed (sec)
Data sources

Sea-ice layer:

Landmass:

Risk-areas: