Modified Dynamic Cell Structures as a Thinning Algorithm

Igor Farkaš & Lucius Chudý
Institute of Measurement Science
Slovak Academy of Sciences
Dúbravská cesta 9, 842 19 Bratislava
Slovakia
e-mail: {farkas,chudy}@neuro.savba.sk

Abstract

We provide preliminary results of our thinning algorithm applied to skeletonization of printed characters. The algorithm is based on Dynamic Cell Structures (DCS) [2], which enable the generation of topology-preserving mapping of (generally high-dimensional) input data onto the constructed graph structure. In order to enhance skeletonization, we modify the original DCS algorithm in two subsequent steps. The one consists in constrained distance calculation in every iteration, the other in "static" graph transformation. Both modifications cause the removal of most of the redundant connections resulting thus in character skeletons which are better than those obtained by original DCS.

1 Introduction

Thinning represents one of the possible preprocessing steps in image analysis. It transforms character patterns (usually of binary type and of elongated shape) into line-like structure called skeleton. Skeleton is usually considered to be of unitary thickness, since this is a necessary requirement of many successive pattern recognition tasks (vectorization of the pattern). The usefulness of the skeleton representation of the pattern follows from the attempt to make a data compression while preserving the topological properties of the pattern and also to facilitate the extraction of morphological features useful in the successive pattern recognition task. Due to their central role in the preprocessing stage, thinning algorithms have been a very active area of research from the sixties (more than 300 published papers until now). Application areas include: (handwritten) character recognition, classification of cells, chromosomes, X-ray image analysis, analysis of coronary arteries, processing of bubble-chamber negatives, visual system of automaton, fingerprint classification, quantitative metallography, measurement of soil cracking patterns, automatic visual analysis of industrial parts and printed circuit boards (for references see a comprehensive survey [5]).

Thinning techniques can be broadly classified into *iterative* and *non-iterative*. The huge majority of developed methods belong to iterative ones. They delete pixels directly by means of some proper "expert rules" either in serial (sequential) or parallel manner. Pixels in an image are examined in predetermined order by raster scans (horizontal and/or vertical) or by following contours. The non-iterative techniques (some of them are referred to in literature also as indirect) do not change the pattern in the course of processing. They extract some (local) properties of the pattern which reflect the topology of the pattern (e.g. link the pixels with some special properties, estimate distribution of pattern points, etc.). As an example belonging to this group we may mention the thinning algorithm [6] which is based on fuzzy clustering of pattern points and the skeleton is constructed by connecting neighbouring cluster centers.

The proposed neural network thinning method belongs to the non-iterative group of techniques which are based on some type of clustering process. The only neural network approach we found in literature is Ahmed's self-organizing graph [1], which is derived from the well-known self-organizing map (SOM) [4]. Since the straightforward application of SOM fails to produce good skeletons, the author developed a modification of SOM which enables the time-varying neuron neighbourhood topology during clustering (learning) in order to capture the underlying input distribution. Since the neural network topology is chosen to be one-dimensional, yielding thus two direct neighbors for every neuron (except the first and the last ones), the clusters represented by neurons can be automatically linked to form the connected skeleton.

Our approach is conceptually similar to the previous one in that it is based on the modified version of topology preserving SOM. However, as a difference, our algorithm does not use the predefined number of neurons (units), but rather a flexible (growing) structure of units that is formed with respect to the shape of input distribution. It exploits the ideas of Dynamic Cell Structures (DCS) [2] which are the extension of Growing Cell Structures (GCS) [3]. Both DCS and GCS yield (undirected) graphs, whereas GCS are restricted to preserving graph structure with fixed, preselected dimension. As a consequence, DCS are more suitable for problems, when we do not have a priori information about the dimensionality of data structure to be approximated or when the data contain regions with different dimensionalities.

Our thinning algorithm is a modified version of DCS, because it is tailored to approximation of character patterns.

