A Dual Bound Algorithm for Very Fast and Exact Template-Matching

Journal:	*Transactions on Pattern Analysis and Machine Intelligence*
Manuscript ID:	TPAMI-2008-12-0846.R1
Manuscript Type:	Regular
Keywords:	I.5.4.c Computational models of vision < I.5.4 Applications < I.5 Pattern Recognition < I Computing Methodologies, I.4.8.e Object recognition < I.4.8 Scene Analysis < I.4 Image Processing and Computer Vision < I Computing Methodologies, I.4.7.d Projections < I.4.7 Feature Measurement < I.4 Image Processing and Computer Vision < I Computing Methodologies

A Dual Bound Algorithm for Very Fast and Exact Template-Matching

Haim Schweitzer, Robert Finis Anderson, and Rui (April) Deng

Abstract—Recently proposed fast template matching techniques employ rejection schemes derived from lower bounds on the match measure. This paper generalizes that idea and shows that in addition to lower bounds, upper-bounds on the match measure can be used to accelerate the search. An algorithm is proposed that utilizes both lower and upper bounds to detect the k best matches in an image. The performance of this dual-bound algorithm is guaranteed; it always detects the k best matches. Theoretical analysis and experimental results show that its run time compares favorably with previously proposed real-time exact template-matching schemes.

Index Terms—Template-matching, Walsh-Transform, Pattern-Matching, Real-time matching, Priority-queues

1 Introduction

Research aimed at developing fast template matching techniques can be traced back to early work in pattern recognition [6]. For example, Goshtasby [9] performs a first stage where matching is computed for a small subset of the template pixels. Exact values are computed in a second stage, but only at image locations that rank high in the first stage. A related approach [18] uses a multi-stage coarse-to-fine search.

Much of the recent work on fast matching is based on the observation that many unlikely candidates can be pruned with little computation. This can be done in a probabilistic sense, rejecting candidates that are most likely not a good match [16], or in an exact sense, utilizing bounds on the match measure [2], [7], [8], [10], [12], [13], [14], [23]. The computational savings come from efficient evaluation of these bounds by various techniques. Examples include sampling [16], multires-olution [8], running sums or integral-images [14], [20], [23], and orthogonal transforms [2], [10], [23].

1.1 The problem being addressed

In a typical template-matching setting one attempts to locate a small template in a big image. Each location in the image is viewed as a candidate for a good match. We denote the template by $\boldsymbol{t}$ and the candidates by $\boldsymbol{y}_1, \ldots \boldsymbol{y}_m$. Our goal is to compute the k best matches for $\boldsymbol{t}$ among the candidates. Let $J(\boldsymbol{t}, \boldsymbol{y})$ be a match measure. We are interested in cases where it is possible to efficiently compute lower and upper bounds on $J(\boldsymbol{t}, \boldsymbol{y})$, with increasing degrees of accuracy. Let d denote the degree of accuracy, and let $l^d(\boldsymbol{t}, \boldsymbol{y})$, $u^d(\boldsymbol{t}, \boldsymbol{y})$ be the lower and the upper bounds on $J(\boldsymbol{t}, \boldsymbol{y})$ computed at accuracy d. For all values of d these bounds satisfy:

$$l^d(\boldsymbol{t}, \boldsymbol{y}) \leq l^{d+1}(\boldsymbol{t}, \boldsymbol{y}) \leq J(\boldsymbol{t}, \boldsymbol{y}) \leq u^{d+1}(\boldsymbol{t}, \boldsymbol{y}) \leq u^d(\boldsymbol{t}, \boldsymbol{y})$$

We also assume that there is a number N such that for any $\boldsymbol{y}$:

$$l^{N+1}(\boldsymbol{t}, \boldsymbol{y}) = J(\boldsymbol{t}, \boldsymbol{y}) = u^{N+1}(\boldsymbol{t}, \boldsymbol{y}) \quad (1)$$

It is easy to make sure that such N exists for any method of computing lower and upper bounds. One can decide on a value of N as a *maximal useful degree*, and calculate the bounds at degree $N+1$ of accuracy by computing the exact matching values. Thus, N can be taken as a parameter and not as a condition that limits the choice of a bounding scheme.

In Section 2 we show that it is possible to compute such bounds using Walsh-Transform kernels. The commonly used match measure that we consider is the Euclidean distance (distance in the l_2 norm). In vector notation the (squared) Euclidean distance can be written as:

$$J(\boldsymbol{t}, \boldsymbol{y}) = \|\boldsymbol{t} - \boldsymbol{y}\|^2 \quad (2)$$

This measure is used, for example, in [2], [7], [10], [23].

1.2 Related work

Without loss of generality we assume that good matches are identified by small values of the match measure. Pruning with lower bounds was used in some recent studies as follows. Suppose T is a match value computed exactly for $\boldsymbol{y}$, one of the candidates. We have: $T = J(\boldsymbol{t}, \boldsymbol{y})$. If a candidate $\boldsymbol{y}_i \neq \boldsymbol{y}$ has a lower-bound value satisfying $l^d(\boldsymbol{t}, \boldsymbol{y}_i) \geq T$ at any d then $\boldsymbol{y}_i$ cannot be better than $\boldsymbol{y}$ and can be pruned. In particular, there is no need to evaluate its bounds at higher degrees of accuracy.

If the user knows what value of T is acceptable, that value can be provided as a parameter. This approach was taken in [2], [10], [23]. It is not clear, however, how easy it is to determine such a value so that a predetermined number of best matches is detected even for $k = 1$. It may be possible to design some type of binary search,

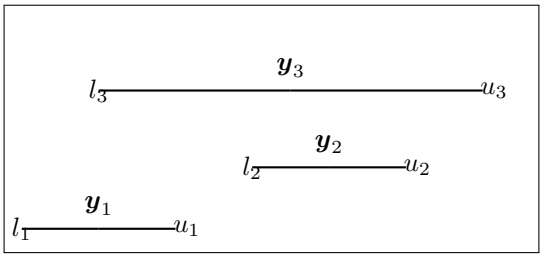

Fig. 1. Example of pruning and early termination. Lower and upper bounds are known for each candidate, but the exact matching values are unknown. If $k = 1$, $\boldsymbol{y}_2$ can be pruned. If $k = 2$, no pruning is possible, but $\boldsymbol{y}_1$ is identified as being part of the solution.

but it was not discussed in the cited references and may lead to a significant increase in the complexity of these algorithms.

