PARALLEL ALGORITHMS FOR EXTRACTING SHAPE FEATURES OF HANDWRITTEN CHARACTERS

H.F. Li, M. Youssef and R. Jayakumar

Department of Computer Science
Concordia University
Montreal, Quebec, Canada H3G 1M8

Abstract

Parallel algorithms to extract shape features of handwritten characters based on edge classification are presented. The start and end points of the edges defining the boundary of a character are first detected in parallel. These edges are sent into the second stage of a pipeline to form edge chains for characterizing the corresponding shapes of a character pattern. The edge chains are identified in parallel using a two phase routing (routing update and data routing) of messages. These parallel feature extraction algorithms are simulated on a sequential computer and a classification scheme for recognizing handwritten numerals is developed. It is noted that the characters can be recognized with very high accuracy even if the characters are rotated by an angle between -15 and +15 degrees.

1. Introduction

Recent advances in hardware technology has made possible the development of special-purpose hardware based character recognition systems. The first step in developing such systems is to design parallel algorithms suitable for VLSI implementation. In this paper some parallel algorithms for extracting features from handwritten characters are proposed.

Various techniques for recognizing handwritten characters have been reported in the literature. These techniques recognize characters by (1) extracting global and statistical features [1,2] or (2) extracting geometrical and structural features [3-6]. However, the latter approach has the advantage of less sensitivity to rotation and are more amenable to natural variations in handwritten characters. Hence, this paper considers extracting the shape features based upon the approach proposed by Ahmed and Suen [5,6].

2. The Approach for Extracting Shape Features

Consider a binary pattern having M rows and N columns of pixels each of them can be either white or dark. Such a pattern is surrounded by a sequence of edges where an edge is a transition from a white region to a dark one or vice-versa, while scanning the pattern. For example, Fig. 1 shows a binary pattern having 11 rows and 12 columns and the edges surrounding this pattern are shown in Fig. 2. In the following it is assumed that the pattern is scanned horizontally.

An edge has a start point and an end point. These start and end points can be ranked such that points occurring in i-th row of the pattern have lower rank compared to points occurring in j-th row if and only if $i < j$. There are basically four types of edges, depending upon whether they occur along the inner periphery or the outer periphery of a pattern: (1) outer left (e^1), (2) outer right (e^2), (3) inner left (e^3) and (4)inner right (e^4). These edge types are exemplified in Fig. 2. The edges in a pattern can be ranked such that an edge having a start point with lower rank than that of another edge is assigned a lower rank than the latter one. A relation can be defined between a pair of edges meeting at a start or end point. The relations between the edges in Fig. 2 are also shown in the figure. Ahmed and Suen [5,6] has derived 14 different relations between a pair of edges meeting at a start or end points. Using the edges and the relations, simple holes and cavities can be detected in a pattern [5,6]. Furthermore, the edges form closed chains in and around the pattern (see Fig. 2). These chains can be used to detect complex holes and cavities in a pattern [5,6]. Ahmed and Suen proposed an approach [5,6] in which the edges along with the relation between pairs of them and the chains in a pattern can be used as shape features in recognizing handwritten characters.

In this paper, systolic algorithms to extract these shape features from a

given pattern are presented. The edges and their start and end points are first detected using the algorithm of Section 3, and the chains are then determined by the algorithm of Section 4.

3. Extracting the Edge Features

In this phase, the start and end points of all the edges in the pattern are extracted. The relations between the corresponding pairs of edges are also determined and the edges are ranked according to their start points.

