

Some Elemental Operations on Linear Quadtrees for Geographic Information Systems

D. J. ABEL

Division of Computing Research, CSIRO, P.M.B., P.O. Aitkenvale, Qld 4814, Australia

Quadrees appear attractive as an alternative to fixed-cell-size representations for areal entities in geographic information systems. This paper considers some aspects of use of linear quadrees in such applications. Compaction (reduction of a quadtree to its maximal leaves) is desirable to minimise space requirements and to simplify some basic operations. An algorithm for compaction as an adjunct to quadtree-generating processes is presented, with analysis showing it to be linear with the sizes of the input and compacted quadrees. Some simple forms of set operations are also described.

1. INTRODUCTION

The quadtree¹² is a form of cellular region representation found effective in such fields as graphics^{10, 18} and image processing.^{13, 9} It can be viewed as a variant of the usual fixed-cell-size representation which offers high compressions in the number of cells required to represent an image⁹ while retaining the simplicity of operations on cellular representations. The quadtree representation is based on a regular hierarchical decomposition of a square region into subquadrants with the decomposition conveniently shown as a tree of outdegree 4 with each node corresponding to a subquadrant of side length 2^{-m} , where m is the depth (or level) of the node designating the root (the whole) region as level 0. A binary-coded image can be mapped by colouring each node black if the corresponding subquadrant lies wholly within the image, white if wholly outside and grey otherwise. Clearly if a node is black or white all its descendants must be of the same colour and the node is treated as a leaf. In practice it is usual to truncate the tree at a level L , where 2^{-L} corresponds to a desired precision of representation.

Quadtree concepts have received a certain degree of attention in geographic information systems (GIS).^{14, 17} Their usage has largely been restricted to providing mechanisms for addressing spatially defined entities by location. For example, Tamminen¹⁷ forms subsets of entities with each subset (of not more than a specified number of entities) contained within a subquadrant. However, use of the quadtree to represent the entities also appears attractive as a means of providing an equivalent set of facilities to a fixed-cell-size representation, while retaining the quadtree's greater efficiency in usage of storage and some operations.

While a number of design choices appear open, a direct application is to represent a spatial entity by one quadtree for each value the attribute can take. That is, an attribute taking p values over the region could be held as a set of p binary-coded quadrees so that each quadtree corresponds to a mapping of a value for an attribute over the region. The quadrees could be accessed by keys derived from the attribute name and the value and possibly by a form of locational key such as that proposed by Abel and Smith.³

The linear quadtree^{8, 2} appears to be a suitable form of the quadtree under this approach. This form essentially represents the quadtree as a list of its black nodes, with a node identified by a unique key derived from its ordered

list of ancestors and with the list of nodes sequenced by that key. The formulation of the key is chosen to enable the keys of nodes bearing adjacency (neighbour) and ancestor/descendant relationships to a given node to be evaluated by arithmetic operations on the key of the node. Examination of a node is then equivalent to determination of its colour. This can be performed by searching the list for the node or an ancestor (with the node black if it is present) with inference required to determine if a node absent from the list is white or grey. Abel^{2, 3} offers formulations for a set of elemental operations on nodes and on linear quadrees.

The linear quadtree appears more attractive than a pointer data structure for the GIS proposed. Gargantini⁸ presents an analysis showing the linear quadtree to be more economical in its space requirements. Where disk-resident quadrees must be considered, the linear quadtree can be readily indexed by a B-tree⁶ to provide efficient access to nodes for such elemental operations as examination of neighbours of a given node. While the combined space requirements of the B-tree and linear quadtree can then exceed those of the pointer data structure, the B-tree's worst-case query cost in disk accesses is $O(\log N)$, where N is the number of black nodes in the linear quadtree. In contrast the pointer data structure is unsuited to disk-resident quadrees and has worst-case costs of some elemental operations of $2L$.^{16, 1} While encoded forms of quadrees¹¹ are more space-efficient than the linear quadtree, these suffer the same disadvantages as the pointer structures when disk storage is required and are essentially oriented towards sequential operations on the whole quadtree.

