Information Retrieval, 5, 353–375, 2002

Generalized Hamming Distance

ABRAHAM BOOKSTEIN a-bookstein@uchicago.edu
Center for Information and Language Studies, University of Chicago, Chicago, IL 60637

VLADIMIR A. KULYUKIN vkulyukin@cs.usu.edu
Computer Science Department, Utah State University, Logan, Utah 84322-4205

TIMO RAITA* raita@cs.utu.fi
Computer Science Department, University of Turku, 20520 Turku, Finland

Received October 13, 1999; Revised May 23, 2002; Accepted July 12, 2002

Abstract. Many problems in information retrieval and related fields depend on a reliable measure of the distance or similarity between objects that, most frequently, are represented as vectors. This paper considers vectors of bits. Such data structures implement entities as diverse as bitmaps that indicate the occurrences of terms and bitstrings indicating the presence of edges in images. For such applications, a popular distance measure is the Hamming distance. The value of the Hamming distance for information retrieval applications is limited by the fact that it counts only exact matches, whereas in information retrieval, corresponding bits that are close by can still be considered to be almost identical. We define a "Generalized Hamming distance" that extends the Hamming concept to give partial credit for near misses, and suggest a dynamic programming algorithm that permits it to be computed efficiently. We envision many uses for such a measure. In this paper we define and prove some basic properties of the "Generalized Hamming distance", and illustrate its use in the area of object recognition. We evaluate our implementation in a series of experiments, using autonomous robots to test the measure's effectiveness in relating similar bitstrings.

Keywords: information retrieval, hamming distance, metrics, computer vision, image retrieval, object recognition, robot vision

1. Introduction

The Hamming distance is often used to quantify the extent to which two bitstrings of the same dimension differ. An early application was in the theory of error-correcting codes (see, e.g., Hamming (1980)) where the Hamming distance measured the error introduced by noise over a channel when a message, typically a sequence of bits, is sent between its source and destination. Bit sequences also appear within the Information Retrieval (IR) environment. For example, bitmaps can indicate the documents a term occurs in, with the Hamming distance quantifying differences in the occurrence patterns of terms. A typical application of such a distance measure is to create term clusters (Bookstein and Klein 1991).

*It is our sad duty to report that Timo Raita passed away shortly before this paper was completed. Timo was a close and valued colleague. He will be greatly missed.

In a traditional application of the Hamming distance, the only concern is whether the corresponding bits in two strings agree. However, in a full text database, more subtle distinctions are required. For example, bitmaps may be used to indicate the units (sentences, paragraphs, etc.) within a single document that a term occurs in; in this context, it is troubling that two terms that tend to occur "close" to one another, even if not in exactly the same units, are assessed by the Hamming distance in the same way as terms that are completely unrelated to one another. In other words, the original Hamming distance does not recognize the idea of neighboring units.

As an example, consider the problem of automatically segmenting a body of text (Hearst 1994, Hearst and Plaunt 1993). In this context, it is useful to have a measure of how well an algorithmic text segmentor agrees with a target partition, as defined, for example, by a judge. In this case, both the target and the source can be represented by bitmaps, with each sentence (or other text unit) being represented by a bit position, and 1-bits indicating segment boundaries. Here, ideally, we should have an exact matching of 1-bits, and the number of discrepancies is the most obvious measure of algorithm failure. This measure is the Classic Hamming distance (CHD).

However, unlike the many coding applications in which the Hamming distance is used, in the text segmentation context we have a notion of bit-site proximity, and a measure of failure can reasonably be expected to take this into account. For example, CHD does not distinguish whether a discrepancy between a target and source 1-bit are separated by one or many textual units. Consider the following target bitmap (a), and the candidate source bitmaps (b) and (c):

(a) 1100100000
(b) 1100010000
(c) 1100000001

As assessed by CHD, both (b) and (c) are equally good efforts to match the target's segment boundaries: both differ from the target by a distance of 2. But intuitively, one is inclined to regard (b) as a better match than (c): while both (b) and (c) fail at matching the last 1-bit, (b) misses by only one unit, which may be quite acceptable for applications such as document segmentation. Similarly, in image matching applications, the noise introduced by the channel, e.g., a camera, makes exact matches highly improbable, so that the best solution is to find matches that are close enough. For such applications, we would like a distance measure that assesses (a) to be closer to (b) than it is to (c).

In general, there is a great deal of arbitrariness in defining a measure of goodness. But a minimal desideratum of a measure of quality is that it at least satisfy such intuitive criteria as suggested by the applications for which it is intended. The Generalized Hamming distance (GHD) we define below introduces this flexibility. In addition, if our measure is to be applied in contexts where noise is significant, we would hope that it is more reliable than the classic measure when the exact placement of 1-bits is influenced by random effects. Experiments described in Bookstein et al. (2001) suggest that this is true for GHD.

While motivated by the problem of assessing the effectiveness of segmentation algorithms, GHD should be applicable to other problems in which the notion of bit-neighborhoods is valid. For example, in the course of computing GHD, our algorithm constructs

an "optimal" sequence of steps that transforms one bitmap into the other. Such a process is similar to that used for fax compression (Witten et al. 1999), and an encoding of the optimized transformation process may permit us to improve compression in applications requiring higher performance. A more flexible Hamming distance can also be of use in situations which require assessing the closeness of pairs of event sequences, in which nearby events are considered to be associated; such a requirement occurs in some data-mining applications (Adriaans and Zantinge 1996, Kulyukin and Burke to appear). In this paper, we illustrate our algorithms by considering the problem of object recognition, such as may occur in image retrieval, or, perhaps, character recognition. We evaluate our implementation in a series of image matching experiments with an autonomous robot.

In the following section we extend CHD to be sensitive to situations in which the concept of neighboring locations is important. This extension will be defined in terms of the operations necessary to transform the source bitmap to the target bitmap. Conditions sufficient for this measure to be a true distance will be derived. The paper concludes by describing an application in object recognition.

2. Generalized Hamming distance

The traditional definition of a Hamming distance is simple and intuitively appealing: given two bitstrings (more generally, strings of symbols) of the same dimension, the Hamming distance is the minimum number of symbol changes needed to change one bitmap into the other. One can view this as a type of edit-distance (Sankoff and Kruskal 1983, Crochemore and Rytter 1994), but with a highly restricted set of edit operations.

Our generalization of the Hamming distance is also a type of edit distance. But by extending the edit-operation set, we are able to recognize a notion of neighborhood that gives credit for near misses. Unlike most edit distances, our measure compares pairs of fixed size bitmaps instead of general strings. The restriction to comparing two bitmaps of equal size allows our measure to focus only on the disposition of 1-bits, which greatly simplifies our task. However, it relies critically on a "shift" operation that, while important for us, has not gotten much, if any, attention in the string literature (Crochemore and Rytter 1994).

Suppose then that we wish to measure the distance between two bitmaps. For many applications, these enter symmetrically, but this need not be the case. For example, one bitmap may represent a target bitmap (B_T); the other may be a source bitmap (B_S) that is the output of an algorithm that is trying to match that target. To assess the success of the algorithm, we wish to measure how similar the two bitmaps are. Ideally, B_S and B_T should be identical.

Notationally, we let B denote a bitmap, and M denote its dimension. We define a bit-site to be a position in a bitmap that contains a 1-bit or a 0-bit. For example, the bit-map might describe a sequence of M sentences, with a bit-site associated with each sentence and a 1-bit indicating that the corresponding sentence ends a segment of text. We shall use the notation $N(B)$ to denote the number of 1-bits in the bitmap B. We next define GHD as an edit distance that measures the difficulty of transforming the source bitmap (B_S) into the target bitmap (B_T).

