Terrain Analysis in Real-Time Strategy Games: An Integrated Approach to Choke Point Detection and Region Decomposition

Luke Perkins
Rensselaer Polytechnic Institute
110 Eighth Street, Troy, New York 12180
lukeperkins@alum.rpi.edu

Abstract

Autonomous agents in real-time strategy (RTS) games lack an integrated framework for reasoning about choke points and regions of open space in their environment. This paper presents an algorithm which partitions the environment into a set of polygonal regions and computes optimal choke points between adjacent regions. This representation can be used as a component for AI agents to reason about terrain, plan multiple routes of attack, and make other tactical decisions. The algorithm is tested on a set of popular maps commonly used in international Starcraft competitions and evaluated against answers made by human participants. The algorithm identified 97% of the choke points that the participants found and also identified a number of bottlenecks that human participants did not recognize as choke points.

Introduction

In recent years the domain of real-time strategy (RTS) games has become a growing area of research due to the complex challenges such environments present. In RTS games, two or more players gather resources in order to research upgrades and construct armies of up to several hundred units with the goal of defeating the other players. Since players can control each of their units individually to perform a number of distinct actions, the number of unique actions a player can chose from at any point in time can be astronomical making traditional AI search algorithms impractical.

In most RTS games, resources are placed in different areas throughout the map. As a result, players that control more territory naturally have access to more resources and can afford to construct larger armies. The player with the most map control is often able to construct the strongest army and defeat his opponent. Thus, a key sub-goal to winning an RTS game is to gain an overwhelming amount of map control.

To gain control of a particular region of the map currently occupied by the enemy, an autonomous agent must be able to reason about different avenues of attack and decide the best directions to attack from. Even if the agent has a superior army, assaults which fail to take into account the bottlenecks in the terrain are unlikely to succeed.

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Once a region has been conquered, the agent must defend the region against possible attacks from the enemy. The agent must be able to determine where to place defensive structures and units so as to best fortify its base. Only with a tactical awareness of choke points can the agent effectively withstand attacks from players with superior armies.

Thus, a major challenge facing any RTS agent is to recognize the tactical choke points in the terrain, partition the map based on these choke points into meaningful connected regions, and reason about the graph structure of these regions and choke points on the map. This paper addresses this challenge with a novel algorithm which transforms a Voronoi diagram of the environment into a representation that consists of polygon-shaped regions connected by choke points.

Problem Definition

The traversability information of a map can be represented as a two dimensional boolean array of tiles where each tile either permits ground unit movement (traversable) or does not (untraversable). Dynamic obstacles such as units and buildings are not considered in this paper.

Regions are connected components of traversable tiles that do not have any bottlenecks or choke points within them that could impede unit movement. The border of a region can be represented as a simple polygon and while regions do not have to be convex, units located inside a region should be able to travel to other parts of the region with relative ease. A region can be connected to one or more other regions via choke points. Mathematically, regions can be represented by vertices which are connected to each other by edges which represent the choke points.

Choke points are corridors that connect exactly two regions. Each choke point has a narrowest gap which, if walled-off, would prevent movement through the choke point. Each choke point falls into one of two categories:

1. Choke points which, if walled-off, would merge two distinct obstacles into one larger obstacle.
2. Choke points which, if walled-off, would split a connected component of traversable tiles into multiple components, preventing movement between the two disconnected components.

The two types of choke points are shown in Figure 1. Observe that walling-off the left choke point merges the two

distinct obstacles while walling off the right choke point splits the free space into an inner component and an outer component.

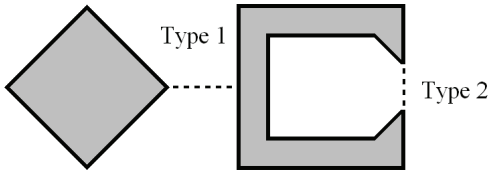


Figure 1: The two types of choke points

Thus, the problem is to partition the terrain into regions via choke points. The accuracy and correctness of the algorithm will be determined by comparing the results against answers made by human participants.

