Search Strategies for Geostatistical Simulation of Unstructured Grids

John Manchuk
Department of Civil & Environmental Engineering,
University of Alberta

Abstract

A common task in geostatistical simulation is a search for nearby relevant data. A variety of indexing schemes such as the spiral search and the super block search have been used in practice. The spiral search is commonly used for regular grids. The super block search is used for data sets that do not change in size. Unstructured grids, in most cases, cannot be indexed for either the spiral or super block search. Alternative search strategies must be considered when dealing with unstructured grids. This note describes the implementation of search trees.

Introduction

Searching for nearby original data at a small scale, regularly gridded soft data, and grid blocks of varying size makes most common indexing schemes impractical. There are many different configurations of grid blocks specified by their centroid locations (x,y,z) and geometric characteristics. There could be millions of locations to be searched including data points and previously simulated nodes.

There are several ways to search for nearby data. Using an exhaustive search is possible for small problems. A n_{gb} by n_{gb} matrix of distances must be searched; where n_{gb} is the number of grid blocks. Implementing a super block search strategy is another option. The centroids of all grid blocks and data locations would be indexed using the conventional super block search method [1]. Another possible method of indexing large data sets is the use of search trees.

Search Tree Structure

There are two types of search trees popular in computer graphics: quadtrees (2-dimensional) and octrees (3-dimensional) [2]. In the 2-dimensional case, a quadtree could be implemented to organize data so that operations such as point location, region location, and neighbor searches can be done quickly. For applying search trees to geostatistics, nearest neighbor searches and region queries would be important for acquiring conditioning data and previously simulated nodes. Point location operations will be needed for inserting simulated nodes into the search tree. Quadtrees work by taking the initial set of data and dividing it up into quadrants. If the number of points within each quadrant exceeds a specified maximum, they are divided into sub-quadrants. This process continues until all quadrants in the tree contain at most the specified maximum. Octrees work identically to quadtrees except there are 8 regions created when an octant is sub-divided. Quadtrees and octrees are often created such that all regions at a specific level in the tree are the same size; however they can be built using other methods. To divide the data as evenly as possible, the quadrants or octants can originate from the median of each data set being divided. Figure 1 shows representations of both a quadtree and an octree.

The first cell in a search tree is called the root cell. If a cell is split into quadrants or octants, it is referred to as a parent cell and it contains a pointer to the first of a set of children cells (there would be four children in a quadtree). Each child cell contains a pointer to its parent. If a child cell no longer needs to be partitioned, it is referred to as a leaf node and it points to the data it contains.

Another popular type of search tree is a multidimensional binary search tree (kd-tree) that can be used to sort data of higher (k) dimensions [3 and 4]. Like the octree or quadtree, data can be sorted by having one or several data points per region. A kd-tree works by splitting the data into two regions per partition. Each cutting plane's location and orientation are chosen based on the median of the longest aligned axis for each region's data. As the process narrows to meeting a specified requirement of one or multiple data per region, a tree is built based on the median locations. Figure 2 shows a schematic of how the data are separated along with a resulting tree.

Nearest Neighbor Searches Using Search Trees

A step in performing simulation is locating nearby relevant data for kriging and search trees can be used efficiently to do this. There are several ways a search tree can be traversed to locate nearest neighboring data. To describe these methods, a quadtree will be referred to. A simple way to construct a quadtree is to store the locations where partitions originate (these locations could be the x and y medians of the data set) and index the parent and children cells in a logical fashion. The extents of each cell also need to be stored. The root cell's index is set to zero and if it is partitioned into four children, these cells receive indexes one through four. If the child cell with index 3 is partitioned, it becomes a parent and the new children cells receive indexes 13 through 16. Equations 1 through 4 show the relationships for indexes in a quadtree and octree:

quadtree Indexes range from: $$I_C = 4I_P + 1 \quad (1)$$

to: $$I_C = 4I_P + 4 \quad (2)$$

octree Indexes range from: $$I_C = 8I_P + 1 \quad (3)$$

to: $$I_C = 8I_P + 8 \quad (4)$$

where I_C is a child index and I_P is the parent index.

