Clique-to-Clique Distance Computation Using a Specific Architecture

J.Climent[1], A.Grau[1], J.Aranda[1], A.Sanfeliu[2]
[1] Automatic Control and Computer Engineering Department. Universitat Politècnica de Catalunya (UPC).
[2] Institut de Robòtica i Informàtica Industrial.
e-mail: CLIMENT@ESAII.UPC.ES

Abstract. In this paper, we present a new fast architecture to compute the distance between cliques in different graphs. The distance obtained is used as a support function for graph labelling using probabilistic relaxation techniques. The architecture presented consists on a pipe-lined structure which computes the distance between an input clique and k reference cliques. The number of processing elements needed is m^2, and the number of cycles required to compute the distance is n_i (being m the number of external nodes in the input clique, and n_i the number of external nodes in the i-th reference clique). The processing elements are very simple basic cells and very simple communication between them is needed, which makes it suitable for VLSI implementation.

1. Introduction

Probabilistic relaxation techniques are often used for graph matching and labelling. The compatibility on the mapping between two different graphs nodes is determined by a support function Q. A support function presented in [SER] uses the Levenshtein distance between sequences that represent the external nodes of two cliques. When this Levenshtein distance has to be computed , substitution costs are not deductible from the node attributes, and so, the existing schemes and architectures for approximate string matching are not useful. In section III, an algorithm for computing such Levenshtein distances between sequences of nodes is presented.

The complexity of computing the support function for each clique is $O(n_i*m^2)$, being n_i the number of external nodes in the i-th reference clique and m the number of external nodes in the input clique. This complexity is too high. In section IV, we present a fast architecture to speed up that computation. The time complexity will be reduced to $O(n_i)$ using m arrays of m elements.

2. Preliminaries

2.1 Graph matching

The graph matching problem can be accomplished by optimising an energy function. One way to define an energy function is to use a probability-based method. The matching configuration will be selected as the one which optimises the joint probabilities of both graph nodes. It is shown in [HUM] that the probabilistic relaxation can be interpreted as the minimisation of an energy function based in the joint probabilities and an heuristic defined support function. In [KIT], this support function is expressed in terms of probability distributions and then established a probabilistic relaxation method entirely in terms of probability distributions.

A new support function is being presented in [SER]. It is defined by the expression:

$Q = \Psi_{sw} * \Psi_{s\varepsilon} * \Psi_{d}$, where $\Psi_{sw} = e^{-d_{sw}}$, $\Psi_{s\varepsilon} = e^{-d_{s\varepsilon}}$, and $\Psi_{d} = e^{-d_{d}}$

d_{sw} is a measure distance between the two node attributes. $d_{s\varepsilon}$ is a *structure measure distance* between the two sequences of node neighbours. It depends on the attributes of the external nodes of both cliques. d_d is also a *structure measure distance* between the two sequences, but it depends on the current joint probabilities of the external nodes of both cliques.

d_{sw} and $d_{s\varepsilon}$ are static distances, they depend on node attributes and can be computed *a priori*. So, their value do not change during the relaxation process. d_d is a dynamic distance. It depends on the current joint probabilities, so, it has to be computed in every iteration of the process. d_{sw} depends on the distance between two single attributes, so, it has no structural contribution. $d_{s\varepsilon}$ and d_d depend on the structure, and the Levenshtein distance is taken as the structure measure distance.

2.2 Levenshtein distance computation

To determine the structure measure distances $d_{s\varepsilon}$ and d_d, techniques of approximate string matching are used. Since these techniques are well-known, we will not discuss them in this paper. Interested readers can find a good survey in [BUN].