2 DCS algorithm

Dynamic Cell Structures is a powerful algorithm for topological approximation of various data distributions, including those consisting of disconnected regions. For future reference, its form is briefly reviewed here:

```
initialization;
do {
    for ($\lambda$ times) {
        getNextExample(&v);
        calculateTwoClosest(v,&wB,&wN);
        updateConnectionStrengths(wB,wN,$\alpha$,$\theta$);
        kohonen(w1,v,$\epsilon$B,$\epsilon$N);
        updateResource(wB,v);
    }
    addNewNeuron();
    decrementResourceValues($\beta$);
} while (!stoppingCriterion());
```

The algorithm is initialized by positioning the first two units (whose position is given by their associated weight vectors $\mathbf{w}_1$, $\mathbf{w}_2$) to match respectively the points $\mathbf{v}_1$, $\mathbf{v}_2$, drawn randomly from input distribution. Further, units **1** and **2** are connected by edge, i.e. the elements $A_{12} = A_{21}$ of the (symmetric) connection matrix A, which defines the current topology of the graph, are set to 1.

Now the algorithm enters its outer loop which is repeated until the criterion measure drops below the certain value. A good measure is for example an average quantization error (E_{target}). Within the outer loop, after finishing the inner one, a unit is inserted to the region with the lowest density of units, which is detected by their resource values (expressing units' quantization errors). Specifically, it is placed between the unit with the highest resource and one of its direct neighbors (given by the current state of the connection matrix A), according to the ratio of their resource values. Subsequently, the resource values of all N units are either decreased (in on-line learning), i.e. $\tau_i(t+1) = \beta.\tau_i(t)$, with $1 \leq i \leq N, 0 < \beta < 1$, or reset to zero (in off-line learning, i.e. when the input distribution is static).

The inner loop starts with random drawing an input $\mathbf{v}$ from the training set, followed by finding the two units whose weight vectors $\mathbf{w}_B$, $\mathbf{w}_N$ are closest to $\mathbf{v}$ in terms of the Euclidean distance. Next, the lateral connections (elements of A) are updated according to a competitive Hebbian learning rule: connection between the two closest units is enforced, i.e. $A_{BN} = 1$ [1], all other connections are decreased using the forgetting factor α: $A_{ij}(t+1) = \alpha.A_{ij}(t)$. Next, every connection A_{ij} whose value falls below threshold θ is removed ($A_{ij} = 0$).

Procedure kohonen(w1,v,ϵB,ϵN) adjusts the weight vector of the winner and of its neighbors, i.e. $\Delta\mathbf{w}_B = \epsilon_B.(\mathbf{v}-\mathbf{w}_B)$ and $\Delta\mathbf{w}_j = \epsilon_N.(\mathbf{v}-\mathbf{w}_j)$, for $j \in \mathcal{N}(B)$, where the neighborhood $\mathcal{N}(B) = \{j | A_{Bj} > 0, 1 \leq j \leq N\}$.

[1]This is the former, computationally a bit simpler version of the DCS algorithm. The alternative way of enforcement sets $A_{BN}(t+1) = max\{A_{BN}(t), y_B.y_N\}$, where the units are considered to have a gaussian-like input-output function $y_i = R(\|\mathbf{v} - \mathbf{w}_i\|)$.

Finally, the winner's resource value, as a measure of local square quantization error, is updated by adding an increment $\Delta\tau_B = ||\mathbf{v} - \mathbf{w}_B||^2$.

2.1 Application of DCS to printed characters

Our simulations have shown that the application of original DCS algorithm to printed characters yields quite good results, as can be seen in Fig. 1. Specifically, if the parameters of the algorithm (mainly the stopping criterion, α and λ) are suitably chosen, the resulting skeleton has the tendency to consist mostly of one-dimensional lines connected at junctions. Parameters were set as follows: $\theta = 0.001, \alpha = \sqrt[\theta]{S}, \lambda = S/2, \beta = 0, \epsilon_B = 0.2, \epsilon_N = 0.01$, where S denotes the size of the training set (the number of black pixels in an input image). Probably the most crucial point — the stopping criterion — needed to be solved with respect to the line thickness of the character in an image, in order to force one-dimensional skeletonization. The learning stopped, as soon as the average squared quantization error decreased below certain value E_{target}. To make the procedure automatic, E_{target} was estimated from the average line thickness that was previously also estimated using a simple procedure (in the simulations, on average $E_{target} = 7$). However, as seen in Fig. 1, some of the connections are redundant. Either they connect close, but disconnected regions (e.g. in characters β, φ or ξ), or they produce unit "polygons", mostly triangles (e.g. in α, κ, ϕ or ψ), which violate 1D skeletonization.