Related Work

An enhanced Space Filling Volumes algorithm named DEACCON (Hale 2008, Heckel 2009) has been developed which automatically decomposes free space into a set of convex polygons. This algorithm generates a navigation mesh which works well for agent navigation; however, it may not be the best algorithm to use for choke point detection. Since the algorithm enforces region convexity, it must split free space into multiple regions even when no significant choke points exist. The simplest example of this is the case of a concave connected component of free space with only one local maximum, such as the shape of a crescent moon. Such a component of free space has no choke points, yet the DEACCON algorithm must split the component into two or more parts to enforce region convexity.

For choke point detection, existing image-processing algorithms grow auras of influence outward from each obstacle (Higgins 2002, Obelleiro 2008). When the auras from two different obstacles come into contact, the point of contact is recognized as a choke point in the environment. These algorithms are great at detecting choke points between distinct obstacles; however, type 2 choke points are missed because they occur between two parts of the same obstacle (see Figure 1). Obelleiro's paper also identified a problem where sometimes no single value for the aura size would yield an optimal region decomposition for the whole map. The approach presented in the next section of this paper evolved from an attempt to extend these algorithms to solve these problems; however, the tile-based aura approach proved unwieldy when additional functionality for detecting type 2 choke points was incorporated.

Another image-processing approach computes a discrete Voronoi diagram of obstacles (Forbus 2002). A Voronoi diagram of obstacles consists of a set of points which are equidistant from two or more distinct obstacles. Forbus's algorithm uses this to find intersection points (pixels equidistant from three or more distinct obstacles), and from that grow auras in all directions to form the open regions. Like Higgins' and Obellerio's algorithms, Forbus's is unable to detect type 2 choke points, so it misses regions with only one entrance. Additionally, in some cases an open region may occur where no Voronoi diagram intersection points occur, such as in an empty room with only two doorways.

The Voronoi diagram representation of open space has been used by others as the basis of a framework which provides agents with a spatial awareness of the environment (Perkins 2008). This framework provides agents with information about the intrinsic spatial qualities of the environment; however, unfortunately it was not developed toward the goal of choke point detection or region decomposition.

Finally, other methods of providing spatial awareness in RTS games have been developed such as influence points, influence maps, and potential fields (van der Sterren 2001, Tozour 2001, Hagelbäck 2008), but these approaches do not seem to be useful for an integrated approach to choke point detection and region decomposition.

Algorithm

The algorithm presented in this paper can be outlined as the following sequence of eight steps:

1. Recognize Obstacle Polygons
2. Compute Voronoi Diagram
3. Prune Voronoi Diagram
4. Identify Region Nodes
5. Identify Choke Point Nodes
6. Merge Adjacent Regions Based On Thresholds
7. Wall Off Choke Points
8. Recognize Region Polygons

Each step shall now be described in detail. For illustrative purposes, the result of each step will be shown for the popular Starcraft map Byzantium 3, shown in Figure 2.

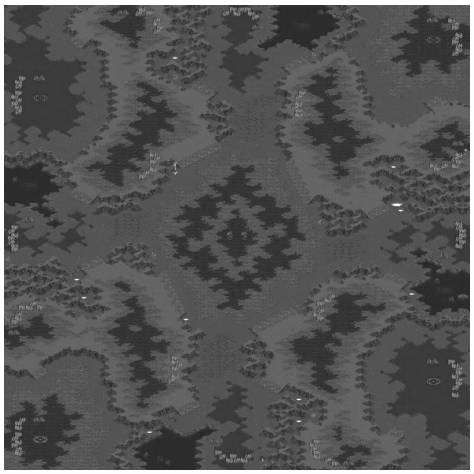
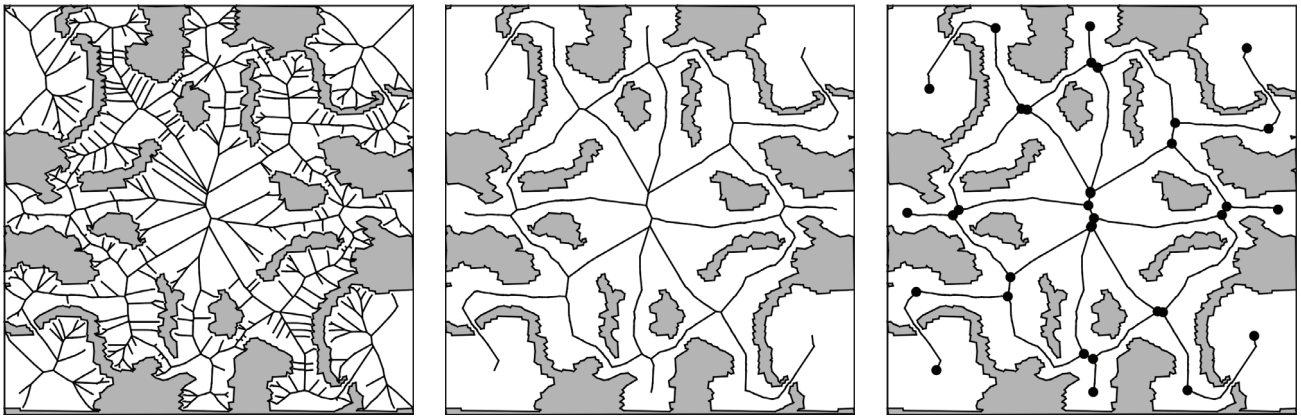


