On the Use of a Quadtree Search for Estimation and Simulation

R. M.Hassanpour and O. Leuangthong

Centre for Computational Geostatistics
Department of Civil & Environmental Engineering
University of Alberta

Reservoir flow simulation involves complex geometry and geology. The use of unstructured grids is advantageous in resolving important features such as faults, channels and deviated wells. It also permits static and dynamic reservoir properties to be resolved at the required fine scale resolution near well bores and along faults, while coarser resolution is often acceptable far away from wells. Inherent to any inference modeling approach is the ability to efficiently search for nearby relevant data and in the case of simulation, any previously simulated values. Different search algorithms have been successfully employed for regular grids; however, under the context of an unstructured grid, these methods may prove to be inefficient and impractical. There is a need to develop a robust, efficient searching method that can quickly search for nearby information within an unstructured grid.

This paper examines the use of a quadtree search for estimation and simulation. This is initially tested on a regular grid and its efficiency is compared against the super block search. Interestingly, the super block search performed just as efficiently in estimation and even faster than the quadtree search in simulation.

Introduction

Reservoir simulation is often implemented on unstructured grids to resolve the complexity of reservoir geometry. Developing a three dimensional non-Cartesian model enables us to quantify reservoir properties with greater precision than was previously possible. The recent reservoir simulators must be capable of accurately representing these complexities.

Current simulation algorithms such as sequential Gaussian simulation (SGS) (Isaaks, 1990) are unable to consider multiscale data on a regular grid, much less on an unstructured grid. The implementation of direct sequential simulation (DSS) is proposed to solve this problem (Manchuk et. al., 2005), however searching for nearby relevant data remains an issue.

A brute-force approach could be implemented for any type of grid; however, this type of search means to consider every possible solution of a problem until a solution is found, or all possible solutions have been exhausted. Generally, brute force refers to any method that does not rely on any intelligence, but tries every possible solution to find the best answer. This method is applicable for small number of data but becomes time-prohibitive when a large data set is considered.

To counter the inefficiencies inherent to a brute force search strategy, numerous search algorithms have been proposed for geostatistical estimation and simulation. The most common strategies are implemented in the GSLIB software (Deutsch and Journel, 1998), and include the super block search, spiral search and the octant search. The super block search algorithm is used to efficiently find non-gridded data. In this technique a template grid is constructed according to the search ellipsoid and centered at the point of interest. Then all data which are located in the template grid are found and considered for estimation/simulation. In the case where data are located on a regular grid, the spiral search is an efficient alternative (Deutsch and Journel, 1998). In this method the search starts at the location of estimation and follows the spiral path to find the relevant data. The search is terminated when the maximum number of data searched or the search radius is reached. Another common search strategy is the octant search, which is useful if there is significant clustering of the samples. For this search, the search neighborhood is divided into eight equal sectors. If there are too many empty adjacent octants around a block then that block will not be estimated. Zanon (2004) discusses each of these search strategies in more detail and examines their impact on simulation time.

Despite the fact that the conventional search methods found in common geostatistical software works quite efficiently for a regular grid, the extension to unstructured grids poses several challenges. Firstly, the data for an unstructured grid will consist of multiscale, irregularly shaped support volumes, any method that is suited for finding data on a regular grid/spacing will likely be inadequate. Secondly, a template approach presumes that the grid can be resolved into some common pattern that is easily translated throughout the model area; however, the flexibility permitted by an unstructured grid is inconsistent with finding any local templates that could be applied throughout the domain.

Implementation of conventional searching algorithms commonly found in existing geostatistical modeling methods to an unstructured grid is certainly an option. Another option is to consider other search strategies that have been used in the computer graphics and gaming industry for their robustness and efficiency. This paper considers one such strategy: search trees. Specifically, we examine the use of search trees for estimation and simulation. The following section provides some background on tree-based search methods. A small example of how a quadtree search works is presented. This is then implemented as a subroutine for kriging and simulation and a small comparative study is presented.

Tree-based Search Methods

Consider the number of data point distributed spatially in area $\boldsymbol{A}$. Tree based methods index data in a way that can be easily accessible when they are needed. The tree structure is based on the location of data in space and the specification of a root node (or the initial tree's root); the root node contains all sub-nodes. Each node could be either a leaf or a parent node. Parent nodes have one or more child nodes and all children of the same node are siblings (see Figure 1). Data points are assigned to the leaf nodes.

