NEIGHBORHOOD METRICS ON n-DIMENSIONAL BLOCKS OF CHARACTERS

Samuel R. Buss
Mathematical Sciences Research Institute
July 1985

Abstract.

A mathematical framework for constructing metrics on arrays of characters is defined. This is inspired by the Yianilos string matching algorithm. New metrics ν_d are defined which provide a fuzzy comparison function on d-dimensional blocks of characters. A sequential algorithm for ν_2 is given with runtime $O(N^2 \cdot (\#A))$ which operates on N×N blocks of characters where #A is the alphabet size. A parallel algorithm for ν_2 is presented which uses N^2 processors and has runtime O(N+(#A)). Possible applications of ν_2 include image recognition.

Research supported in part by NSF Grant 8120790.

Introduction.

Fuzzy or approximate string matching is becoming more and more widely used as the domain of computer applications expands to include pattern recognition and artificial intelligence. The most widely used versatile fuzzy string matching algorithms are the Demerau-Levenstein metrics [1], which are computed via a dynamic programming algorithm. A less well-known fuzzy string matching function is due to Yianilos [3]. Yianilos' metric seems to be a good substitute for Demerau-Levenstein metrics, yet is much easier to compute.

This paper presents a generalization of the Yianilos metric to higher dimensions. We shall focus on the 2-dimensional case since it seems to have useful applications.

We first present a mathematical framework for defining a large variety of what we call "neighborhood metrics" and "neighborhood similarity measures", and briefly show how the simple Yianilos similarity measure may be defined as a neighborhood metric. Next, new neighborhood metrics and similarity measures are introduced, which act on d-dimensional blocks of characters; these are in some sense a generalization of the Yianilos metric. Since the two-dimensional case is the most interesting, we describe a serial algorithm and a parallel algorithm for computing the two-dimensional block metric. We conclude by sketching a possible application to image recognition.

A General Framework for Defining Metrics.

The Yianilos and the Demerau-Levenstein metrics measure the similarity between two strings of characters. Formally speaking, if A is an alphabet (i.e. a non-empty set), then a *string* of n characters is a mapping f:[n]⟶A where [n] denotes the set {0,1,2,...,n-1}.

Instead of restricting ourselves to strings of characters, we shall deal with *arrays* of characters. An *array of characters* U consists of a *domain* G and a mapping f:G⟶A, where A is the *alphabet*. We write U = <G,f> to denote an array of characters. For example, if G = [n], then U = <G,f> is a string of characters; whereas if $G = [n]^d$ then U is an d-dimensional *block* of characters.

The members of the alphabet A are *characters*. Each character $a \in A$ is assigned a nonnegative integer *weight* w_a. Frequently it is convenient to include a blank character, denoted "Δ", in A, with weight $w_\Delta = 0$.

When H is a finite set, let #H denote the cardinality of H. If U = <G,f> and V = <H,k> are arrays of characters with the same alphabet A and if $D \subset G$ and $E \subset H$ then define $\text{Common}_{U,V}(D,E)$ to be equal to

$$\sum_{a \in A} w_a \cdot \min(\#\{i \in D : f(i)=a\}, \#\{i \in E : k(i)=a\}).$$

So $\text{Common}_{U,V}(D,E)$ is equal to the weighted sum of the number of characters common to the subarrays <D,f> and <E,k>.

We define

$$\text{Weight}_U(D) = \sum_{a \in A} w_a \cdot \#\{i \in D : f(i)=a\}.$$

The identity

$$\text{Weight}_U(D) = \sum_{i \in D} w_{f(i)}$$

shows that $\text{Weight}_U(D)$ is equal to the sum of the weights of the characters in the subarray <D,f>.

To simplify notation, we shall frequently suppress the subscripts and just write Common(D,E) and Weight(D).

If G and H are domains then a *neighborhood correspondence* ♦ *between*

G *and* H is a set of ordered pairs <D,E> such that D ⊂ G and E ⊂ H. When U = <G,f> and V = <H,k> we define Dist(U,V,Φ) to be equal to

$$\sum_{<D,E>\in\Phi} \text{Weight}_U(D) + \text{Weight}_V(E) - 2\cdot\text{Common}(D,E).$$

It is easy to see that an equivalent definition for Dist(U,V,Φ) is

$$\sum_{\substack{<D,E>\in\Phi \\ a\in A}} w_a \cdot \left| \#\{i\in D: f(i)=a\} - \#\{i\in E: k(i)=a\} \right|$$

where the vertical bars denote "absolute value".

The function Dist gives us a notion of "distance" or "discrepancy" between two arrays of characters, provided we have an appropriate neighborhood correspondence. We want to define metrics which will give a notion of "distance" between *any* two arrays of characters. Our metrics will be defined by specifying the neighborhood correspondences between pairs of domains. However, the types of neighborhood correspondences must be restricted in order to obtain useful metrics.

Let $\mathcal{D}$ be a set of domains. Then a *neighborhood metric* δ *over* $\mathcal{D}$ consists of an *enumeration space* $\mathcal{I}$ and a *neighborhood enumeration function* Γ. This is denoted by δ = <$\mathcal{I}$,Γ>. The enumeration space $\mathcal{I}$ is a countable set and Γ is a function with domain $\mathcal{I}\times\mathcal{D}$. In addition, the following properties must be satisfied:

(1) For all $j\in\mathcal{I}$ and $G\in\mathcal{D}$, Γ(j,G) ⊂ G.

(2) For all $G\in\mathcal{D}$, the support of Γ at $\mathcal{D}$, i.e. the set $\{j\in\mathcal{I}:\Gamma(j,G)\neq\emptyset\}$, is finite.

The idea is that for every fixed domain $G\in\mathcal{D}$, Γ(j,G) enumerates a finite set of neighborhoods (i.e. subsets) of G, indexed by $j\in\mathcal{I}$.

