PERGAMON

Computers & Geosciences 28 (2002) 763–774

COMPUTERS & GEOSCIENCES

www.elsevier.com/locate/cageo

Rapid extraction of image texture by co-occurrence using a hybrid data structure

David A. Clausi*, Yongping Zhao

Department of Systems Design Engineering, University of Waterloo, 200 University Avenue West, Waterloo, Ont., Canada N2T 3G1

Received 2 April 2001; received in revised form 21 September 2001; accepted 25 September 2001

Abstract

Calculation of co-occurrence probabilities is a popular method for determining texture features within remotely sensed digital imagery. Typically, the co-occurrence features are calculated by using a grey level co-occurrence matrix (GLCM) to store the co-occurring probabilities. Statistics are applied to the probabilities in the GLCM to generate the texture features. This method is computationally intensive since the matrix is usually sparse leading to many unnecessary calculations involving zero probabilities when applying the statistics. An improvement on the GLCM method is to utilize a grey level co-occurrence linked list (GLCLL) to store only the non-zero co-occurring probabilities. The GLCLL suffers since, to achieve preferred computational speeds, the list should be sorted. An improvement on the GLCLL is to utilize a grey level co-occurrence hybrid structure (GLCHS) based on an integrated hash table and linked list approach. Texture features obtained using this technique are identical to those obtained using the GLCM and GLCLL.

The GLCHS method is implemented using the C language in a Unix environment. Based on a Brodatz test image, the GLCHS method is demonstrated to be a superior technique when compared across various window sizes and grey level quantizations. The GLCHS method required, on average, 33.4% ($\sigma = 3.08\%$) of the computational time required by the GLCLL. Significant computational gains are made using the GLCHS method.

Keywords: Texture features; Hash table; Linked list; Co-occurrence probabilities; Remote sensing imagery

1. Introduction

Currently, large volumes of remotely sensed imagery are captured to study many aspects of the earth's surface and atmosphere. As a result, there exist ever increasing demands to analyze data generated from satellite platforms (e.g. LANDSAT, SPOT, RADARSAT, ERS-1, JERS-1, etc.) more efficiently and accurately. A popular technique used to perform spatial analysis in remotely sensed imagery is texture analysis.

Grey-level co-occurrence matrices (GLCMs), developed by Haralick et al. (1973), are widely utilized in image texture feature extraction within remotely sensed imagery (Barber and LeDrew, 1991; Shokr, 1991; Baraldi and Parmiggiani, 1995; Soh and Tsatsoulis, 1999). The GLCM technique employs the following steps. The probability of co-occurrence between two grey levels i and j given a relative orientation (θ) and distance (δ) can be computed for all possible co-occurring grey level pairs in an image window. The GLCM stores these probabilities and, as such, is dimensioned to the number of grey levels available. Then, selected statistics are applied to the GLCM by stepping through the entire matrix (i.e. over all probabilities) to calculate the texture features. For image segmentation considerations, the texture features are assumed to belong to the centre pixel of the subwindow. A primary computational drawback of

*Corresponding author. Tel.: +519-888-4567 x2604; fax: +519-746-4791.

E-mail address: dclausi@uwaterloo.ca (D.A. Clausi).

0098-3004/02/$ - see front matter
PII: S0098-3004(01)00108-X

GLCMs is that they require demanding computational requirements when applying the statistics, especially when attempting to segment entire remote sensing images.

There are a number of approaches to reduce the computational requirements when calculating texture features based on co-occurrence probabilities stored in GLCMs. First, quantization of the image grey levels reduces the size of the GLCMs and thus reduces the total computational load when applying the statistics. In practice, this quantization is always used, often reducing the number of grey levels from 8 bits (256 grey levels) to 4 or 5 bits (16 or 32 grey levels). Second, when determining texture features on a pixel-by-pixel basis throughout an image, shifting the window of interest by one column does not change many of the co-occurring probabilities. Only redundant probabilities are retained and new probabilities (introduced by the new column) are added. This reduces the number of computations when determining the co-occurrence probabilities. Third, it is advised to capture only a minimal number of texture features based on minimizing the number of parameters used (orientations, distances, and statistics). Not only does this reduce the total computational requirements, but it can also improve the quality of the feature set since it is recognized that statistical correlations do exist between co-occurrence texture features. Barber and LeDrew (1991), for example, only recommend using three statistics based on texture analysis of SAR sea ice imagery. The objective of this paper is to introduce an algorithm that improves the computational requirements of determining co-occurrence texture features so that the user can adjust the parameters to suit their own needs with significantly less concern to the overall computational time.

Other methods exist to represent sparse matrices (Duff et al., 1986). By using a grey level co-occurrence linked list (GLCLL) to store the co-occurring probabilities, a significant reduction in computational demand is achieved (Clausi and Jernigan, 1998). This is achieved since, unlike the GLCMs, the GLCLLs do not store zero probabilities for co-occurring pairs. However, to efficiently access grey level pairs on the linked list to update their probabilities, the list is kept sorted. Although tremendous computational gains over the GLCM are achieved, the sorting compromises the efficiency of the GLCLLs. In this paper, a grey level co-occurrence hybrid structure (GLCHS) based on a hash table and linked list integrated approach is presented that does not require sorted lists. This algorithm shows a significant, consistent improvement in computational performance relative to GLCLLs. The texture features captured by the GLCHS, GLCLL, and GLCM methods are identical for a given set of parameters.

This paper is arranged in the following manner. Section 2 details existing algorithms used to capture co-occurrence probability texture features. The GLCHS method is then described in Section 3. Section 4 provides computational speed test results and comparisons while the final section (Section 5) summarizes and concludes the paper.