This paper considers two aspects relevant to the generation and management of areal entities represented by quadrees in a GIS. First, a process for compacting² a quadtree is presented. Some processes do not necessarily generate a quadtree whose leaves are of maximal size. For example, if a quadtree is generated by performing a union of two quadrees, the quadtree might include the four sons of a node, all of the same colour. It is clearly more efficient in terms of space to replace the four sons by their father. Additionally some operations on linear quadrees are made more complex if internal (non-leaf) nodes are permitted to be black, white or grey rather than grey only. If all internal nodes are grey, then a node absent from the list and with no ancestor appearing in the list must be white or grey, with these conditions distinguished by the presence of descendants of the node in the list (in which

case the node is grey). If absent nodes can be black, and if some descendants of the queried node appear in the list, then all those descendants must be examined to prove or disprove the node as black. The process of reducing a quadtree to its maximal leaves is termed 'compaction' by Hunter.² Woodward¹⁸ refers to an analogous process as 'reassembly' while Gargantini⁸ terms the process 'compression'.

Secondly, the paper also considers some set operations (union, intersection and difference) on quadtrees. These are of particular importance as such GIS operations as overlaying can be framed as combinations of these set operations.¹⁵ While algorithms for some of these have been reported,^{8,1} the algorithms proposed here have some advantages in their simplicity.

2. DEFINITIONS

While it will later be suggested that the algorithms can be used with alternative forms of keys and linear quadtrees, for simplicity of presentation the linear quadtree will be assumed to be implemented using the form of key defined by Abel and Smith.³ Under this keys are defined recursively as follows. Let k be the key for a node of level m , where m is non-zero and the key for the ancestor of k at level $(m-1)$ be k' . Then k is given by

$$k = k' + S(k)p(m)$$

where

$$\begin{aligned} S(k) &= 1 \text{ if } k \text{ is the SW son of } k' \\ &= 2 \text{ if } k \text{ is the NW son of } k' \\ &= 3 \text{ if } k \text{ is the SE son of } k' \\ &= 4 \text{ if } k \text{ is the NE son of } k' \end{aligned}$$

and

$$P(m) = 5^{L-m}$$

1222	1224	1242	1244	1422	1424	1442	1444
1220		1240		1420		1440	
1221	1223	1241	1243	1421	1423	1441	1443
1200				1400			
1212	1214	1232	1234	1412	1414	1432	1434
1210		1230		1410		1430	
1211	1213	1231	1233	1411	1413	1431	1433
1000							
1122	1124	1142	1144	1322	1324	1342	1344
1120		1140		1320		1340	
1121	1123	1141	1143	1321	1323	1341	1343
1100				1300			
1112	1114	1132	1134	1312	1314	1332	1334
1110		1130		1310		1330	
1111	1113	1131	1133	1311	1313	1331	1333

Figure 1. Three-level decomposition with locational keys to base 5.

The key for the node of level 0 is taken as $P(0)$. A key thus has the general form

$$k = \sum_{j=0}^m S_j(k) P(j), \quad S_j \in (1, 2, 3, 4) \quad \forall_j (1 \leq j \leq m),$$

$$S_0 = 1.$$

Fig. 1 shows keys assigned to nodes from a quadtree decomposition with an L of 3. Note that key values have been given to base 5; this convention is followed throughout this paper where numeric key values are given. The values of S will be referred to as the 'digits' of the key for a node.

This form of key imposes a pre-order traversal sequence on a list of nodes held in ascending sequence by key. That is, a node is followed immediately by its descendants.

3. COMPACTION

The algorithm presented here performs an extended form of compaction. The usual form assumes the quadtree is presented in a valid form. In terms of the linear quadtree, a quadtree is valid if a node may not appear in the same quadtree as any of its descendants. Compaction of this style is usually performed after definition of the quadtree by a procedure such as union which might yield the quadtree in a valid but uncompact form. The form presented here allows the input quadtree to include any set of black nodes provided that the nodes are passed serially in ascending sequence by key. The quadtree output is in a strictly correct form and consists of the minimal set of black nodes which together cover all nodes of the input set and no node not covered by members of the input set. For example, two valid input sets of nodes are (1310, 1320, 1330, 1340, 1410) and (1300, 1320, 1410); both yield (1300, 1410) as the compacted quadtree. Ramifications of this extended form will become apparent in consideration of the set operations.