It remains to define the numerical value of the neighborhood metric δ. Let U = <G,f> and V = <H,k> be arrays of characters with $G,H\in\mathcal{D}$. Define Φ(G,H) to be the neighborhood correspondence between G and H such that <D,E>∈Φ(G,H) if and only if

(1) $D \neq \emptyset$ or $E \neq \emptyset$, and

(2) $\exists j \in \mathcal{J}$ so that $\Gamma(j,G) = D$ and $\Gamma(j,H) = E$.

Then $\delta(U,V)$ is defined to be equal to

$$\delta(U,V) = \text{Dist}(U,V,\Phi(G,H)).$$

It is easy to verify that δ satisfies the following properties:

(1) For all U,V as above, $\delta(U,V) \geq 0$. In fact, since character weights are non-negative integers, $\delta(U,V)$ must be a non-negative integer.

(2) $\delta(U,V) = \delta(V,U)$. This is because the neighborhood correspondence δ is specified by a neighborhood enumeration function.

(3) For all appropriate arrays of characters U,V,W;

$$\delta(U,V) \leq \delta(U,W) + \delta(W,V).$$

This is the "triangle inequality". This is proved by noting that each term in the summation defining Dist satisfies the triangle property.

Thus a neighborhood metric is indeed a metric.

With a neighborhood metric δ, we associate a norm $\| \cdot \|_\delta$ which is defined by

$$\| \langle G,f \rangle \|_\delta = \sum_{j \in \mathcal{J}} \text{Weight}_U(\Gamma(j,G)).$$

Note that the summation is finite since there are only a finite number of $j \in \mathcal{J}$ such that $\Gamma(j,G) \neq \emptyset$.

If H is any domain and if the alphabet A contains the blank symbol Δ, let $0_H = \langle H, \Delta_H \rangle$ where Δ_H is the unique function $\Delta_H: H \longrightarrow (\Delta)$. Then it is readily seen that

$$\| U \|_\delta = \delta(U,0_H).$$

A neighborhood metric δ has an associated *neighborhood similarity measure* θ_δ which is defined by

$$\theta_\delta(U,V) = 1 - \frac{\delta(U,V)}{(\|U\|_\delta + \|V\|_\delta)} .$$

In general a *similarity measure* θ satisfies the following properties for all U,V:

(1) $0 \leq \theta(U,V) \leq 1$

(2) $\theta(U,V) = \theta(V,U)$

(3) $\theta(U,U) = 1$

The idea is that θ provides a fuzzy measure of the similarity between two arrays of characters on a scale of 0 to 1. So $\theta(U,V)$ near 0 indicates U and V are very dissimilar whereas $\theta(U,V) \approx 1$ denotes U and V are nearly identical. Note that a similarity measure may not satisfy the triangle inequality.

The above "abstract nonsense" definition of neighborhood metrics and similarity measures is somewhat tedious, so we attempt to liven things up by giving some examples in the next two sections. However, the mathematical viewpoint adopted above is very useful, since we will be able to use our geometric intuitions to construct neighborhood enumerations which yield useful neighborhood metrics and similarity measures.

The Simple Yianilos Metric.

The Yianilos metric operates on strings of characters so we let $\mathcal{U}_1$ be the set $\{[n]:n\geq 0\}$. The enumeration space $\mathcal{S}_1$ is defined to be

$$\mathcal{S}_1 = \{\langle i,m\rangle : 0\leq i\leq 1,\ 0\leq m,\ i \text{ and } m \text{ integers}\}.$$

The neighborhood enumeration function Γ_1 is defined so that

$$\Gamma_1(\langle 0,m\rangle,[n]) = \{i:\ 0\leq i<n-m,\ i \text{ an integer}\}$$
$$\Gamma_1(\langle 1,m\rangle,[n]) = \{i:\ m\leq i<n,\ i \text{ an integer}\}.$$

It is obvious that $\Gamma_1(j,[n]) \subset [n]$ for all $j\in\mathcal{S}$. Also, if $m \geq n$ and $i = 0$ or 1, then $\Gamma_1(\langle i,m\rangle,[n])$ is the empty set. Hence, for all n, the support of Γ_1 at $[n]$ is finite.

By definition, the *simple Yianilos metric* is $\nu_1 = \langle \mathcal{S}_1,\Gamma_1\rangle$ with $\mathcal{S}_1$ and Γ_1 as above. We let θ_1 be the *simple Yianilos similarity measure* associated with ν_1. Incidentally, it can be shown that if $U = \langle [n],f\rangle$ is a string of characters of length n, then

$$\|U\|_{\nu_1} = (n+1)\cdot \text{Weight}(U).$$

The Yianilos similarity measure is a useful and practical function, since it gives results which agree well with human intuition and, in addition, can be computed by a *linear-time* algorithm. Thus it seems to be a good replacement for the Demerau-Levenstein metrics since the best known algorithms for computing the latter are square time. In fact, the algorithm for the Yianilos similarity measure has been implemented in a custom VLSI circuit manufactured by Proximity Technology, Inc. which was founded by Peter Yianilos.

The similarity measure θ_1 defined above is called the *simple* Yianilos similarity measure since more complicated and more versatile similarity measures have been defined by Yianilos. Although it would be possible to extend the mathematical framework of the previous section to incorporate the properties of the general Yianilos similarity measure, we

refrain from doing this, as the primary interest of this paper is metrics on blocks of characters.

The Yianilos similarity measure was first defined in [3] and [4] by a very different approach than ours. In [5], the equivalence of our definition to Yianilos's is shown. See [2] for a description of the VLSI implementation of the Yianilos's similarity measure.

A Metric on Blocks of Characters.

Let d be fixed positive intger, called the *dimension*. Let $n_0,\ldots,n_{d-1}$ be a vector of positive integers, denoted $\vec{n}$. We write $[\vec{n}]$ to denote the domain

$$[\vec{n}] = \{\langle i_0,\ldots,i_{d-1}\rangle : \text{for all } j,\ i_j \text{ is an integer, } 0 \leq i_j < n_j\}.$$

Intuitively, $[\vec{n}]$ is a d-dimensional grid of lattice points of size n_0 by n_1 by n_2 ... by n_{d-1}.

A *d-dimensional block of characters* is defined to be an array of characters of the form $\langle[\vec{n}],f\rangle$ for some d-tuple $\vec{n}$.