Let C_F^{λ} be a reference clique with a central node v^{λ} and n external nodes represented by the node sequence V_F^{λ}. Let C_A^{γ} be an input clique with a central node v^{γ} and m external nodes represented by the node sequence V_A^{γ}. The complexity of computing the Levenshtein distances between an input sequence of m nodes and k reference sequences of n_i nodes is $O(k*n_i*m)$. Since there is no *a priori* knowledge of input clique orientation, V_A^{γ} will be a cyclic sequence. It is needed to compute the distance for any rotation of the input sequence, and the minimum distance obtained is selected. Since there are m different rotations of V_A^{γ}, the complexity is $O(k*n_i*m^2)$. This cost can be reduced to $O(k*n_i*m*log(m))$ using the schemes presented in [MAE] and in [GRE], but its hardware implementation is not obvious and we are not aware of such an architecture. In order to speed up the computation of these distances, a new specific architecture will be presented in section IV.

When the dynamic distance (d_d) is being computed, the cost of substituting node v^{λ} by node v^{γ}, is determined by its current joint probability (1- P ($v^{\lambda} = v^{\gamma}$)). Then, the hardware implementation becomes rather complicated because substitution costs cannot be deducted from node attributes, and then, the matrix of substitution costs should also be input to the system. The architecture presented in section IV, permits to compute the Levenshtein distance between two symbol sequences using external substitution costs.

2.3 Related Architectures

In this section, we discuss the advantages and disadvantages of the different architectures proposed to compute the Levenshtein distance between strings and their possible applications to compute the dynamic clique distance d_d.

The architectures presented in [LIP] and [LOP] are efficient implementations for the case where the edit costs for substitution, insertion and deletion are fixed to 2, 1, and 1 respectively. However, the applications where this architecture can be used are limited, due to this constraint on the cost of the edit operations. The architecture

presented in [CHE] is two dimensional and requires $m \times n$ processors to process strings of length m and n. Another disadvantage of this architecture is that $m+n$ inputs need to be provided in parallel during each clock cycle. In these architectures, the lengths of the strings that can be matched are constrained by the maximum value that a matrix element can take.

The architecture presented in [SAS] overcomes the disadvantages of the architectures presented before. It does not place any restriction on the length of the costs that the edit operations can take. The encoding scheme presented computes incremental costs instead of absolute costs . It permits to process arbitrary size strings and it also minimises the data flow between adjacent processors. The number of processing elements used is $m + n - 1$, and the number of cycles required to obtain the final result are $m + n - 1 + 2max\{m,n\}$ cycles. The costs are pre-loaded in registers into the processing element during an initialisation phase, thus, they become fixed after the initialisation stage. However, this architecture is not able to compute the dynamic clique distance d_d, since it cannot deal with variable substitution costs non-deductible from node attributes.

The architecture presented in this paper solves the problem of matching sequences of symbols using variable and non-deductible substitution costs. These costs are input to the system. Since node labels and attributes are not needed to calculate substitution costs, the systolic data flow is different from the ones presented in [SAS] and [LIP]. The number of processing elements needed to compute the distance between a reference sequence of n symbols and an input sequence of m symbols is only m, and the number of cycles required to complete the distance calculation is n . The external nodes of a clique are represented with a cyclic sequence of nodes. Since V_A^γ is a cyclic sequence, distance d_d is determined by the minimum distance obtained from the computation for the m different rotations of V_A^γ. Then, The computation of d_d can be finished every n cycles using m^2 processing elements, or every $m*n$ cycles using only m processing elements. The encoding scheme used is the same than in [SAS] , thus, there are not any restrictions on sequence sizes and the data flow between processors is also minimised.

3. Proposed Algorithm

In this section, we describe the algorithm used to calculate the dynamic distance between cliques . An example of matching an input clique with two different reference cliques is detailed to clarify the use of the algorithm.