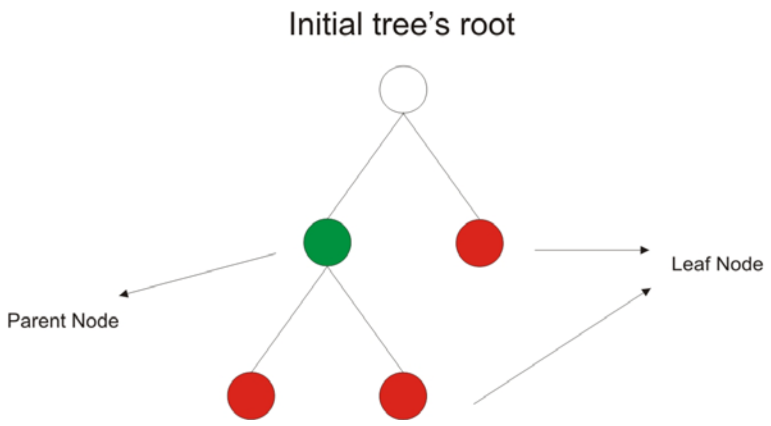

Figure 1: Schematic illustration of Tree data Structure.

Depending on the area decomposition and the way data are indexed, several tree-based data structures are developed and used in computer science applications. Some of the most useful of them are discussed as follows.

Quad-trees

These types of structures are used for two dimensional data spaces. In this structure the regions are defined by squares in the plane, which are subdivided into four rectangles if any regions contain more than a specified number of data inside them. So each internal node in the underlying tree has four children. Figure 2 shows 10 data point in a square region. The space is divided in a way that each region has a maximum of one data. The location of each data is tree is also shown. Data in smaller squares are located in the lower tree's layer. The process of point insertion and deletion is fairly simple in quad-trees. This kind of structure is commonly used in image processing techniques and its advantage in memory storage is highly appreciated.

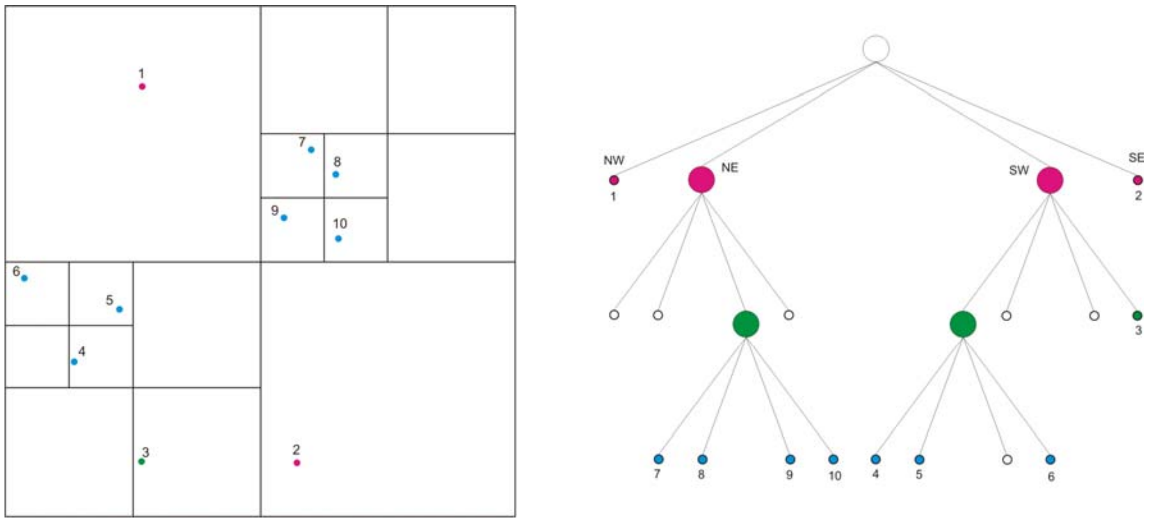

Figure 2: Data indexing using Quadtree. On the left: The space is recursively divided into four squares. On the right: The quadtree structure showing the location of data in tree.