2. Existing co-occurrence implementations

Of the 14 original statistics developed by Haralick et al. (1973) for generating texture features based on co-occurrence probabilities, only a limited number are used in practice. Dissimilarity (DIS), uniformity (UNI), entropy (ENT), contrast (CON), and correlation (COR), presented in Table 1, are often used in practice since they are (a) scale and shift invariant and (b) effective texture discriminators. These five statistics will be used exclusively in this paper. Each of these five statistics has a qualitative meaning with respect to the structure within the GLCM, and as a result, with respect to the underlying texture. Dissimilarity and contrast measure the degree of texture smoothness, uniformity and entropy reflect the degree of repetition amongst the grey level pairs, and correlation describes the correlation

Table 1
Statistics frequently applied to co-occurrence probabilities for texture feature generation

Statistics	Equations
Dissimilarity	$\mathrm{DIS} = \sum_{i,j=1}^{G} C_{ij}\lvert i-j\rvert$
Uniformity	$\mathrm{UNI} = \sum_{i,j=1}^{G} C_{ij}^2$
Entropy	$\mathrm{ENT} = -\sum_{i,j=1}^{G} C_{ij}\log C_{ij}$
Contrast	$\mathrm{CON} = \sum_{i,j=1}^{G} C_{ij}(i-j)^2$
Correlation	$\mathrm{COR} = \sum_{i,j=1}^{G} \frac{(i-\mu_x)(j-\mu_y)C_{i,j}}{\sigma_x\sigma_y}$

C_{ij} represents co-occurring probabilities stored inside GLCM. G represents quantized grey level. Note that (μx, μy) and (σx, σy) represent means and standard deviations for row i and column j within GLCM.

2	2	2	3	3
3	2	2	3	3
2	2	2	2	2
3	0	0	2	2
0	0	0	0	2

(A)

i \ j	0	1	2	3
0	0.2	0	0.1	0
1	0	0	0	0
2	0	0	0.4	0.1
3	0.05	0	0.05	0.1

(B)

i \ j	0	1	2	3
0	0.2	–	–	–
1	0	0	–	–
2	0.1	0	0.4	–
3	0.05	0	0.15	0.1

(C)

Fig. 1. (A) 5×5 image window with four grey levels values (0–3). (B) Corresponding GLCM given $\delta = 1$ pixel and $\theta = 0^\circ$. Full matrix required. Twenty co-occurring pairs exist. Pair (3,2) occurs only once so its probability is 0.05. (C) Corresponding GLCM given $\delta = 1$ pixel and $\theta = 0^\circ$ and 180°. Only lower triangle matrix required. Forty co-occurring pairs exist. Pairs (3,2) and (2,3) occur together six times which generate probability of co-occurrence to be 0.15.

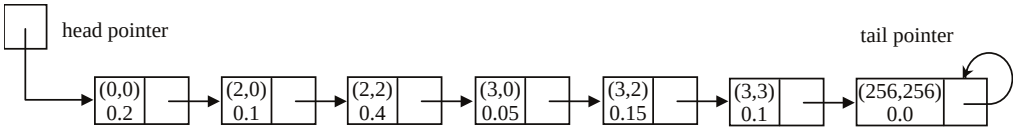

Fig. 2. GLCLL structure for determining image texture features. To reduce search times, nodes are sorted according to grey level pairs (i,j). GLCLL is created based on data in Fig. 1A.

between the grey-level pairs (Shokr, 1991; Baraldi and Parmiggiani, 1995).

Within Table 1, element C_{ij} is the grey level co-occurrence probability for co-occurring pixels with grey levels i and j given an interpixel distance δ and interpixel orientation θ. Thus, for each unique (δ, θ) pair, a different GLCM is required. Each GLCM is dimensioned to G, the number of quantized grey levels. In general, relative interpixel distances are short when applied to remotely sensed imagery and often only setting $\delta = 1$ is required to generate preferred texture features (Barber and LeDrew, 1991). The relative interpixel orientation is usually set to either $\{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$ or the average of all four orientations. Co-occurring pairs oriented at 0° are also oriented at 180°, which generates a symmetrical GLCM. This concept extends to 45° and 225°, 90° and 270°, as well as 135° and 315°. As a result, only the lower triangular portion of the GLCMs (i.e. only co-occurring pairs where $i \geqslant j$) needs to be retained.

The generation of co-occurrence probabilities is illustrated in Fig. 1. Here, given $G = 4, \theta = 0^\circ$, and $\delta = 1$ pixel, co-occurring probabilities are generated for a 5×5 sample window (Fig. 1A) and represented within a GLCM (Fig. 1B). For example, the co-occurring pair (2,3) occurs two times in the window. Since there are a total of 20 possible co-occurring pairs, this generates a probability of 0.10 at $i = 2$ and $j = 3$ in the GLCM. With increasing G, the matrix becomes more sparse with a corresponding $O(G^2)$ increase in the computational requirements to apply each statistic. If only non-zero probabilities are stored and used to determine the statistics, a dramatic improvement in the computational performance can be obtained. In Fig. 1C, since symmetrical pairs are considered, only a lower matrix needs to be stored. In this example $\theta = 0^\circ$ and 180°, so $i = 2$ and $j = 3$ occurs six times out of a possible 40 leading to a probability of 0.15. Features generated using a symmetrical matrix are comparable to the results obtained using a non-symmetrical matrix.

Instead of using a matrix to store the co-occurrence probabilities, a linked list structure (Reingold and Hansen, 1983; Deitel and Deitel, 1994) can be used. This particular linked list structure is referred to as the GLCLL (Clausi and Jernigan, 1998). The GLCLL's structure is illustrated in Fig. 2 using the image window indicated in Fig. 1A. The list must be kept sorted to allow rapid searching for existing (i,j) pairs. These nodes are sorted with the grey level pairs in ascending order. Searching begins at the head of the list by looking for the first instance of the ith grey level. If found, then the algorithm searches for the corresponding jth grey level. If the (i,j) pair is found, then the probability stored inside that node is incremented. If the (i,j) node is not found at the expected location, then a node must be entered at that location that stores the appropriate probability for that grey level pair.