2.1. *Edit distance: Overview*

To compute an edit distance we first define a set of elementary edit-operations, and associate with each a cost. The edit operations we define must certainly include the *insert/delete* operations of CHD. In addition, we introduce a *shift* operation that allows us to transfer a 1-bit in B_S to a nearby 1-bit in B_T at less cost than deleting the 1-bit in B_S and inserting it in B_T. The shift operation is an abstraction of the concrete task of attempting to match a 1-bit in a target and missing, but getting close—it thus captures, for our measure, the notion of neighboring bit-sites.

Given a set of elementary operations, with a cost assigned to each, a measure of the difference between two bitmaps can be computed by using these operations to transform one bitmap to the other, and adding up the costs of the operations used. To eliminate the ambiguity attached to the multiplicity of possible transformation sequences, we quantify how close one bitmap is to another by the *minimum* cost over sequences of elementary operations that transform the source bitmap into the target bitmap.

2.2. *Edit distance: Details*

Computing edit distances typically requires processing whole strings. However, the special nature of our problem will allow us to focus on only the 1-bits of the bitmap, which reduces the computational cost of our algorithm. Thus, to develop our algorithm, it is convenient to describe a participating bitmap by a list of the index values, in ascending order, of its 1-bits. For example, we might represent the bitmap B_S by $\mathcal{S} = \langle s_1, s_2, \ldots, s_{N(\mathcal{S})} \rangle$, where s_i is an integer denoting the position of the i-th 1-bit of the bitmap B_S, and $N(\mathcal{S}) = N(B_S)$. Below we shall refer to both representations, B_S and $\mathcal{S}$, as "the source bitmap"; we use $\mathcal{T}$ to denote the corresponding representation of the bitmap B_T.

In general, let $c(i, j)$ denote the minimum cost of transforming the first i 1-bits of the source bitmap, B_S, into the first j 1-bits of a target bitmap, B_T, where $1 \leq i \leq N(B_S)$ and $1 \leq j \leq N(B_T)$. Our objective, then, is to compute the distance $d(\mathcal{S}, \mathcal{T})$ between the bitmaps represented by $\mathcal{S}$ and $\mathcal{T}$ as $c(N(B_S), N(B_T))$, the minimum cost of transforming the source bitmap B_S into the target bitmap B_T. We will represent the function c in a tableau, whose values will be computed by means of a dynamic programming technique, as is standard in the string processing literature (Sankoff and Kruskal 1983, Crochemore and Rytter 1994).

The dynamic programming technique we employ assures that we satisfy the desired constraint that no two shift operations cross: i.e., if s_i is shifted to t_j and s'_i is shifted to t'_j, then the dynamic programming technique assures that if $s_i < s'_i$, then $t_j < t'_j$ (See next section).

Such a constraint is consistent with the applications for which we envision this measure; for example, if we are trying to match segment boundaries in text, we anticipate missing the correct boundary by a slight amount, but not actually interchanging boundaries. Consider, for example, the following configuration:

$$\begin{array}{lccc} & & \overset{a}{} & \overset{b}{} \\ B_S = & 0 & 1 & 1 \\ B_T = & 1 & 1 & 0 \end{array}$$

We would not like the algorithm to compute the matching error on the assumption that the corrections required shifting the 1-bit labeled b on B_S as follows:

$$B_S = 0 \quad \overset{a}{1} \quad \overset{b}{1} \rightarrow \overset{b}{1} \quad \overset{a}{1} \quad 0,$$

passing the 1-bit labeled by b *over* the 1-bit labeled a.

To define our edit distance, we use the following elementary operations, motivated by reference to a problem in which a source bitmap is created with the intention of matching a target bitmap:

- *Insertion*: Here the algorithm generating the source is considered to have completely missed the j-th 1-bit of the target, located on B_T at location t_j. To correct this oversight, a 1-bit is inserted into the source, B_S, at location t_j, incurring a cost $c_I > 0$. If this operator is applied in the course of a truly optimal sequence of operations taking $\langle s_1, \ldots, s_i \rangle$ to $\langle t_1, \ldots t_j \rangle$, then

  $$c(i, j) = c_I + c(i, j - 1),$$

  and the j-th target bit is considered disposed of. Note that an insertion is an option only when we are matching bit-sites s_i and t_j where $s_i < t_j$.

  Note that the possibility of this insertion does not preclude the possibility of a 1-bit already being present at this site in B_S, as might be the case if $s_i > t_j$. But accepting the possibility of an insertion *at this stage*, even in this case, greatly simplifies our evaluating this measure, since we are able to focus on only the "end" bits. Further, such a possibility is conceptually desirable, since we have to take into account the possibility that the bit in B_S that matches t_j has been placed there erroneously, and should itself be shifted to match a different bit in B_T. For example, we can return to the following configuration:

  $$\begin{array}{lccc} B_S = 0 & \overset{a}{1} & \overset{b}{1} \\ B_T = 1 & 1 & 0 \end{array}$$

  The algorithm might properly shift the 1-bit labeled b to the left, apparently over the 1-bit labeled a; of course, the 1-bit labeled a will itself be deleted or moved at another stage in the algorithm. Whether precisely this sequence of steps is realized is of course taken care of automatically in the course of the optimization procedure, to generate that sequence that produces the smallest cost.

  We also note that our use of the terms *insertion* and *deletion*, while natural in this context, is somewhat non-standard. An insertion changes the value of a bit from zero to one, while the lengths of the bitmaps are fixed—that is, only the number of 1-bits is changed. In the string-algorithm literature, an insertion usually refers to an operation that increases the size of the string. Similar considerations apply below, when we define deletions.
- *Deletion*: Here the i-th 1-bit of the source is assessed as spurious. The 1-bit is changed to a 0-bit, incurring a cost $c_D > 0$. If this is optimal, then

  $$c(i, j) = c_D + c(i - 1, j),$$

and the i-th source bit is considered disposed of. We consider a deletion only when $s_i > t_j$.

- *Shift*: Here the j-th 1-bit of the target and the i-th 1-bit of the source are considered to represent the same bit value, but misaligned by a small amount. Continuing with the example of trying to match segment boundaries, in this case the source generator correctly sensed the need for a 1-bit, but its exact location may have been in error. Now the source 1-bit is shifted Δ locations to align it with the target 1-bit. The cost incurred by this operation is given by $c_S(\Delta)$, a non-negative function, monotonically increasing with $|\Delta|$. If the match was accurate, then $\Delta = 0$, with $c_S(0) = 0$, will denote the null operation. If a shift operation is optimal, then, for $\Delta = t_j - s_i$,

$$c(i, j) = c_S(\Delta) + c(i - 1, j - 1),$$

and the i-th source bit and j-th target bit are considered disposed of. Most simply, we can assess $c_S(\Delta) = A|\Delta|$, for some non-negative constant A. We would want to adjust A relative to c_I and c_D so that for $|\Delta|$ "large," it is cheaper to *delete* and *insert* rather than shift, while the opposite is true for small $|\Delta|$. We will adopt this form for the shift cost below, though in other contexts a different form might be used. For example, if our algorithm is to be used for data compression, the cost would be made to conform with the number of bits required to encode the operation; this would be true also for the cost of insertion and deletion, which might in this context not be equal.

Although the shift operation is unconventional in the string processing literature, it is easily accommodated within the dynamic programming framework.

Computing edit distances typically requires processing whole strings. But, the special nature of our problem will allow us to focus on only the 1-bits of the bitmap, which reduces the computational cost of our algorithm. We are assuming here that the bitmaps we are processing have already been constructed, for example during indexing, and that they are stored, for efficiency, as pointers to the 1-bits. However, under certain circumstances the bit-sites may have to be constructed at run time, in which case the cost of linearly reading both bitmaps once must be added to the overall cost of the algorithm.