Octrees

In case of 3-D data, the octree data structure can be applied in order to index data. The main concept of this method is the same as a quadtree, but regions are defined by cubes in 3 dimensions, which are subdivided into eight equal-sized cubes (like an octant) for any regions containing more than a single point (Figure 3). So each internal node in the underlying tree has eight children and, like quad-trees, point insertion/ deletion is simple.

k-d trees

This search algorithm is designed to use in multidimensional spaces. Regions are defined by hyperrectangles which are subdivided into two hyperrectangles by cutting a hyperplane through the median point, for any regions containing more than two points. So the underlying tree can be made a binary tree; compared to the octree structure, it may have more layers or depth which considerably affects the efficiency of searching through the tree. Although some variants were proposed to help, such as, cutting along the longest side instead of along each axis recursively, the octree is generally preferred.

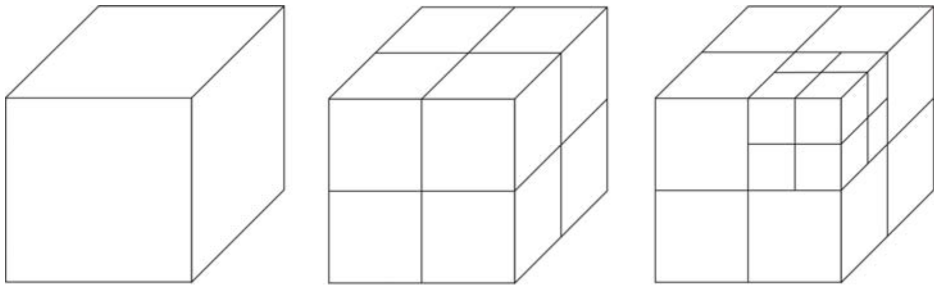

Figure 3: Octree Data Structure. 3-D data space (left end) is divided into eight equal sized cube and recursively repeated to have maximum one point in each cube (right end).

Compressed quadtree or Octrees

Regions are defined in the same way as in a quadtree or octree (depending on the dimensionality), but paths are compressed so that all parent nodes that have just one non empty child are eliminated. Figure 4 shows a quadtree structured and the resulting compressed tree. This tree structure has all advantages of quadtree or octree but it is more memory efficient.

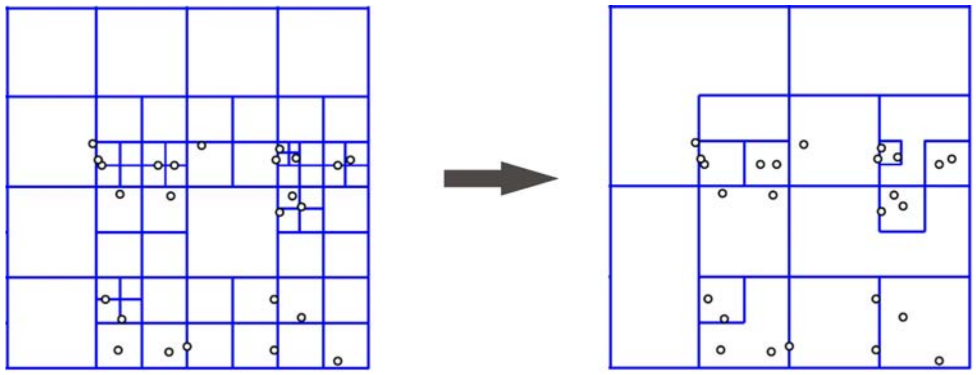

Figure 4: A quadtree structure (left) and the resulting compressed tree (right). (Redrawn from Eppstein *et al.*, 2005)

Algorithm of Point Insertion in Quadtree

There are different types of quadtree structures which are differentiated by the type of data (point, area or volume) and the decomposition process. Commonly, quadtrees are used for point data, curves, surfaces and volume. The decomposition process also may be into equal parts or it may be governed by the input. Due to this variety, the algorithms for data insertion and search would be different. One of the most powerful and efficient algorithm for point data is discussed by Samet (1990).

Data are inserted in the tree one by one. Three different conditions may occurred when a point is inserted in the tree structure: (1) data falls in an empty quadrant and is assigned to the node, (2) data falls in a quadrant which already has a point assigned to it, so the corresponding quadrant is recursively decomposed until each sub-quadrant contains a maximum of one point, or (3) data falls in a quadrant which contains a root and in this case the point is assigned to the relevant quadrant of that root. The algorithm is summarized in Figure 5.