The edges in a M*N pattern are detected using a column of M+1 processing elements (PE) as exemplified in Fig. 3. The input to the PE array is the M rows of the binary pattern and a row of zeros (in the (M+1)-th row). A column of reset signals are entered at the end of the N-th column of the pattern which reset the PE's to their initial state. Each PE_i, $1 \leq i \leq M+1$, is responsible for detecting all the start and end points occurring in the i-th row. A PE can be set to detect a transition from either a white pixel to a dark one (a 0 -> 1 transition) or from a dark pixel to a white one (a 1 -> 0 transition). A PE can detect a start point when an appropriate transition occurs, (as explained below) and generate a corresponding pair of edges. In order to detect end points, a PE has to trace these edges. This is achieved by sending a message token about an edge downward to the next PE, while maintaining one itself to search for the other edge. When PE_i, $1 \leq i \leq M$, detects a start point, it generates two edge messages, stores the left edge message within itself and sends the right edge message to PE_{i+1} as depicted at point A in Fig. 3. Note that PE_i, can associate a newly detected transition with an edge generated by a lower numbered PE (an edge continuation) if it receives an edge message from PE_{i-1} at the same time it detects a transition. In order to ensure proper identification of straight edges slanted at ± 45 degrees as single edges, the pattern is skewed such that the i-th row is delayed by two clock cycles with respect to the (i-1)-th row, as shown in Fig. 3. This can be achieved by passing the i-th row of the pattern through a row of 2(i-1) delays before entering it into PE_i.

Suppose two edge message tokens are meeting at PE_i. If PE_i encounters a transition at the same time as the two tokens, then the token from PE_{i-1} along with the transition at PE_i correspond to an edge continuation, as depicted at point B in Fig. 3. On the other hand, if PE_i does not encounter a transition, then the two tokens signify two edges meeting at PE_i, and hence an end point should be detected at PE_i, as shown at point C in Fig. 3.

Initially all the PE's are set to detect a 0 -> 1 transition and the vertical (message) input of PE_1 is set to null. During every clock cycle, PE_i, $1 \leq i \leq M$, performs the following operation.

Algorithm for extracting the edge features.

```
for 2M+N cycles do
  for all PEi, 1 ≤ i ≤ M+1, pardo
    if the expected transition occurs
    then
      if PEi receives a message from
      PEi-1 then
        if PEi contains a stored message
        then
          send the message stored in PEi
          to PEi+1;
          store the message received from
          PEi-1 in PEi
        else {No stored message in PEi.}
        {The transition corresponds to an
        edge continuation.}
          send the message received from
          PEi-1 to PEi+1
      else {No message from PEi-1.}
        if PEi contains a stored message
        then
          send the message stored in PEi
          to PEi+1
        else {No stored message in PEi.}
        {The transition corresponds to a
        start point.}
          detect the start point;
          generate two edges (of type e1
          and e2 if this is a 0 ->1
          transition or of type e3 and e4
          if this is a 1 -> 0 transi-
          tion);
          output the coordinates of the
          start point and the types of
          the two edges generated;
          store the left edge message in
          PEi;
          send the right edge message to
          PEi+1
    else {No transition}
      if PEi receives a message from
      PEi-1 then
        if PEi contains a stored message
        then
        {This corresponds to two edges
        meeting at a PE when there is no
        transition -- an end point.}
          detect the end point;
          output the coordinates of the
          end point and the two edges
          meeting at it;
          delete the messages stored in
          PEi and received from PEi-1
        else {No stored message in PEi.}
          store the message received from
          PEi-1 in PEi
```

This algorithm is illustrated for the binary pattern of Fig. 1 in Fig. 3. The various edge messages are shown in the skewed pattern itself by arrows between pixels. A horizontal arrow in the i-th row corresponds to a message stored in PE_i and an arrow in the west-south direction from the i-th row to the (i+1)-th row indicates a message from PE_i to PE_{i+1}. The start and end points and the edge information output by the column of PE's are shown in Fig. 4 along with the detected edges in the pattern of Fig. 1.

It can be seen that more edges than shown in Fig. 2 are detected. This is due to the fact that the input pattern is skewed. Since the edge chains and not the individual edges are used in recognizing the pattern, this will not create any problem later on.

The relations between the pair of edges generated at a start point or meeting at an end point in a PE can also be determined by the PE based on the type of these edges. Also, the edges can be ranked according to the order of appearance of their start points.

This computation requires a flush time of 2(M+1) cycles, a delay time of 2(M+1) + N-2 = 2M+N cycles, and one more cycle is required to reset the PE's to their initial state before the next pattern can be processed. However, since the computation is pipelined, a different pattern can be input to the PE array every N+1 cycles. Furthermore, each PE in the array is processing at most two edge tokens and a transition at a time, and hence its area is a constant independent of M and N.