Figure 2. A typical Starcraft map

1. Recognize Obstacle Polygons

First the algorithm converts the two-dimensional binary traversability array into a geometric representation of obstacles. This process, called vectorization, can be implemented in many different ways and in general is a rather complex problem. In this case the pixels are binary values (traversable or untraversable), so a simple algorithm will suffice. First, the map is flood-filled to determine the connected components on the map. Second, the algorithm traces



A B C

Figure 3: Steps one through four (A) Computed Voronoi diagram (B) Pruned Voronoi diagram (C) Identified region nodes

the outline of each connected component to construct a polygon border. Third, polygons of traversable components that are inside polygons of untraversable components are represented as holes in those polygons, so that each obstacle on the map is represented as a polygon with zero or more holes. Fourth, obstacles that are smaller than a certain threshold are discarded. Finally, all the obstacles are simplified to reduce the number of vertices in each polygon, while still approximating the shape of the actual obstacles to sufficient detail. The obstacle polygons are shown in grey in Figure 3.

2. Compute Voronoi Diagram

The second step is to compute the Voronoi diagram of line segments from the edges of the polygonal obstacles. This is accomplished using the efficient and robust 2D Segment Delaunay Graph implementation in the open source Computational Geometry Algorithms Library (CGAL). This implementation is based on the work of (Karavelas 2004) among others and is beyond the scope of this paper. The output of this step is shown in Figure 3(A).

3. Prune Voronoi Diagram

The third step of the algorithm is to trim off unnecessary portions of the Voronoi diagram. First, the radius of each vertex in the graph is computed, where the radius of a vertex is defined as the distance from the vertex to the nearest obstacle or map border. Next, the algorithm iterates over each leaf in the graph, and if the radius of the vertex is less than the radius of its parent, the vertex and adjacent edge are removed from the graph. Removing this edge may result in the adjacent vertex becoming a leaf, so this process is repeated until no more edges or vertices can be removed from the graph. As a result, the only leaves that remain in the graph have a greater radius than their parent vertices. Last, the algorithm removes all isolated vertices that have a radius less than a certain threshold. In Figure 3(B) it is apparent that this step greatly reduces the complexity of the Voronoi diagram while still representing the overall structure of the map.