When capturing texture features on a pixel-by-pixel basis from an image, a sliding window is employed. The algorithm begins with the window at the top left-hand

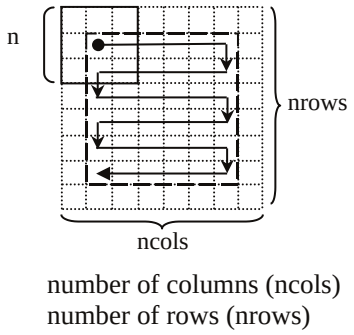

Fig. 3. Zig–zag window path to determine texture features for entire image. See text for details.

corner of the image. The co-occurring probabilities are determined for this window and then the features are calculated using the statistics (Table 1). After the features are calculated, the window slides one column to the right. At this point, most of the co-occurring probabilities remain the same. The grey level pair probabilities introduced by the new column are then included in the GLCLL and the probabilities associated with the column that the window just moved past can be subtracted from the GLCLL. For new grey level pairs introduced by the new column, new nodes will be created. For grey level pairs whose probability reaches zero as a result of reducing the probability, the associated nodes are deleted from the linked list. A similar procedure holds when the window is moved down a single row when the end of a row is reached. For even rows, the window is moved from left to right and for odd rows the window is moved from right to left. The window moves efficiently in this zig–zag pattern until the entire image is covered (Fig. 3).

This algorithm spends considerable time maintaining sorted linked lists. Note that a separate GLCLL is maintained for every (δ, θ) pair. The application of the statistics to calculate the texture features occurs rapidly by traversing each linked list from head to tail.

3. Hybrid data structure

To improve on the GLCLL, the reliance on maintaining sorted lists should be removed. A combined hash table and linked list structure is designed to meet this criterion. A hash table is a data structure that stores elements in locations that are easily computed from the value or representation of the elements (Reingold and Hansen, 1983). That is, the hash table directly transforms an element into an address where it will be stored. Since the element itself is used to access the location, the search order for this data structure is $O(1)$. The function that is used to perform the transformation is known as the hash function. Here, the grey level pairs (i and j) are used to quickly access the hash table which is created in the form of a matrix. Each element in the hash table has a pointer that can point to an element on the linked list, should that co-occurring pair exist in the window.

Based on the combination of the hash table and the linked list, this structure is referred to as the GLCHS. Using the GLCHS, a two-dimensional hash table **struct** is created to point to linked list nodes (Fig. 4). A doubly linked list is used to allow easy insertion and deletion of nodes. Both the hash table and the linked list are necessary. The hash table allows for rapid access of any node in the linked list, if that node exists. The linked list allows for rapid application of the statistics by traversing the linked list from head to tail. The C **struct** definition for nodes in the linked list is:

```
typedef struct ListNode
{
  int x1, x2;  //co-occurring grey level pairs
  struct ListNode *prev;
  struct ListNode *next;
} ListNode;
```

and the struct definition for nodes in the hash table is

```
typedef struct HashNode
{
  float pr;  //co-occurrence probability
  struct ListNode *list_ptr;
} HashNode;
```

In the ListNode **struct**, four members are defined. Each instance of the **struct** represents a node on the doubly linked list. The two integer members (**x1**,**x2**) store the grey level pairs. Two self-referential pointers are defined to access previous (***prev**) and next (***next**) ListNode nodes. Linked list nodes are defined to represent the first (**head**) and the last (**tail**) nodes. In the hash table **struct**, one float member (**pr**) stores the grey level co-occurrence probability and the other stores the linked list pointer (***list_ptr**). The **list_ptr** points to the corresponding node on the linked list associated by the grey level pair.

The following three variables are used throughout the code and are defined globally.

```
HashNode **Glchs[MAX]; //pointers of hash table
                       //structure
ListNode *head[MAX];   //head of linked list
ListNode *tail[MAX];   //tail of linked list
```

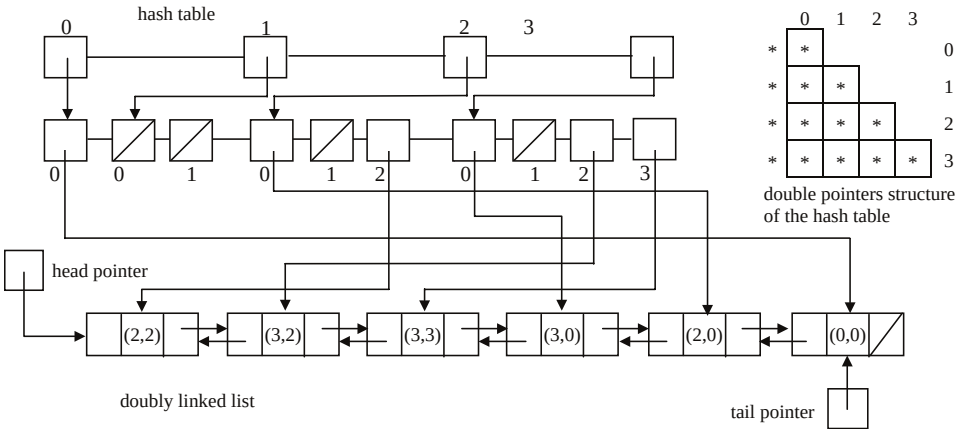

Fig. 4. GLCHS structure for determining image texture features. GLCHS created is based on sample image in Fig. 1A.