We next define a neighborhood metrics ν_d on d-dimensional blocks of characters. The set of domains $\mathfrak{D}_d$ will contain all domains of the form $[\vec{n}]$ with $\vec{n}$ a d-tuple of positive integers. Let $\vec{2}$ be the d-tuple $\langle 2,2,\ldots,2\rangle$. The enumeration space $\mathcal{S}_d$ is defined to be $[\vec{2}]\times\mathbb{N}^d$ where $\mathbb{N}$ is the natural numbers. In other words,

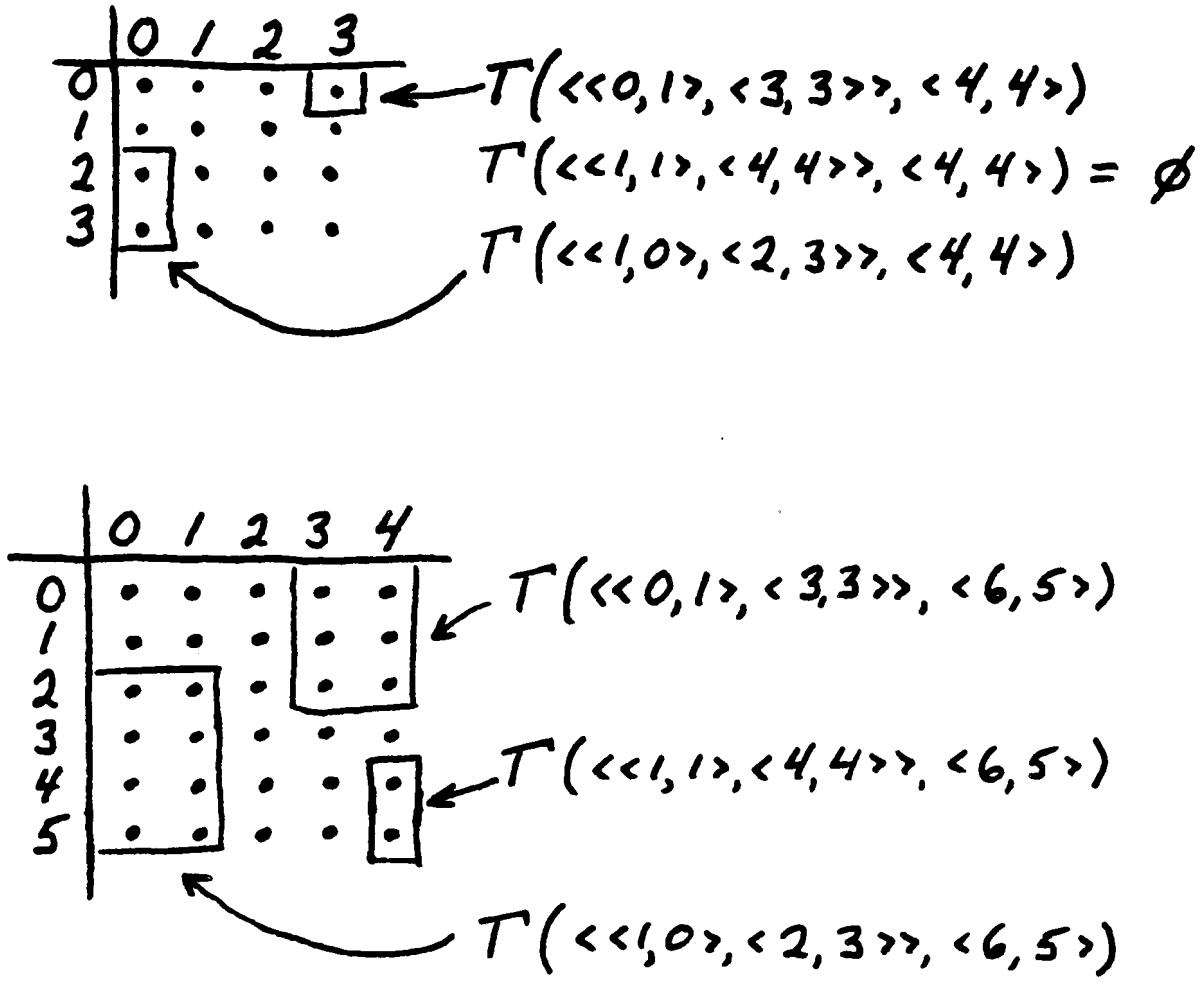
$$\mathcal{S}_d = \{\langle \langle j_0,\ldots,j_{d-1}\rangle,\langle m_0,\ldots,m_{d-1}\rangle \rangle : 0 \leq i_r \leq 1,\ m_r \geq 0 \text{ for all } r\}.$$

The neighborhood enumeration function Γ_d is defined by

$$\Gamma_d(\langle\vec{j},\vec{m}\rangle,[\vec{n}]) = \{\langle i_0,\ldots,i_{d-1}\rangle : m_r \cdot j_r \leq i_r < n_r - m_r \cdot (1-j_r)\}.$$

The d-*dimensional block metric* ν_d is $\langle\mathcal{S}_d,\Gamma_d\rangle$ and its associated similarity measure is θ_d. The associated norm is denoted $\|\cdot\|_{\nu_d}$.

Note that when $m_r > n_r$ for some r, then $\Gamma_d(\langle\vec{j},\vec{m}\rangle,[\vec{n}])$ is the empty set. Since $[\vec{2}]$ is a finite set, it follows that Γ_d has finite support at $[\vec{n}]$. Hence ν_d is a well-defined neighborhood metric. Figure 1 shows some examples of the neighborhood enumeration Γ_2.



Some examples of the neighborhood enumeration Γ_d function for 2-dimensional domains.