The index of a point or block being estimated can be determined by making comparisons starting at the root cell and traversing the tree in a downward fashion. When a leaf node is reached, the index of the point is known. Having this index, the x and y extents (for a quadtree) of a cell can be retrieved. Using the extents, adjacent cells can be found by adding or subtracting a small value to the cell boundaries and traversing the tree using these locations. If a cell smaller than that containing the point to be estimated is found, the tree is traversed upward until the same size cell is reached, then all leaf nodes within that cell are searched. Figure 3 shows a schematic of this process.

Another method of quickly searching for adjacent cells in a quadtree or octree is given by [2]. The method uses location codes which are determined using a cell's left, bottom, and back corner coordinates or by using x, y, and z coordinates of a point. In a quadtree, there will be two location codes, one for x and one for y. In an octree there will be three for x, y, and z coordinates. A cell's location code can be found by multiplying the left, bottom, and back (x,y,z)

corner coordinates by 2^{RL}, where RL is the root level ($RL=N_{levels}-1$), and then representing the resulting number in binary. Acquiring location codes from points will result in binary numbers that do not immediately lead to which leaf node they are in and traversal of the tree is necessary. Figure 4 shows a quadtree with binary indexing.

The location of a point to be estimated can be determined by multiplying the points coordinates by 2RL, truncating the result to an integer, and representing the result in binary. Using the bits in the binary number, the tree can be traversed until a leaf node is reached. Once the cell the point belongs to is known, neighboring cells can be found using the location codes of the cell's boundaries, which are used to traverse the tree. For an example, to determine a cell's upper left neighbor, the x location code of the smallest possible left neighbor and the y location code of the top boundary are used to traverse the tree until a leaf node is reached. The smallest possible left or bottom location codes are determined by subtracting the x or y location codes of the cell by the binary representation of one.

Once a leaf node is determined for a point to be estimated and the neighboring cells have been determined, the data contained within them can be taken and used for kriging or sent to a secondary filtering algorithm to reduce screening or other potential problems. After simulating each point, they are inserted into the search tree in the correct leaf node and if at any point a leaf node contains more data than the specified maximum it gets partitioned.

Region Queries Using Search Trees

Having the ability to quickly select a region of leaf nodes to acquire nearby data from is important to simulation since there will be a specific ellipsoid from which relevant data will come. Currently, there are methods of determining all the leaf nodes within a specific region of space in a search tree; however, these regions take the shape of the cells used to construct the tree. With quadtrees or octrees, rectangular or box shaped regions surrounding the search ellipsoid can be queried quite easily.

Using the method of indexing data explained by Equations 1 through 4 and refereeing to a quadtree, a rectangular region query can be made by using the lower left and upper right coordinates of the rectangle. These coordinates are acquired by adding the proper search radius values to the location to be estimated. Using the coordinates, the tree can be traversed from the root cell until a leaf node is reached, then the common parent of both leaf nodes must be determined. Every leaf node below the common parent will approximately outline the region and can be searched for conditioning data and previously simulated nodes.

[2] describes a method for region queries as well using binary location codes. For a quadtree, the smallest possible enclosing cell is determined by XOR'ing the left and right location codes and the top and bottom location codes of the desired region. The first one bit from the left in the XOR'ed location codes indicates the level in the quadtree where the initial location codes differ. The level of the desired enclosing cell is one above the previously determined level. Using any location code within the region, such as the bottom left corner, the quadtree can be traversed until a leaf node is reached or the desired level is reached, whichever comes first. This will be the smallest possible cell enclosing the desired region. Figure 5 depicts this process.

In Figure 5, if just the ellipsoid was used as the region to pick leaf nodes, perhaps the leaf node on level 1 with x and y location codes 0100 and 0100 respectively would have been rejected. A method of performing region queries, where the region does not follow the shape of the cells used

to construct the search tree, would be useful for application to geostatistical simulation. The result would be fewer leaf nodes to search for nearby data.