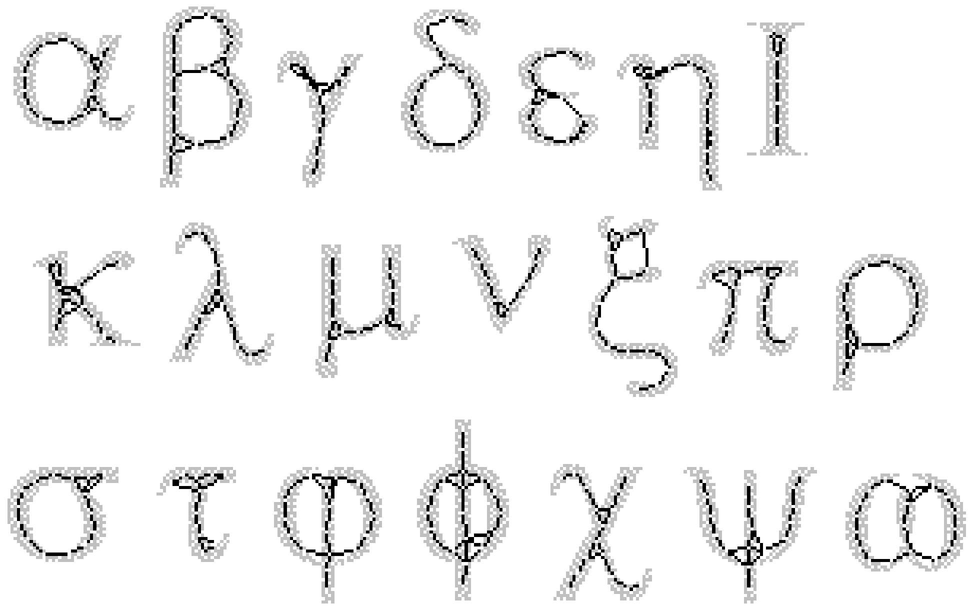

Figure 1: Character skeletons obtained by original DCS algorithm. Black lines denote existing connections between units (white points at the place of graph vertices). Input patterns were randomly drawn from the shaded areas. For the purpose of display, the essentially floating-point-valued unit coordinates were rounded to integers.

3 The thinning algorithm

We intentionally classify the observed polygons in two categories: *cross-boundary* polygons and *within-boundary polygons.* The polygons of the former type (unlike the latter) have the property that at least one of the participating connections passes through the non-input region. This is the property that has been exploited in our modification of DCS algorithm. Its goal was to possibly prevent the creation of such connections (including false connections, mentioned above, which do not violate 1D skeletonization).

3.1 Removing redundant cross-border connections

The modification is based on estimating the constrained Euclidean distance of the units from the current input $\mathbf{v}$. By imposed constraint we mean the estimation of the shortest path from $\mathbf{v}$ to a unit without leaving the regions of inputs. Calculation of such a constrained path in general does not appear to be trivial, therefore we chose a simplified algorithm instead. The idea is sketched in Fig. 2.

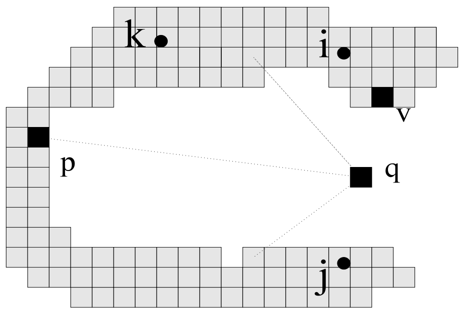

Figure 2: Example of the situation when the constrained distance between unit j and input $\mathbf{v}$ is much longer than the Euclidean distance. For explanation see the text.