Figure 1

Theorem 1. Let $\vec{n}$ be a d-tuple and $U = \langle [\vec{n}], f \rangle$ be a d-dimensional block of characters. Then

$$\|U\|_{\nu_d} = \text{Weight}(U) \cdot \prod_{r=0}^{d-1} (n_r+1).$$

Proof. By the definition of $\|U\|_{\nu_d}$ and of Weight we have

$$\begin{aligned} \|U\|_{\nu_d} &= \sum_{j \in \mathcal{J}_d} \text{Weight}_U(\Gamma_d(j,U)) \\ &= \sum_{j \in \mathcal{J}_j} \sum_{x \in \Gamma_d(j,U)} w_{f(x)} \\ &= \sum_{x \in [\vec{n}]} (w_{f(x)} \cdot (\#\text{ of } j \in \mathcal{J}_d \text{ s.t. } x \in \Gamma_d(j,U))). \end{aligned}$$

Since $\sum_{i \in [\vec{n}]} w_{f(i)}$ is equal to Weight(U), it will suffice to show that, for all $x \in [\vec{n}]$, the number of $j \in \mathcal{J}_d$ such that $x \in \Gamma_d(j,U)$ is equal to $\prod_{r=0}^{d-1} (n_r+1)$.

The proof is by induction on d. When d=1, $[\vec{n}]$ is just $[n_0] = (\langle i \rangle : 0 \leq i < n_0)$. Let $x = \langle k \rangle$ be an arbitrary member of $[\vec{n}]$. Then for $j \in \mathcal{J}_1$,

$$x \in \Gamma_1(j, [\vec{n}]) \iff \begin{cases} j = \langle 0, m \rangle \text{ with } 0 \leq m < n_0 - k, \text{ or} \\ j = \langle 1, m \rangle \text{ with } 0 \leq m \leq k \end{cases}$$

So there are exactly $(n_0-k) + (k+1) = n_0+1$ members j of $\mathcal{J}_1$ such that $\langle k \rangle \in \Gamma_1(j, [\vec{n}])$. This proves the assertion for the case d = 1.

Now suppose the assertion is true for all d < N; we prove it for d = N. An arbitrary element x of $[\vec{n}]$ has the form $\langle k_0, \ldots, k_{N-1} \rangle$. Now, if $j \in \mathcal{J}_N$, j has the form $\langle \langle i_0, \ldots, i_{N-1} \rangle, \langle m_0, \ldots, m_{N-1} \rangle \rangle$. Let $\tilde{x}$, $\tilde{n}$, $\tilde{j}$ denote $\langle k_0, \ldots, k_{N-2} \rangle$, $\langle n_0, \ldots, n_{N-2} \rangle$ and $\langle \langle i_0, \ldots, i_{N-2} \rangle, \langle m_0, \ldots, m_{N-2} \rangle \rangle$, respectively. Then

$$x \in \Gamma_N(j,[\vec{n}]) \Leftrightarrow \begin{cases} i_{N-1} = 0, \ 0 \le m_{N-1} < n_{N-1} - k_{N-1}, \text{ and } \tilde{x} \in \Gamma_{N-1}(\tilde{j},[\vec{n}]); \text{ or} \\ i_{N-1} = 1, \ 0 \le m_{N-1} \le k_{N-1}, \text{ and } \tilde{x} \in \Gamma_{N-1}(\tilde{j},[\vec{n}]). \end{cases}$$

By the induction hypothesis, there are exactly $\prod_{r=0}^{N-2}(n_r+1)$ different $\tilde{j} \in \mathcal{I}_{N-1}$ such that $\tilde{x} \in \Gamma_{N-1}(\tilde{j},[\vec{n}])$. Each of these $\tilde{j}$'s can be extended in $(n_{N-1}-k_{N-1}) + (k_{N-1}+1) = (n_{N-1}+1)$ different ways to yield a $j \in \mathcal{I}_N$ such that $x \in \Gamma_N(j,[\vec{n}])$. Hence there are exactly $\prod_{r=0}^{N-1}(n_r+1)$ such j's. This completes the proof. ■

The next theorem gives a basic property of the d-dimensional block metric which should be satisfied by any reasonable metric on blocks of characters.

Theorem 2. Let $d \geq 1$. Suppose the alphabet A has at most one character of zero weight (i.e. the blank). If U and V are d-dimensional blocks of characters and $\nu_d(U,V) = 0$, then either

(a) U is identical to V, or

(b) Weight(U) = Weight(V) = 0 (in other words, every character of U and V is a blank).

Note that case (b) may hold, yet $U \neq V$ since U and V may have different domains.

Proof. Let $U = \langle[\vec{n}],f\rangle$ and $V = \langle[\vec{m}],g\rangle$ be d-dimensional blocks of characters. If D and E are subsets of $[\vec{n}]$ and $[\vec{m}]$ respectively, then $\text{Differ}_{U,V}(D,E)$ is defined to be

$$\text{Weight}_U(D) + \text{Weight}_V(E) - 2 \cdot \text{Common}_{U,V}(D,E).$$

By the definition of neighborhood metric,

$$\nu_d(U,V) = \sum_{j \in \mathcal{I}_d} \text{Differ}_{U,V}(\Gamma_d(j,[\vec{n}]),\Gamma_d(j,[\vec{m}])).$$

Since Differ is always non-negative, it will suffice to show that if $U \neq V$ and if one of U or V has non-zero weight then for some $j \in \mathcal{I}_d$ $\text{Differ}_{U,V}(\Gamma_d(j,[\vec{n}]),\Gamma_d(j,[\vec{m}]))$ is non-zero.

Assume that U and V have the same domain so $\vec{n} = \vec{m}$. Since $U \neq V$, it is easy to see that there is some $\vec{i} = \langle i_0,\ldots,i_{d-1}\rangle$ in $[\vec{n}]$ such that (1) for all $\vec{k} = \langle k_0,\ldots,k_{d-1}\rangle$ such that $\vec{k} \neq \vec{i}$ and $k_r \leq i_r$ for $0 \leq r < d$, $f(\vec{k}) = g(\vec{k})$ and (2) $f(\vec{i}) \neq g(\vec{i})$. For instance, $\vec{i}$ can be chosen to be the lexicographicly first element of $[\vec{n}]$ such that $f(\vec{i}) \neq g(\vec{i})$. Let $\vec{p}$ be the d-tuple such that $p_r = n_r - i_r - 1$ for $r = 0,1,\ldots,d-1$. Then

$$\Gamma_d(\langle \vec{0},\vec{p}\rangle,[\vec{n}]) = \{\langle k_0,\ldots,k_{d-1}\rangle : 0 \leq k_r \leq i_r \text{ for } 0 \leq r < d\}.$$

Then $\text{Differ}_{U,V}(\Gamma_d(\langle \vec{0},\vec{p}\rangle,[\vec{n}]),\Gamma_d(\langle \vec{0},\vec{p}\rangle,[\vec{n}]))$ is equal to $w_{f(\vec{i})} + w_{g(\vec{i})}$. Since A has at most one character of weight zero, $w_{f(\vec{i})} + w_{g(\vec{i})} \neq 0$.