The algorithm conceptually assembles maximal nodes in a buffer defined by two items only: a tentative node (denoted by LANC in the Appendix, derived from largest ancestor) which is the largest able to be formed from the nodes passed to the buffer since the last flush together with nodes possibly following, and the last node (LNODE) added to the buffer. At any stage, the strict interpretation of the buffer contents is the descendants of the tentative node through to and including the last node under a pre-order traversal. The buffer is established by setting its two nodes as null. The following observations provide controls on operation of the buffer. Let the incoming node be k . Note that k must have a key value greater than LNODE as nodes are presented in ascending sequence by key.

If k is a descendant of the last node passed, then k can be discarded as implicitly included in the nodes in the buffer.

If the buffer is currently empty, then the largest node to be formed from k is the largest node of which k is a direct SW descendant. (For example, if k is 1211, then its largest direct SW ancestor is 1200.) Clearly any larger ancestor must also be an ancestor of nodes already considered or which are white by virtue of their absence from the input list of nodes. If k is identical to the tentative node, then k can be passed directly to the output quadtree and the buffer retained as empty. Otherwise the buffer is established.

The buffer must be flushed when the tentative largest node is proven grey. This arises when absent (white) nodes can be detected between the last node added to the buffer: that is, when a pre-order traversal beginning at the last node added and continuing to k passes through nodes of which k is not a direct SW descendant. The flushing process requires extraction of the descendants of the tentative node beginning with the SW descendant of that node and proceeding to the last node added to the buffer. The traversal can be performed by progressing through the siblings of the SW son of the tentative node to the node which is an ancestor of the last node added or the last node added itself. In an ancestor of the last node added is encountered, the traversal is recommenced with its SW son, and so on. When the last node added has been reached the buffer is treated as empty and k used to establish it.

The tentative node is proven black (i.e. complete) if there are no white nodes between the last node added to the buffer and k , and k is not a descendant of the tentative node. The tentative node is then passed to the output as a single node and k acted upon as if the buffer were empty.

The Appendix gives a formal statement of the algorithm as the Pidgin ALGOL⁴ procedures COMPACT, FLUSH and INITBUF. Implementation requires a number of elemental operations. Abel² presents formulations for evaluation of the level of a node given its key, for determining the son of a node in a given quadrant and for testing a node as a descendant of another. Abel and Smith³ show that the next node in a pre-order traversal is given by adding 1 to the digit corresponding to the level of a current node. (That is, 1244 is followed by 1300 and 1300 by 1400.) The largest ancestor of which k is a direct SW descendant can be evaluated by replacing any trailing 1s of the key of k by 0s. Thus for $k = 1211$ the required ancestor is 1200, and for $k = 1342$, 1342. The presence of white nodes between the last node passed and k can be tested by comparing the next node from the last node with the largest node of which k is a direct SW descendant; if these nodes are not identical then white nodes are present.

The algorithm can be shown to have average solution costs which are $O(I+N)$ where I is the number of black nodes in the input quadtree and N the number in the compacted quadtree. We observe that, with the exception of evaluation of the largest node of which a given node is a direct SW descendant, the required operations require constant time to execute provided the level of the node is known. Determination of the level requires examination of the $(L-m+1)$ digits of the key of a node of level m . As there are 4^m nodes of level m , T the number of nodes of all levels is trivially $2^{2L+2}/3$. D the total number of digits examined in evaluating the level for all possible nodes can similarly be shown to be 4^{2L+3} for large L . If all nodes of all levels are assumed equally likely to occur, then the average number of operations to evaluate the level of a node is T/D , which is a constant.

A similar analysis shows the average number of operations to evaluate largest ancestors is 4. The total compaction process thus has average solution times linear with I and N and is independent of L .

4. SET OPERATIONS

A number of subregion-subregion (i.e. area-area) operations in a GIS can be framed as combinations of the set operations of union, intersection and difference. Consider, for example, a GIS database of 20 binary-valued linear quadtrees modelling mappings over the region of soil types 1-10, rainfall ranges A-F and altitude ranges 1-4. A representative query could be to display the areas suitable for wheat growing. If an area is taken to be suitable for wheat growing if the soil type is 1 or 2, if the rainfall is within range A and if the altitude is within range 1, then the entity 'area suitable for wheat' could be evaluated as

$$((\text{soil type 1} \cup \text{soil type 2}) \cap \text{rainfall range A}) \cap \text{altitude range 1}$$

The ability of the extended compaction procedure to accommodate lists of black nodes without restrictions on the presence of descendants allows union to be performed simply as a merge of two input quadtrees, with the nodes from the merge passed serially for compaction. Thus a union of the quadtrees (1200, 1310, 1320, 1330, 1410) and (1120, 1220, 1300, 1420, 1430, 1440) would be formed by serially forming the nodes list (1120, 1200, 1220, 1300, 1310, 1320, 1330, 1410, 1420, 1430, 1440) with compaction generating the quadtree (1120, 1200, 1300, 1400).