2.3. Implementation

To implement the dynamic programming method, we most conveniently work with the representations $\mathcal{S}$ and $\mathcal{T}$. We first initialize the tableau by inserting the following boundary values:

$$c(0, j) = j\, c_I \quad \text{and} \quad c(i, 0) = i\, c_D,$$

reflecting that we begin the process with the source bitmap $\mathcal{S}$, or target bitmap $\mathcal{T}$, being empty. The optimal costs are then developed recursively as indicated in the definition of the operations: starting with row 1, going from left to right, we compute the costs of the various possible operations, and select that with the smallest cost. When a cell in the tableau

is evaluated, the costs to its left and above have already been computed, so the optimal decision for that cell is possible. The optimal value is then stored in the cell (as well, if appropriate, as the operation applied to get that value), and the next cell evaluated. When the last cell, at the south-east corner of the tableau, has been evaluated, the task is completed.

Example. We once more consider the following configuration:

$$\begin{array}{lcc} & \overset{a}{} & \overset{b}{} \\ B_S = 0 & 1 & 1 \\ B_T = 1 & 1 & 0 \end{array}$$

So we are trying to convert $\mathcal{S} = \langle 2, 3 \rangle$ into $\mathcal{T} = \langle 1, 2 \rangle$. For this problem, we let $c_I = 20$, $c_D = 30$, and $c(\Delta) = A|\Delta|$, for $A = 10$.

We begin by initializing the tableau:

		$\mathcal{S}$	
	0	30	60
$\mathcal{T}$	20		
	40		

For example, the corner of the table (row 0, column 0) is assigned the value 0, since it gives the cost of changing a null source into a null target.

The other values on top margin (row 0) of the table give the costs of transforming a source vector to a null target vector, while the left margin (column 0), below the corner, gives the costs of transforming a null source vector to a target vector. For example, the value 30 in the top margin assesses the cost of changing the source bitmap `01`, i.e., $\mathcal{S} = \langle 2 \rangle$, into the null target vector. We can do this only by deleting the 1-bit in $\mathcal{S}$ at site 2, which incurs a cost of $c_D = 30$.

Similarly, the value 20 in row 1 of the left margin is the cost of transforming a null source vector to the target bitmap `1`. Since the only way to achieve this transformation is to insert a 1-bit into the source at position 1, the cost is $c_I = 20$. The other two values in the top margin (left margin) represent the costs of deleting 1-bits from the source bitmap (inserting 1-bits into the source bitmap).

To continue with our example, let us number the cells constituting the body of the table with integers in the order in which the tableau cells are filled by the algorithm:

		$\mathcal{S}$	
	0	20	40
$\mathcal{T}$	30	(1) $\rightarrow$	$\swarrow$(2)
	60	(3) $\rightarrow$	(4)

Thus, the algorithm first computes the values in row 1: first for cell 1, then cell 2; it then goes on to evaluate row 2: first cell 3, then 4. The value placed into cell 4 is the GHD value of the distance between the two bitmaps.

To compute the value for cell 1, the algorithm is presented the task of transforming $\mathcal{S}' = \langle 2 \rangle$ into $\mathcal{T}' = \langle 1 \rangle$. As always, the algorithm tries to reconcile, optimally, the rightmost bits of $\mathcal{S}'$ and $\mathcal{T}'$, taking into account values already computed. (Here, these 1-bits are the only 1-bits.) Since bit-site 2 is greater than the target's bit-site 1, we have to decide whether to delete the 1-bit at $s_1 = 2$ or to shift it one position to the left.

- If we delete the 1-bit, we incur a cost of $c_D = 30$, plus the cost of converting the empty source into $\mathcal{T}$, which has already been evaluated in the course of the initiation procedure as 20, for a total cost of $c(1, 1) = c_D + c(0, 1) = 50$.
- If we shift the 1-bit, we incur an incremental cost of 10 for the shift, plus the cost of transforming an empty source into an empty target, which has already been evaluated (see corner of tableau) as 0, for a total cost of $c(1, 1) = c_S(1) + c(0, 0) = 10$.

We select the minimum-cost option, and insert its value, 10, into cell 1 of the tableau (and, if useful, the fact that a shift was used).

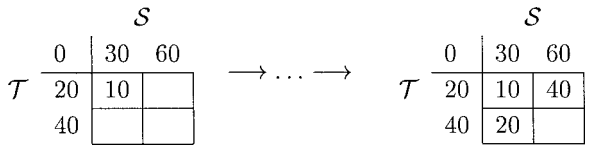
To compute the value in cell 2, $c(1, 2)$, we give the algorithm the task of transforming $\mathcal{S} = \langle 2, 3 \rangle$ into $\mathcal{T}' = \langle 1 \rangle$. Sicne $t_1 = 1 < s_2 = 3$, we consider either deletion or shift.

- The cost of deletion is $c_D + c(1, 1) = 30 + 10 = 40$.
- The cost of the shift is $c_S(\Delta) + c(0, 1) = A|3 - 1| + 30 = 20 + 30 = 50$.

We insert the minimum of these (40) into cell 2.

Having completed the first row, we are in a position to compute the second row. The value in cell 3, $c(2, 1)$, is the cost of transforming $\mathcal{S}' = \langle 2 \rangle$ into $\mathcal{T} = \langle 1, 2 \rangle$. Since $s_1 = 2 = t_2 = 2$, we can accept the cost (0) of a zero shift as optimal, and insert the value $c(1, 0) + 0 = 20$ into cell 3.

The transformation of the tableau described above leaves only the terminal cell, cell 4, empty:



At this point, our task is to transform $\mathcal{S} = \langle 2, 3 \rangle$ into $\mathcal{T} = \langle 1, 2 \rangle$. As usual, we first decide how best to dispose of the right-most bits. Since 3 is greater than 2, we have the option of deleting the 1-bit at 3 or shifting it to bit position 2.

- If we delete it, we incur a cost of $c_D = 30$ plus the cost of transforming $\langle 2 \rangle$ into $\langle 1, 2 \rangle$, already evaluated as 20, for a total cost of $c(2, 2) = c_D + c(2, 1) = 50$.
- If we shift it by one place, we incur a cost of 10 for the shift, plus the cost of transforming $\langle 2 \rangle$ into $\langle 1 \rangle$, already evaluated as 10, for a total cost of $c(2, 2) = c_S(1) + c(1, 1) = 20$.

We put the minimum of the two values, or 20, into the final cell:

		$\mathcal{S}$	
	0	30	60
$\mathcal{T}$	20	10	40
	40	20	20

Thus, we conclude that the GHD for this problem is 20.

Note that the size of the tableau is $N(\mathcal{S})N(\mathcal{T})$, rather than M^2, which may dramatically reduce the complexity of the computation. Further efficiencies are possible if the bitmaps are sparse or the maximal shift distance is sufficiently small.

Below, when we wish to reveal all parameters in the edit distance between two arbitrary source and target bitmaps of the same dimension, represented as above by $\mathcal{S}$ and $\mathcal{T}$, we adopt the notation $d(\mathcal{S}, \mathcal{T}; c_I, c_D, A)$.

3. Properties

GHD has some very interesting properties and relations to other functions. We now examine some that are particularly striking.

3.1. Crossing restraint

The character of the problems we deal with would make it awkward if a 1-bit in $\mathcal{S}$ had to be shifted so it exchanged relative positions with another source 1-bit. Fortunately, our dynamic programming algorithm automatically prevents this. This is because the algorithm, at any stage, considers only the rightmost bits of initial components of the two bitmaps. We now examine this in detail, implicitly arguing by induction.

Once the tableau is complete, we can take the two bitmaps being compared and, by backtracking, describe the disposition of each 1-bit in the source. We start with the rightmost bit and work backward through the source and target bitmaps. Suppose it has been shown that after disposing of a number of the terminating bits, no crossing has occurred, and examine the remaining source and target bitmaps at that stage. Certainly the terminal source bit, say s, can't be shifted to cross any of the source 1-bits to its right that have already been disposed of, for these have either been deleted, or shifted to target bits to the right of the rightmost target bit remaining in the current target. So we need ask only whether s might cross a 1-bit to its left.