The above argument proves the theorem when $\vec{n} = \vec{m}$. Then case $\vec{n} \neq \vec{m}$ is proved by a slightly more complicated argument which we leave as an exercise for the reader. ■

To get a feel for how the block metrics work, consider the following simple example. Let $U = \langle [\vec{n}], f\rangle$ and $V = \langle [\vec{n}], g\rangle$ be d-dimensional blocks of characters. Let the alphabet A contain the blank character Δ and a character "a" with weight $w_a = 1$. Let $\vec{i}$ and $\vec{j}$ be elements of $[\vec{n}]$ and suppose f and g are defined so that

$$f(\vec{p}) = \begin{cases} a & \text{if } \vec{p} = \vec{i} \\ \Delta & \text{otherwise} \end{cases}$$

$$g(\vec{p}) = \begin{cases} a & \text{if } \vec{p} = \vec{j} \\ \Delta & \text{otherwise.} \end{cases}$$

So U and V each contain only a single instance of a nonzero weight character. We wish to determine the values of the distance $\nu_d(U,V)$ between U and V and the similarity $\theta_d(U,V)$ of U and V.

Let $\|\vec{i}-\vec{j}\|_1$ denote $\sum_{r=0}^{d-1} |i_r-j_r|$, the "one-norm" distance between i and j. Then we have the following theorem:

Theorem 3. Let U and V be as above. Then

(1) $$\nu_d(U,V) = 2\cdot\prod_{r=0}^{d-1}(n_r+1) - 2\cdot\prod_{r=0}^{d-1}(n_r+1-|i_r-j_r|)$$

(2) $$\theta_d(U,V) = \prod_{r=0}^{d-1}\left(1 - \frac{|i_r-j_r|}{n_r+1}\right).$$

So when $\vec{i}$ and $\vec{j}$ are relatively close,

$$\theta_d(U,V) \approx 1 - \sum_{r=0}^{d-1}\frac{|i_r-j_r|}{n_r+1}.$$

In other words, θ_d is "locally $\mathscr{L}^1$".

Proof. We first prove (1). By the definitions of ν_d, Differ and $\|\cdot\|_{\nu_d}$ and by Theorem 1, we have

$$\begin{aligned}\nu_d(U,V) &= \sum_{k\in\mathscr{L}_d} \mathrm{Differ}_{U,V}(\Gamma_d(k,[\vec{n}]),\Gamma_d(k,[\vec{n}])) \\ &= \|U\|_{\nu_d} + \|V\|_{\nu_d} - \sum_{k\in\mathscr{L}_d} 2\cdot\mathrm{Common}_{U,V}(\Gamma_d(k,[\vec{n}]),\Gamma_d(k,[\vec{n}])) \\ &= 2\cdot\prod_{r=0}^{d-1}(n_r+1) - 2\cdot\sum_{k\in\mathscr{L}_d}\mathrm{Common}_{U,V}(\Gamma_d(k,[\vec{n}]),\Gamma_d(k,[\vec{n}])).\end{aligned}$$

For r = 0,...,d-1, let $\delta_r = |i_r-j_r|$. It will suffice to show that

$$\sum_{k\in\mathcal{L}_d} \text{Common}_{U,V}(\Gamma_d(k,[\vec{n}]),\Gamma_d(k,[\vec{n}])) = \prod_{r=0}^{d-1}(n_r+1-\delta_r).$$

First note that since U and V contain only the single character "a", each term of the sum will be equal to 0 or 1. So the sum will be equal to the number of elements k of $\mathcal{L}_d$ such that both $\vec{i}$ and $\vec{j}$ are in $\Gamma_d(k,[\vec{n}])$.

To count the number of such k's in $\mathcal{L}_d$, let k be $\langle\vec{p},\vec{m}\rangle$. Then both $\vec{i}$ and $\vec{j}$ are in $\Gamma_d(k,[\vec{n}])$ if and only if, for r = 0,...,d-1, either

(a) $p_r = 1$ and $0\leq m_r\leq \min(i_r,j_r)$, or

(b) $p_r = 0$ and $0\leq m_r< n_r-\max(i_r,j_r)$.

Since the conditions on the d coordinates are independent of each other and since there are

$$(\min(i_r,j_r)+1) + (n_r-\max(i_r,j_r)) = n_r+1-\delta_r$$

possible values for p_r and m_r to satisfy (a) and (b), it follows that number of $k\in\mathcal{L}_d$ with both $\vec{i}$ and $\vec{j}$ in $\Gamma_d(k,[\vec{n}])$ is equal to $\prod_{r=0}^{d-1}(n_r+1-\delta_r)$. This completes the proof of (1).

We now prove (2) from (1). By (1),

$$\theta_d(U,V) = 1 - \frac{\nu_d(U,V)}{2\cdot\prod_{r=0}^{d-1}(n_r+1)}$$

$$= \frac{\prod_{r=0}^{d-1}((n_r+1)-\delta_r)}{\prod_{r=0}^{d-1}(n_r+1)}$$

$$= \prod_{r=0}^{d-1}\left(1 - \frac{\delta_r}{n_r+1}\right).$$

By the binomial theorem, if each $\frac{\delta_r}{n_r+1}$ is close to zero, then the similarity value can be approximated by

$$\theta_d(U,V) \approx 1 - \sum_{r=0}^{d-1} \frac{\delta_r}{n_r+1}.$$

When $n_0 = n_1 = \ldots = n_{d-1}$, then $\theta_d(U,V) \approx 1 - \frac{\|\vec{I}-\vec{J}\|_1}{(n_0+1)}$. Note that in the case d = 1 (i.e. the simple Yianilos metric) the approximation is exact. ■

The situation vastly more complicated when U and V contain multiple occurrences of the same character. It is very important to note that the block neighborhood metric does not even implicitly assign a one-to-one correspondence between characters in U and characters in V.