In the case when the connecting line between unit j and input $\mathbf{v}$ leaves the input region, we find the midpoint among those lying in non-input region (point $\mathbf{q}$). From this point the closest black pixel is sought in (two mutually opposite) perpendicular directions, of which the minimum is taken. In this way, we get point the $\mathbf{p}$. [2]

If we denote the Euclidean distances $d_E(\mathbf{v}, \mathbf{w}_j) = a$, $d_E(\mathbf{q}, \mathbf{p}) = b$ and $d_E(\mathbf{v}, \mathbf{p}) = d_E(\mathbf{p}, \mathbf{w}_j) = c$, then the squared constrained distance is approximated using simple triangle geometry

$$d_C^2(\mathbf{v}, \mathbf{w}_j) = 4c^2 = 4b^2 + a^2.$$

Thus, in the algorithm, $d_C^2(\mathbf{v}, \mathbf{w}_j)$ is computed instead of $d_E^2(\mathbf{v}, \mathbf{w}_j)$. As an effect, unit j becomes more distant to $\mathbf{v}$ than the unit k and therefore, not the connection i-j, but i-

[2] Due to variety of simulations, this searching procedure had to be made more exhaustive. Namely, in both directions, the search in done within an angle of 45 degrees (the three dotted lines) in order to make the estimate more robust.

k will be enforced in current iteration. Statistically, this distance constraint will cause the connection i-j to die off even though the following inequality may mostly hold during learning: $d_E(w_j, v) < d_E(w_k, v)$ for inputs $\mathbf{v}$ for which unit i wins the competition.

Calculation of constrained distances considerably slows down the learning process. Moreover, it is intended for penalizing cross-border connections aimed at their removal. For the two reasons, it was applied only in the final phase of learning. In our simulations, it replaced the original procedure calculateTwoClosest(v,&wB,&wN) as soon as the average quantization error dropped below 1.5 multiple of E_{target}.

3.2 Removal of within-boundary polygons

It may be obvious that the emergence of within-boundary polygons is impossible to hinder using the above DCS approach with constrained distance calculation. Within-boundary polygons appear in regions of greater line thickness (note that the stopping criterion refers to *average* square quantization error). As a matter of fact, the placement of units (and existence of connections between them) is in accordance with tendency of DCS algortihm to approximate input distribution.

We applied a non-neural algorithm, which subsequently, after training, transforms detected polygons in the three following steps: (1) add a unit into the center of gravity of the polygon; (2) connect all units in the polygon with the new unit and (3) remove mutual connections between units (except the new one) in the polygon. However, an undesired side-effect, which may appear, is the emergence of new branche(s) (e.g. in bottom character γ).

Due to the above graph-transformation procedure the number of units is increased and also their distribution may deteriorate. For these reasons, it is followed by fine-tuning phase using procedure kohonen(w1,v,ϵB,ϵN), this time with both ϵ_B and ϵ_N decreasing to zero.

In summary, the thinning algorithm consists of 3 phases:

1. DCS algorithm with included distance calculation constraint in its final phase
2. procedure for removal of within-boundary polygons
3. fine-tuning.

3.3 Simulation results

We performed tests on various sets of printed characters. As an example, we have included the Greek alphabet (Fig. 3). Typical duration of processing of one character was in range 4000–8000 iterations, taking overall a few seconds on a conventional 486-based PC. The enhancement of character skeletons can be observed when comparing Fig. 3 with Fig. 1.