This procedure requires space only for a current member of each of the input quadtrees, the compaction buffer and the current node of the output quadtree. The merge component is clearly simpler than a union forming a valid quadtree through avoiding the need to test pairs of nodes as a node and a descendant. While an ancestor test is made within compaction for each incoming node from the merged list, such a test must be made even for the restricted sequences of nodes from a full union.

Intersection can be simply performed by extracting the nodes common to the two input quadtrees or nodes which are descendants of nodes in the other. This can be easily performed as a serial process. We note that intersection always yields nodes of maximal size, if inputs are already compacted.

The difference of two linear quadtrees will be recognized as the list of nodes not common to both and with no descendant or ancestor in the other quadtree, together with the difference of a node in the one list and its descendants in the other. The first set of nodes can be readily derived; attention here centres on evaluation as a set of maximal nodes, the difference of a node and one or more of its descendants. Evaluation can be performed as a tree traversal in a similar way to the buffer flush for compaction.

Let K be a node from the first quadtree and M a node from the second, where K is an ancestor of M . Assume for the moment that there are no other nodes in the second quadtree which are descendants of K . The difference between K and M can then be evaluated by a traversal beginning at the SW son of K and continuing until M itself or an ancestor of M is encountered. If an ancestor of M is met, then the traversal recommences at that ancestor's SW son, and so on. When M is met, the traversal recommences at the next sibling of M until a node not a descendant of K is encountered. If there are more descendants of K in the second quadtree, then the

traversal is interrupted when the next descendant or one of its ancestors is encountered, and so on.

The evaluation of the difference of two nodes will be recognized as closely related to the inference of the white nodes between two black nodes. The procedure advanced here is somewhat simpler than that described by Gargantini⁸ for this operation. Additionally the set of maximal nodes are generated serially in ascending sequence by key, rather than as two distinct sets, one in ascending sequence and the other in descending sequence.

5. DISCUSSION

The algorithms presented appear able to be modified to accommodate different forms of keys and definitions of the linear quadtree with only minor alterations. Clearly the elemental operations required can be framed for other key forms. Trivial changes would be needed where both black and white nodes are to be represented explicitly in the linear quadtree. Essentially a colour must be associated with the tentative node with changes in the colour of incoming nodes triggering a flush.

REFERENCES

1. D. J. Abel, *A B⁺-tree Structure for Large Quadrees*. Technical Report No. 13, CSIRO Division of Computing Research, Canberra, Australia (1983).
2. D. J. Abel, *A B⁺-tree structure for large quadrees*. *Computer Vision, Graphics and Image Processing* (to appear).
3. D. J. Abel and J. L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem. *Computer Vision, Graphics and Image Processing* **20**, 1, 1–24, (1983).
4. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass. (1974).
5. M. A. Comeau, *A Coordinate Reference System for Spatial Data Processing*. CLDS Technical Bulletin 3, Lands Directorate, Environment Canada, Ottawa (1981).
6. D. Comer, The ubiquitous B-tree. *Association for Computing Machinery Computing Surveys* **11**, 2, 121–137 (1979).
7. B. G. Cook, The structural and algorithmic basis of a geographic data base. In *Proceedings of the First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems*, edited G. Dutton. Harvard Papers on Geographic Information Systems, Harvard (1978).
8. I. Gargantini, An effective way to represent quadrees, *Communications of the Association for Computing Machinery* **25**, 12, 905–910 (1982).
9. I. Gargantini and Z. Tabakman, Linear quad-and oct-trees: their use in generating simple algorithms for image processing. *Proceedings Graphics Interface '82, 17–21 May 1982, Toronto, Ontario*, pp. 123–127 (1982).
10. G. M. Hunter, Efficient computation and data structures for graphics, *Ph.D. dissertation*, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey (1978).
11. E. Kawaguchi and T. Endo, On a method of binary picture representation and its application to data compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2**, 1, 27–35 (1980).
12. A. Klinger, Patterns and search statistics. In *Optimizing Methods in Statistics*, edited J. S. Rustagi. Academic Press, New York (1971).
13. A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition. *Computer Graphics and Image Processing* **5**, 1, 68–205 (1976).
14. J. J. Martin, Organization of geographical data with quad trees and least squares approximation. *Proceedings, IEEE Conference on Pattern Recognition and Image Processing, Las Vegas*, pp. 458–463 (1982).
15. G. Nagy and S. Wagle, Geographic data processing. *Association for Computing Machinery Computing Surveys* **11**, 2, 139–181 (1979).
16. H. Samet, Neighbor finding techniques for images represented by quadrees. *Computer Graphics and Image Processing* **18**, 1, 37–57 (1982).
17. M. Tamminen, *Efficient Geometric Access to a Multi-representation Geo-database*. Report HTKK-TKO-B52, Laboratory of Information Processing, Helsinki University of Technology, Espoo, Finland (1983).
18. J. R. Woodwark, The explicit quad tree as a structure for computer graphics. *The Computer Journal* **25**, 2, 235–238 (1982).