Efficient Algorithms for computing the 2-dimensional Block Metric.

We give below two algorithms for computing the 2-dimensional block neighborhood metric ν_2: the first is a sequential algorithm, and the second is a parallel algorithm using $n_0 \cdot n_1$ processors. Part of the reason this author thinks the block metrics ν_d are important is that there are efficient and practical algorithms for computing them. We concentrate on the 2-dimensional metric ν_2 partly for notational and conceptual convenience and also because it has potentially useful applications. One possible application of ν_2 is discussed in the next section.

An important property of any neighborhood metric is that distinct characters contribute independently to the metric value. This is expressed more precisely by Theorem 4 below.

Definition. Let A be a alphabet containing a blank character Δ. If f is a function with range A and if $a \in A$ then f_a is the function

$$f_a(x) = \begin{cases} a & \text{if } f(x) = a \\ \Delta & \text{otherwise.} \end{cases}$$

If $U = \langle G,f \rangle$ is an array characters then U_a is the array of characters $\langle G,f_a \rangle$.

Theorem 4. Let ν be a neighborhood metric and θ_ν its associated similarity measure. Then

$$\nu(U,V) = \sum_{\Delta \neq a \in A} \nu(U_a,U_a).$$

Furthermore, if $\|U\|_\nu \neq 0$ or $\|V\|_\nu \neq 0$, then

$$\theta_\nu(U,V) = 1 - \frac{\sum_{\Delta \neq a \in A} \nu(U_a, V_a)}{\sum_{\Delta \neq a \in A} (\|U_a\|_\nu + \|V_a\|_\nu)}$$

Proof. This is easily seen by examining the definition of a neighborhood metric. ■

A consequence of Theorem 4 is that to compute $\nu(U,V)$ we need merely compute $\nu(U_a,V_a)$ for all $a \in A$ and take this sum. Accordingly, we can restrict ourselves to the case where the alphabet A contains only the blank symbol Δ and one other character.

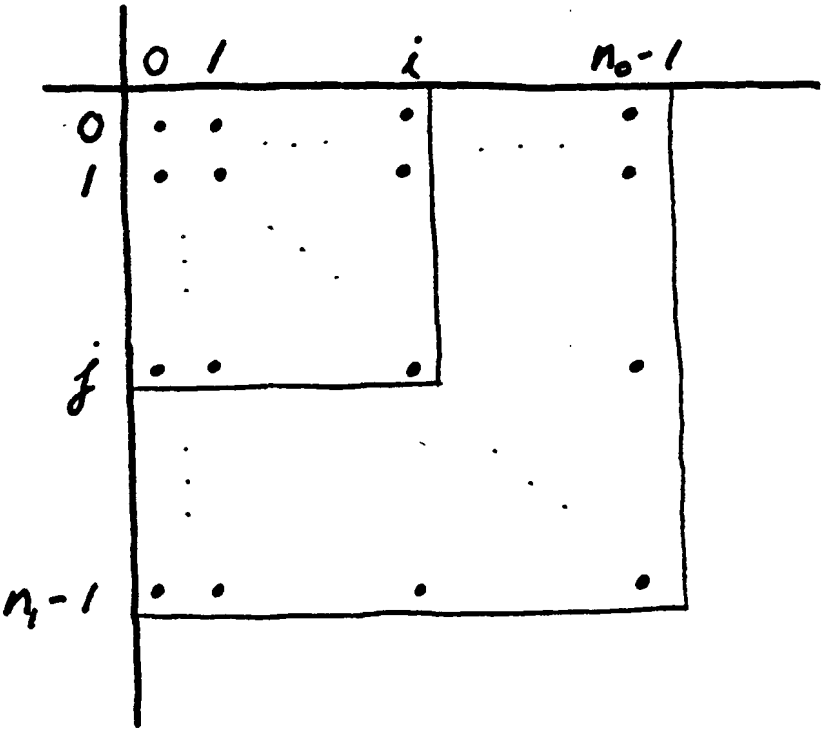
We now describe a (sequential) algorithm to compute ν_2. As input, we have two strings $U = \langle [n_0,n_1],f \rangle$ and $V = \langle [n_0,n_1],g \rangle$. We are assuming U and V have the same domain partly for simplicity and partly because this is the most interesting case. A is assumed to have only one nonzero weight character which without loss of generality may be assumed to have weight equal to 1.

The algorithm scans the $n_0 \times n_1$ domain in a rasterscan pattern four times. Each step of the scan considers a rectangular neighborhood (see Figure 2). The variable i varies faster than j. The algorithm has the following variables: (The C programming language is used in this paper.)

```
int R[n0];          /* number of matching characters
                       in rectangular neighborhood */
int T[n0];          /* signed tally of non-matching character
                       in rectangle */
int TOTR[n0];       /* total number of characters
                       in rectangle */
int PR;             /* Partial row sum for R[] */
int PT;             /* Partial row sum for T[] */
int PTOTR           /* Partial row sum for TOTR[] */

int M;              /* Value of ν(U,V) */
int TOTM;           /* Value of ||U||ν+||V||ν */

int i,j;            /* Work variables */
```



The sequential algorithm for 2-dimensional block matching scans in raster fashion. Shown above is the neighborhood denoted (i,j) during the first of the four scan phases. Each dot denotes a point where a character may appear.

Figure 2.