A practical approach for computing a pruning threshold T is to run a preliminary crude search for a good match. The exact match value of the candidate found can then be used for initial pruning. That value can be updated whenever a better solution is discovered. See, e.g., [8], [14], [23].

1.3 Our contribution

1.3.1 Dual bounds

We show that upper bounds can be used together with lower bounds to improve the speed of the search. The dual bounds can be used to perform pruning and to detect early termination conditions. Fig. 1 illustrates this for a particular case of three candidates. Each candidate is shown as a line segment connecting its lower bound value with its upper bound value. The true matching value of each candidate is at an unknown position along the line segment. If the goal is to detect the best matching candidate ($k = 1$), we can prune $\boldsymbol{y}_2$ since its lower bound value is above the upper bound value of $\boldsymbol{y}_1$, so that it can never be better than $\boldsymbol{y}_1$. If the goal is to detect the best two matching candidate ($k = 2$), pruning cannot be applied but we can determine that $\boldsymbol{y}_1$ is one of the two best candidates since its upper bound value is below the lower bound value of $\boldsymbol{y}_2$.

To the best of our knowledge the use of dual bounds for accelerating the search is new in the context of fast template-matching.

1.3.2 The dual-bound algorithm

As discussed in the beginning of this section the main approach used in previous studies was to incrementally update the pruning threshold, examining the candidates one by one. The order in which the candidates are examined affects the run time of the algorithm. The best case occurs when the first examined candidate is the best match, so that the initially computed pruning threshold turns out to be the correct one.

We show that there is an optimal ordering for examining the candidates, which can be determined by a priority queue. An algorithm that uses this ordering spends the same amount of time evaluating bounds as an algorithm that is given the best possible pruning threshold as input. The implementation of this idea leads to an algorithm with the best possible run time in terms of the cost of computing bounds. However, this algorithm has an additional overhead of maintaining the priority queue.

1.4 Paper organization

The remainder of the paper is organized as follows: Lower and upper bounds that can be computed using orthogonal projections are derived in Section 2. The use of Walsh-Transform kernels is discussed in Section 3. Pruning and early termination conditions are discussed in Section 4. The dual-bound algorithm is described in Section 5. Experimental results with the algorithm are reported in Section 6. Comparison with some alternative methods is given in Section 7. Conclusions and future work are discussed in Section 8.

2 Bounding Match Measures

In this section we derive lower and upper bounds on the Euclidean match measure defined in Eq. (2). The results are described in terms of general projections, and the lower bounds that we derive unify and generalize results that appear in [8], [10], [23].

The template $\boldsymbol{t}$ and the candidates $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$ are viewed as vectors in $\mathbb{R}^n$, where n is the number of template pixels. We write $\|\boldsymbol{x}\|$ for the l_2 norm of the vector $\boldsymbol{x}$. We need the following inequalities:

$$| \, \|\boldsymbol{x}\| - \|\boldsymbol{y}\| \, | \leq \|\boldsymbol{x} - \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\| \tag{3}$$

These inequalities follow from the Minkowski's inequality (e.g., [22]). This form of the left-hand side was used by Tombari et. al [23] to derive their lower bounds.

Let $V = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d)$ be an $n \times d$ matrix with orthonormal columns. The projection of a vector $\boldsymbol{x}$ on the subspace spanned by the columns of V is given by:

$$\boldsymbol{x}_p = P\boldsymbol{x} = VV^T\boldsymbol{x} = V\boldsymbol{x}_\gamma$$

where $P = VV^T$ is called the projection matrix, and $\boldsymbol{x}_\gamma = V^T\boldsymbol{x}$ is called the coefficients vector. Any vector $\boldsymbol{x} \in \mathbb{R}^n$ can be expressed as:

$$\boldsymbol{x} = \boldsymbol{x}_p + \boldsymbol{x}_q \tag{4}$$

where $\boldsymbol{x}_q = \boldsymbol{x} - \boldsymbol{x}_p$ is orthogonal to $\boldsymbol{x}_p$. Our goal is to approximate the match in terms of projections on the subspace spanned by V. For efficient computation such projections should be easy to compute, and the column vectors of V should have good "energy compaction". By this we mean that the projection $P\boldsymbol{x}$ is a good approximation of $\boldsymbol{x}$ when $\boldsymbol{x}$ is a typical image. Three

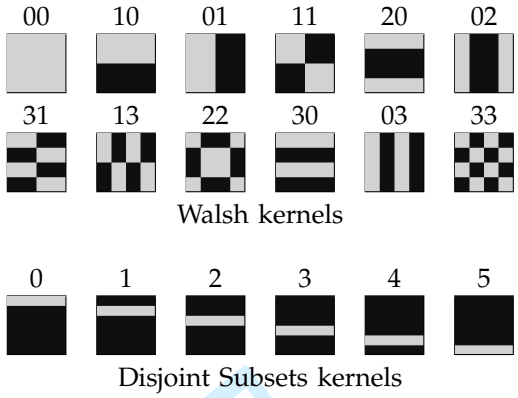

Fig. 2. Several Walsh and Disjoint Subsets kernels

explicit examples from previous studies can be described in this way.

- **DCT kernels.** This choice gives very good energy compaction, but it is inefficient to compute at each point in the image. See [7].
- **Walsh-Transform kernels.** This choice gives slightly worse energy compaction than the DCT [1], but there are methods for computing projections on the subspace spanned by these kernels efficiently at each point in the image. See [2], [10], and Section 3.
- **Disjoint subsets.** Tombari et. al [23] and Mattoccia et. al [14] proposed dividing the image into r disjoint subsets of pixels. Let $s_1, \ldots, s_r$ be these subsets. To enable fast computation of projections, these subsets are selected as pixels in axis parallel rectangles. Define the vectors $\boldsymbol{v}_j$, for $j = 1, \ldots, r$ as follows:

$$\boldsymbol{v}_j(i) = \begin{cases} 1 & \text{if } i \in s_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

  Clearly these vectors are orthogonal. They may have bad energy compaction, but as was shown in [23] they enable very fast rejection in cases where near exact matches exist.