If s is deleted, then the issue of crossing is meaningless. So suppose the terminal source bit, s, is shifted to the current terminal target bit. But then both are removed from any further consideration, and no opportunity exists for a source 1-bit to the left of s be moved to the right of s. Note that in moving s to its final site, s may very well pass over source 1-bits *currently* to its left. But, considering the details of the algorithm, we see that any 1-bits in the source passed over in this way will soon either be deleted, or shifted further to the left.

We conclude that at no stage can crossing occur.

3.2. *GHD is a true distance*

An immediately appealing quality of the traditional Hamming Distance is that it is indeed a true distance function: that is, it is a real valued function of two bitmaps that is (a) Symmetric; (b) Positive, unless the two bitmaps are equal, in which case it takes the value zero; and (c) Satisfies the triangle inequality. We begin by examining the conditions under which GHD can be shown to be a distance. To do this, it is convenient to define a function on the integers as *concave* if it satisfies the following condition: given integers r, s, t, u, with $r < s \leq t < u$, a function f defined on the integers is concave provided,

$$\frac{f(s) - f(r)}{s - r} \geq \frac{f(u) - f(t)}{u - t}.$$

Below, we denote the distance between bitmaps $\mathcal{S}$ and $\mathcal{T}$, as determined by GHD, by $d(\mathcal{S}, \mathcal{T})$. Then we claim:

Theorem. *$d(\mathcal{S}, \mathcal{T})$ is a true distance function, if, when $x \geq 0$ denotes the absolute size of a shift*:
(a) *c_S depends only on x, i.e., it is symmetric around zero*;
(b) *$c_S(x) = 0$, if $x = 0$, and $c_S(x) > 0$ otherwise*;
(c) *$c_S(x)$ increases monotonically*;
(d) *$c_S(x)$ is concave on the integers*; *and*
(e) *$c_D = c_I > 0$.*

Proof: For GHD to be a distance function, three conditions must be satisfied.

Positivity: *$d(\mathcal{S}, \mathcal{T}) > 0$ if $\mathcal{S} \neq \mathcal{T}$; $d(\mathcal{S}, \mathcal{T}) = 0$ if $\mathcal{S} = \mathcal{T}$.*

Clearly, if $\mathcal{S} = \mathcal{T}$, no operations (technically, only shifts of length zero) are required to transform one to the other, so for this case $d(\mathcal{S}, \mathcal{T}) = 0$. On the other hand, if the bitmaps are not identical, at least one non-trivial operation must be applied, incurring a positive cost. Thus, $d(\mathcal{S}, \mathcal{T}) \geq 0$, taking the value zero only for identical bitmaps.

Symmetry: *$d(\mathcal{S}, \mathcal{T}) = d(\mathcal{T}, \mathcal{S})$.*

For any sequence of operations, $o_1, o_2, \ldots, o_n$, taking $\mathcal{S}$ to $\mathcal{T}$, a complementary sequence of operations can be defined: $o'_n, o'_{n-1}, \ldots, o'_1$, where

$$o'_i = \begin{cases} \textit{delete} & \text{if } o_i = \textit{insert} \\ \textit{insert} & \text{if } o_i = \textit{delete} \\ \textit{shift}(-j) & \text{if } o_i = \textit{shift}(j). \end{cases}$$

Clearly, the complementary sequence systematically undoes the effect of the original sequence, so, when applied to the bitmap $\mathcal{T}$, it transforms it to $\mathcal{S}$. Under the conditions of the theorem, it incurs the identical cost. Since this is true as well for an optimal sequence, the symmetry condition is satisfied.

Triangle inequality: *For bitmaps* $\mathcal{S}, \mathcal{T}$, *and* $\mathcal{U}$, $d(\mathcal{S}, \mathcal{T}) \leq d(\mathcal{S}, \mathcal{U}) + d(\mathcal{U}, \mathcal{T})$.

Consider an optimal sequence of operations, τ, that transforms $\mathcal{S}$ to $\mathcal{T}$. Each operation either deletes an "extraneous" 1-bit of $\mathcal{S}$; inserts an "overlooked" 1-bit into $\mathcal{S}$ to match one in $\mathcal{T}$; or shifts a "misplaced" 1-bit in $\mathcal{S}$ to match a 1-bit in $\mathcal{T}$.

To prove the triangle inequality, the costs of operations that transform $\mathcal{S}$ to $\mathcal{T}$ via the intermediary $\mathcal{U}$ must be evaluated and compared to the cost of τ. So consider the sequence of operations, τ', first taking $\mathcal{S}$ to $\mathcal{U}$; then the sequence, τ'', taking $\mathcal{U}$ to $\mathcal{T}$.

Consider, then, a 1-bit in $\mathcal{S}$. It could be considered acted on in any of several ways. It can be:

(i) left alone in both τ' and τ'';
(ii) deleted in τ', and left alone by τ'';
(iii) left alone in τ' and deleted in τ'';
(iv) shifted in τ' and left alone in τ'';
(v) left alone in τ' and shifted in τ'';
(vi) deleted in τ', then reinserted in τ'';
(vii) shifted in τ' then deleted in τ''; or
(viii) shifted in both τ' and τ''.

An additional set of operations would be required to complete the description of how a 1-bit in $\mathcal{T}$ might matched, for example by insertions. Since they don't introduce any new possibilities, they won't be considered explicitly.

Pairs of operations similar to (i)–(v) are easily disposed of: in effect they directly transform a 1-bit in $\mathcal{S}$ to a 1-bit in $\mathcal{T}$, and could have been performed as a single operation in τ, and thus must have been considered in the process of constructing τ, and incorporated or rejected. Thus, since τ is optimal, these operations, collectively can only increase the cost or leave it unchanged; as such they are compatible with the triangle inequality.

Any sequence of two operations that cancel, as in (vi), or a single shift combined with a non-null operation, as in (vii), can only increase the cost relative to the disposition of the bit in τ, again in accordance with the triangle inequality. By systematically examining all possibilities in this manner, it is straightforward to conclude that the only way the triangle inequality can break down is if the concatenated sequence $\tau'\tau''$ shifts a bit twice: in effect shifting the bit from its initial location in $\mathcal{S}$ to its final destination in $\mathcal{T}$ by means of two shifts (a disposition illegal when directly transforming $\mathcal{S}$ to $\mathcal{T}$).

Thus, to prove the triangle inequality we need to examine only the case when two successive shifts, say of (positive) lengths A and B, are encountered. Consider two cases: (a) both shifts are to the same direction, resulting in the combined shift length $A + B$ and (b) one shift is followed by a shift in the reverse direction for the second, resulting in the combined shift length $|A - B|$. Recalling that c_S depends only on the absolute value of its argument, we observe that these conditions imply that for the triangle inequality to be valid, the following inequalities must hold:

$$c_S(A) + c_S(B) \geq c_S(A + B)$$
$$c_S(A) + c_S(B) \geq c_S(A - B).$$

Since $c_S(A - B) = c_S(B - A)$, we can assume, without loss of generality, that $A > B$. Thus we proceed assuming that $A, B \geq 0$ and $A \geq B$.

We first note that the second inequality is a direct consequence of the monotonicity property, since $c_S(A)$ alone is greater than $c_S(A - B)$, and gives nothing new. So we need focus only on the consequences of the first inequality.