APPENDIX

Pidgin ALGOL statement of the compaction algorithm

The following elemental operations are assumed:

- LEVEL (k), to return the level of node k;
- ANCESTOR (k,p), to return TRUE if k is a descendant of p or is identical to p;
- SON (k,q), to return the son of k in the quadrant q of k;
- MAXFF (k), to return the largest node of which k is a direct SW descendant;
- PASS_TO_OUT (k), to insert k in an output quadtree.

The compaction algorithm described offers a simple means of performing compaction of a linear quadtree to retain economy of storage and simplicity of some basic operations. As the algorithm accepts nodes serially and generates the compacted quadtree node-by-node, it can be applied as an adjunct to operations yielding quadrees. Where the quadrees of interest are large enough to demand use of auxiliary storage, this is preferable to generating a quadtree in full and then applying compaction as a separate operation, as it saves the two sets of disk transfers needed to store and fetch the interim quadtree. More generally, the node-by-node treatment of input and output quadrees for compaction and set operations raises the possibility of evaluating a quadtree for an area defined by a set of qualifications (as in Section 4) and immediately processing or displaying it node by node, without any requirement to store the quadtree.

Dr J. L. Smith and Dr D. M. Mark made useful comments on earlier forms of this paper. Dr J. Woodwark also suggested some improvements to the structure and content.

procedure COMPACT (K):

(* COMPACT acts upon the next mode K *)

integer K, LANC, LNODE;

begin

(* If the buffer is currently empty, insert K *) Pass if LANC = NULL then INITBUF (K)

(* Only act on K if an ancestor of it has not previously been presented. Any such ancestor must be LNODE *)

else if ANCESTOR (K,LNODE) = FALSE

(* Flush the buffer if LANC is complete or proven grey *)

then if (ANCESTOR (K,LANC) ≠ TRUE) or (INCR

```

(LNODE) ≠ MAXFF (K))
then begin
  FLUSH (LANC, LNODE);
  INITBUF (K):
  end
  (* Otherwise update the end of the black descendants of
  LANC*)
  else LNODE := K;
  end;

procedure INITBUF (K, LANC, LNODE):
(* INITBUF establishes the buffer after a flush with K as the
next node *)
integer, K, LANC, LNODE;
begin
  LANC := NULL;
  LNODE := NULL;
  if K = MAXFF (K)
  then PASS_TO_OUT (K)
  else begin
    LANC := MAXFF (K);
    LNODE := K;
  end
end;

```

```

procedure FLUSH (LANC, LNODE):
(* FLUSH identifies the maximal descendants of LANC to and
including LNODE*)
integer LANC, LNODE, K, K2;
begin
  (* Consider the case where LANC can be passed as a single
  node *)
  if (LNODE = LANC) or (ANCESTOR(INCR(LNODE),
  LANC) = FALSE)
  then PASS_TO_OUT (LANC)
  else begin
    (* Perform tree traversal by sons until an ancestor of LNODE
    is reached *)
    LEVK2 := LEVEL(LANC)+1;
    K2 := SON(LANC, 'SW');
    until (ANCESTOR(LNODE,K2) = TRUE do
    begin
      PASS_TO_OUT (K2)
      move to sibling of K2
    end;
    FLUSH (K2, LNODE);
  end;
end;

```