The encoding scheme used in [SAS] needs, for every element in the distance matrix, two incremental costs instead of one absolute cost. One cost is the difference between the matrix element and its top neighbour, which will be named *vertical incremental cost*. The other one is the difference between the matrix element and its left neighbour, and will be named *horizontal incremental cost*. Our algorithm constructs two different matrices, one for the vertical incremental costs (C_v) and another one for the horizontal incremental costs (C_h). The values of the elements of these matrices for a given row i and a given column j, are determined by the expressions:

$$C_v[i,j] = min\{C_h[i-1,j] + Del, C_v[i,j-1] + Ins, Sub_{\alpha,\beta}[i,j]\} - C_h[i-1,j]$$
$$C_h[i,j] = min\{C_h[i-1,j] + Del, C_v[i,j-1] + Ins, Sub_{\alpha,\beta}[i,j]\} - C_v[i,j-1]$$

$Sub_{\alpha,\beta}$ is the matrix of substitution costs between sequences α and β. *Del* and *Ins* are deletion and insertion costs, which are programmable costs and, for simplicity, have been considered constant for all nodes. The distance between two non-cyclic sequences is determined by both the formulae:

$$d(\alpha,\beta) = m \cdot Ins + \sum_{i=1}^{i=n} C_v[i,m] = n \cdot Del + \sum_{j=1}^{j=m} C_h[n,j]$$

The input sequence, V_A^γ, is a cyclic sequence. Hence, the dynamic distance d_d between a reference clique C_F^λ and an input clique C_A^γ is determined by the expression:

$$d_d = min\left\{d\left(V_F^\lambda, \left(V_A^\gamma\right)^r\right); r = 0..m-1\right\},$$

where $\left(V_A^\gamma\right)^r$ is the r-th rotation of the input sequence.

Figure 1 shows an example of an input clique and two reference cliques , with their respective node label sequences. The Substitution cost matrices are shown in figure 2. The costs are represented within the [0..7] interval, and are determined by the current joint probability.

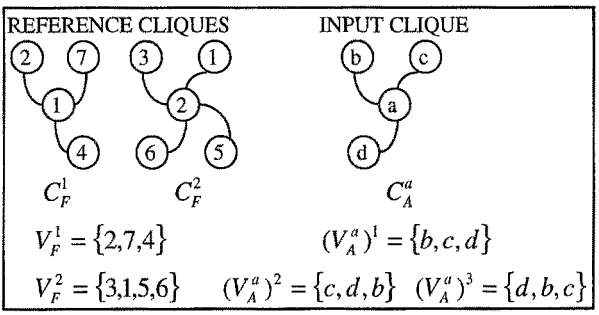

Fig.1. Example

$$Sub_{V_F^1,V_A^a} = \begin{bmatrix} 4 & 5 & 4 \\ 5 & 5 & 5 \\ 5 & 4 & 5 \end{bmatrix}$$

$$Sub_{V_F^2,V_A^a} = \begin{bmatrix} 4 & 7 & 6 \\ 4 & 7 & 7 \\ 7 & 0 & 3 \\ 6 & 7 & 5 \end{bmatrix}$$

Fig.2. Substitution costs matrices

The edit matrices obtained using the algorithm described are shown in figure 3. The horizontal and vertical incremental costs matrices are both represented on the same table, in each table position C_h is the bottom-left value and C_v is the top-right value. Deletion and insertion costs have been fixed to 4.

It can be seen in figure 3 that $d(V_F^1,(V_A^a)^0) = 14$, and $d(V_F^2,(V_A^a)^0) = 13$. There can also be found the distances for all possible rotations of the input sequence: $d(V_F^1,(V_A^a)^1) = 15$, $d(V_F^1,(V_A^a)^2) = 13$, $d(V_F^2,(V_A^a)^1) = 17$, $d(V_F^2,(V_A^a)^2) = 14$. Hence, the dynamic distances between the input clique and the two references are determined by:

$$d_d(V_F^1,V_A^a) = min\{14,15,13\} = 13 \qquad d_d(V_F^2,V_A^a) = min\{13,17,14\} = 13$$

	$(V_A^a)^1$	b	c	d	
V_F^1		+4	+4	+4	**+12**
2	+4	0 0	0 +4	0 +4	**0**
7	+4	+4 0	+1 +1	+1 +4	**+1**
4	+4	+4 0	+3 0	+1 +2	**+1**
	+12	**0**	**0**	**+2**	**14**