where MAX represents the maximum possible number of linked lists. The following is the function used to initialize the hybrid data structure:

```
//int ng;      no. quantized grey levels
//int na;      no. angles
//int nd;      no. pixel displacements
//float inc;   probability increment
void initialize_hybrid (int ng, int na, int nd,
                        float inc )
{
  int i, j, k, inc, nsum;
  //nsum=n+(n-1)+(n-2)+···+2+1
  nsum=ng*(ng+1)/2.0;
  //create Glchs for each (distance,
  //  orientation) pair
  for (i=0; i < nd * na; i++)
  {
    //allocate sufficient pointers to rows
    //  in hash table
    Glchs[i]=(HashNode **) malloc((size_t)
             ng*sizeof (HashNode *));
    if (Glchs[i]==NULL)
    {
      fprintf(stderr,
        ''ERROR:initialize_hybrid
          (Glchs[i])'');
      exit(0);
    }
    //allocate memory for entire hash table
    Glchs[i][0]=(HashNode *)malloc((size_t)
                nsum * sizeof(HashNode));
    if (Glchs[i][0]==NULL)
    {
      fprintf(stderr,
        ''ERROR:initialize_hybrid
          (Glcms[i][0])'');
      exit(0);
    }
    k=1;
    inc=1;
    //assign pointers to rows
    for (j=1; j<ng; j++)
    {
      Glchs[i][j]=Glchs[i][0]+k;
      inc++;
      k+=inc;
    }
    //initialize all nodes in hash table
    for (j=0; j < nsum; j++)
    {
      Glchs[i][0][j].pr=0.0;
      Glchs[i][0][j].list_ptr=NULL;
  }
  //initialize head and tail of
  //  doubly linked list
  head[i]=(ListNode*)
            malloc(sizeof(ListNode));
  if (head[i]==NULL)
  {
     fprintf(stderr,
        ''ERROR:initialize_hybrid
          (head[i])'');
     exit(0);
  }
  head[i]->next=NULL;
  head[i]->prev=NULL;
  head[i]->x1=-1;
  head[i]->x2=-1;
  tail[i]=head[i];
  }
}/*end initialize_hybrid*/
```

In this function, dynamic memory allocation (using the **malloc** command) is used to create the hash table. The total number of entries in the lower triangle of the hash table is **nsum**. The double pointer defined by **Glchs[i]** is set as a list of pointers to the rows in the hash table. Sufficient memory to represent the entire lower triangle of the hash table is created and pointed to by the first entry in the hash table (**Glchs[i][0]**). The first loop (from 1 to **ng**) indexes the row pointers properly. Then, for the loop indexing **nsum** times, all of the nodes are initialized (**pr** set to zero and pointer set to NULL). Finally, the head and tail are initialized to appropriate values to represent an empty doubly linked list.

The function to include the next co-occurring pair into the given GLCHS is indicated as follows:

```
//int indx;  glchs index
//int x1, x2;  co-occurring pair
//float inc;  probability increment
void find_and_insert (int indx, int x1, int x2,
float inc)
{
  ListNode*x;
  int temp;
  //to ensure hash table is lower triangular,
//force x1>=x2
  if (x1 < x2)
    {temp=x1; x1=x2; x2=temp;}
  //if zero probability, not yet placed on doubly
//linked list
  if (Glchs[indx][x1][x2].pr==0.0)
  {
    x=(ListNode*) malloc(sizeof(ListNode));
    x->x1=x1;
    x->x2=x2;
    x->next=NULL;
    x->prev=tail[indx];
    tail[indx]->next=x;
    tail[indx]=x;
    Glchs[indx][x1][x2].list_ptr=x;
  }
  //increment probability associated with
//co-occurring pair
  Glchs[indx][x1][x2].pr+=inc;
}//end find_and_insert
```

The function first ensures that the co-occurring pair has the relationship $x1 \geqslant x2$ so that only a lower triangular hash structure is required. Then, if the hash table has a zero entry for the particular grey level pair (**x1**,**x2**), then that particular co-occurring pair does not have a representative node on the linked list. As a result, a new ListNode is created, its grey level values are set, and it is inserted at the end of the linked list. The **list_ptr** is then set to point to this ListNode to establish the relationship between the HashNode and its corresponding ListNode. If the hash table entry is not zero, then that HashNode already points to an existing ListNode on the linked list. Whether or not the ListNode was created, the probability associated with HashNode is incremented by **inc**. A similar function is used to decrement a probability for a certain grey level pair. In this situation, if the probability reaches zero, the ListNode is removed from the linked list and its associated HashNode's **list_ptr** is set to NULL.

Fig. 4 illustrates the structural arrangement for the GLCHS. Nodes from the hash table point to nodes on the linked list. As a result, the linked list does not have to be kept sorted, in contrast to the GLCLL. This design is expected to significantly and consistently reduce the completion times when determining co-occurrence probability texture features. To calculate the texture features, the statistics (Table 1) are applied directly to the linked list which only stores those co-occurring pairs with non-zero probabilities.

A flowchart outlining the entire algorithm is presented in Fig. 5. At the start of the algorithm, the given image will be read. Also, user-defined parameters such as the window size (**n**), grey level quantization (**ng**), interpixel distance (δ), orientation (θ), and statistics are read in at the start of the program. The zig–zag process that the window follows to capture texture features from the entire image is presented in Fig. 3. To calculate texture features for each window, the algorithm must loop through all selected angles, displacements, and statistics. Each statistic is applied to each separate linked list.

4. Algorithm comparisons

4.1. Order comparison

The computational order of the algorithms is an important consideration. Table 2 shows the order of each of the algorithms. Each order estimate is broken into two components. The first component is the order for capturing the co-occurring probabilities and the second component is the order for determining the statistics. In this table, n represents the window dimension, s represents the number of statistics, G represents the number of quantized grey levels, and L represents the length of the linked lists. In other words, L represents the number of distinct grey level pairs found in a window, which is dependent on G and n as well as the textural characteristics in that window.