Several Walsh and Disjoint Subsets kernels are shown in Fig. 2.

2.1 Bounding the Euclidean Distance measure

Simple linear algebra shows that if a vector $\boldsymbol{x}$ is partitioned as in 4 then the following relation holds:

$$\|\boldsymbol{x}\|^2 = \|\boldsymbol{x}_p\|^2 + \|\boldsymbol{x}_q\|^2 \tag{6}$$

In particular, for the match measure $J(\boldsymbol{t}, \boldsymbol{y})$:

$$J(\boldsymbol{t}, \boldsymbol{y}) = \|\boldsymbol{t} - \boldsymbol{y}\|^2 = \|\boldsymbol{t}_p - \boldsymbol{y}_p\|^2 + \|\boldsymbol{t}_q - \boldsymbol{y}_q\|^2$$

Applying (3) to $\|\boldsymbol{t}_q - \boldsymbol{y}_q\|$ we get:

$$\begin{aligned} \|\boldsymbol{t}_p - \boldsymbol{y}_p\|^2 + (\|\boldsymbol{t}_q\| - \|\boldsymbol{y}_q\|)^2 \leq \\ \|\boldsymbol{t} - \boldsymbol{y}\|^2 \\ \leq \|\boldsymbol{t}_p - \boldsymbol{y}_p\|^2 + (\|\boldsymbol{t}_q\| + \|\boldsymbol{y}_q\|)^2 \end{aligned} \tag{7}$$

We proceed to show that these bounds can be computed efficiently. For rectangular templates the norms $\|\boldsymbol{t}\|, \|\boldsymbol{y}\|$ can be computed efficiently at each location using running sums or integral-images [24]. We show that the bounds in (7) can be computed from the following information:

$$\|\boldsymbol{t}\|, \|\boldsymbol{y}\|, \boldsymbol{t}_\gamma, \boldsymbol{y}_\gamma$$

It is easy to verify the following formulas:

$$\begin{aligned} &\|\boldsymbol{t}_p - \boldsymbol{y}_p\|^2 = \|\boldsymbol{t}_\gamma - \boldsymbol{y}_\gamma\|^2 \\ &\|\boldsymbol{t}_q\|^2 = \|\boldsymbol{t}\|^2 - \|\boldsymbol{t}_\gamma\|^2, \quad \|\boldsymbol{y}_q\|^2 = \|\boldsymbol{y}\|^2 - \|\boldsymbol{y}_\gamma\|^2 \end{aligned} \tag{8}$$

These formulas can be computed for consecutive degrees of accuracy by a very simple iterative procedure. The bounds at accuracy d are computed from projections on the subspace spanned by $V_d = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d)$. Initially, for $d = 0$ we have:

$$\|\boldsymbol{t}_p^0 - \boldsymbol{y}_p^0\|^2 = 0, \quad \|\boldsymbol{t}_q^0\|^2 = \|\boldsymbol{t}\|^2, \quad \|\boldsymbol{y}_q^0\|^2 = \|\boldsymbol{y}\|^2$$

Assuming known values for $d-1$, the update for d can be computed by the following steps. Compute:

$$\begin{aligned} f &= \boldsymbol{v}_d^T \boldsymbol{t} = \textstyle\sum_{ij} \boldsymbol{v}_d(i,j)\boldsymbol{t}(i,j), \\ g &= \boldsymbol{v}_d^T \boldsymbol{y} = \textstyle\sum_{ij} \boldsymbol{v}_d(i,j)\boldsymbol{y}(i,j) \end{aligned} \tag{9}$$

where the summation is over all coordinates (pixel values). It is shown later that these summations can be calculated very fast when the $\boldsymbol{v}_d$ are Walsh Transform kernels. (The value of f is independent of $\boldsymbol{y}$ and can be precomputed.) Once f, g are calculated apply the following update formulas:

$$\begin{aligned} &\|\boldsymbol{t}_p^d - \boldsymbol{y}_p^d\|^2 = \|\boldsymbol{t}_p^{d-1} - \boldsymbol{y}_p^{d-1}\|^2 + (f-g)^2 \\ &\|\boldsymbol{t}_q^d\|^2 = \|\boldsymbol{t}_q^{d-1}\|^2 - f^2, \quad \|\boldsymbol{y}_q^d\|^2 = \|\boldsymbol{y}_q^{d-1}\|^2 - g^2 \end{aligned}$$

From (7) it follows that at accuracy d the lower bound $l^d(\boldsymbol{t}, \boldsymbol{y})$ and the upper bound $u^d(\boldsymbol{t}, \boldsymbol{y})$ can be computed from:

$$\begin{aligned} (l^d(\boldsymbol{t}, \boldsymbol{y}))^2 &= \|\boldsymbol{t}_p^d - \boldsymbol{y}_p^d\|^2 + (\|\boldsymbol{t}_q^d\| - \|\boldsymbol{y}_q^d\|)^2 \\ (u^d(\boldsymbol{t}, \boldsymbol{y}))^2 &= \|\boldsymbol{t}_p^d - \boldsymbol{y}_p^d\|^2 + (\|\boldsymbol{t}_q^d\| + \|\boldsymbol{y}_q^d\|)^2 \end{aligned}$$

3 Walsh-Transform kernels

To compute bounds as described in Section 2 we need a set of orthonormal vectors $V = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d)$. Following [10] we select them to be low sequency Walsh-Transform kernel functions. The one dimensional Walsh-Transform kernel $\boldsymbol{w}_\alpha$ of length 2^h can be defined as follows [1]:

$$\boldsymbol{w}_\alpha(i) = \begin{cases} 1 & \text{if } \sum_{k=0}^{h-1} b_k(i) b_{h-1-k}(\alpha) \text{ is even} \\ -1 & \text{otherwise} \end{cases}$$

where $b_k(z)$ is the k^{th} bit in binary representation of z.

The two dimensional Walsh-Transform kernel is defined by:

$$\boldsymbol{w}_{\alpha\beta}(i,j) = \boldsymbol{w}_\alpha(i)\boldsymbol{w}_\beta(j)$$

Several Walsh-Transform kernels are shown in Fig. 2. Observe that Walsh-Transform kernels consist of uniform

rectangular regions of "1" or "-1" values. It was pointed out in [24] and [10] that running sums, or integral-images, can be used to efficiently compute projections on Walsh-Transform kernels, although this requires time proportional to the kernel sequency. (The sequency is a measure of the sign changes in the Walsh-Transform kernel.)