First note that the triangle inequality is trivially valid if one term, say B, is zero. So consider two integers, $A \geq B > 0$. Then $0 < B \leq A < A + B$. If $c_S(.)$ is concave on the integers, then, by the definition of concavity given above,

$$\frac{c_S(B)}{B} = \frac{c_S(B) - c_S(0)}{B} \geq \frac{c_S(A + B) - c_S(A)}{B}$$

Thus, $c_S(A + B) \leq c_S(A) + c_S(B)$. That is, the concavity property is sufficient to assure the triangle inequality, as was to be proved. □

The class of integer concave functions is quite broad. In particular, if $c_S(.)$ is linear, or the integer restriction of a traditionally concave function defined over the reals, then we can assume that our measure is a distance, and benefit from the intuition that this provides. However, we do wish to observe that under some conditions, it may be necessary to abandon the condition that the GHD is a true distance function. For example, there can be situations where inserting a bit is more expensive than deleting it. Some image processing operations allow for pixel insertions but disallow pixel deletions. Also, if we are computing the distance in order to compress a bitmap, the costs of insertion and deletion are the number of bits required to encode the operation, and these may not be the same for an optimal encoding.

3.3. Optimality

We first argue that the procedure described above does produce an optimal result, among alternatives subject to the constraint that 1-bits in B_S not cross. The logic is simple: given two bitmaps, there are only three operations that are permitted in disposing of one or both of the rightmost bits. Each such operation reduces the bitmap(s) to a simpler pair. If we know the minimum cost for each of the simpler pairs, then we can consider in turn each permitted disposition of the rightmost bits and add its cost to the optimal cost of the resulting simpler pair. If we choose the minimum of these costs, then we have made the optimal decision for the initial pair of bitmaps. We now argue that, if we fill the tableaux as described above, each cell contains values consistent with this logic.

Consider, then, in more detail the various steps in the algorithm, assuming i 1-bits for B_T and j for B_S:

- Initialization: The margins (i and/or j being 0) involve changing a bitmap to an empty bitmap, or vv. Here we have no choice. For example, if B_S has i bits and B_T is empty, our only option is to delete the i bits in B_S, incurring a cost of ic_D.
- $s_j = t_i$: Here the rightmost bits match. We can with no cost pass on to the $i - 1$, $j - 1$ case (with a shift of zero bits). This value already appears in the table as $c(i - 1, j - 1)$,

and is assumed to be the optimal value for the reduced problem. Thus we insert into the tableau the value $c(i, j) = c(i-1, j-1)$.

- $s_j > t_i$: There are three options, which we consider in turn.
  (a) If we delete the bit at s_j, we incur the cost c_D, to which we add the known, and presumed optimal, cost of $c(i, j-1)$ for transforming the remaining source bitmap into the target, for a total cost of $c_D + c(i, j-1)$.
  (b) An insertion in B_S at t_i is precluded. For ultimately, the rightmost bit in B_S must be dealt with. If it will sooner or later be deleted, we can with no extra cost do that now. On the other hand, if the terminal bit in B_S will eventually be shifted to match the terminal bit in B_T, then again an insertion is not necessary. Nor can an insertion occur, eventually followed by the terminal bit in B_S being shifted to more leftmost bit in B_T, since this would violate our restraint that no bits cross.
  (c) This leaves a shift as the only remaining possibility. If we shift the source bit to the target bit, we incur the cost $c_S(t_i - s_j)$, which is added to the value $c(i-1, j-1)$, available and presumed optimal; in this case $c(i, j) = c_S(t_i - s_j) + c(i-1, j-1)$. We cannot shift this bit to a bit to the left of t_i, since the target bit ultimately must be matched, and whether this be by an insertion or shifting an earlier bit, it would violate the restriction that no 1-bit in B_S cross another.

  Thus, if, as assumed, the cells evaluated earlier are optimal, then the minimum of the two costs considered above will be again optimal, and $c(i, j) = \min(c_D + c(i, j-1), c_S(t_i - s_j) + c(i-1, j-1)$.
- $s_j < t_i$: By an argument parallel to that just presented, we conclude that the optimal value for $c(i, j)$ is given by $c(i, j) = \min(c_I + c(i-1, j), c_S(t_i - s_j) + c(i-1, j-1)$.

3.4. Relation to other measures

The edit distance we defined above is very general, and it is interesting to relate it to other distance-measures between bitmaps.

3.4.1. Hamming distance. An obvious measure for the distance between two bitmaps is the simple Hamming distance: the minimum number of bits that must be changed so the source and target agree. The Hamming distance between the bitmaps represented by $\mathcal{S}$ and $\mathcal{T}$ may be expressed as a special case of the edit distance: $d(\mathcal{S}, \mathcal{T}; 1, 1, \infty)$. That is, our measure is a genuine generalization of the traditional Hamming distance.

3.4.2. Recall/precision type measures. In IR, one customarily uses two measures to evaluate performance. *Recall* indicates the fraction of all relevant items, e.g., documents or images, that appear in a retrieval set, while *precision* indicates the fraction of items in a retrieval set that are relevant. These measures can be adapted to evaluating the distance between a source and target bitmap (Hearst 1994). To do this, we define two functions:

$$p(\mathcal{S}, \mathcal{T}) = \frac{N(\mathcal{S} \texttt{ AND } \mathcal{T})}{N(\mathcal{S})}, \quad r(\mathcal{S}, \mathcal{T}) = \frac{N(\mathcal{S} \texttt{ AND } \mathcal{T})}{N(\mathcal{T})},$$

where the AND operator acts on the bitmaps indicated by $\mathcal{S}$ and $\mathcal{T}$.

Thus $p(\mathcal{S}, \mathcal{T})$ is the fraction of the 1-bits of the bitmap represented by $\mathcal{S}$ to be evaluated which indeed match the 1-bits of the target bitmap represented by $\mathcal{T}$, e.g., the percentage of segment boundaries produced by our algorithm, which are "correct" in the sense that they are also boundaries in the given reference partition; similarly $r(\mathcal{S}, \mathcal{T})$ is the fraction of the 1-bits of the target bitmap that have corresponding 1-bits in the source bitmap (e.g. the percentage of segment boundaries of the reference partition that are detected by our algorithm).

We thus assign, after having fixed $\mathcal{T}$, a pair of numbers (r, p) to each partition. Relative to $\mathcal{T}$, we consider a partition represented by $\mathcal{S}_1$ to be better than another partition, represented by $\mathcal{S}_2$, if

$$p(\mathcal{S}_1, \mathcal{T}) \geq p(\mathcal{S}_2, \mathcal{T}) \quad \text{and} \quad r(\mathcal{S}_1, \mathcal{T}) \geq r(\mathcal{S}_2, \mathcal{T}).$$

Generally, however, the values r and p tend to be inversely related, and trying to raise one usually lowers the other. By changing the parameters in a segmentation algorithm, one could get a series of (r, p)-pairs, and produce curves similar to the recall/precision curves in IR.

While these measures are interesting because of their relation to the tradition of research in IR, they can be simply expressed in terms of our edit distance. Letting $\mathbf{0}$ denote the zero-bitmap and recalling the notation $d(\mathcal{S}, \mathcal{T}; c_I, c_D, A)$, we find

$$p(\mathcal{S}, \mathcal{T}) = \frac{d(\mathcal{S}, \mathbf{0}; 0, 1, \infty) - d(\mathcal{S}, \mathcal{T}; 0, 1, \infty)}{d(\mathcal{S}, \mathbf{0}; 0, 1, \infty)}$$

$$r(\mathcal{S}, \mathcal{T}) = \frac{d(\mathcal{S}, \mathbf{0}; 0, 1, \infty) - d(\mathcal{S}, \mathcal{T}; 0, 1, \infty)}{d(\mathcal{T}, \mathbf{0}; 0, 1, \infty)}.$$

Since recall and precision can be expressed in terms of imprecise costs, this representation immediately suggests an interesting generalization. By relaxing the infinity value in the cost, we can define new versions of the classic recall and precision measures as used in this context.

4. Experiments

Many areas of modern computer science are in constant need of robust similarity metrics. One such area is image retrieval. To assess the role GHD might play in image processing, we conducted a number of experiments which required a robot to use GHD to identify target objects in images captured by the robot. The use of robots allowed us to objectively evaluate the value of our measure for the task of finding objects in images, as well as introduce GHD for use in robot vision more generally, another area in which GHD could be of interest.