Applying Search Trees to Simulation

There are two ways that search trees could be applied to organizing data for geostatistical simulation purposes. The tree can be built once to include all of the point data and nodes to be simulated or the tree can be built with just the conditioning data and gradually updated as simulation advances. Developing a search tree using the second method would be more efficient as the number of neighboring cells to search through would remain fairly constant. Leaf nodes are progressively split as simulated values are inserted into the search tree making the cells more refined; however, the number of neighboring leaf nodes around any location will remain fairly constant. Building the tree progressively will also ensure than most leaf nodes contain data so one should not have to search beyond the nearest neighboring leaf nodes for conditioning data or previously simulated nodes, see Figure 6.

A quadtree or octree would be ideal for simulation since their structure could also be used to filter data (to reduce screening effects). Limiting the allowable number of data in each leaf node and carrying out simulation as shown in Figure 6 will act similarly to using the octant search within the current version of SGSIM [1]. The allowable number of data per leaf node could also be based on the maximum data and previously simulated nodes to use for kriging.

The way the quadtree is implemented in Figure 6 does not consider the search ellipsoid used or limiting distances imposed by input variograms; however, the distances from all data to the estimate can be calculated and used to filter the data acquired from the search tree. Distances calculated at this point can simply be reused for covariance calculations rather than be recalculated.

Conclusions

Implementing quadtrees or octrees to organize spatial data for simulation purposes would allow for efficient acquisition of nearby data for each node to be simulated. The search for nearest neighboring leaf nodes is not setting dependent (the grid type has no effect on it) and tree traversal for finding and inserting points is a simple process. Having these characteristics along with low memory requirements makes search trees good candidates for organizing and retrieving data during the simulation process.

As mentioned earlier, searching the nearest neighbors does not incorporate any search radius or variogram limits and region queries require the shape of the region follow that of the cells used to construct the tree. An efficient method of determining which leaf nodes of a search tree lie within the specified search ellipsoid may be beneficial; searching only the adjacent leaf nodes may be leaving out some data influential to the estimate.

References

[1] C. V. Deutsch and A. G. Journel. GSLIB, Geostatistical Software and Library and User's Guide. Second Edition. Oxford University Press. 1998

[2] Sarah F. Frisken and Ronald N. Perry. *Simple and Efficient Traversal Methods for Quadtrees and Octrees.* Mitsubishi Electrical Research Laboratories. 2002.

[3] Jon L. Bently. *Multidimensional Binary Search Trees Used for Associative Searching.* Association for Computing Machinery, Inc. 1975.

[4] Tapas Kanungo et al. *An Efficient k-Means Clustering Algorithm: Analysis and Implementation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 7, July 2002.