Recently it was shown that projections on the subspace of Walsh-Transform kernels can be computed very fast, regardless of their sequency [2], [10]. Exploiting redundancy in the computation of projections from neighboring image windows and contiguous kernel orders, Ben-Artzi et. al. [2] have shown that it is possible to compute the projection of a single window on a single Walsh-Transform kernel with only two operations per pixel. This is a very nice and surprising result. Unfortunately, the speed of the fast approaches of [2], [10] is only fully apparent when one needs to compute projections on the same Walsh-Transform kernels at every location in the image. When used with selective or pruning techniques such as the one proposed in this paper, one may end up computing many unnecessary projections [5].

To fully exploit the selective power of our algorithm, and to enable a clear analysis of its complexity, we implemented an integral-images approach to evaluate projections on the subspace of Walsh-Transform kernels. It computes each projection independently of other projections, with cost proportional to the number of corners in the corresponding Walsh-Transform kernel.

3.1 The corners method

The corners method was mentioned as a possible algorithm for computing projections on the subspace of Walsh-Transform kernels in [24] and [10]. Let $\boldsymbol{w}$ be a Walsh-Transform kernel. Since $w(i,j)$ is either 1 or -1 The projection of an image window $\boldsymbol{p}$ (of the same dimensions as $\boldsymbol{w}$) on $\boldsymbol{w}$ can be expressed as follows:

$$\boldsymbol{p}^T\boldsymbol{w} = \sum_{i,j}\boldsymbol{p}(i,j)\boldsymbol{w}(i,j) = 2\sum_{\boldsymbol{w}(i,j)=1}\boldsymbol{p}(i,j) - \sum_{i,j}\boldsymbol{p}(i,j)$$
$$= 2R_w - \alpha_{00}$$

Here α_{00} is the sum of all pixel values in $\boldsymbol{p}$, and R_w is the sum of the pixel values in $\boldsymbol{p}$ over the regions defined by rectangles of "1" value in $\boldsymbol{w}$. These sums can be computed using the "integral-images" approach (see [24]), adding or subtracting the integral image value at the four corners of the rectangle. The complexity of this method is the number of corners, which grows linearly in the order of the kernel. This linear relation is shown in Fig. 3. (The kernels order was computed by ranking the kernels according to the number of corners.) For additional detail see [5].

4 Pruning and Early Termination

As was illustrated in Fig. 1, lower and upper bounds can identify both pruning and early termination conditions.

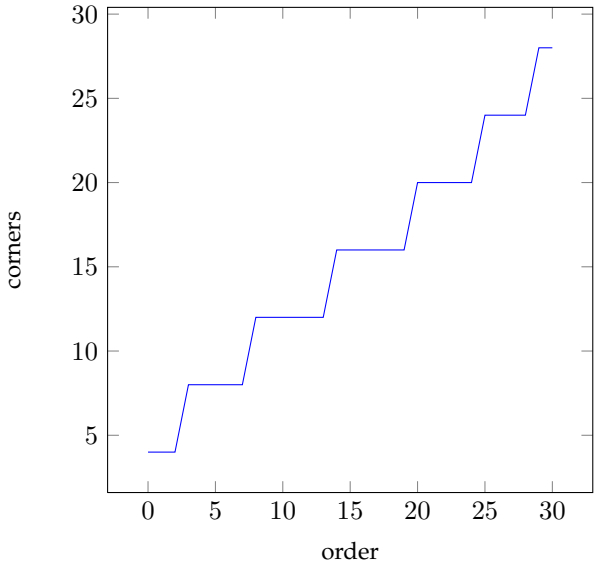

Fig. 3. Number of corners of a Walsh kernel as a function of its order.

In this section we discuss how to use these conditions to detect a set of k matches that forms an optimal solution.

Consider an algorithm for detecting the best k matches of the template $\boldsymbol{t}$ among the m candidates $\boldsymbol{y}_1,\ldots,\boldsymbol{y}_m$. An optimal solution to this problem is a set A of k candidates such that the matching value of any candidate in A is at least as good as the matching value of any candidate not in A.

Bounds on the matching values of these candidates can be computed at different degrees of accuracy during the run time of the algorithm. At any time during the run we write $u(\boldsymbol{y})$ for the most current upper bound of $\boldsymbol{y}$, and $l(\boldsymbol{y})$ for its most current lower bound. The list of all candidates with their associated bounds at a given time is denoted by G.

Consider a partition of G into the two disjoint subsets A, B, where A contains k candidates and B contains $m-k$ candidates. We identify two "extreme" candidates. The extreme candidate in A, denoted by $\boldsymbol{y}_A$, has the largest upper-bound value among all candidates in A. The extreme candidate in B, denoted by $\boldsymbol{y}_B$, has the smallest lower-bound value among all candidates in B.

$$\boldsymbol{y}_A = \arg\max_{\boldsymbol{y}\in A} u(\boldsymbol{y}), \quad \boldsymbol{y}_B = \arg\min_{\boldsymbol{y}\in B} l(\boldsymbol{y}) \qquad (10)$$

These values can be used to detect pruning and early termination as follows:

Pruning. $\boldsymbol{y}\in B$ can be pruned if $l(\boldsymbol{y}) \geq u(\boldsymbol{y}_A)$.

Global Pruning. If all members of B can be pruned then the set A is an optimal solution. (There may be several equally good solutions.)

Early Termination. If $\boldsymbol{y}\in B$ satisfies $u(\boldsymbol{y}) \leq l(\boldsymbol{y}_B)$ then y can be taken as a part of an optimal solution.

Global Early Termination. If $u(\boldsymbol{y}_A) \leq l(\boldsymbol{y}_B)$ then the set A is an optimal solution.

Clearly, the global early termination condition holds if and only if the global pruning condition holds. Early termination suggests a different approach to designing matching algorithms than pruning. Trying to prune all candidates in B suggests intermediate steps in which as much pruning as possible is applied. This is the approach taken by [2], [7], [8], [10], [14], [23]. By contrast, an algorithm that attempts to establish the global early termination condition may choose to focus on steps that evaluate the two extreme candidates $\boldsymbol{y}_A$ and $\boldsymbol{y}_B$. We describe such an algorithm in the next section.