	$(V_A^a)^1$	b	c	d	
V_F^2		+4	+4	+4	**+12**
3	+4	0 0	0 +4	0 +4	**0**
1	+4	+4 0	+3 +3	+3 +4	**+3**
5	+4	+4 0	-3 -4	-3 +4	**-3**
6	+4	+4 0	+4 -4	+1 +1	**+1**
	+16	**0**	**-4**	**+1**	**13**

Fig 3. Edit distance matrices

4 Architecture Description

In this section the proposed architecture is presented. First, in figure 4 is shown a simple processing element to calculate the two incremental costs of a single matrix position. Then, in order to design the architecture to compute the whole edit matrix, the dependencies between matrix positions must be determined. Figure 5 shows the dependencies between matrix elements. It can be seen that each element depends only on elements that are located above and to its left. Then, all elements along a 45° diagonal can be calculated simultaneously.

The *Ins* and *Del* registers in figure 4 are pre-loaded with the costs of insertion and deletion respectively during an initialisation phase. At the same time that incremental costs, C_h and C_v, are input to the processing element, the corresponding substitution cost is also input. The processing element calculates new incremental costs based on the results of the algorithm proposed. The incremental vertical cost is then transmitted to the adjacent processor to the right, while the incremental horizontal cost is transmitted down.

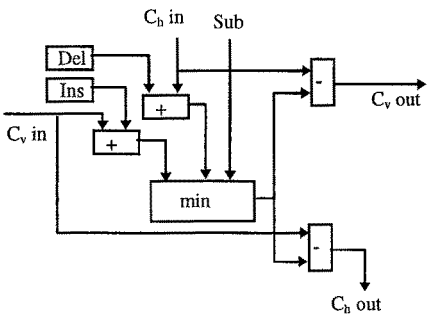

Fig.4. Processing Element

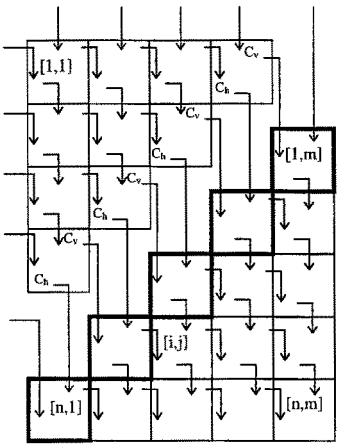

Fig. 5. Dependencies

The block diagram of the architecture needed to calculate the distance between a non-

cyclic input sequence, and k reference sequences is shown in fig. 6. With such a path configuration, each processing elements performs computations along its respective column in the edit distance matrix. The edit distance matrix for comparing sequences of lengths n and m has m such columns. Therefore, m processing elements are needed at every cycle to compute the incremental costs. An accumulator is used to compute the edit distance. The output of the last column processing element is added to the accumulator. The width of the accumulator depends on the maximum value that the edit distance can take, and is, therefore, dependent on the edit costs and the length of the sequences. For this reason, the accumulator is not a part of the architecture and is provided externally.

At every new clock cycle, substitution costs corresponding to a new 45° diagonal of the cost matrix are input to the system. At the same time, incremental costs computed by each processing element are latched into the corresponding horizontal and vertical registers. And the incremental vertical cost coming from the last processing element is added to the accumulator at the same cycle too. The accumulator should be initialised with the value m^* *Ins*. After the first $m + n_l - 1$ cycles, the accumulator contains the distance between the input sequence and the first reference, and every n_i cycles, the distance with a new reference is computed .

An advantage of this architecture is that minimal cost is required. A single phase is needed to control all data flow. Since k reference sequences are being compared, a initialisation signal (*Ini*) is required to reset the registers, each time a new reference sequence is input to the system. The delay elements, named δ, are also controlled by the same clock phase. Figure 6 shows the state of the array after the i-th clock cycle.