Actually, some of these variable are redundant as PTOTR is always equal to the sum of the absolute value of PT and twice PR. Likewise, TOTR[i] is always equal to |T[i]|+2·R[i]. The algorithm for computing $\nu_2(U,V)$ and $\theta_2(U,V)$ is:

```
/* Initialize all variables to zero */
   M = TOTM = 0;
   "zero all entries of R[], T[], TOTR[]";
/* First Scan (rectangles in upper left corner) */
   for (j=0; j<n1; j++) {
      PR = PT = PTOTR = 0;
      for (i=0; i<n0; i++) dopixel(i,j);
   }
/* Second scan (rectangles in upper right corner) */
   for (j=0; j<n1; j++) {
      PR = PT = PTOTR = 0;
      for (i=n0-1; i>=0; i--) dopixel(i,j);
   }
/* Third scan (rectangles in lower left corner) */
   for (j=n1-1; j>=0; j--) {
      PR = PT = PTOTR = 0;
      for (i=0; i<n0; i++) dopixel(i,j);
   }
/* Fourth scan (rectangles in lower right corner) */
   for (j=n1-1; j>=0; j--) {
      PR = PT = PTOTR = 0;
      for (i=n0-1; i>=0; i--) dopixel(i,j);
   }

/* Exit conditions:      M = ½(||U||ν2 + ||V||ν2 - ν2(U,V))
                         TOTM = ||U||ν2 + ||V||ν2
                         θ2(U,V) = 2·M/TOTM          */
   exit ();
```

where the subroutine dopixel is

```
dopixel (i, j) {
     if ( "f(i,j)==a" ) {
          PTOTR++;
          if ( PT < 0 ) PR++;
          PT++;
     }
     if ( "g(i,j)==a" ) {
          PTOTR++;
          if ( PT > 0 ) PR++;
          PT--;
     }
     if ( PT*T[i] < 0 ) {
          R[i] += min( abs(PT), abs(T[i]) );
     }
     T[i] += PT;
     R[i] += PR;
     M += R[i];
     TOTR[i] += PTOTR;
     TOTM += TOTR[i];
     return ();
}
```

It is not too difficult to verify that the above algorithm does compute ν_2 and θ_2 correctly. It should be noted that the rectangular neighborhood indexed by (i,j) in the first scan (see Figure 2) is the neighborhood enumerated as $\Gamma_2(x,[n_0,n_1])$ where

$$x = < <0,0>,<n_0-(i+1),n_1-(j+1)> >.$$

The runtime of the above sequential algorithm is easily computed; the subroutine dopixel is called $4 \cdot n_0 \cdot n_1$ times. In general, the alphabet A may contain many characters and the runtime of the sequential algorithm is $4 \cdot n_0 \cdot n_1 \cdot (\#A)$ where #A is equal to the number of nonzero weight characters of A.

We next describe a parallel, systolic algorithm for computing $\nu_2(U,V)$. We no

longer require that U = $\langle[n_0,n_1],f\rangle$ and V = $\langle[n_0,n_1],g\rangle$ contain only a single character; instead, the alphabet A is the first (#A)+1 integers (0,1,2,...,#A), where 0 is the blank symbol.

The algorithm operates on an n_0 by n_1 rectangular array of processors; each processor can communicate only with its 4 immediate neighbors. Each processor corresponds naturally to a point (i,j) in the domain $[n_0,n_1]$ and has the following registers:

(1) F	- equals the value of f(i,j); $0 \leq F \leq \#A$
(2) G	- equals the value of g(i,j); $0 \leq G \leq \#A$
(3) CURCHAR	- current character; $0 \leq CURCHAR \leq \#A$
(4) R	- sums (both row and column)
(5) T	- tallies (both row and column)
(6) COLFLAG	- one bit column status flag
(7) ROWFLAG	- one bit row status flag
(8) OUTFLAG	- one bit indicating "Output Ready"
(9) EDGE	- one bit flag indicating edge of array

We will adopt the convention that there are imaginary processors around the edge which are labelled (i,j) with i = -1, i = n_0, j = -1, or j = n_1. For these imaginary processors, every register value is hardwired to zero, except for EDGE which is hardwired to one. The real processors have EDGE hardwired to zero.

The processors run synchronously; that is to say, at time t, each processor has access to the contents of the registers of its four neighbors as computed during the previous clock cycle at time t-1. Actually, only the registers numbered (4)-(7), (9) need be accessible. The results of the calculation are extracted by an additional master processor; the master processor has access to the registers OUTFLAG and R of the four corner processors labelled (0,0), (n_0-1,0), (0,n_1-1) and (n_0-1,n_1-1).

Just as in the sequential algorithm, there are four different scan phases. During first phase, the upper left scan, each of the $n_0 \cdot n_1$ processors runs the following program:

```
if ( CURCHAR != 0 ) {
    int A=0,B=0;      /* temporary variable */
    if ( CURCHAR == F ) A=1;
    if ( CURCHAR == G ) B=1;
```

```
        OUTFLAG = 0;
        if ( !COLFLAG && !ROWFLAG &&
                (left->ROWFLAG || left->EDGE) ) {
            R = LEFT->R;
            if ( A && B ) R++;
            else if ( A && !B ) R += (left->T < 0);
            else if ( !A && B ) R += (left->T > 0);
            T = A - B + left->T;
            ROWFLAG = 1;
        }
        else if ( !COLFLAG && ROWFLAG &&
                (up->COLFLAG || up->EDGE) ) {
            R = up->R;
            if ( T*up->T < 0 )
                R += min ( abs(T), abs(up->T) );
            T += up->T
            COLFLAG = 1;
        }
        else if ( ROWFLAG && COLFLAG &&
                ( !left->ROWFLAG || left->EDGE) ) {
            R += left->R;
            ROWFLAG = 0;
        }
        else if ( !ROWFLAG && COLFLAG ) {
            R += up->R;
            CURCHAR --;
            COLFLAG = 0;
            OUTFLAG = 1;
        }
    }
```

As can be observed each processor has very little work to do at each clock cycle. In brief, the activities of the processors are as follows:

When ROWFLAG == 0 and COLFLAG == 0, the partial row sums of R and T are computed.

When ROWFLAG == 0 and COLFLAG == 1, the true values of R and T are computed.