5 The Dual-Bound Algorithm

This section describes our main result: an algorithm that uses lower and upper bounds to compute the k best matching candidates. The main idea is to create the partitioning of the candidates into the sets A, B, as discussed in Section 4, so that at any given time the set A consists of the k candidates with the smallest lower bound values. As we show later, this enables us to easily identify bound values that must be computed at higher accuracy.

The input to the algorithm is the template $\boldsymbol{t}$, the list G of candidates, and the parameters k, N, d_0. The parameter k is the number of desired matches, N is the maximal useful degree of bound accuracy, and d_0 is an initial degree of accuracy. The algorithm starts by computing initial bounds at degree d_0 for all candidates. The list G is then pruned with the threshold U, the k^{th} smallest upper bound. Among the remaining candidates, those with the k lowest lower-bound values are put in the "A" list, and all other candidates are put in the "B" list.

Once the lists A, B are initialized, the extreme candidates $\boldsymbol{y}_A, \boldsymbol{y}_B$ are computed according to Eq. (10). An optional pruning step and the global early termination condition are evaluated. If termination conditions are satisfied, the algorithm terminates with the candidates in A as an optimal solution. Otherwise, new bounds at a higher degree of accuracy are calculated for the extreme candidate $\boldsymbol{y}_A$. This continues until the global early termination condition holds, or different extreme candidates must be selected. The algorithm steps are detailed in Fig. 4.

Notes

- The template $\boldsymbol{t}$ and the parameter N are needed for computing the bounds.
- The notation $l(\boldsymbol{y}), u(\boldsymbol{y})$ always refer to the most accurate lower and upper bounds of $\boldsymbol{y}$ available to the algorithm.
- Step 4.1 is optional. It does not affect the correctness of the algorithm, but may improve its run time. The value of U is the maximum of the upper bounds of k candidates. As was shown in Section 4 it can be used for pruning. A pruning step takes place whenever a tighter (smaller) upper bound is found. See Section 4.
- The algorithm makes sure that the list A always contains the k candidates with the smallest lower bounds. If this is violated at Step 4.3.1, it is immediately fixed at Step 4.3.2.

Input: a template $\boldsymbol{t}$, a list G of candidates, and the values of the parameters k, N, d_0.

1. Compute bounds at degree d_0 for all $\boldsymbol{y} \in G$.

2. Set U to be the k^{th} smallest upper bound in G. Remove from G all $\boldsymbol{y}$ satisfying $l(\boldsymbol{y}) > U$.

3. Compute A, B, and the extreme candidates:

3.1 A = k candidates $y \in G$ with smallest $l(y)$.

3.2 $B = G \setminus A$.

3.3 $\boldsymbol{y}_A = \arg\max_{\boldsymbol{y} \in A} u(\boldsymbol{y}), \quad \boldsymbol{y}_B = \arg\min_{\boldsymbol{y} \in B} l(\boldsymbol{y})$

4. Iterate:

4.1 (*Optional Step*) If $u(\boldsymbol{y}_A) < U$ then:

4.1.1 $U = u(\boldsymbol{y}_A)$.

4.1.2 Prune $\boldsymbol{y} \in B$ satisfying $l(\boldsymbol{y}) > U$.

4.1.3 If B is empty terminate with A.

4.2 If $u(\boldsymbol{y}_A) \leq l(\boldsymbol{y}_B)$ terminate with A.

4.3 Else

4.3.1 Calculate $\boldsymbol{y}_A$ bounds at a higher degree of accuracy.

4.3.2 If $l(\boldsymbol{y}_A) > l(\boldsymbol{y}_B)$ then:

4.3.2.1 Swap $\boldsymbol{y}_A$ in A with $\boldsymbol{y}_B$ in B.

4.3.2.2 Compute new $\boldsymbol{y}_A, \boldsymbol{y}_B$:

$\boldsymbol{y}_A = \arg\max_{\boldsymbol{y} \in A} u(\boldsymbol{y}), \ \boldsymbol{y}_B = \arg\min_{\boldsymbol{y} \in B} l(\boldsymbol{y})$

4.3.3 Else we must have $l(\boldsymbol{y}_A) \leq l(\boldsymbol{y}_B)$. Compute new $\boldsymbol{y}_A = \arg\max_{\boldsymbol{y} \in A} u(\boldsymbol{y})$

Fig. 4. The dual-bound algorithm

5.1 Correctness

The algorithm terminates either at Step 4.1 or at Step 4.2, when the global pruning or the global early termination condition of Section 4 hold. As shown in Section 4, these conditions guarantee the candidates in A are an optimal solution. It remains to show that the algorithm always terminates.

Observe that Step 4.3.1 must be executed at each non-terminating iteration. It is always reached with $l(\boldsymbol{y}_A) < u(\boldsymbol{y}_A)$, since at that point $u(\boldsymbol{y}_A) > l(\boldsymbol{y}_B)$ and $l(\boldsymbol{y}_B) \geq l(\boldsymbol{y}_A)$. This means that the degree of accuracy of $\boldsymbol{y}_A$ bounds at that point cannot exceed N. This shows that if there are m candidates in G then Step 4.3.1 cannot be executed more than $m(N+1)$ times.

5.2 Complexity of the dual-bound algorithm

In this section we analyze the run-time of the dual-bound algorithm. From Section 5.1 we know that the worst-case run time is $O(mN)$, but this is no better than computing the exact matching values for all candidates. The analysis in this section shows that the number of estimates of

lower and upper bounds required by the algorithm is nearly the best possible.

The run time has two components. The first is the computation of lower and upper bounds, carried out in steps 1 and 4.3.1. We call this component the *inherent cost* of the algorithm. The second component is the creation and maintenance of the lists A, B, including the repeated computation of the extreme candidates $\boldsymbol{y}_A$ and $\boldsymbol{y}_B$. We show that this can be done with priority queues, and refer to this component as the *queue cost* of the algorithm.