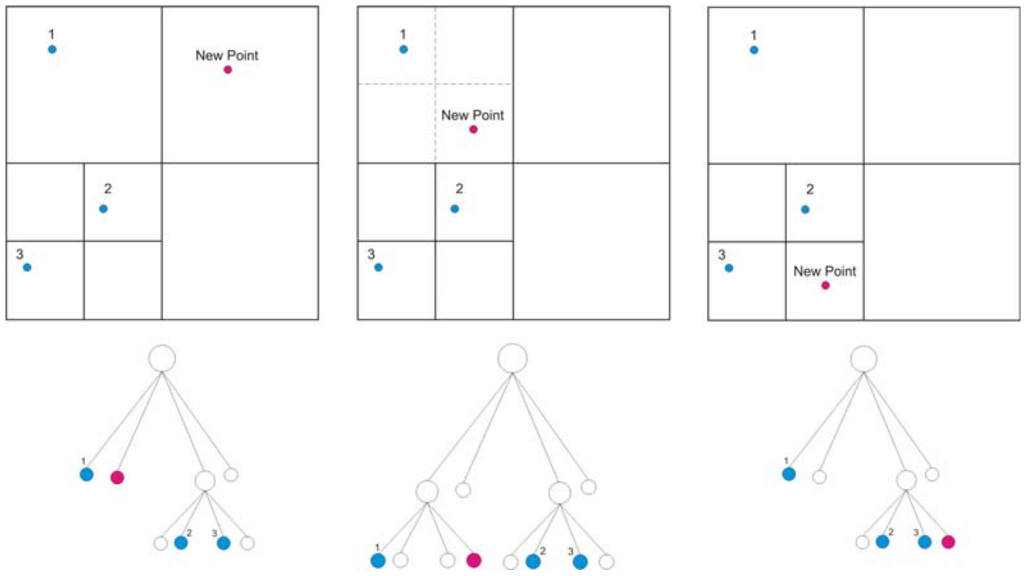

Figure 5: Three cases which may occurred when a new point inserted in a tree. Point inserted in an empty quadrant (left), a quadrant contains data (center) and a quadrant contain a root (right).

Searching within Quadtrees

In geostatistical estimation and simulation, data in a local neighborhood must be identified. Quadtree structure can be used as an option. The main idea of search in quadtree is to reduce the number of quadrants that should be considered for the search. In order to do this, the coordinate of the initial tree's root is compared with the desired point of estimation to find the quadrants of the tree that should be searched by using the template figure (See Figure 6). After all quadrants are found then a check is performed to see if data within those quadrants lie within the search neighbourhood. The following example explains detail of point insertion and search algorithm.

Let's consider six points with the coordinates and their porosity value given in Table 1 . First, let's consider insertion of two first points into tree (one by one). According to the input data, the boundaries of data space and also initial tree's root are defined (X=6.5, Y=5.75). Comparing the first point coordinate to the tree's root, point one is inserted in *SW* quadrant. The same comparison is performed for point 2 and shows that point 2 also resides in *SW* quadrant with respect to the tree's root. Since the *SW* quadrant is already consists of point 1, this quadrant is divided into four sub-quadrants. A new root (parent node) is defined (X=3.5, Y=2.75) and both points 1 and 2 are compared with this new root to define into which quadrant they may be assigned (point 1 in *SW* and point 2 in *NW*). Other data points are entered into tree in the same way (Figure 7). It is clear that the shape of the final tree is independent of the order in which data points are inserted into it.

X	Y	Porosity
1	1.5	6.98
1.5	4.2	7.61
8	10	7.42
6	3.1	11.83
12	7.35	8.49
5	10	13.38

Table 1: Coordinates and value of 6 data used in example.

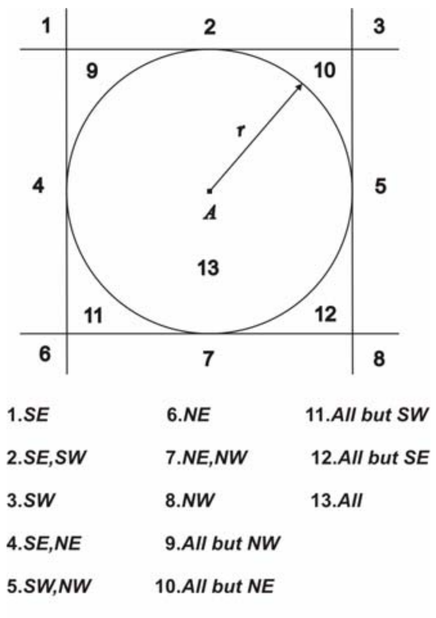

Figure 6: Template figure showing a circular search space and regions in which a root of a quadtree may reside (Redrawn from Samet, 1990). For example if all nodes within the radius r of point *A* are desired and the tree's root, say *R*, is in region 7, then the search can be restricted to the NW and NE quadrants of *R*.