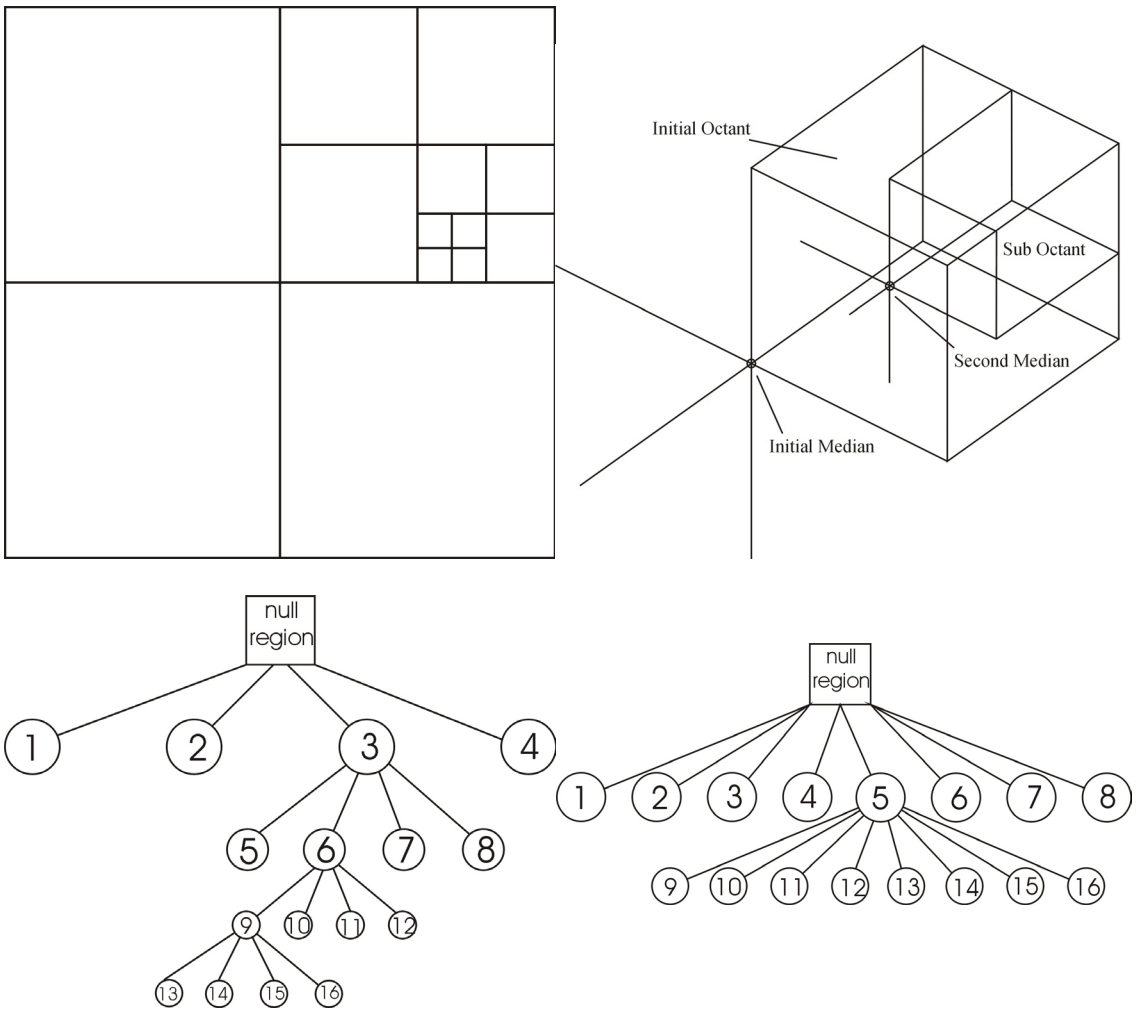

Figure 1: Quadtree (top-left) and octree (top-right) data organization. The corresponding tree representation is shown for each tree structure (bottom). The quadtree is regular whereas the octree is created based on median locations. The leaf nodes (1, 2, 4, 5, 7, 8, 10-12, and 13-16 for the quadtree and 9-16 for the octree) would contain pointers to the data within them. Cells that are split like octant 5 in the octree (bottom-right) contain pointers to the first of a set of children (cell 9).