5.2.1 Inherent cost

We show that in terms of inherent cost the dual-bound algorithm performs as well as algorithms that have access to a perfect pruning threshold. Without loss of generality assume that the candidates are numbered so that $J(\boldsymbol{y}_i, \boldsymbol{t}) \leq J(\boldsymbol{y}_{i+1}, \boldsymbol{t})$ for $i = 1, \ldots, m-1$. Define:

$$T_r = J(\boldsymbol{y}_k, \boldsymbol{t}), \quad T_a = J(\boldsymbol{y}_{k+1}, \boldsymbol{t})$$

Thus, T_r is the k^{th} smallest match value, and T_a is the $k+1^{\text{th}}$ smallest match value. If we know these values then it is possible to apply the following *best possible* acceptance/rejection scheme:

$$\begin{array}{l} \text{if for some } d,\ u^d(\boldsymbol{y}) < T_a \text{ then } \boldsymbol{y} \text{ belongs to solution} \\ \text{if for some } d,\ l^d(\boldsymbol{y}) > T_r \text{ then reject (prune) } \boldsymbol{y} \end{array} \tag{11}$$

Therefore, the bounds of $\boldsymbol{y}$ need only be calculated up to an accuracy d where one of the conditions in (11) holds. We refer to T_r as the perfect pruning threshold, and to T_a as the perfect acceptance threshold. We show that the dual-bound algorithm behaves as if T_r is known, so that the bounds of candidates that can be pruned are never evaluated unnecessarily.

Theorem. *Let d be the smallest degree of accuracy in which the candidate $\boldsymbol{y}$ can be pruned with the perfect pruning threshold T_r. The dual-bound algorithm never calculates the bounds of $\boldsymbol{y}$ to a degree higher than d.*

Proof: We need to show that if bounds are calculated for $\boldsymbol{y}_A$ in Step 4.3.1, then $l(\boldsymbol{y}_A) \leq T_r$. Suppose this condition does not hold so that $l(\boldsymbol{y}_A) > T_r$. Observe that at Step 4.3.1 $l(\boldsymbol{y}) \geq l(\boldsymbol{y}_A)$ for all candidates in B, and therefore if $\boldsymbol{y} \in B$ then $l(\boldsymbol{y}) \geq l(\boldsymbol{y}_A) > T_r$. This means that all candidates in B and at least one in A can be pruned, which leaves at most $k-1$ unpruned candidates. This contradicts the fact that there are at least k candidates that cannot be pruned. □

The proof of the theorem holds as long as the candidate chosen in step 4.3.1 belongs to A, the set of the k candidates with the smallest lower bounds. The extreme candidate y_A is chosen since other candidates in A are more likely to be identified as a part of the solution by the early termination condition.

As shown in Fig. 5, the dual-bound algorithm is not optimal with respect to the perfect acceptance threshold T_a. Candidates that end up as part of the solution may be evaluated to higher accuracy than necessary (if T_a is known). This applies to at most k candidates.

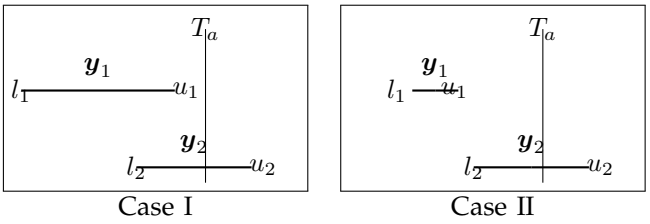

Fig. 5. Here $k = 1$, and $A = \{\boldsymbol{y}_1\}$. In Case I, if the value of T_a is known, the algorithm can terminate with $\boldsymbol{y}_1$ as the best candidate. The dual-bound algorithm will not terminate, and will keep improving the bounds of $\boldsymbol{y}_1$ until $u_1 \leq l_2$, as shown in Case II.

5.2.2 Queue cost

Step 2 of the algorithm requires that the value of the k^{th} smallest upper bound is computed. This can be done in time linear in m when k is a small constant, by using a binary heap of size k [3]. Similarly, identifying the sets A, B in Step 3 can be achieved by first building a MIN-HEAP of size m, ranking all candidates according to their lower-bound values, and then extracting the k elements of A. When k is a small constant this process requires only linear time in m.

The following operations are performed on the lists A, B during the iterations (Step 4):

- Select the extreme candidates as the largest ($\boldsymbol{y}_A$ in A) or the smallest ($\boldsymbol{y}_B$ in B) in the list.
- Update bound values of extreme candidates.
- Replace extreme candidates.
- Remove (not necessarily extreme) candidates, at Optional Step 4.1.

With the exception of the candidates removal at Optional Step 4.1, these are standard operations on a priority-queue, a heavily researched abstract data type [17]. We describe a simple implementation using the binary heaps data-structure with a partial implementation of Step 4.1.

We implemented the priority queues of both A and B as binary heaps [3]. The A heap was implemented as a MAX-HEAP, ranking the candidates in A according to their upper-bound values. The B heap was implemented as a MIN-HEAP, ranking the candidates in B according to their lower-bound values.

The creation of a heap takes linear time. Retrieving the values of the extreme candidates $\boldsymbol{y}_A, \boldsymbol{y}_B$ as the top elements of the heaps is instantaneous. The two expensive operations are the repeated deletions of candidates (at Step 4.1) and the replacement of the candidate at the top of the heap (Steps 4.3.2.2, 4.3.3). We provide some additional details on the partial implementation of these steps and their expected performance.

5.2.3 Handling deletions

From the heap property (see [3]) it follows that the deletions at Step 4.1 are of entire subtrees with root values above U. There does not seem to be an efficient method

of performing these deletions in a binary heap. Since this step affects only the queue cost and not the correctness of the algorithm, we have decided not to fully implement it. Instead, we just ignore subtrees of the heap when the root value is above U. We found that this reduces the queue cost by about 5%, which is not significant. It appears that ignoring Step 4.1 altogether does not significantly change the run-time of the algorithm. Not implementing Step 4.1 also allows for an implementation using off-the-shelf binary heap software.

5.2.4 Updating the top element

The cost of updating the candidate at the top of a size m heap is at worst $\log_2 m$. We argue here that the average cost in our case should be much smaller, and support this claim by experimental evidence in Section 6.

The updating of the heap top element (when it needs to be updated) is carried out by swapping it with it's child of smallest value (see [3]). This is continued until the "percolated down" element is no bigger than its two children (the heapify-down procedure).