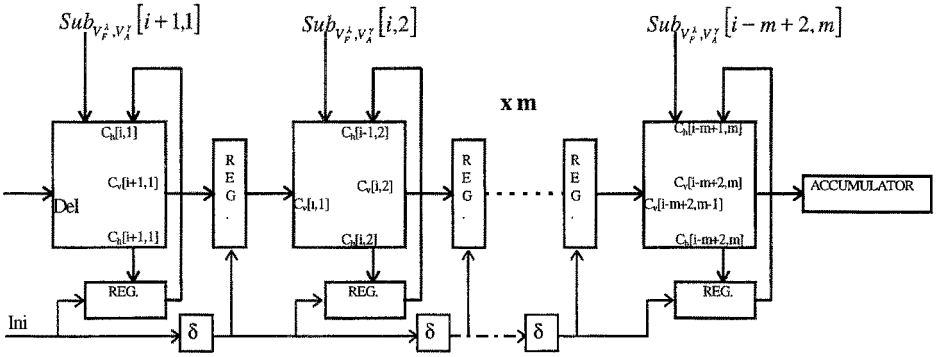

Fig. 6. Array of Processing elements

Only m processors are needed. In those cases where the lengths of input sequences are longer than the number of processing elements in the system array, a cascading strategy can easily be performed just appending more processing elements before the accumulator. This is possible because, as mentioned earlier, the accumulator is the only element whose size is dependent on string lengths, and it does not belong to the architecture. This makes our architecture flexible and adequate for sequences of any length.

For matching cyclic sequences, we present a new architecture which uses m processor arrays like the one in figure 6. Each array of processing elements, will compute the

distance between the references and a concrete rotation $\left(V_A^\gamma\right)^r$ of the input sequence. This array will be identified with the number of the respective rotation, *r*. Since the input sequence $\left(V_A^\gamma\right)^r$ is obtained just rotating V_A^γ *r* times, the substitution cost matrix, $Sub_{V_F^\lambda,\left(V_A^\gamma\right)^r}$ will be obtained rotating $Sub_{V_F^\lambda,V_A^\gamma}$ *r* times too. It can easily be demonstrated that:

$$Sub_{V_F^\lambda,\left(V_A^\gamma\right)^r}[row][col] = Sub_{V_F^\lambda,V_A^\gamma}[row][((col+r-1) \bmod m)+1]; r = 0..m-1$$

At every new clock cycle, substitution costs corresponding to a 45° diagonal of the cost matrix need to be input to every array. This means that the substitution cost being input to the first processing element of every array *r*, during the i-th cycle of operation, will be given by the expression:

$$Sub_{V_F^\lambda,\left(V_A^\gamma\right)^r}[i][1] = Sub_{V_F^\lambda,V_A^\gamma}[i][(r \bmod m)+1]; r = 0..m-1 = Sub_{V_F^\lambda,V_A^\gamma}[i][r+1]$$

This means that the cost to be input to the first processor in each array is the corresponding element of the i-th row of the unrotated substitution costs matrix. It can also be seen that the j-th processor in each array, during the i-th cycle, needs the substitution cost: $Sub_{V_F^\lambda,V_A^\gamma}[i][((j+r-1) \bmod m)+1]$.

Hence, the Substitution matrix will be input row by row to the complete system at each clock cycle. To clarify this concept, figure 7 shows the data path of substitution costs during the i-th clock cycle ($Sub_{V_F^\lambda,V_A^\gamma}$ [i][j] has been written as Sub[i][j] to simplify the notation).

Using this architecture, after every n_i cycles (being n_i the number of nodes in the reference sequence), all the distances between the reference and all possible rotations of input sequence have already been computed.

5 Conclusions

A very fast and simple architecture for computing a support function for graph labelling and matching has been presented. This architecture is able to compute the dynamic distance between an input clique and *k* model cliques. In only n_i cycles, (being n_i the number of nodes of the i-th model) the dynamic distance is computed. An approximate string matching algorithm has been modified to permit to work with external substitution costs. These substitution costs are input to the system. Only a new row of the substitution cost matrix is needed at each new clock cycle. The architecture presented consists on *m* cascadable arrays of *m* processing elements (being *m* the number of nodes of the input clique) and does not place any restrictions on the lengths of node sequences being compared.