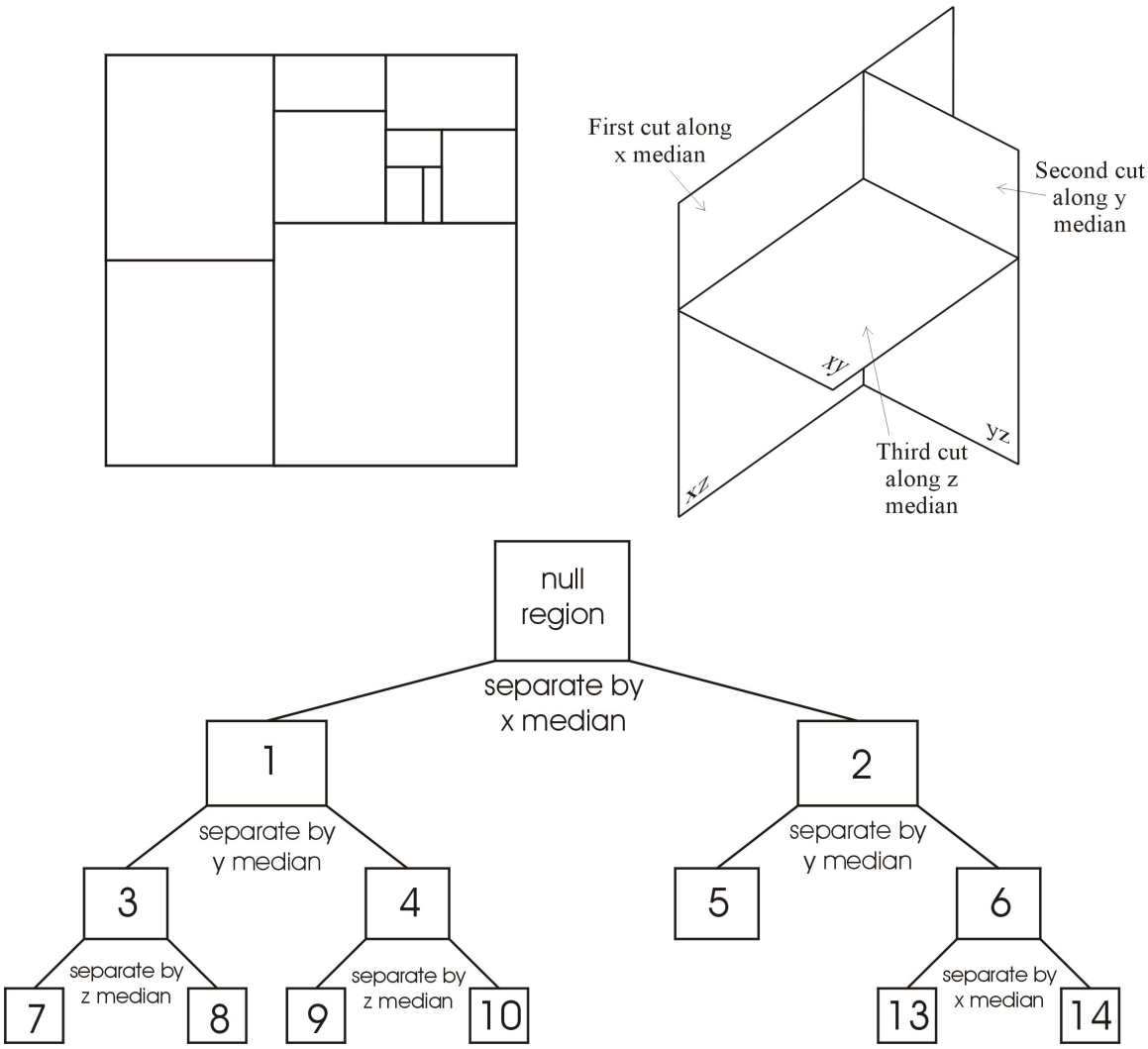

Figure 2: A kd-tree applied to two dimensions (top-left) and three dimensions (top-right) along with a representation of the tree (bottom). The data is partitioned based on median locations until there are at most a specified maximum number of points per region.