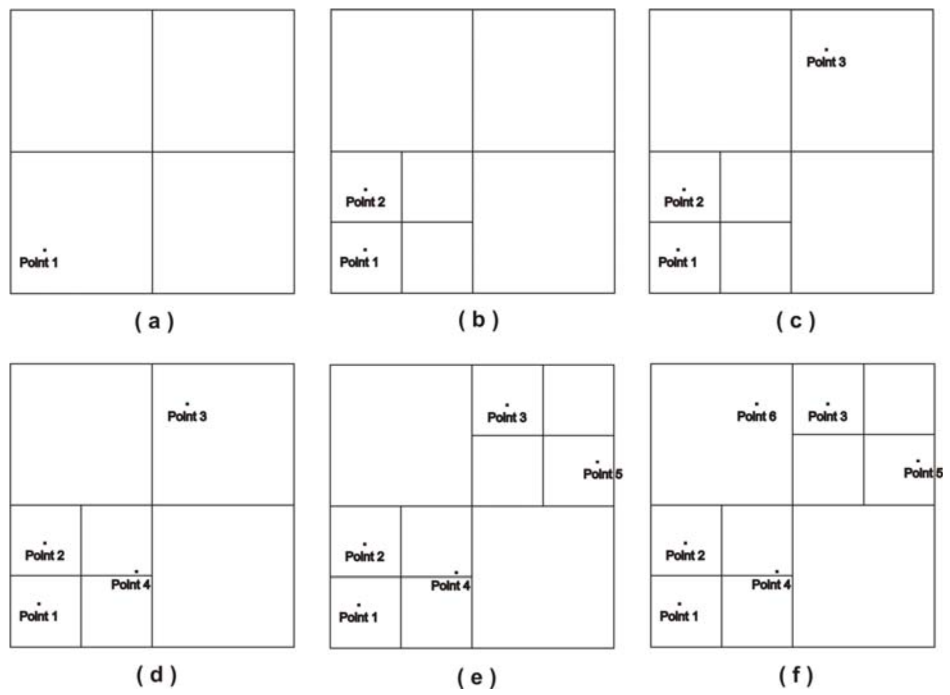

Figure 7: 6 data points are inserted in a tree.

Now let’s consider that we want to search for all data points that fall in within a circle centered at (6.5,8.5) with radius of 2.5 (See Figure 8). According to the search circle and the template figure, the initial tree’s root reside in region 7 and the search area is restricted to the NW and NE quadrant of this root. Thus there is no need to search the data points of SW and SE quadrants. Indeed, the NW quadrant is also a parent node and by referring to the template figure, the NE and SE child of this node should not be considered for search.