Note that since the input pattern to the PE array is skewed, the output from the array will also be skewed in time. However, the output can be easily deskewed by using additional delays in a storage stage in which there are 2(M-i) delays in the i-th row. Thus, if the pattern for Fig. 1 is input to the array at cycle 1, all the feature points will be available in the storage stage at cycle 35. In the next cycle the storage stage receives the reset signal thereby passing the feature points to the next stage where the edge chains are extracted.

4. Extracting the Edge Chain

Given the set of edges and the start and end points in the pattern, the various chains formed by these edges and the sequence of relations between adjacent edges in these chains are now to be determined. This edge extraction is performed by another column of M+1 PE's (these PE's are different from the PE's of Section 3). The PE in the i-th row, PE_i, collects and outputs a chain if the start point with the lowest rank in this chain occurs at the i-th row in the pattern under consideration. Without loss of generality, it is assumed that at most three start and three end points only can occur in any row of a valid pattern. Thus, there are three identical processors in a PE, one for a pair of start and end points.

For simplicity, consider now a pattern with a single chain, even though the algorithms given latter in this section are applicable with multiple chains. During the chain extraction, each PE has to forward the start and end points it has identified to the PE that holds the start point with the lowest rank of the corresponding chain. Each start/end point carries the inherent attributes, namely, the types and thus the relation between the edges incident at that point. Assume that PE_i contains an end point and PE_j and PE_k contain the start points of the edges meeting at PE_i, and let $j \leq k$. Initially all such PE_i's can send their end points to the corresponding PE_j's. In addition, the PE_k's should be informed to forward the start points to the PE_j's as well. Moreover, all the feature points received by a PE_i can in turn be redirected to another PE_j, $j < i$, which holds a start point in the same chain. Thus, before sending the feature points, all the PE's should know the PE containing the start point with the lowest rank in each chain. This can be achieved using a two phase message routing, as presently explained. In the first phase, called routing update phase, PE_i computes a routing tag T_i such that all the edges sent to PE_i should be redirected to PE_j where $j = T_i$. In the second phase, called the data routing phase, all the PE's send the feature points to the PE indicated by their routing tags. All such actions are performed systolically.

4.1 Routing Update

Initially, the routing tags of all the PE's are empty and PE_i contains the start and end points in row i of the pattern. In the following finding the routing tags of all the PE's containing start points is explained. The routing tags for the PE's containing end points can similarly be obtained.

Assume PE_i contains an end point and let PE_j and PE_k contain the start points of the edges meeting at PE_i, and, without loss of generality, assume that $j \leq k$. During the first clock cycle all such PE_i's generate a message (j <- k) and send it to PE_{i-1}. During clock cycle 2, 3, ..., M all the PE's systolically update their routing tags and send appropriate messages to the PE's above them as given in the following algorithm.

Algorithm for routing update of the start points.

```
during the first cycle do
  for all PE_i, 1 ≤ i ≤ M+1, pardo
    if PE_i contains an end point then
      let PE_j and PE_k contain the start
      points of the edges meeting at PE_i;
      generate a message min{j,k} <-
      max{j,k} and sent it to PE_i-1

during the 2-nd, 3-rd, ..., M-th cycles
do
  for all PE_i, 1 ≤ i ≤ M, pardo
    let PE_i receive the message (j <- k)
    from PE_i+1;
    if i = k then
      send message min{j,T_i} <- max{j,T_i}
      to PE_i-1;
```

set T_i = min{T_i,j}
else
send the message received from PE_{i+1} to PE_{i-1}

This routing update is exemplified in Fig. 5 for the start and end points shown in Fig. 4. As stated before the routing tags for the PE's containing end points can be computed in a similar way. This computation takes M cycles (cycles 37 to 48 in Fig. 5) and then the PE's are reset to their initial state (in cycle 49). Moreover, each PE requires constant area independent of M and N.

It can be verified that at the end of this routing update phase, for any PE_i either T_i is non-empty or PE_i contains the start point with the lowest rank in the chain (and hence collects a chain). Once the routing tags for all the PE's are computed, the feature points can be systolically sent to the PE's collecting the chains.

4.2 Data routing

In the data routing phase all the PE's perform the following concurrently and systolically.