4. Identify Region Nodes

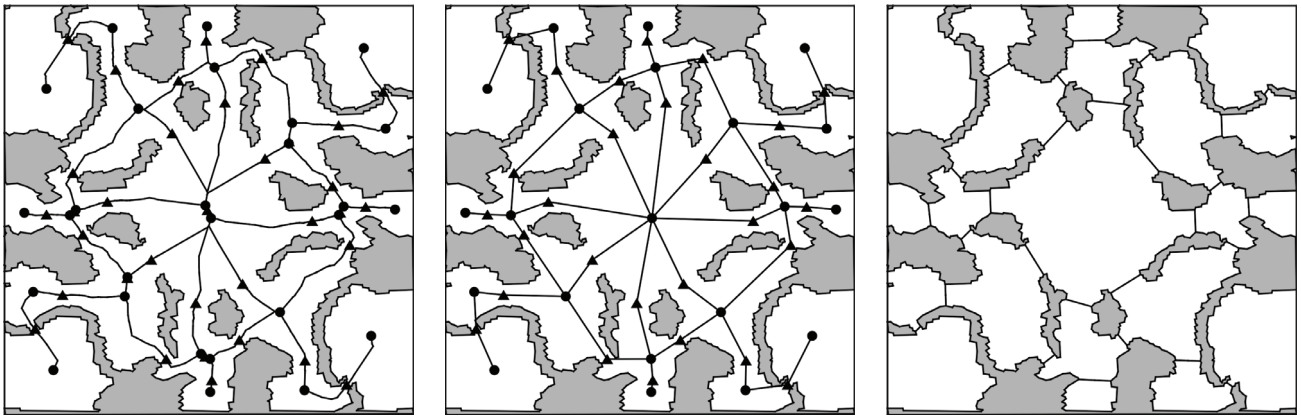
The fourth step begins the process of partitioning the map into regions. This begins with marking certain nodes as region nodes. All nodes of degree other than two are marked as region nodes because they represent important areas in the structure of the map such as intersections, leaves, and isolated components. Nodes of degree two are marked as region nodes only if they are locally maximal and above a certain radius. A node A is locally maximal if for every other node B that is within A's radius, the radius of B is less than the radius of A. In Figure 3(C) the region nodes are depicted as large black dots.

5. Identify Choke Point Nodes

Since all vertices with degree not equal to two are now region nodes, it follows that all remaining unlabeled vertices in the graph must have a degree of two and thus must be located along some path that connects two region nodes. Thus, the fifth step of the algorithm walks along each path, finds the vertex with the smallest radius, labels it as a choke point node, and links it to the pair of region nodes it connects. It is possible that the vertex with the smallest radius is one of the end points of the path (such as when two region nodes are adjacent) so sometimes a vertex is both a region node and a choke point node at the same time. The other odd situation that can occur is when a path forms a loop with a single region vertex. In this case the algorithm discards the path, which makes more sense than adding a choke point node that connects a region node to itself. In Figure 4(A) the choke point nodes are depicted as black triangles.

6. Merge Adjacent Regions Based On Thresholds

Since the map has more region nodes and choke point nodes than is necessary to represent the terrain, it is necessary to merge adjacent regions based on thresholds. If the algorithm were to proceed directly from step five to step seven, the algorithm would partition the map into an excessive number of regions and place choke points where no bottlenecks exist. To rectify this, step six of the algorithm merges adjacent



A B C

Figure 4: Steps five through eight (A) Identified choke point nodes (B) Merged adjacent regions (C) Final result

regions and removes unnecessary choke points. The problem of deciding which regions to merge currently appears to be rather complex, and it has yet to be determined what are the best criteria to use. However, the following criteria have been shown to produce suitable results.

Two adjacent regions are merged when either of the following two conditions are met. First, two regions are merged if the radius of the choke point connecting them is larger than 90% of the radius of the smaller region node, or larger than 85% of the radius of the larger region node. The second criterion applies specifically in the case where one of the regions has exactly two choke points. For a region with two choke points, the region is merged with the adjacent region that is connected to the larger of the two choke points if the radius of the larger choke point is greater than 70% of radius of the original region node.

These criteria appear to work well in many cases; however, they are not perfect. Furthermore, if adjacent regions are merged one choke point at a time, the order in which they are merged can impact the final result. While it has not yet been determined what order of merge operations is best, to keep the algorithm deterministic the algorithm currently examines each choke point in order of decreasing radius. The result of this step is shown in Figure 4(B), where the Voronoi diagram has been replaced with line segments to show which region nodes are linked to which choke point nodes.