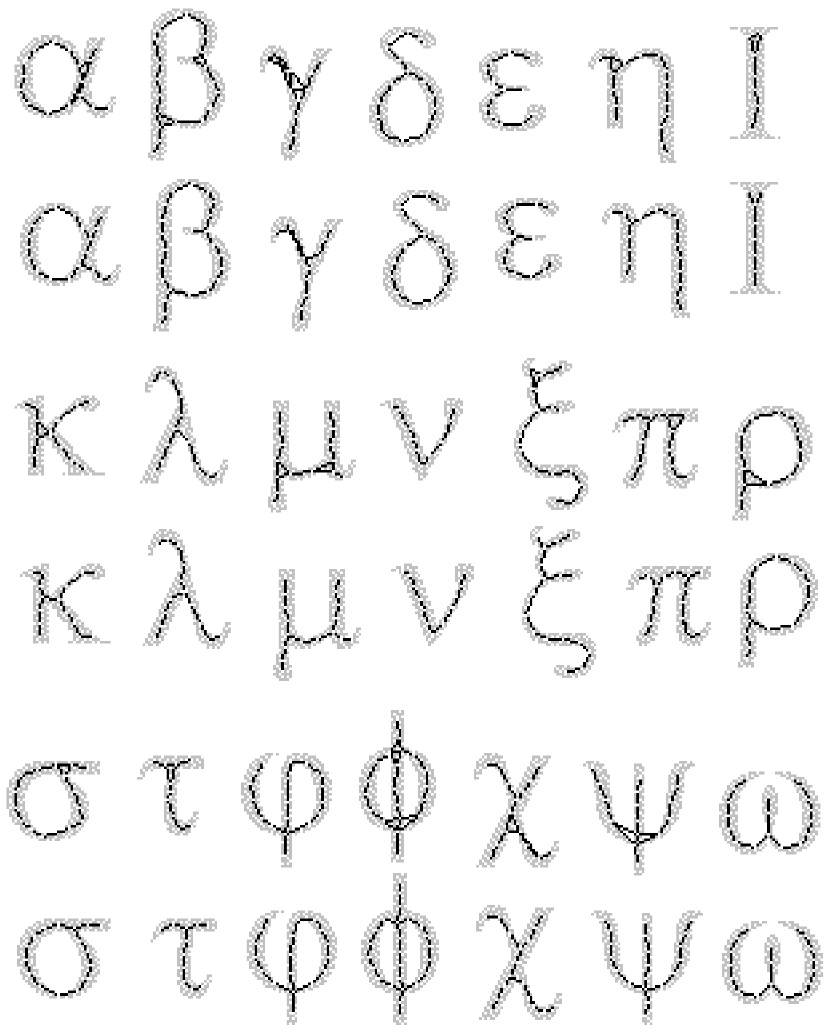
Figure 3: Example set of printed characters obtained by proposed thinning algorithm. in each couple, the upper row displays the outputs after phase 1, the bottom rows are final graphs (after phase 3).

4 Conclusion

We provided preliminary results of our hybrid thinning algorithm derived from Dynamic Cell Structures. According to our simulations, it is difficult to obtain 1D skeletons using original DCS, due to varying thickness of characters, as well as their complicated (curved) structure. Removal of redundant connections could be partially maintained by induced distance evaluation constraint (cross-border connections). Within-border connections had to be deleted (with a side-effect) with a static graph-modification procedure.

Acknowledgement

This work was partially supported by Slovak Grant Agency for Science (grants No. 2/2040/95 and No. 95/5305/468).

References

[1] P. Ahmed. A neural network based dedicated thinning method. *Pattern Recognition*, 16:585–590, 1995.

[2] J. Bruske and G. Sommer. Dynamic cell structure learns perfectly topology preserving map. *Neural Computation*, 7:845–865, 1995.

[3] B. Fritzke. Growing cell structures - a self-organizing network for unsupervised and supervised learnig. *Neural Networks*, 7(9):1441–1460, 1994.

[4] T. Kohonen. *Self-Organizing Maps.* Springer Verlag, 1995.

[5] L. Lam, S. Lee, and C.Y. Suen. Thinning methodologies – a comprehensive survey. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(9):869–885, 1992.

[6] S. Mahmood, I.S. Abuhaiba, and R.G. Green. Skeletonization of arabic characters using clustering based skeletonization algorithm (CBSA). *Pattern Recognition*, 24:453–464, 1991.