Algorithm for data routing.

```
during the first cycle do
  for all PE_i, 1 ≤ i ≤ M+1, pardo
    send the feature points and the
    routing tag to PE_i-1

during the 2-nd, 3-rd, ..., M-th cycles
do
  for all PE_i, 1 ≤ i ≤ M+1, pardo
    if T_i is empty then
    {This PE is collecting a chain}
      collect all the feature points
      received from PE_i+1
    else
    {This PE has to redirect the feature
    points.}
      if the routing tag received from
      PE_i+1 is i then
      {Redirect the feature points.}
        send the feature points received
        from PE_i+1 along with T_i to PE_i-1
      else
        send the feature points and the
        routing tag received from PE_i+1
        to PE_i-1
```

This algorithm is illustrated in the time space diagram of Fig. 6 for the given pattern. The empty routing tags are shown as a hyphen in this diagram. This computation also takes M cycles (cycle 50 to 61 in Fig. 6) and each PE takes constant area.

At the end of the data routing phase, the start and end points and the edges in a chain will be output by the PE containing the start point with lowest rank in the chain. These edges should then be arranged in a proper order (anticlockwise or clockwise) before they can be used to detect complex holes and cavities in the pattern.

It can be verified that the total computation time to extract the edge features and the chains in a pattern with M rows and N columns using these parallel algorithms is 4M+N+5 cycles.

5. Classification of Numerals

The feature extraction algorithms presented in Sections 3 and 4 are simulated on a sequential computer to classify handwritten numerals for the purpose of recognition using the approach presented. Each of the 10 numeral is simulated for 120 handwritten samples. The classification algorithm examines the chains of edge relations of a character for the purpose of distinguishing among different characters. Four-directional scan is used to remove almost all ambiguities in recognition. The classifier has a simple PLA-type architecture in implementation, but the details of the classifier is beyond the scope of this paper. The following are some observations from our experiment.

(1) Most of the numerals can be recognized by using the features extracted during a horizontal and vertical scan on them. To differentiate between the numerals 1 and 7, and 4 and 9, these two scans are not enough and an additional diagonal scan is necessary. With a horizontal, vertical, and a diagonal scan all the numerals in the available samples are recognized.
(2) The handwritten numerals are correctly recognized when they are rotated by an angle between -15 and +15 degrees. Thus, the features extracted by these parallel algorithms have less sensitivity to rotation.

6. Conclusion

This paper has presented parallel algorithms to extract shape features of handwritten characters. These features can be used to recognize the characters using the approach proposed by Ahmed and Suen [5,6]. The start and end points of edges in a pattern and the relations between a pair of edges generated at a start point or meeting at an end point are first extracted by the parallel algorithm of Section 3. The chains formed by these edges are then extracted by the parallel algorithm of Section 4. A classification scheme for recognizing handwritten numerals using these shape features has been developed and an experiment was conducted with 120 samples for each of the 10 numerals. It has been observed that using the features extracted during a horizontal, a vertical, and a diagonal scan all the handwritten numerals can be correctly recognized. Moreover, these numerals can

be correctly recognized even if they are rotated by an angle between -15 and +15 degrees.

REFERENCES

[1] T.W. Calvert, "Nonorthogonal Projections for Feature Extraction in Pattern Recognition", IEEE Trans. on Computers, Vol. C-19, 1970, pp. 447-452.

[2] B.V. Dasarathy and K.P.B. Kumar, "Chitra: Cognitive Handprinted Input-Trained Recursively Analyzing System for recognition of alphanumeric characters", Intl. Journal of Comp. and Infor. Sci., Vol. 7, 1978, pp. 253-282.

[3] M.T.Y. Lai and C.Y. Suen, "Automatic Recognition of Characters by Fourier Descriptors and Boundary Line Encoding", Pattern Recognition, Vol. 14, 1981, pp. 383-393.

[4] M. Shridhar and A. Badreldin, "High Accuracy Character Recognition Algorithms using Fourier and Topological Descriptors", Pattern Recognition, Vol. 17, 1984, pp. 515-524.

[5] P. Ahmed, "Computer Recognition of Totally Unconstrained Handwritten ZIP Codes", Ph.D. Thesis, Dept. of Computer Science, Concordia University, Montreal, Canada, July 1986.

[6] P. Ahmed and C.Y. Suen, "Edge Classification and Extraction of Shape Features", 7th Intl. Conf. on Pattern Recognition, 1984, pp. 593-596.