For each algorithm, determining the co-occurring probabilities is proportional to n. This holds true when a

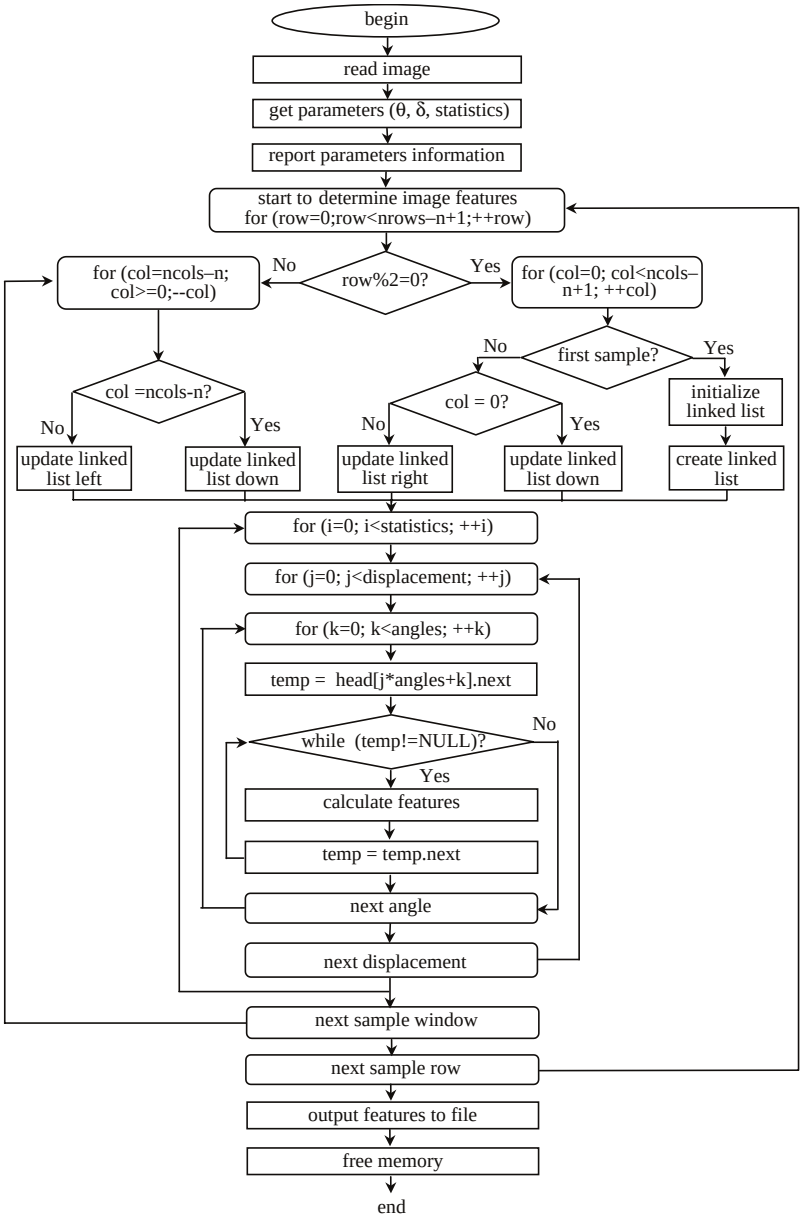

Fig. 5. Flowchart of GLCHS routine for determining image texture features.

sliding window is used (Fig. 3) since a new column introduces new co-occurring grey level pairs, which must be included in the existing linked lists. Similarly, a column of grey level pairs must be removed from the existing linked lists. In the case of the GLCLL, this term must be multiplied by L to account for searching the linked list. The application of the statistics also varies for each method. Since each statistic has a different order associated with its determination, a generic estimate of s^2 is used. For the GLCM, this term is multiplied by G^2 since the calculations are performed using the entire co-occurrence matrix. For both the GLCLL and GLCHS, the s^2 term is multiplied by L since the entire linked list must be traversed to apply each statistic. Overall, the key differences between the algorithm orders are: (a) the GLCM must loop through the entire matrix to calculate the statistics while GLCLL and GLCHS only have to traverse the linked list and (b) the GLCLL must search the linked list to create the probabilities while the GLCM and GLCHS do not. As a

Table 2
Computational orders to determine image texture features for GLCM, GLCLL and GLCHS algorithms

Method	Order
GLCM	$O(n) + O(s^2G^2)$
GLCLL	$O(nL) + O(s^2L)$
GLCHS	$O(n) + (O(s^2L)$

Terms: n—window dimension, G—number of grey level quantization levels, s^2—generic order to apply statistics, L—linked list length.

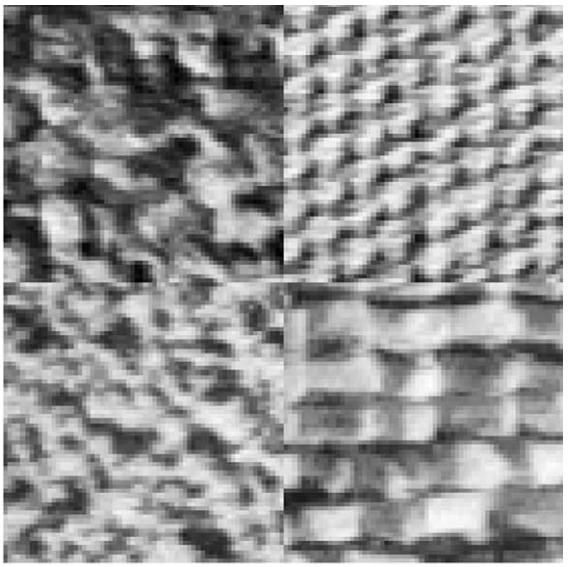

Fig. 6. Brodatz demonstration image (128×128 pixels). Four textures are included. Clockwise from top left-hand corner: cork (D4), cotton (D77), raiffa (D84), and paper (D57). D** indicates texture number in Brodatz book.

result, the GLCHS has the preferred order for calculating co-occurrence texture features.