Many problems in robotics depend on reliable object recognition (Murphy 2000). To be practical, object recognition must use metrics that are robust in the presence of sensor noise and account not only for exact matches but also for approximate ones. A key component of image retrieval research is the automatic generation of keyword indexes for individual

images in image collections (Tam and Leung 2001). Many keyword index generation techniques used in image retrieval rely on accurate object recognition. Images can, for example, be indexed in terms of objects found in them and, if feasible and necessary, relationships among found objects. In this section, we present object recognition experiments with GHD. Our objective in performing these experiments was to show that GHD can be used to create robust object recognition similarity metrics.

Most current object recognition metrics are either model-based or sample-based. Model-based metrics, such as correlation (Parker 1993), template matching (Parker 1993), and color histogram matching (Swain and Ballard 1991, Chang and Krumm 1999), rely on model libraries. Sample-based metrics, such as Bayesian Networks (BN) (Boykov and Huttenlocher 1999) and Artificial Neural Networks (ANN) (Young et al. 1994), require large pre-classified samples for training.

We used GHD to define a model-based object recognition metric: that is, we created a library of images for objects that must be recognized, and used GHD and other measures to attempt to match a new image with those in the library. GHD complements the other metrics insomuch as it offers a principled way to specify the degree of approximation that can be tolerated in matches. The metric is implemented and integrated with the control system of a mobile robot. In our test, the robot patrols an office area and looks for soda cans, coffee cups, and crumpled pieces of paper. When an object is recognized in the image, the robot must pick it up and place it in an area of the floor designated for that type of object. Thus, it is important that the robot be able to distinguish each type of object by matching the images it creates as it moves around the room with those in the library.

4.1. Object models

Objects are represented by model libraries created from images taken from different distances and with different camera tilts. The camera used is the Pioneer Pan-Tilt-Zoom (PTZ) Robotic Camera mounted on a Pioneer 2DX mobile robot manufactured by ActivMedia, Inc. (www.activmedia.com). The video capture card used in the experiments was Winnov Video VO PCI (www.winnov.com). Figure 1 depicts the robot used in the experiments. The camera has a horizontal angle of view of 48.8 degrees and a vertical angle of view of 37.6 degrees. The images are saved as 120 by 160 color bitmaps. The origin of the image coordinate system is in the bottom left corner of the image. Distances are measured in meters

Figure 1. Pioneer 2DX robot.

from the robot's gripper to the object. The models were taken from the following distances: 0.5, 1.0, 1.5, and 2 meters. The camera tilt for the distances 1.0, 1.5, and 2 was 0 degrees. The camera tilts for 0.5 were 0 and -10 degrees. The negative tilt, which indicates that the camera is tilted 10 degrees downward, was chosen for the distance 0.5, because, as the robot approaches an object, the pickup *skill* tilts the camera in order to ensure that the object is still present or to correctly identify the object. (In the robotics literature, a *skill* is a program that runs in response to a command given to the robot either by a human observer or by another program. For example, the observer may ask the robot to fetch a pepsican, which will trigger the activation of the "pickup skill.") Each object is represented in the library by two types of representations, which we will refer to below as *models*: Hamming distance models (HDMs) and color histogram models (CHMs) (Swain and Ballard 1991, Chang and Krumm 1999).

4.1.1. Hamming distance models. To create an HDM, an object's image is taken from a given distance and with a given camera tilt. We now wish to convert this image into a binary one. We first process the image with the "gradient edge detection mask" (Parker 1993). This process transforms the image in a manner that helps us to detect edges, which are characterized by sharp changes in intensities. Once we have identified those changes, we can turn the image into a bit array. This is done by transforming intensity values to 1's and 0's. If the intensity value is above a threshold, the edge is there so the corresponding pixel becomes 1; otherwise the value 0 is assigned. The object's model is the smallest region of this bit array containing the object. Thus, we finally produce HDMs, which are 2D bit arrays. Figure 2 contains part of an image containing a soda can. Figure 3 contains the image from figure 2 after the application of the gradient edge detection mask. The white lines are the edges detected by the mask. Figure 4 contains the bit array obtained from the image from figure 3. In summary, an HDM model is obtained from the 2D bit array computed from

Figure 2. A soda can.

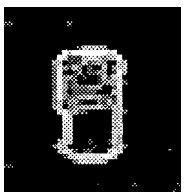


Figure 3. Edges of the soda can.

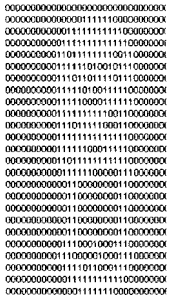


Figure 4. Bit array of the soda can.

the manually cropped region containing the object. In this case, the region is a rectangle containing all of the bits of the soda can.

An object has five HDMs: two models for the distance 0.5, one with tilt 0 and one with tilt -10, and three models with tilt 0 for the remaining distances. In addition, we add two parameters to each model: these are row constraints that indicate approximately where in the robot's image the bottom row of the model would be found. For example, the row constraints of the two meter pepsican model are 40 and 45. These row constraints were added to the models to reflect the camera's calibration, and to speed up the image processing. For example, when the robot is looking for a pepsican approximately 2 meters away from it on the floor, the base of the object will be between rows 40 and 45. Thus, the appropriate models can be restricted to that region of the image. It was assumed that all objects are on the floor.

4.1.2. Color histogram models. CHMs are models of objects that are based on the distribution of colors. They are created for the same distances and tilts as the HDMs. To create a CHM, an object's image is taken, as before, from a given distance and with a given camera tilt. The smallest region of the image containing the object is manually cropped. Three color histograms are created from the cropped region. The first histogram records the relative frequencies of red intensities over the rows comprising the cropped image. The other two do the same for blue and green.

Specifically, the image is made up of pixels, each pixel giving the intensity values for red, green and blue. To create a histogram for a color, we divide the full intensity range, [0, 255], into sixteen non-overlapping subranges, e.g., [0, 15], [16, 31], etc. We proceed by computing the percentages of pixels whose intensities for the given color fall into each subrange. Since this computation is done for red, green, and blue, each CHM consists of three color histograms, each with sixteen values. The same row constraints as for the HDMs hold.

4.2. Object recognition

4.2.1. Hamming distance models. To recognize objects in images using HDM models, we convert images into 2D bit arrays. Formally, let $M_H(O, D, T, L, U)$ denote an HDM

for object O at distance D and with tilt T and two row constraints L and U for the lower and upper rows, respectively. For example, $M_H(pepsican, 2, 0, 40, 45)$ is the model of a pepsican two meters from the robot and with tilt 0 and row constraints 40 and 45.

Let H and W be the model's height and width. Suppose we are given an image I, and are given the task of deciding whether an object O can be found in I. We first process I with the gradient edge detection mask. To recognize O in I, the model $M_H(O, D, T, L, U)$ is matched with the region of I whose bottom row is L and whose top row is $U + H$. For example, if the height of $M_H(pepsican, 2, 0, 40, 45)$ is 13 rows, the matching happens between rows 40 and 58 $(45 + 13)$. The matching is done by moving the model from left to right C columns at a time and from top to bottom R rows at a time. For this reason, R and C are referred to as *model shift parameters*. After each shift, a tentative similarity coefficient is computed for each region of the image that overlaps with the model. We then take the region with the highest similarity coefficient.

We now indicate how each candidate similarity coefficient is computed. To compute a similarity coefficient between the model and an image region covered by the model, we take the sum of distances computed between the bit strings comprising the corresponding rows of the model and the image region, normalized by the model's size. In this paper, the distance measures used will the classic Hamming distance, and the generalization, GHD, introduced here; we also test two other traditional metrics. Formally, let S be the image region covered by $M_H(O, D, T, L, U)$. The similarity between M_H and S, $sim(M_H, S)$, is given by $sim(M_H, S) = K \times \sum_{R=0}^{H-1} ghd(M_R, S_R; c_I, c_D, c_S)$, where M_R and S_R are the two corresponding bit strings, i.e., rows, from M_H and S, respectively, and K is the normalization coefficient.