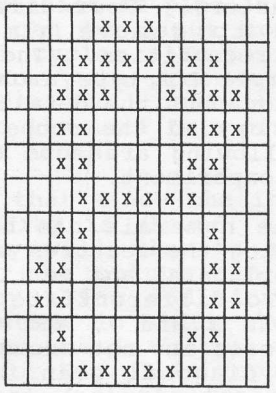
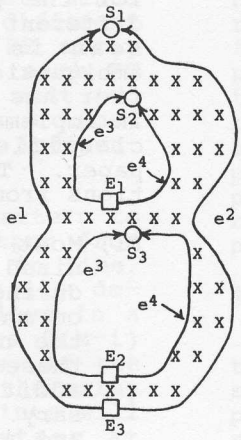


Figure 1

A binary pattern of pixels



Feature Point	Type of Meeting Edges	Relation
S_1	e^1,e^2	R_1
S_2	e^3,e^4	R_7
E_1	e^3,e^4	R_8
S_3	e^3,e^4	R_7
E_2	e^3,e^4	R_8
E_3	e^1,e^2	R_2

Chain 1: $((S_1,E_3), (E_3,S_1))$,
Chain 2: $((S_2,E_1), (E_1,S_2))$,
Chain 3: $((S_3,E_2), (E_2,S_3))$

Figure 2

The set of edges
surrounding the pattern of Fig. 1

Start points are shown as small circles
End points are shown as small squares

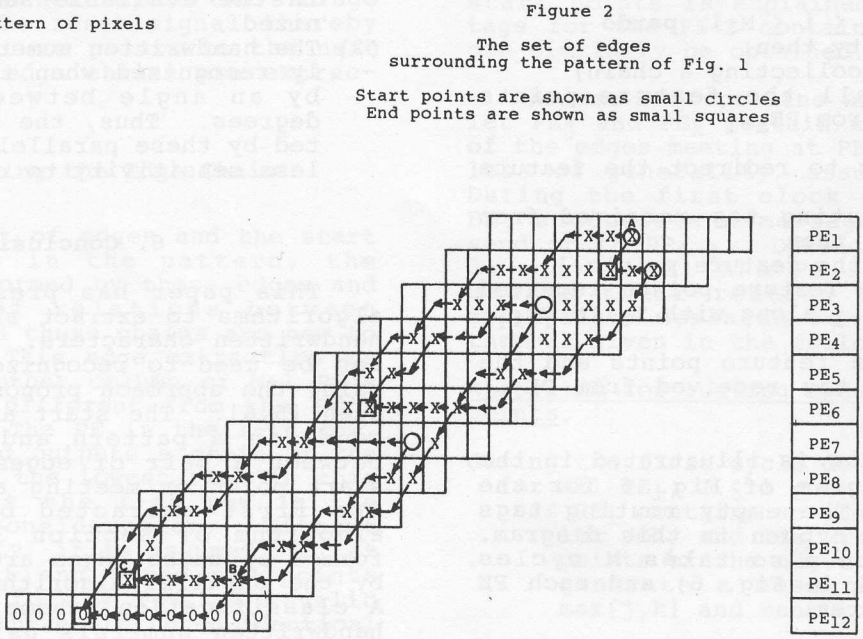


Figure 3

Edge messages during the extraction of start and end points
in the pattern of Fig. 1

Arrows show motion of message token as data are pumped into the systolic array. Coincidence of some combination of token pairs yields an end point (marked as a square.

PE	Extracted Feature Point
PE$_1$	S;1,6;e^1,e^2
PE$_2$	S;2,3;e^1,e^2 E;1,6,e^2;2,3,e^1
PE$_3$	S;3,5;e^3,e^4
PE$_6$	E;3,6,e^3;3,6,e^4
PE$_7$	S;7,5;e^3,e^4
PE$_{11}$	E;7,5,e^3;7,5,e^4
PE$_{12}$	E;1,6,e^1;2,3,e^2