When ROWFLAG == 1 and COLFLAG == 1, the partial row sums of TOTR are computed.

When ROWFLAG == 0 and COLFLAG == 1, the true value for TOTR is computed. OUTFLAG is set to signal this and CURCHAR is decremented to begin again with the next character.

The master processor runs the first scan as follows:

First Scan (left & up):

(1) Each processor is initialized so that ROWFLAG = COLFLAG = 0 and CURCHAR = (#A).

(2) A globally distributed clock runs one step of the above program.

(3) If the lower right processor, labelled (n_0-1,n_0-1), has OUTFLAG = 1 then the master processor computes the product $R \cdot w_{CURCHAR}$ of the contents of its R register and the weight of the current character. This product is added to a register M of the master processor.

(4) Return to steps (2) until steps (2) and (3) have been performed $2(n_0+n_1)+4 \cdot (\#A)-2$ times. Or equivalently, until the lower right processor has OUTFLAG = 1 and CURCHAR = 0.

The second, third and fourth scans are similar, except that the directions are different. The directions are "*right & up*", "*left & down*", and "*right & down*". The programs for the processors are changed by replacing each occurrences of "left" and/or "up" by "right" and/or "down". Also in steps (3), the master processors inspects the lower left, upper right, and upper left processor, respectively.

So overall the master processor computes the 2-dimensional block metric by

(1) Load character array by assigning correct numbers to F and G registers throughout the array of processors.

(2) Set M = 0

(3) Run first scan (left and up)

(4) Run second scan (right and up)

(5) Run third scan (left and down)

(6) Run fourth scan (right and down)

(7) Terminate: the value of M is equal to $\frac{1}{2}(\|U\|_{\nu_2}+\|U\|_{\nu_2}-\nu_2(U,V))$.

Not counting the time to load the character array (i.e. step 1), the total runtime is

$$8(n_0+n_1) + 16\cdot(\#A) - 2.$$

This is quite satisfactory when #A is not too large, especially if $\#A \leq n_0+n_1$. In order to compute $\nu_2(U,V)$, the value of TOTM is also needed. This can be computed either simultaneously with M or by using the identity of Theorem 1.

In summary, the sequential algorithm presented above takes time $O(n_0\cdot n_1\cdot \#A)$ and the parallel algorithm with $n_0\cdot n_1+1$ processors takes time $O(n_0+n_1+\#A)$. Further algorithms should be developed. Firstly, many applications seem to involve very sparse blocks of characters, so it would be desirable to have a sequential algorithm which works well when there are a lot of blanks present. Of course this would require representing sparse blocks of characters in a more compact form. Secondly, a parallel algorithm that works well when $\#A \approx n_0\cdot n_1$ could be useful.

A Possible Application to Image Recognition.

The main reason for our interest in neighborhood metrics in the feeling that they may be useful. To justify this feeling we present a rough sketch or "back-of-the-envelope-calculation" of how the 2-dimensional block metric ν_2 could be used for image recognition using parallel processing.

The image recognition problem is briefly as follows: a TV camera views a single object; the camera supplies its image as a digitized frame to a computer. The task of the computer is to decide what the object is by using a reference database of possible objects. This task is logically divided into two parts. First, feature extraction is performed. An example of a feature could be the presence of a vertical, left edge at a certain point. We assume there are 32 possible features. Once the features are extracted, they are mapped down to a 32×32 grid; in effect, we decrease the resolution of the picture after features have been detected. Our alphabet A will be the blank symbol plus 32 feature symbols. So the process of feature extraction maps the image to a 32×32 block of characters from the alphabet A.

The reference database of images is, of course, preprocessed and stored as blocks of characters of the same type. The image U is compared to each of these and the reference image V with the highest similarity $\theta_2(U,V)$ is the answer. The computation of the similarity measure θ_2 will be on a 32×32 array of processors.

Suppose there are 1000 reference images. Each computation of similarity measure θ_2 takes time

$$8(32+32) + 16 \cdot 32 - 2 = 1022$$

clock cycles. Before each computation, the reference image must be loaded into the array of processors; we assume this takes 64 clock cycles. Thus the entire sequence of 1000 comparisons takes 1.086×10^6 clock cycles. If we assume a clock speed of 4 MHz, the elapsed time is 0.272 seconds.

If we assume that the feature extraction process takes about the same time, then the entire recognition takes about 0.6 seconds. With a 1000 database images we could save representations of 62 items in each of two different sizes and 8 different orientations (since $62 \cdot 2 \cdot 8 = 992 \leq 1000$). Thus we estimate that 62 different items, arbitrarily oriented, could be distinguished.

Our assumption that we have a 32×32 array of processors is not unreasonable since

the processors are very simple. Indeed, wafer-scale integration technology is probably already capable of producing such an array on a single wafer. Or, a less ambitious project would be to build an integrated circuit containing an 4×4 array of processors; 64 of these could be combined to effect a 32×32 array.

In summary, we feel that the θ_2 similarity measure may have practical applications. The scenario presented above is very speculative and is most likely not the most efficient use of θ_2 for image recognition.

Summary.

Neighborhood metrics and neighborhood similarity measures provide a useful and intuitive means for defining metrics on arrays of characters. The Yianilos similarity measure is one example. The n-dimensional block metrics and similarity measures defined in this paper are another. The 2-dimensional block metric may be useful for image recognition applications.

REFERENCES

[1] Joseph B. Kruskal, "An overview of sequence comparison", in *Time warps, String Edits and Macromolecules: the theory and practice of sequence comparisons*, ed: D. Sankoff & J.B. Kruskal, pp. 1–44.

[2] Proximity Technology, Inc., PF474 *Product Data Book*, Ft. Lauderdale, Florida, 1984.

[3] Peter N. Yianilos, "The definition, computation and application of symbol string similarity functions", Masters Thesis Emory University 1978.

[4] Peter N. Yianilos, nine volume business plan, 1979.

[5] Peter N. Yianilos and Samuel R. Buss, "Associative Memory Circuit System and Method", Continuation in part, U.S. Patent #4490811, issued Dec. 25, 1984.