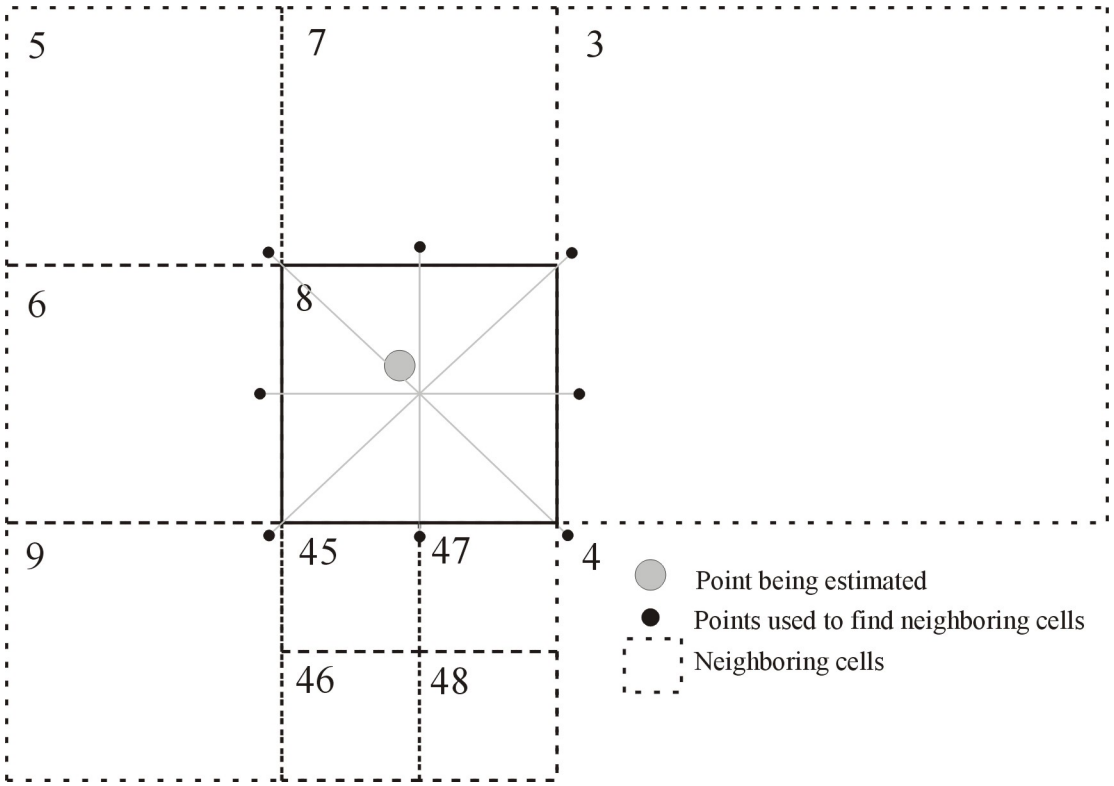

Figure 3: A search method for finding neighboring cells in a quadtree. Numbers in the upper left corner of each cell are its index. The neighboring cells found would point to data used in kriging or data passed to some other filtering algorithm prior to kriging. Since the leaf node with index 45 was smaller than leaf node with index 8, the tree was traversed upward from 45 to 11 and all leaf nodes within cell 11 were taken (indexes 45 to 48).

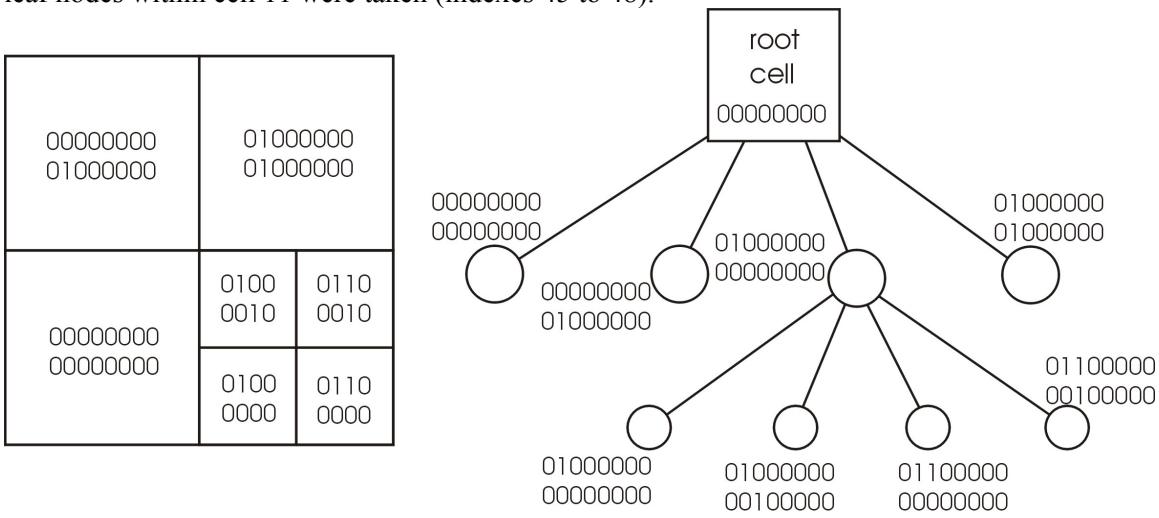

Figure 4: Quadtree representation with binary indexing. The top numbers represent x location codes and the bottom numbers y location codes. For x, 0 indicates left and 1 right. For y, 0 indicates bottom and 1 top. Using binary numbers with 8 bits as shown here would allow a tree a depth of 8 levels including the root level.