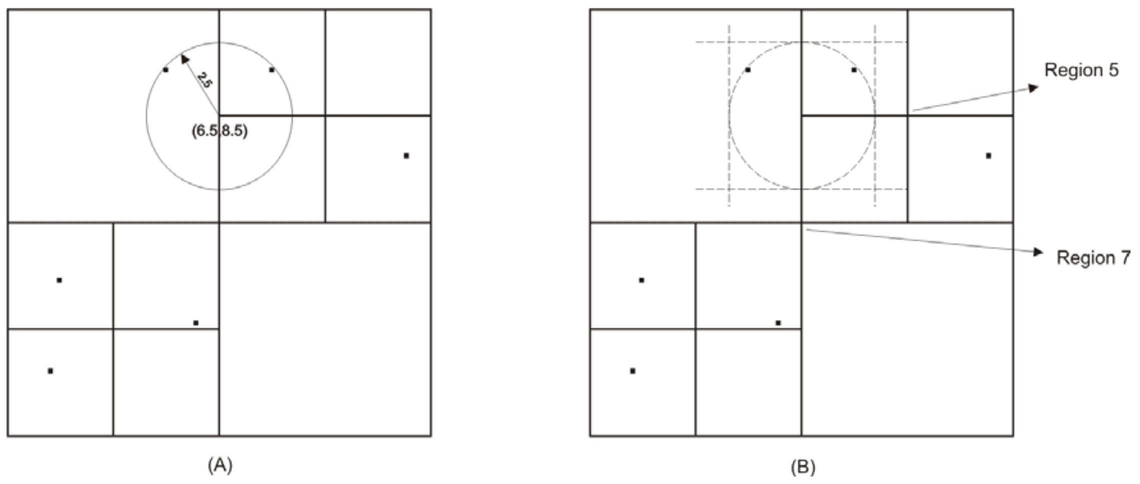

Figure 8: Searching around point (6.5,8.5) with radius of 2.5.

Implementation

The subroutine `srchqdt` was developed to implement the quadtree search. This subroutine returns the number of close data to the estimation point as well as the index of those data in ascending order. This routine can be easily called in any estimation and simulation code for a circular search area.

In order to compare the performance and efficiency of `srchqdt` with the available search algorithms in GSLIB software, a data set with 310 data points is considered (Figure 9).

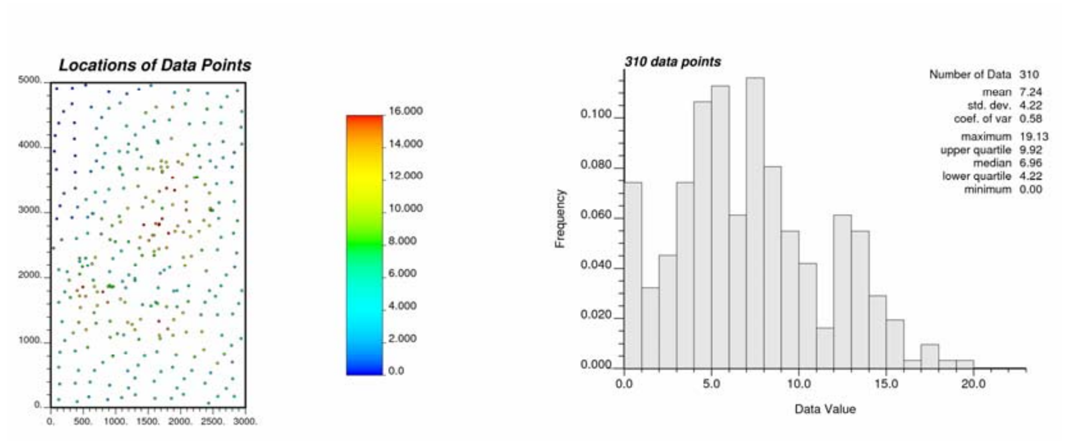

Figure 9. Location map and histogram of 310 samples used in example.

The `kb2d` and `sgsim` codes are modified to use the quadtree search. Estimation (simple kriging) and simulation is performed on the sample data with a quadtree search and the super block search. Figure 10 shows the results of the search algorithm in estimation, and we see clearly that there is no impact on the estimation results due to the search strategy. Figure 11 shows the results of applying the quadtree search and the super block search in simulation. Minor differences are observed between the super block search and quadtree search in simulation which is not significant.

The primary motivation for this study is to examine the computational efficiency of a different search algorithm. As such, the execution time for this example is recorded and tabulated in Table 2. Surprisingly, results show that the super block search is more efficient than quadtree search in terms of execution time.

	Code	Grids	Execution Time (s)
Estimation	kb2d-qt	300x500	147
	kb2d	300x500	48.6
Simulation	sgsim-qt	300x500	48.1
	sgsim	300x500	14.37

Table 2. Comparison of execution time for quadtree search in estimation and simulation.

Conclusion

Compared to the conventional super block search in many of the GSLIB estimation and simulation algorithms, the quadtree search does not show any improvement in terms of execution time for estimation or simulation. The quadtree algorithm is inherently efficient, and the inefficiency appears in this paper may be due to the coding. Improving the `srchqdt` code may considerably affect the execution time.

References

Eppstein D., Goodrich M., Sun J. , June 2005,"The Skip Quadtree: A Simple Dynamic Data Structure for Multidimensional Data", the twenty-first ACM symposium on Computational Geometry.