Consider figures 5 and 6. The former contains the top of HDM model for a pepsican. The latter contains a region of a 2D array obtained from an image that contains another pepsican. We now overlap the model with various regions of the image. For each overlap, we compute the similarity between the model top and the region, as described above, by summing the measured distances between the corresponding horizontal bit strings. The sum thus obtained is normalized by multiplying by the reciprocal of the product of the model's height and width. By giving a "per pixel" value, we are able to relate models of different

```
0000111111010000
0001101111101100
1011110111110100
0110111111100110
0111111010110111
```

Figure 5. Bit model.

```
0000001111110000
0001111111111000
0011111111111100
0110111111100110
0111111010010110
```

Figure 6. Bit image.

size. This similarity coefficient represents our confidence that the model's object is in the region covered by the model. We then shift and obtain another similarity coefficient for another row and column pair.

We label each of these similarity coefficients with the row and column of the bottom left corner of the image region covered by the model. The result is a set of 4-tuples $\langle x, y, m, s\rangle$ where x and y are the coordinates of the bottom left corner of the image region that matches the model m with the similarity score s. We sort by the value of s and take the top N, where N can be between 1 and 5, for further evaluation as described below.

4.2.2. Color histogram models. CHMs are matched as follows. Let $M_C(O, D, T, L, U)$ be the $H \times W$ CHM of object O at distance D and with tilt T and two row constraints L and U. Let I be an image. To recognize O in I with $M_C(O, D, T, L, U)$, the $H \times W$ mask is matched with the appropriate region of the image R. The similarity between M_C and R is the weighted sum of the similarities between their corresponding color histograms: we define $h(H_1, H_2)$, the similarity between two color histograms H_1 and H_2, as the sum of the absolute differences of their relative frequencies; thus $0 \leq h(H_1, H_2) \leq Q$, where Q is a suitably chosen constant.

Let $RH(M_C)$ denote the model's red histogram, $BH(M_C)$ denote its blue histogram, and $GH(M_C)$ denote its green histogram. Let $RH(R)$ be the red intensity histogram of R, and let $BH(R)$ and $GH(R)$ be the blue and green intensity histograms, respectively, of the portion of the image being matched. Then $sim(M_C, R) = 1/Q[A \times h(RH(M_C), RH(R)) + B \times h(BH(M_C), BH(R)) + C \times h(GH(M_C), GH(R))]$, where A, B, and C are the weights assessing the relative importance of each color that sum up to 1. In the experiments, $A = .34$, $B = C = .33$. Thus, $0 \leq sim(M_C, R) \leq 1$. This matching metric is different from the standard intersection metric used in color histogram matching (Swain and Ballard 1991), because it provides a way to control the relative importance of different colors.

4.3. Evaluation

We evaluated four metrics on 425 images of various objects taken by the robot's camera from different distances, under different lighting conditions, and with different occlusions. The following metrics, were evaluated: GHD, CHD, the color histogram metric, and normalized correlation (Parker 1993). The normalized correlation similarity between a bit model and the region of a bit image covered by the model is the sum of the products of the corresponding bits, normalized by the size of the model. Normalized correlation serves as a typical basis of comparison in many object recognition experiments.

We used the following parameters to compute the GHD: $\mathrm{ghd}(S, T; c_I = 1, c_D = 1, A = 0.5)$. The cost of shift was computed as $c_S(\Delta) = A|\Delta|$, with Δ the distance between a target bit and a source bit being shifted. The small value of A was chosen, because quite a few images taken by the robot contain multiple objects. Therefore, the degree of approximation was minimized to avoid false positives as much as possible. When an image contains a single object, a correct model is likely to give a large matching score on the sub-image containing the object even if the corresponding bits are misaligned to a significant degree. On the other hand, in images with multiple objects, the value of A should be small, because

large values may lead to numerous false positives. The model shift parameters C and R were set to 1. The threshold for all metrics was .6.

The main question our experiments was to answer was how likely it is that an object is correctly recognized in an image that contains it. The answer to this question estimates how well a metric detects objects when they are present in images. Let $1 \leq N \leq 5$. Let $REC(O, D, T, N)$ be the event when the metric recognizes O at D and with T within the first N matches. Let $IC(O, D, T)$ be the event that a captured image actually contains O at D and with T. The answer to the question above can be formulated as $P_{rec} = \Pr\{\text{REC}(O, D, T, N) \mid \text{IC}(O, D, T)\}$, the probability that an object model recognize an object, given that an image contains the object the model is to recognize. Different metrics can be compared in terms of how close they come to the ideal that $P_{rec} = 1$. In the context of image retrieval, this evaluation procedure can be interpreted as follows. If a metric correctly recognizes an object in a image, the image is retrieved. Otherwise, the image is not retrieved.

We estimated P_{rec} as the ratio L_1/L_2, where L_1 is the number of images that contain O at D and with T and where the metric correctly recognizes O within the first N matches; L_2 is the number of images that contain O at D and with T.

The justification of this estimation method is twofold. First, in the context of information retrieval, a growing body of experimental evidence (Jansen et al. 1998) suggests that most users (80–90%) examine only the top 2–4% of retrievals. Consequently, little benefit is gained from retrieving all relevant items, if none of them makes it to the small group at the top. Second, in the context of robot vision, a robot must make real time decisions on the basis of object recognition. Inspecting a small number of top matches does not result in performance deterioration. On the other hand, if the number of matches becomes large, the robot loses the ability to respond to its environment in real time.

Figures 7–10 contain experimental data. Each cell contains a value of P_{rec}. In each table, columns 1 to 4 present data for $N = 1$, and columns 5 to 8 present data for $N = 5$. As

dist and tilt	Pepsi	RootBeer	Coffee	Paper	Pepsi	RootBeer	Coffee	Paper
0.5(−10)	1	1	.91	.18	1	1	1	.21
0.5(0)	.48	.75	.91	.18	.58	.81	1	.23
1.0(0)	.51	.61	.34	.15	.77	.77	.69	.21
1.5(0)	.82	.85	.57	.325	.96	.96	.75	.83
2.0(0)	.71	.75	.61	.67	.81	.82	.87	1

Figure 7. Generalized Hamming distance. $N = 1$ and $N = 5$.

dist and tilt	Pepsi	RootBeer	Coffee	Paper	Pepsi	RootBeer	Coffee	Paper
0.5(−10)	.83	.79	.81	0	.85	.8	.82	0
0.5(0)	.38	.46	.8	0	.38	.46	.83	0
1.0(0)	.41	.12	.15	0	.77	.65	.69	0
1.5(0)	.82	.86	.42	.12	.96	.91	.75	.74
2.0(0)	.5	.64	.25	.22	.5	.64	.38	.71

Figure 8. Crisp Hamming distance. $N = 1$ and $N = 5$.

dist and tilt	Pepsi	RootBeer	Coffee	Paper	Pepsi	RootBeer	Coffee	Paper
0.5(−10)	1	.91	1	.91	1	.91	1	.91
0.5(0)	.86	.8	1	.91	.86	.8	1	.91
1.0(0)	.4	0	0	0	.4	0	0	0
1.5(0)	0	0	0	.43	0	0	0	.43
2.0(0)	0	0	0	0	0	0	0	.33

Figure 9. Color histogram matching. $N = 1$ and $N = 5$.

dist and tilt	Pepsi	RootBeer	Coffee	Paper	Pepsi	RootBeer	Coffee	Paper
0.5(−10)	.33	.33	0	0	.37	.33	0	0
0.5(0)	.17	.17	0	0	.25	.33	0	.33
1.0(0)	.41	.12	.1	0	.71	.47	.31	0
1.5(0)	.61	.66	.25	.43	.71	.71	.58	.22
2.0(0)	.71	.67	.25	.21	.75	.71	.25	.22

Figure 10. Normalized correlation. $N = 1$ and $N = 5$.

can be seen from the table, the P_{rec} values for GHD are best overall, while those for the normalized correlation are worst.