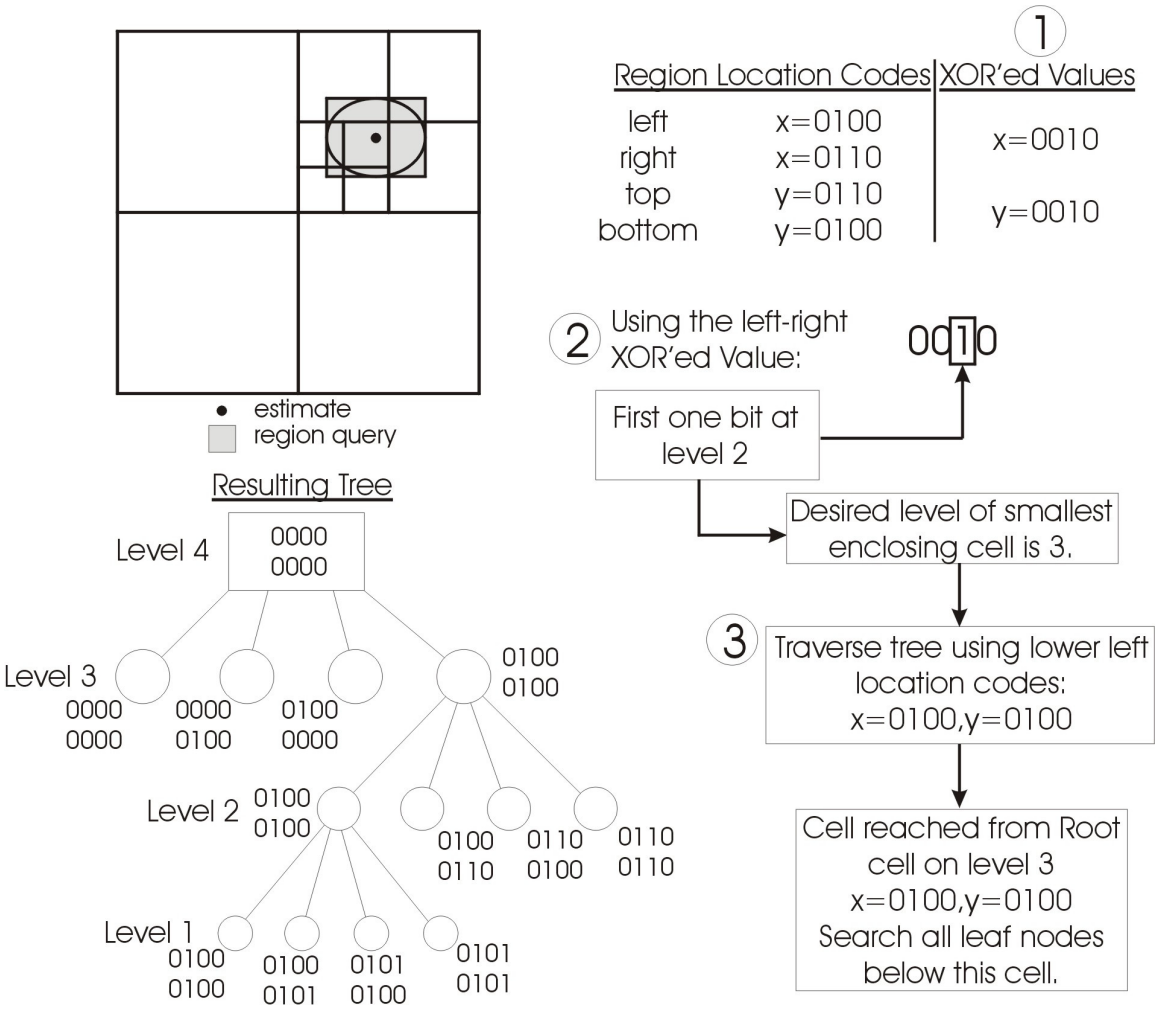

Figure 5: The region query process as defined by [2]. The ellipsoid within the region query might represent the ellipsoid defined by variogram ranges in the x and y directions.