The updating at Steps 4.3.2.1, 4.3.2.2 replaces $\boldsymbol{y}_B$ with an upgraded $\boldsymbol{y}_A$, whose upgraded lower bound value is bigger than $l(\boldsymbol{y}_B)$. That upgraded $\boldsymbol{y}_A$ is percolated down the heap B. Observe that before the upgrade $l(\boldsymbol{y}_A)$ was smaller than the lower bound of all candidates in B. One would expect that the upgraded lower bound value would still be among the smallest in B. Suppose the lower bound of the upgraded $\boldsymbol{y}_A$ is the t-smallest in B. Then it is known that its *expected* location in the heap would be at depth $\log_2 t$ [4]. This suggests expected run time of $\log_2 t$ which can be much shorter than $\log_2 m$.

6 Experimental results

This section describes experimental results with the dual-bound algorithm. The implementation that we use did not include the optional pruning in Step 4.1 of the algorithm. With the exception of Experiment 5, all runs were with $d_0 = 1$. In evaluating the inherent cost we ignored terms that are independent of the candidates and can be precomputed, such as the values of f in Eq. (9). The following parameters were measured in each run:

- p_d : fraction of candidates requiring bounds at degree d of accuracy.
- e : fraction of candidates that had to be computed exactly (at degree $N+1$).
- s : total number of heap swaps.

The inherent cost was calculated per image pixel as follows:

$$\text{Inherent cost per pixel} = \textbf{InitCost} + \sum_{d=1}^{N} c_d p_d + \textbf{ExactCost}$$

The value of c_d is the cost of updating the bounds from degree $d-1$ to d. In our case the number of arithmetic operations that the upgrade takes is the number of corners of the d^{th} Walsh kernel (see Section 3).

The value of **InitCost** was taken as 5 operations per pixel. It is the cost of computing two integral images. The first integral image is used for computing sums of squared pixel values over rectangles, that is needed for computing norms. Its creation cost is 3 operations per pixel. The second is for computing pixel sums over rectangles, that is needed for computing projections on Walsh kernels. Its cost is 2 operations per pixel.

Computing the exact matching value as defined in (2) takes 3 operations per template pixel. The value of **ExactCost** was computed as: $\textbf{ExactCost} = 3 \cdot n \cdot e$, where n is the number of template pixels.

As discussed in Section 5.2.2 all operations involving the priority queues can be implemented with heaps. (This includes Step 2 of the algorithm.) The cost parameter of heap operations is the number of swaps. The value of the parameter s gives the total number of heap swaps during the run of the algorithm. There are two "machine-dependent" parameters that affect the run-time:

- H_1 : The cost of an arithmetic operation.
- H_2 : The cost of swapping two array elements.

The inherent cost is proportional to H_1, and the queue cost is proportional to H_2. In order to represent the queue cost in the same units as the inherent cost we multiply the queue cost by the following ratio:

$$r = \frac{H_2}{H_1} \tag{12}$$

This gives the following expression for the queue cost per pixel, measured in the same units as the number of arithmetic operations:

$$\text{Queue cost per pixel} = \frac{sr}{m}$$

The total cost per pixel was computed as the sum of the inherent cost per pixel and the queue cost per pixel.

We found the value of r to vary roughly in the range of 2 to 10, depending on the hardware used, the amount of memory available, and the size of the array allocated for the heaps. In reporting the experimental results we use $r = 5$.

6.1 Experiment 1

In this experiment we evaluated the performance of the algorithm as a function of the template size. Keeping all other parameters unchanged, we ran the algorithm on image/template pairs obtained by scaling both the template and the image by the same scaling factor. The image size itself is not relevant since the cost was measured per image pixel. In this experiment we used the "man" image from the USC database, as shown in Fig 6. The original image size was 1024×1024, and the template size was 128×128. Zero-mean Gaussian noise with $\sigma = 10$ was added to the template. Both image and template were shrunk by factors of 1,2,4,8,16, and 32.

Fig. 6. Image and template in Experiment 1.

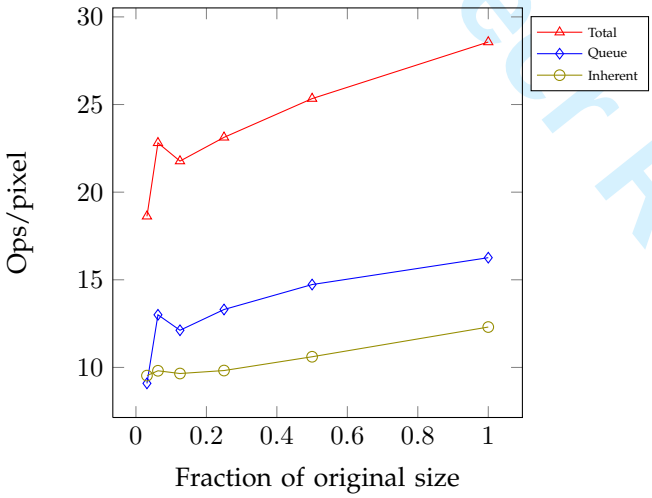

Fig. 7. Algorithm cost as a function of template size. (Experiment 1.)

Results are shown in Fig. 7. The total cost of the algorithm grows very slowly with the template size. While the size grows by a factor of 32, the total cost increases only by a factor of 1.5. The inherent cost grows even more slowly, going from 9.5 ops/pixel to 12.3 ops/pixel, a factor of 1.3.

6.2 Experiment 2

In order to evaluate the performance of the algorithm on inexact matches we added various levels of zero mean Gaussian noise to the template. The experiment was performed on 1,200 images from the MIT's LabelMe image database [19]. The Harris corner detector was used to automatically select 2 random templates with a large number of corners from each image. (This is similar to the experimental framework described in [10].) Noise

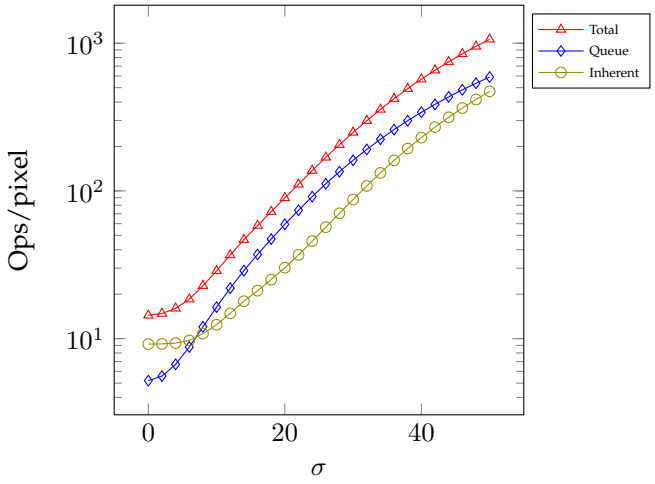

Fig. 8. Operations per pixel as a function of noise level. (Experiment 2.)