7. Wall Off Choke Points

Once the region and choke point nodes have been established, the final goal is to compute the exact polygon shape of each region and compute the two sides of each choke point. To do this, the algorithm iterates over each choke point and looks up the set of nearest points that lie on the neighboring obstacle polygons. CGAL's 2D Segment Delaunay Graph implementation facilitates this step by caching the pairs of obstacle line segments that define each edge in the Voronoi diagram. Once the two end points are retrieved, a line segment connecting them is inserted into the 2D arrangement. Figure 4(C) shows the obstacle polygons along with these new choke point walls.

8. Recognize Region Polygons

Once the choke points are walled off, each region node is in a unique face in the arrangement of obstacles and choke point walls, so the algorithm looks up the corresponding face for each region node and iterates over the edges of the face to determine the polygon border of the region. This completes the process of region decomposition and the final result is shown in Figure 4(C) where each white polygon is a region, and the black lines between two adjacent white polygons represent the choke points.

Implementation

This algorithm has been implemented as a free open source C++ library which can be used by other AI researchers and developers under the General Public License, version 2. The library, called the Brood War Terrain Analyzer (BWTA), reads map data using the Brood War Application Programming Interface (BWAPI) and thus currently only works for the Starcraft: Broodwar RTS game. BWTA uses CGAL to handle the construction of the Voronoi diagram of line segments and the processing of the 2D arrangement of obstacles and choke point walls. Additionally, BWTA uses the Qt framework to draw the results of each step as well as the final region decomposition. Figures 3, 4, and 5 were generated by the BWTA library. Source code, user documentation, and binary releases of this library are available at http://code.google.com/p/bwta/.

Results

The BWTA library has been evaluated on a set of RTS maps known as the iCCup Map Pack which is freely available from http://www.iccup.com. The iCCup Map Pack is a compilation of 48 Starcraft maps that are commonly used in international Starcraft competitions. In the interest of space only six of the 48 maps have been included in Figure 5; however, the full results of the algorithm for all 48 maps in both graphical form and XML form as well as raw map data for future research on the iCCup Map Pack data set are archived at http://code.google.com/p/bwta/.