References

[BUN] H. Bunke. *"String Matching for Structural Pattern Recognition"*. H.Bunke and A.Sanfeliu Eds, *Syntactic and Structural Pattern Recognition. Theroy and Applications*, pp.381-414, World Scientific, 1990.

[CHE] H.D. Cheng, and K.S. Fu. *"VLSI Architectures for String Matching and Pattern Matching"*. Pattern Recognition, vol.20, n.1, pp. 125-141. 1987.

[GRE] J.Gregor, and M.G. Thomason. "Efficient Dynamic Programming Alignment of Cyclic Strings by Shift Elimination". Pattern recognition, vol.29, n.7, pp.1179-1185, 1996.

[HUM] R.A. Hummel, and S.W. Zucker. *"On the Foundations of Relaxation Labelling Processes"*. IEEE Trans. Pattern Anal. Mach. Intell. Vol.5, N.3, pp. 267-286. 1983.

[KIT] J.Kittler, W.C.Christmas, M.Petrou. *"Probabilistic Relaxation for Maching of Symbolic Structures"*. Advances in structural and syntactic pattern recognition. Ed. by H. Bunke pp. 471-480. 1995.

[LIP] R.J. Lipton, and D. Lopresti. *"A Systolic Array for Rapid String Comparison"*. 1985 Chapel Hill Conf. on VLSI, H. Fuchs, de., Rockville, Md.: Computer Science Press, pp. 363-376. 1985.

[LOP] D. Lopestri. *"P-NAC: A Systolic array for comparing nucleic acid sequences"*. Computer, vol.20, pp. 98-99. 1987.

[MAE] M. Maes. *"On a Cyclic String-to-String Correction Problem"*. Information Proc. Letters, 35, pp.73-78. June 1990.

[SAS] R.Sastry, N. Ranganathan, and K. Remedios. *"CASM: A VLSI Chip for Approximate String Matching"*. IEEE Trans. Pattern Anal. Mach. Intell. Vol.17, N.8, pp. 824-830. 1995.

[SER] F. Serratosa and A. Sanfeliu. *"Function-Described Graphs Applied to 3D Object Representation"*.Image Analysis and Processing, proc. 9th International Conference ICIAP, Florence, pp. 701-708,1997.

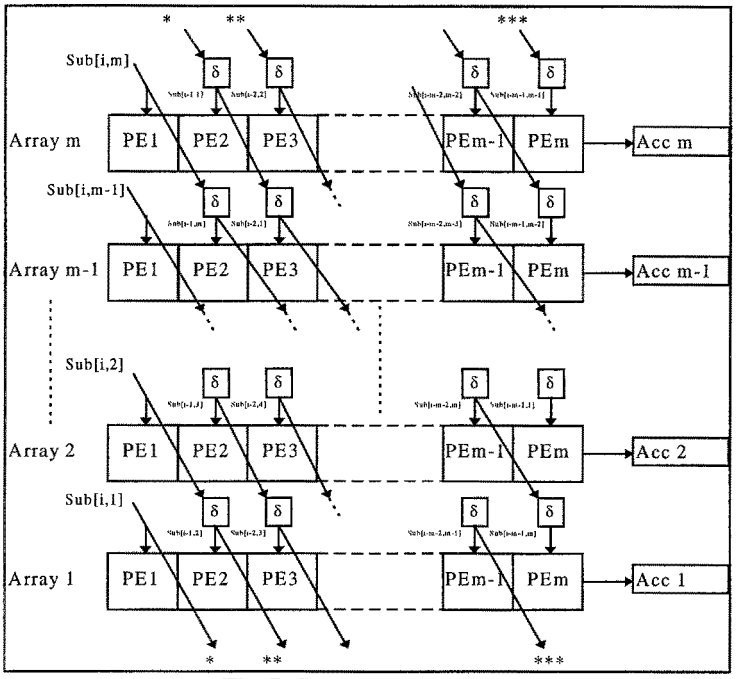

Fig. 7. Complete Architecture