4.2. Computational comparisons

Each of the three algorithms (GLCMs, GLCLLs, and GLCHSs) has been implemented using the C programming language. Tests were performed on a Sun Sparc Ultra 1 200E (200 MHz, 128 Mbytes RAM, 322 SPECint, 462 SPECfp). Each implementation uses the same programming foundation based on the GLCM code. For example, each implementation is based on the same library routines for reading the image data and writing the texture features, preventing any bias. The only distinctions between the implementations are the algorithms themselves.

Each algorithm was applied to a multi-class 128×128 (Brodatz, 1966) image (Fig. 6) which illustrates a diversity of textural characteristics. The user defined parameters were set as follows: $\delta = 1$; $\theta = 0^\circ$, 45°, 90°, 135°; and all the five statistics found in Table 1 (a total of 20 texture features). Varying window sizes were used (5, 10, 15, 20, 25, and 30 pixels). The results are presented using Table 3 (numerical times), Table 4 (percentage ratios of numerical times), Table 5 (average length of linked lists), and Fig. 7 (graphical plot of numerical times). Table 3 presents the comparison of the computational requirements for each of the algorithms. Each window size produces a different number of samples from the fixed image size (texture features are not determined for border pixels). To make proper comparisons, each result is normalized to determine the total time (in μs) to capture texture features per window. Comparisons are made across five quantization levels: 128, 64, 32, 16, and 8 grey levels. Table 4 takes the results from Table 3 and generates percentage ratios comparing times for each pair of algorithms. Table 5 represents the average length of the linked lists required given G and n. Fig. 7 plots the results presented in Table 3.

The results show that the GLCHS is an improvement on the GLCLL, which is an improvement on the GLCM. The GLCHS is always significantly faster than the GLCLL and orders of magnitude faster than the traditional GLCM. The relationships with respect to G, n, and L are now discussed in the context of the Tables 3–5 as well as Fig. 7.

4.2.1. Role of grey level quantization (G)

The greater the number of grey levels, the greater the improvement of the GLCLL and GLCHS over the GLCM method (Tables 3 and 4). This is due to the quadratic order dependence of the GLCM on the number of grey levels (Table 2). The results show that there is a square relationship between the number of grey levels and the total computation time using the GLCM method (Table 3). That is, for every doubling of the number of grey levels, the computational speed is multiplied by four. Varying the number of grey levels does not change the relative performance between the GLCLL and GLCHS methods, where the GLCHS/GLCLL percentage ratio has only minor variations as a function of G (Table 4).

Table 5 indicates the average length of the linked lists (identical linked lists are created using GLCLL and GLCHS) across the grey level quantizations and window sizes. With additional grey levels, the linked list lengths are longer and the GLCLL method spends relatively more time maintaining the sorted list. As a result, the greater the number of grey levels the better the performance of the GLCHS with respect to the GLCLL. For example, given a window size of 30×30

Table 3
Average computation time required to determine texture features per window given G and n for each of GLCHS, GLCLL, and GLCM GLCHS consistently generates lower computational times for all examples

Computational time (μs)	Grey level (G)	Window size ($n \times n$)					
		5×5 (15 376 samples)	10×10 (14 161 samples)	15×15 (12 996 samples)	20×20 (11 881 samples)	25×25 5×5 (10 816 samples)	30×30 (9801 samples)
GLCHS method	128	29.6	124.3	290.9	496.6	734.1	990.2
	64	28.9	113.3	239.3	383.4	536.2	662.2
	32	26.0	89.0	158.1	228.1	260.7	303.0
	16	20.8	53.7	76.6	98.9	98.9	109.2
	8	14.6	27.2	33.5	39.1	42.5	46.4
GLCLL method	128	79.0	351.0	834.5	1547.0	2419.6	3508.8
	64	76.1	322.4	717.9	1242.3	1778.4	2354.4
	32	69.3	255.6	483.6	702.8	899.1	1048.9
	16	55.9	156.8	232.4	287.9	331.5	375.5
	8	38.7	74.9	95.0	109.8	122.5	135.2
GLCM method	128	21564.5	21428.9	21713.6	21997.3	22298.4	22410.5
	64	5397.0	5479.8	5591.3	5735.2	5888.5	5978.5
	32	1373.9	1433.9	1510.1	1544.5	1589.3	1625.3
	16	362.3	393.7	417.1	426.7	439.2	451.0
	8	101.8	115.1	123.5	129.2	136.8	143.9

Table 4
Percentage ratios based on results in Table 3 for GLCHS vs. GLCLL, GLCHS vs. GLCM, and GLCLL vs. GLCM. GLCHS averages 33.4% of GLCLL computation time

Comparison (%)	Grey levels (G)	Window size ($n \times n$)					
		5×5	10×10	15×15	20×20	25×25	30×30
Percentage of GLCHS computational time compared to GLCLL computational time	128	37.5	35.4	34.9	32.1	30.3	28.2
	64	38.0	35.1	33.3	30.9	30.2	28.1
	32	37.5	34.8	32.7	32.5	29.0	28.9
	16	37.2	34.2	33.0	34.4	29.8	29.1
	8	37.7	36.3	35.3	35.6	34.7	34.3
Percentage of GLCHS computational time compared to GLCM computational time	128	0.1	0.6	1.3	2.3	3.3	4.4
	64	0.5	2.1	4.3	6.7	9.1	11.1
	32	1.9	6.2	10.5	14.8	16.4	18.6
	16	5.7	13.6	18.4	23.2	22.5	24.2
	8	14.3	23.6	27.1	30.3	31.1	32.2
Percentage of GLCLL computational time compared to GLCM computational time	128	0.4	1.6	3.8	7.0	10.9	15.7
	64	1.4	5.9	12.8	21.7	30.2	39.4
	32	5.0	17.8	32.0	45.5	56.6	64.5
	16	15.4	39.8	55.7	67.5	75.5	83.3
	8	38.0	65.1	76.9	85.0	89.5	94.0

pixels, the GLCHS method requires 34.3% of the GLCLL method's computational time (Table 4) given a quantization level of 8. At 128 grey levels there is an increased sorting load for the GLCLL algorithm and this percentage drops to 28.8%.