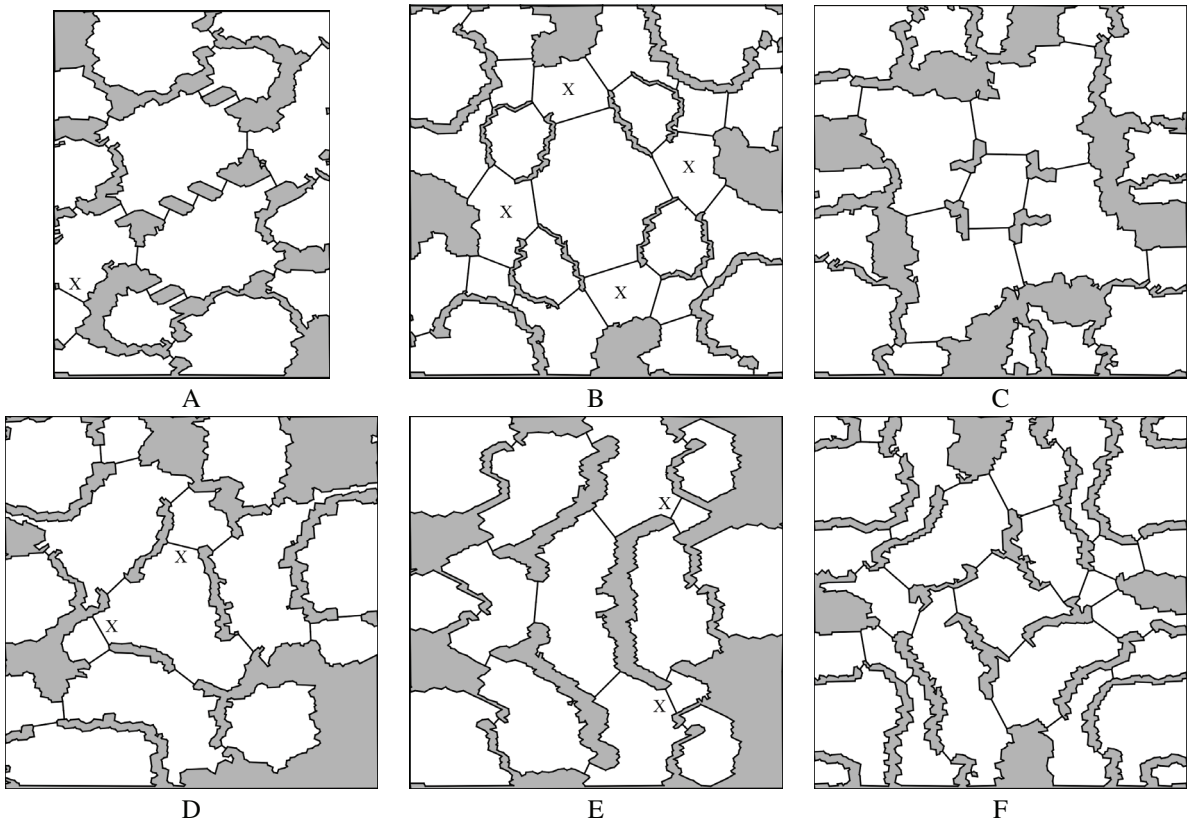


Figure 5: (A) Destination 1.1 (B) Enarey 1.3 (C) God's Garden (D) Longinus (E) Rush Hour 3.1 (F) Return of the King

Since there is no known mathematical formula that can objectively and quantitatively evaluate the accuracy of these results, they have been compared to a majority rule analysis assembled from seven independent human participants. In the majority analysis, if at least four of the seven participants found a choke point between two particular obstacles or points of the same obstacle, then that choke point was inserted into the majority rule analysis for the map.

The two main types of errors to look for when comparing the results of the algorithm against the human majority rule analysis are false positives and false negatives. False positives occur when the algorithm inserts a choke point where the majority of human participants did not, and false negatives occur when the algorithm fails to insert a choke point where the majority of human participants did. The total number of correct choke point matches, false positives, false negatives, and running time of the algorithm in seconds on each map are shown in Table 1:

Map	Match	False +	False -	Time
Destination 1.1	23	1	0	19
Enarey 1.3	23	12	0	31
God's Garden	16	4	0	27
Longinus	15	2	0	28
Rush Hour 3.1	15	2	0	26
Return of the King	20	11	4	43

Table 1: Algorithm Accuracy and Running Time (seconds)

With the exception of Return of the King, the algorithm found every choke point that human participants found; however, it also identified a number of additional choke points. Each of the six maps will now be examined.

The false positive in Destination 1.1 is marked with an X in Figure 5 (A). This choke point is only very slightly smaller than the regions it connects, so its easy to see why most participants would not consider it significant.

In Enarey 1.3, the three largest choke points adjacent to each of the four regions marked with an X were not present in the human majority rule analysis. However, to call these choke points erroneous may be overly simplistic since leaving out these 12 choke points would make the center region very large, highly concave, and thus arguably more difficult to reason about from a tactical view point.

Most of the participants did not place the four radial choke points in God's Garden that touch the four obstacles in the center of the map, making them false positives in Figure 5 (C). However, leaving these four choke points out results in a single outer region with a hole in it. This ring topology would likely add unnecessary complexity to tactical algorithms with little benefit, if any. These four choke points are also significantly narrower than the regions they connect, which means they would still prove to be a bottleneck once a player's army reached sufficient size.

In Longinus, the two false positives are each marked with an X. While the lower left false positive is almost certainly

useless given the slightly narrower choke point right below it, the choke point next to the upper X could potentially be used in certain situations, such as when an opponent is trying to gain control of the center region. This map also has a type-2 choke point which is correctly identified.

Each of the two false positives in Rush Hour 3.1 are marked with an X. These choke points are clearly unnecessary because they each have a much smaller choke point right next to them which walls off the same path.

In Return of the King, most participants placed no radial choke points to the four obstacles in the center of the map, resulting in another outer region with a ring topology. Participants also placed two choke points rather than one in each of the 4 major corridors, technically resulting in 4 false negatives. However, when participants placed two choke points in a single bottleneck they appeared to do so in order to mark the end-points of the corridor, rather than the narrowest cross-section of the gap. From this it is clear a more descriptive representation of these terrain features is needed in order to be able to express the end points of long corridors.