CHD does well on pepsicans and root beer cans, but performs worse on coffee cups and crumpled pieces of paper. GHD has the same tendency but shows improvement over CHD, because, unlike the CHD or normalized correlation, GHD can handle misaligned objects and models. GHD performs better on images with multiple objects where the object contours are not well defined due to occlusions.

GHD and CHD do equally well on images with single objects where object contours are well delineated. The color histogram metric does well on objects that are close to the robot. However, its performance degrades as the objects get further away from the robot, especially when the lighting conditions change. GHD appears to be more robust than the color histogram metric in the presence of occlusions and changes in lighting. The color histogram metric did not recognize a number of objects in images where the lighting conditions were different from the lighting conditions of the image from which the models were taken.

The experiments suggest that GHD can be used as a basis for object recognition metrics. GHD appears to perform at least as well as or slightly better than several model-based and sample-based metrics discussed in the object recognition literature. For example, Young et al. (1994) present an approach to object recognition based on a multi-layer Hopfield neural network structured as a cascade of several single layer Hopfield networks with links between adjacent layers. The network is evaluated on a set of 51 images of door keys. The success rate on images with single objects is 82 percent and on images with occluded objects is 31 percent. Although Young et al. (1994) do not offer any distance or lighting information, both numbers are, on average, below the recognition rates achieved by GHD.

Boykov and Huttenlocher (1999) present a Bayesian approach to object recognition that explicitly accounts for dependencies between features of the object. The approach is evaluated with Monte Carlo techniques to estimate Receiver Operating Characteristic (ROC) curves that plot the probability of detection along the y-axis and the probability of false alarms along the x-axis. The recognition rates of GHD are no worse than the rates reported by Boykov and Huttenlocher. One advantage of GHD is that it does not require the expensive computation of the a priori probabilities. It is difficult to make further content-based comparisons, because Boykov and Huttenlocher use synthetically generated images of very simple objects, e.g., single contour airplanes, in their experiments.

Chang and Krumm (1999) extend the normal color histogram by adding geometric information to it and obtaining the color co-occurrence histogram. The color co-occurrence histogram keeps track of the number of pairs of colored pixels that occur at certain separation distances. The recognition rates of GHD are as good as than the recognition rates of the color co-occurrence histograms. One advantage of GHDs is that their model libraries are significantly smaller and less complex than the model libraries needed for the color co-occurrence histograms. As with Boykov and Huttenlocher's approach, further content-based comparisons are hard to make, because Chang and Krumm use cartoon images for evaluation.

5. Conclusion

The Hamming distance was developed to measure the similarity of two bitstrings used for coding. It is tempting to adopt this measure for a broader range of contexts in which bitmaps are used. A problem is that the traditional Hamming distance may be more sensitive to slight deviations than is appropriate for many potential applications. Our intention in this paper was to extend the traditional Hamming distance to be sensitive to situations in which the concept of neighboring locations is important. This extension, called Generalized Hamming distance, was defined in terms of the operations necessary to transform a source bitmap to a target bitmap. One important motivation for developing GHD is that we believe it to be more reliable than the crisp Hamming measure when the exact placement of 1-bits is influenced by random effects. This is especially true when we deal with noisy channels, as diverse as robot cameras and imperfect text segmentation algorithms.

We derived conditions sufficient for this measure to be a true distance, as is the traditional Hamming distance: in this sense it is a true generalization. We showed that the algorithm we presented for computing GHD does produce optimal results. We also showed how standard recall and precision can be expressed in terms of GHD.

To test the insensitivity of the GHD to noise, we introduced it into a image recognition process, implemented by means of standard robot vision techniques. To do this, we developed a GHD-based object recognition metric and evaluated it in a series of object recognition experiments on a mobile robot. We compared our results with that using the traditional Hamming distance, as well as two metrics commonly use in the robotics literature. Our experiments suggest that GHD can be used as a basis for object recognition similarity metrics. We envision many other uses of GHD-based similarity metrics in information retrieval and related fields.

References

Adriaans P and Zantinge D (1996) Data Mining. Addison Wesley Longman Ltd, Harlow, England.

Boykov Y and Huttenlocher D (1999) A new Bayesian framework for object recognition. In: proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society.

Bookstein A and Klein ST (1990) Information retrieval tools for literary analysis. In: Tjoa AM, Ed., Database and Expert Systems Applications. Springer-Verlag, Vienna, pp. 1–7.

Bookstein A and Klein ST (1991) Compression of correlated bit-vectors. Information Systems, 16:387–400.

Bookstein A, Klein ST and Raita T (1998) Clumping properties of content-bearing words. Journal of the American Society for Information Science, 49:102–114.

Bookstein A, Klein ST and Raita T (2001) Fuzzy hamming distance: A new dissimilarity measure. In: Amir, Amihood and Landau, G Eds., Proceedings: 12th Annual Symposium on Combinatorial Pattern Matching (CPM2001). Jerusalem, Israel, 1–4 July, 2001.

Chang P and Krumm J (1999) Object recognition with color cooccurrence histograms. In: proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society.

Cormen TH, Leiserson CE and Rivest RL (1990) Introduction to Algorithms. MIT Press, Cambridge, MA.

Crochemore M and Rytter W (1994) Text Algorithms. Oxford University Press, New York.

Doyle L (1961) Semantic road maps for literature searchers. Journal of the ACM, 8(4):553–578.

Hamming RW (1980) Coding and Information Theory. Prentice-Hall, Englewood Cliffs, NJ.

Hearst MA (1994) Multi-paragraph segmentation of expository text. In: proc. ACL Conf. Las Cruces.

Hearst MA and Plaunt C (1993) Subtopic structuring for full-length document access. In: proc. 16-th ACM-SIGIR Conf. Pittsburgh, pp. 59–68.

Jansen BJ, Spink A, Bateman J and Saracevic T (1998) Real life information retrieval: A study of user queries on the web. SIGIR Forum, 32(1):5–18.

Knuth DE (1973) The Art of Computer Programming, Vol I, Fundamental Algorithms, Addison-Wesley, Reading, MA.

Kulyukin V and Burke A (in press) Mining free text for structure. In: wang J, Ed., Data Mining: Opportunities and Challenges, Idea Group Publishing, Pensylvania.

Kulyukin V and Morley N (2002) Integrated object recognition in the three-tiered robot architecture, In: Proc. of the International Conference on Artificial Intelligence, Las Vegas.

Kulyukin V and Zoko A (2000) Hamming distance for object recognition by mobile robots. In: Proceedings of the Research Symposium of the DePaul University School of Computer Science, Telecommunications and Information Systems (CTIRS-2000). DePaul University.

Murphy R (2000) AI Robotics. MIT Press, New York.

Parker, JR (1993) Practical Computer Vision Using C. John Wiley and Sons, New York.

Roman S (1992) Coding and Information Theory. Springer-Verlag, New York.

Sankoff D and Kruskal JB (1983) Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison. Addison-Wesley, Reading, MA.

Swain MJ and Ballard DH (1991) Color Indexing. International Journal of Computer Vision, 7:11–32.

Tam AM and Leung CHC (2001) Structured natural-language descriptions for semantic content retrieval of visual materials. Journal of the American Society for Information Science and Technology, 52(11):930–937.

Witten IH, Moffat A and Bell TC (1999) Managing Gigbytes. Morgan Kaufmann Publishers, San Fransisco.

Young SS, Scott PD and Nasrabadi N (1994) Multi-layer hopfield neural network. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recogntion. IEEE Computer Society.