4.2.2. *Role of the window dimension (n)*

The window size has a negligible affect on the computational performance of the GLCM. This is because the GLCM spends most of its time calculating the statistics and the size of the window

Table 5
Average linked list length as function of n and G. This applied to both GLCLL and GLCHS

Grey levels (G)	Window Size ($n \times n$)					
	5×5	10×10	15×15	20×20	25×25	30×30
128	17.4	80.2	183.4	318.9	478.4	653.9
64	16.7	73.4	157.4	253.8	351.2	442.5
32	15.0	57.6	105.4	145.3	175.1	197.1
16	11.8	34.3	49.3	57.9	63.6	68.1
8	7.5	15.2	18.4	20.2	21.4	22.3

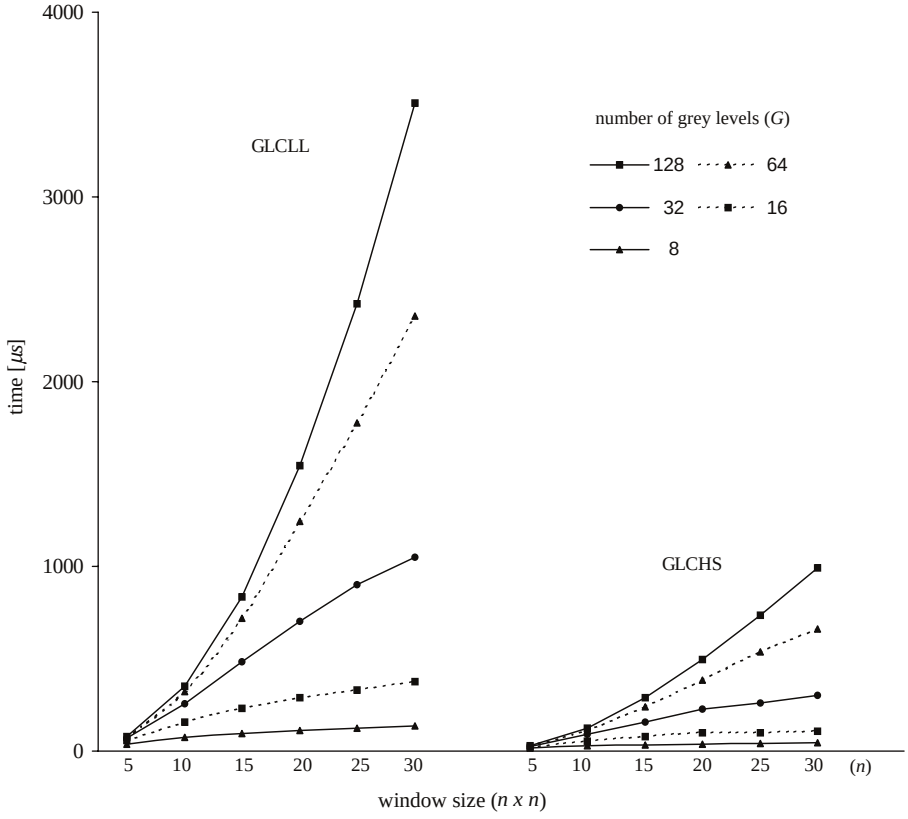

Fig. 7. Graphical plot of GLCHS and GLCLL results from Table 3.

affects only the determination of the co-occurrence probabilities.

On the other hand, the window size has a dramatic affect on the GLCLL and GLCHS computational speeds since the greater the window size, the greater the number of co-occurring probabilities (generally speaking), and the greater the time spent on determining statistics (both GLCLL and GLCHS) and determining the probabilities (GLCLL only). The GLCLL spends more time sorting during creation of the probabilities when larger windows are used since the linked lists become longer. As a result, the GLCHS completion times improve relative to the GLCLL completion times with increasing window size (Table 4). For example, for $G = 128$ grey levels in Table 4, the ratio between GLCHS and GLCLL computation times for 5×5 windows is 37.5% and, with increased window size, gradually decreases to 28.2% for window size 30×30. Fig. 7 illustrates that, with increasing window size, the GLCLL method increases at a higher rate than the GLCHS. A linear increase in the window size can lead to up to a quadratic increase in the number of grey level

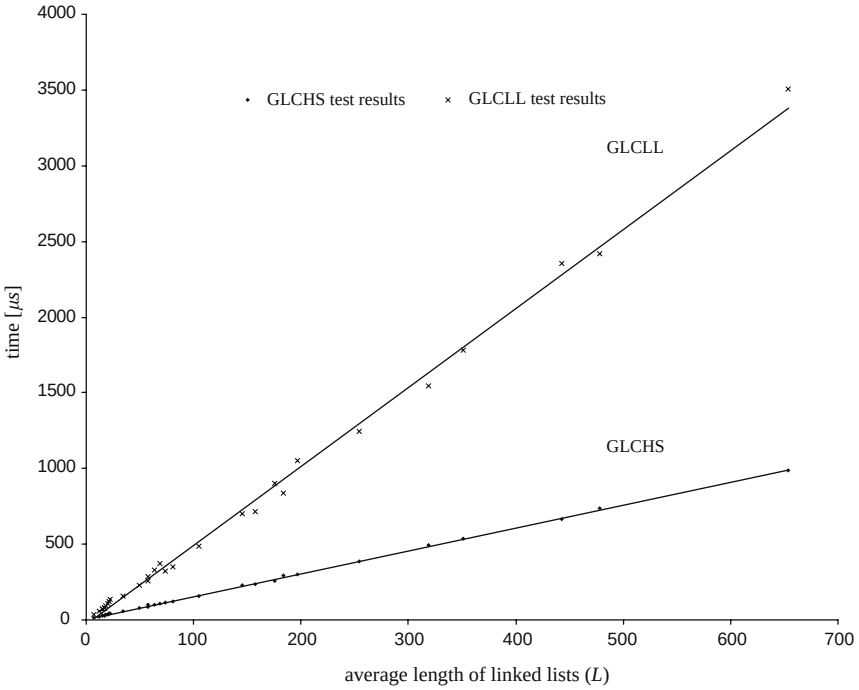

Fig. 8. Linear regression plots of average length of linked lists (L) versus computation time (μs) for GLCLL and GLCHS using data from Tables 3 and 5.

pairs (depending on the underlying texture and the quantization level). As a result, the GLCHS method is relatively more efficient for larger windows compared to the GLCLL method.