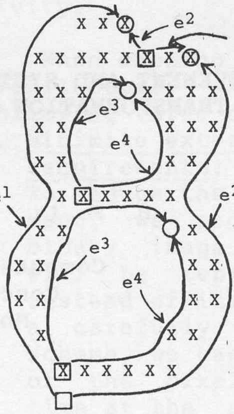


Figure 4

Features extracted by the edge extraction algorithm for the pattern of Fig. 1

PE	Cycle 37	Cycle 38	Cycle 39	Cycle 40	Cycle 41	Cycle 42	Cycle 43	Cycle 44	Cycle 45	Cycle 46	Cycle 47	Cycle 48
1												
2	1<-2	1	1	1	1	1	1	1	1	1	1 1<-2	1
3				3<-3	3	3	3	3	3	3 1<-2	3	3
4			3<-3						1<-2			
5		3<-3						1<-2				
6	3<-3						1<-2					
7					7<-7	7 1<-2	7	7	7	7	7	7
8				7<-7	1<-2							
9			7<-7	1<-2								
10		7<-7	1<-2									
11	7<-7	1<-2										
12	1<-2											

Figure 5

Time space diagram illustrating the routing update algorithm

Cycle 50	Cycle 51	Cycle 52	Cycle 53	Cycle 54	Cycle 55
S;1,6;e^1,e^2;-	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1
S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1					
S;3,6;e^3,e^4,-	S;3,6;e^3,e^4,-	S;3,6;e^3,e^4,-	S;3,6;e^3,e^4;- E;3,6,e^3;3,6,e^4;3	S;3,6;e^3,e^4,- E;3,6,e^3;3,6,e^4;3	S;3,6;e^3,e^4;- E;3,6,e^3;3,6,e^4;3
		E;3,6,e^3;3,6,e^4;3			
	E;3,6,e^3;3,6,e^4;3				
E;3,6,e^3;3,6,e^4;3					
S;7,5;e^3,e^4,-	S;7,5;e^3,e^4,-	S;7,5;e^3,e^4;-	S;7,5;e^3,e^4;-	S;7,5;e^3,e^4;- E;7,5,e^3;7,5,e^4;7	S;7,5;e^3;e^4;- E;7,5,e^3;7,5,e^4;7 E;1,6,e^1;2,3,e^2;1
			E;7,5,e^3;7,5,e^4;7	E;1,6,e^1;2,3,e^2;1	
		E;7,5,e^3;7,5,e^4,7	E;1,6,e^1;2,3,e^2;1		
	E;7,5,e^3;7,5,e^4;7	E;1,6,e^1;2,3,e^2,1			
E;7,5,e^3;7,5,e^4;7	E;1,6,e^1;2,3,e^2,1				
E;1,6,e^1;2,3,e^2;1					

Cycle 56	Cycle 57	Cycle 58	Cycle 59	Cycle 60	Cycle 61
S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1	S;1,6;e^1,e^2;- S;2,3;e^1,e^2;1 E;1,6,e^2;2,3,e^1;1 E;1,6,e^1;2,3,e^2;1
				E;1,6,e^1;2,3,e^2;1	
S;3,6;e^3,e^4;- E;3,6,e^3;3,5,e^4;3	S;3,6;e^3,e^4;- E;3,6,e^3;3,6,e^4;3	S;3,6;e^3,e^4;- E;3,6,e^3;3,6,e^4;3	S;3,6;e^3,e^4;- E;3,6,e^3;3,6,e^4;3 E;1,6,e^1;2,3,e^2;1	S;3,6;e^3,e^4;- E;3,6,e^3;3,6,e^4;3	S;3,6;e^3,e^4;- E;3,6,e^3;3,6,e^4;3
		E;1,6,e^1;2,3,e^2;1			
	E;1,6,e^1;2,3,e^2;1				
E;1,6,e^1;2,3,e^2;1					
S;7,5;e^3,e^4;- E;7,5,e^3;7,5,e^4;7	S;7,5;e^3,e^4;- E;7,5,e^3;7,5,e^4;7	S;7,5;e^3,e^4;- E;7,5,e^3;7,5,e^4;7	S;7,5;e^3,e^4;- E;7,5,e^3;7,5,e^4;7	S;7,5;e^3,e^4;- E;7,5,e^3;7,5,e^4;7	S;7,5;e^3,e^4;- E;7,5,e^3;7,5,e^4;7

Figure 6

Time space diagram illustrating the data routing algorithm to collect the chains in the pattern