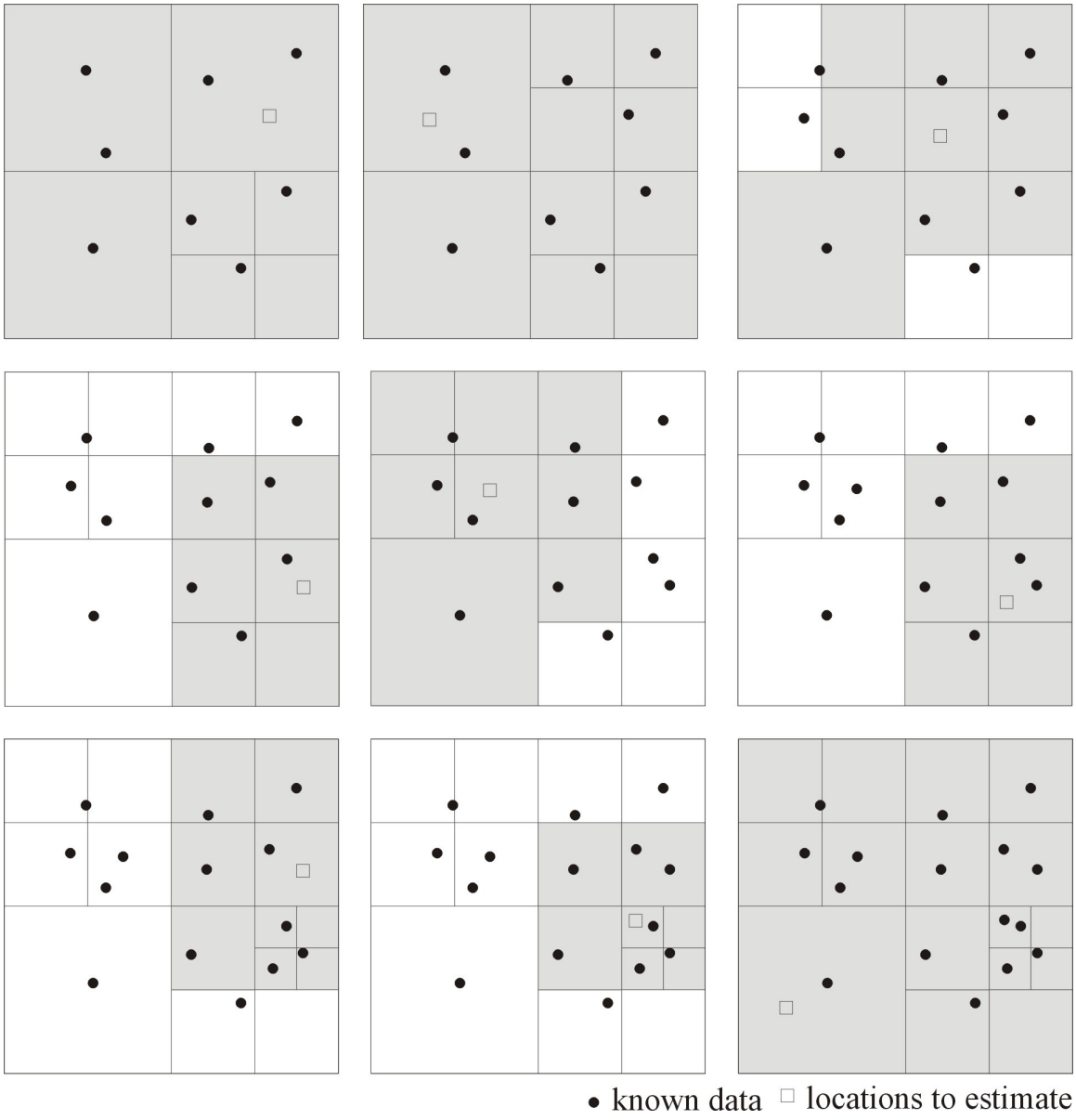

Figure 6: Progression of a quadtree as simulation takes place. The maximum allowable data per leaf node would be two in this case. Solid grey cells are those that would be searched for data for each location being estimated.