4.2.3. Role of the length of the linked lists (L)

The computational time of the GLCLL is proportional to the length of the linked lists and the GLCHS should have a computational time partially proportional to L (Table 2). For example, according to Table 5, for $G = 128$, an increase in n from 10 to 30 pixels increases the average L by a factor of $653.9/80.2 \approx 8$. Using Table 3, the computational requirements for the GLCLL increase by approximately the same ratio ($3508.8/351.0 \approx 10$) and the GLCHS is affected to a lesser extent ($990.2/124.3 \approx 8$) (Table 3), as expected.

With larger images (typical of remote sensing imagery), the computational impact of using the GLCHS algorithm is extremely important. Granted, the computational savings will be a function of the textural characteristics, window size, number of statistics, and quantization level. However, on average across all the test cases, GLCHS required 33.4% ($\sigma = 3.08\%$) of the computational time compared to GLCLL, which strongly supports the use of the GLCHS algorithm.

Fig. 8 plots the relationship of the average length of the linked lists (L) with respect to the average computation time required to calculate texture features for a single window. Data for all G and n are plotted for each of GLCLL and GLCHS. Linear regression clearly shows a linear relationship between L and computation time. This linear relationship can be used to assist estimation of the total time required to calculate all of the texture features in the image. Software can be written to analyze certain windows distributed across the image. The amount of time to calculate the texture features using a variety of grey levels and window sizes is recorded. After a linear regression of this data is performed, the equation of the line produced can be used to predict how long it will take to generate texture features for the entire image. This would give the user the opportunity to redefine the parameters to reduce the overall computation prior to running the GLCHS on the entire image.

5. Conclusions

Image texture segmentation is often applied to information extraction from remotely sensed imagery. Co-occurrence probabilities are frequently used for this task. The advantage of the GLCHS methodology for determining co-occurrence texture features relative to the GLCM or GLCLL method is clearly demonstrated

in the results. For every possible example, the new GLCHS method is consistently faster than the GLCLL method and orders faster than the GLCM method. For any given image, the GLCHS method requires on average 33.4% of the computation time of the GLCLL method. Determining co-occurrence texture features for an entire 1000×1000 pixel image using a window of 25×25 pixels, a quantization level of 64 grey levels, and 12 statistics $\{\theta = 0°, 45°, 90°, 135°; \delta = 1$; statistics = dissimilarity, uniformity, entropy, contrast, and correlation) would require approximately 6 min of computation on a 200 MHz computer. Current CPUs with faster clock speeds would obviously be able to accelerate this process.

We are currently planning to implement this algorithm into the PCI EasiPace© remote sensing software environment. This will hopefully lift the current restriction in the EasiPace© environment on the user to quantize the number of grey levels to 16 when generating co-occurrence texture features. This methodology will give more freedom for the user to select their desired quantization level. This luxury has not been available using the frequently employed GLCM technique. The GLCHS is also preferable over the GLCLL technique especially with respect to using larger window sizes. The GLCLL method's computation speed is more dependent on the window dimension relative to the GLCHS, allowing the GLCHS to have improved computational performance with large windows. Since the GLCHS computation speeds have a linear relationship with the size of the linked lists, the total amount of time required by the algorithm can be estimated for a given image prior to actually capturing texture features for the entire image.

Acknowledgements

Support for this project was provided by Geomatics for Informed Decisions (GEOIDE), (http://www.ula-val.geoide.ca) and Cryosphere System in Canada (CRYSYS) (http://www.crysys.uwaterloo.ca/index.cfm).

References

Baraldi, A., Parmiggiani, F., 1995. An investigation of the textural characteristics associated with Gray level cooccurrence matrix statistical parameters. IEEE Transactions on Geosciences and Remote Sensing 33 (2), 293–304.

Barber, D.G., LeDrew, E.F., 1991. SAR sea ice discrimination using texture statistics: a multivariate approach. Photogrammetric Engineering & Remote Sensing 57 (4), 385–395.

Brodatz, P. 1966 Textures: A Photographic Album for Artists and Designers. Dover, New York, 112pp.

Clausi, D.A., Jernigan, M.E., 1998. A fast method to determine co-occurrence texture features. IEEE Transactions on Geosciences and Remote Sensing 36 (1), 298–300.

Deitel, H.M., Deitel, P.J., 1994. C How to Program, 2nd edn. Prentice-Hall, Englewood Cliffs, NJ, 926pp.

Duff, I.S., Erisman, A.M., Reid, J.K., 1986. Direct Methods for Sparse Matrices. Clarendon Press, Oxford, London, 341pp.

Haralick, R.M., Shanmugam, K., Dinstein, I., 1973. Texture features for image classification. IEEE Transactions on Systems, Man, and Cybernetics 3 (6), 610–621.

Reingold, E.M., Hansen, W.J., 1983. Data Structures. Little, Brown and Company, Boston, 450pp.

Shokr, M.E., 1991. Evaluation of second-order texture parameters for sea ice classification from radar images. Journal of Geophysical Research 96 (6), 10625–10640.

Soh, L.-K., Tsatsoulis, T., 1999. Texture analysis of SAR sea ice imagery using gray level co-occurrence matrices. IEEE Transactions on Geosciences and Remote Sensing 37 (2), 780–795.