These results have shown that overall the algorithm works well on a variety of popular competitive Starcraft maps, yet may also have room for improvement in future research.

Future Research

Several directions for future research have been identified.

First, the optimal criteria for merging adjacent regions have yet to be determined. The criteria presented in step six of the algorithm have been shown to produce suitable results; however, these criteria were chosen by randomly trying different thresholds, so the criteria are most liked not optimal. If future research developed a method that could analyze and numerically score a given region decomposition, then region merging criteria could be evaluated by determining the average score of the results obtained a wide variety of maps. With the ability to evaluate a given set of region merging criteria, optimization algorithms could be employed to determine the optimal criteria.

Next, the algorithm in its current form only represents long corridors as choke points which span the narrowest cross-section of each corridor. Autonomous agents would likely benefit from a more descriptive representation of this terrain feature such as a polygon which includes a start edge at one end of the corridor and an end edge at the other end. Obelleiro's algorithm has already solved this so heuristics inspired from a geometric interpretation of his image-processing approach could be applied here.

Last, while the algorithm has been presented in the context of RTS games, it may prove useful in a wide variety of other domains. For example, choke points often play a major role in first person shooters and other action games. If the algorithm were extended to analyze the traversability of these 3D environments, computer-controlled opponents could work together to block off the player's escape routes. Similarly, the algorithm could give indoor mobile robots an awareness of choke points and prevent the robots from stopping in door ways and other bottlenecks. Finally, the regions generated by this algorithm are bottleneck-free so the algorithm may be useful for pathfinding algorithms.

Conclusion

This paper has presented a novel algorithm and accompanying open source framework for choke point detection and region decomposition. An analysis of the results has shown that the algorithm detects choke points in locations that closely match that of human participants. This algorithm addresses a major challenge that most autonomous agents face in the complex environment of modern RTS games. With the representation of the terrain produced by this algorithm, AI agents are able to reason about the terrain and make tactical decisions that take into account and exploit the significant choke points on the map, enabling AI researchers to create more formidable autonomous agents.

References

Buro, M., and Furtak, T. M. 2004. RTS Games and Real-Time AI Research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*: 6370.

Forbus, K. D.; Mahoney, J. V.; and Dill, K. 2002. How Qualitative Spatial Reasoning Can Improve Strategy Game AIs. *IEEE Intelligent Systems* 17(4): 25-30.

Hagelbäck, J., and Johansson, S. J. 2008. Using Multi-agent Potential Fields in Real-time Strategy Games. In Padgham, L., and Parkes, D., eds., *Proceedings of the Seventh International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*.

Hale, D. H.; Youngblood, G. M.; and Dixit, P. N. 2008. Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds. In *Articial Intelligence and Interactive Digital Entertainment (AIIDE)*: 173-178.

Heckel, F. W. P.; Youngblood, G. M.; and Hale, D. H. 2009. Influence Points for Tactical Information in Navigation Meshes. In *Proceedings of the Fourth International Conference on Foundations of Digital Games*: 79-85.

Higgins, D. 2002. "Terrain Analysis in an RTS-The Hidden Giant." *Game Programming Gems 3.* Hingham Mass.: Charles River Media: 268–284.

Karavelas, M. I. 2004. A robust and efficient implementation for the segment Voronoi diagram. In *Proceedings of the International Symposium on Voronoi diagrams in Science and Engineering (VD2004)*: 51-62.

Perkins, S.; Jacka, D.; and Marais, P.; and Gain, J. 2008. A Spatial Awareness Framework for Enhancing Game Agent Behaviour. In *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video games*: 15-22.

Obelleiro, J.; Sampedro, R.; and Cerpa, D. 2008. "RTS Terrain Analysis: An Image-Processing Approach." *AI Game Programming Wisdom 4.* Boston, Mass.: Charles River Media: 361–372.

Tozour, P. 2001. "Influence Mapping." *Game Programming Gems 2.* Hingham, Mass.: Charles River Media: 287–297.

van der Sterren, W. 2001. "Terrain Reasoning for 3D Action Games." *Game Programming Gems 2.* Hingham, Mass.: Charles River Media: 307–316.