Manchuk, J., Leuangthong, O., and Deutsch, C.V. 2004, "Direct Geostatistical Simulation on Unstructured Grids", in O. Leuangthong and C. Deutsch, eds., *Geostatistics Banff 2004*, vol. 1, Springer Science+Business Media, p. 85-94.

Pyrcz M. J., Deutsch C. V.," Building Blocks for Direct Sequential Simulation on Unstructured Grids", *Centre for Computational Geostatistics Report: 4,* University of Alberta, 2002.

Samet H., 1990, *The Design and Analysis of Spatial Data Structures*, Addison Wesley, Reading, MA, 493 p.

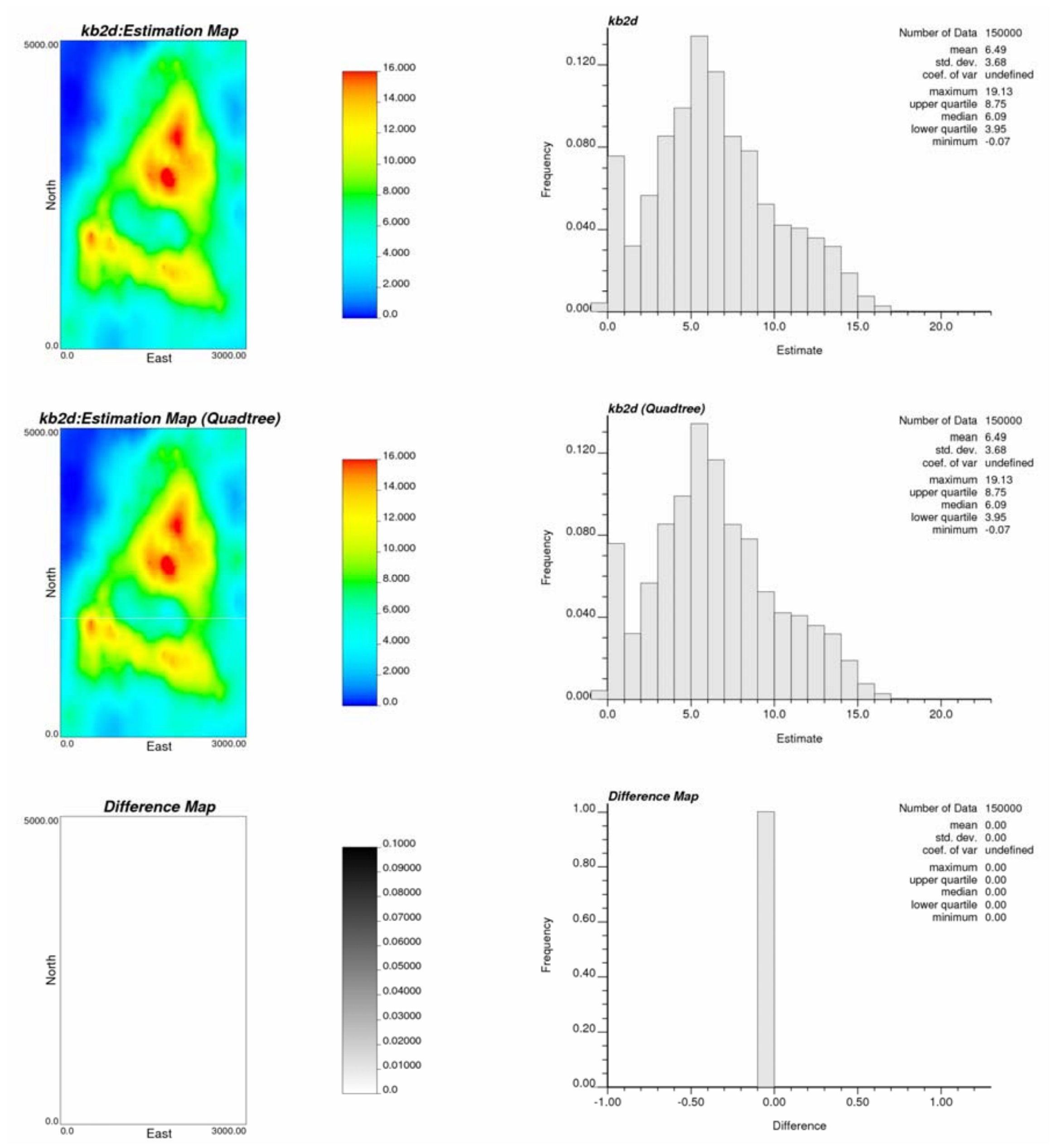

Figure 10. kriging results coming out of kriging with conventional search (Top), kriging using quadtree search (middle) and difference map (bottom).

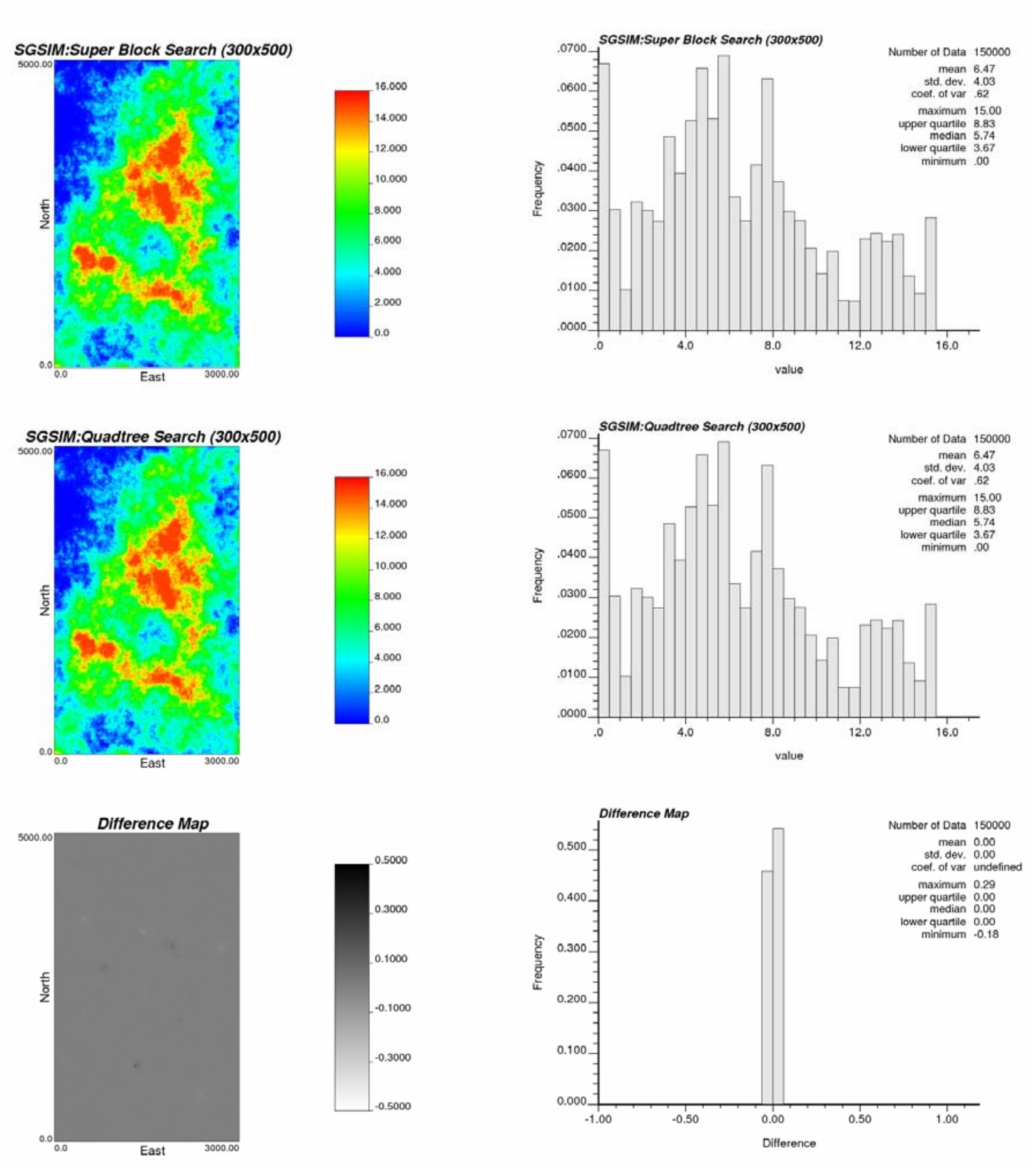

Figure 11: Simulation results coming out of SGSIM with Super block search (Top), SGSIM using quadtree search (middle) and difference map (bottom).