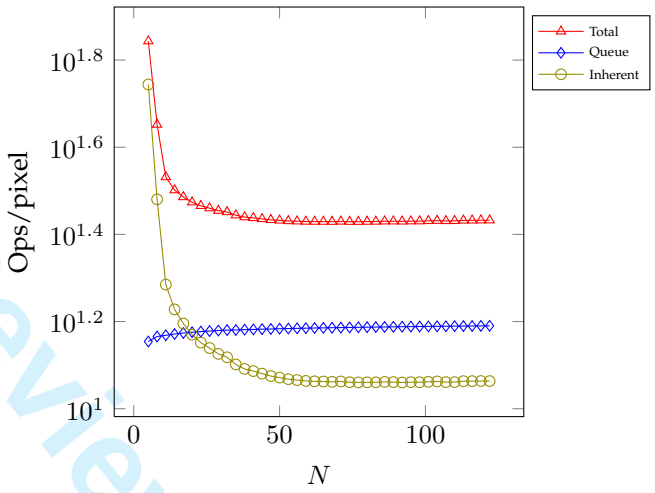

Fig. 9. Operations per pixel as a function of the maximum number of Walsh kernels, N. (Experiment 3.)

was added at 26 different levels to each template giving a total of 62,400 image/template pairs. All images were 512×512, and the templates were 64×64. The value of N was kept constant at $N = 100$, and d_0 was set to 1.

The results are shown in Fig. 8. At low noise levels the queue cost is negligible. At higher noise levels the queue cost is larger than the inherent cost.

6.3 Experiment 3

In this experiment we evaluated the algorithm with different values of the parameter N, the maximum number of Walsh kernels. We used the same images and templates as in Experiment 2, running the algorithm 96,000 times. The amount of Gaussian noise was held constant at $\sigma = 10$.

The results are shown in Fig. 9. They suggest that the number of Walsh kernels has only a minor effect on the total cost of the algorithm unless N is very small. We

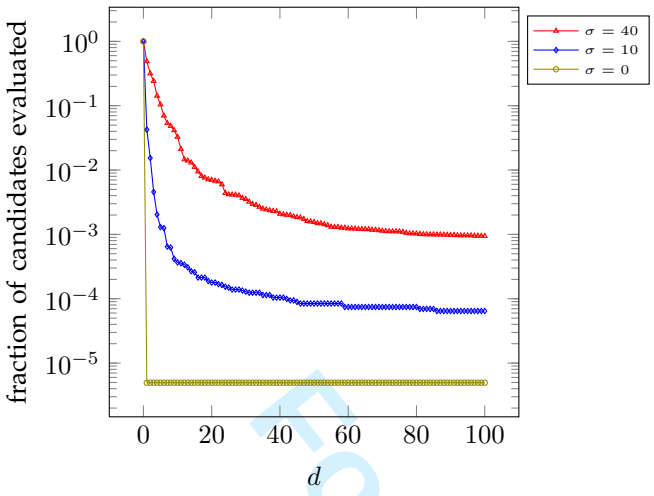

Fig. 10. Fraction of candidates vs. d.(Experiment 4.)

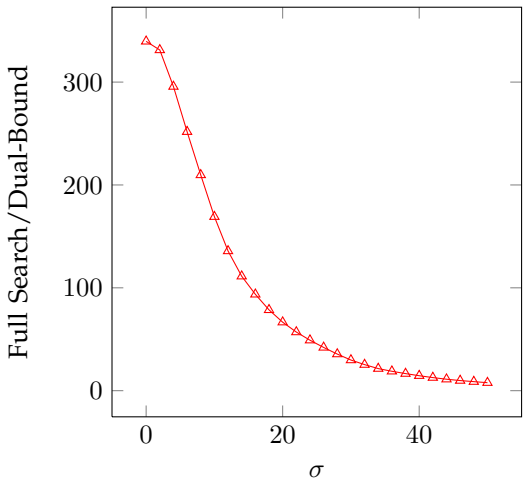

Fig. 12. Ratio of brute force search run time to the dual-bound run time.
(Experiment 6.)

observed that for very large values of N (not shown on the plot), the cost starts growing as a function of N.

6.4 Experiment 4

The savings of the algorithm come from its tendency to evaluate very few candidates at high degrees of bound accuracy. We plot the number of candidates evaluated at bound accuracy d for three noise levels. The experiment was performed on the "lenna" image, with the template taken from the image center. Observe that the number of candidate bounds evaluated at accuracy d decreases very rapidly with d, especially at low noise levels. In particular, in the noise free case it was enough to evaluate bounds based on a single Walsh kernel for all but 1 candidate. Even at $\sigma = 40$ the number of candidates with bounds evaluated at accuracy above 20 was less than 1%.

We observed a very similar behavior with other image/template pairs. In particular, consider the case of $k = 1$ when exact matching exists. In this case we have $T_r = 0$ for the pruning threshold that discussed in Section 5.2.1. According to the theorem, bounds of candidates that are not the best match need only be evaluated until their lower bound value is above T_r, i.e., nonzero. In all our experiments nonzero lower bounds were obtained for $d = 1$.

6.5 Experiment 5

This experiment evaluates the effect of the initial bound accuracy, d_0, on the total cost. A table of the results of runs performed on the "lenna" image is shown in Fig. 11. Observe that, as expected, the inherent cost always increases as a function of d_0, and the queue cost always decreases. For small and moderate noise levels the choice $d_0 = 1$ is clearly the best, but this may not be the case for high noise levels. Similar behavior was observed with other image/template pairs.

6.6 Experiment 6

In this experiment we measure the actual run time of the algorithm and compared it to the run time of the brute force approach in which all matching values are computed exactly. All experiments were conducted on a 3.0 GHz Intel Pentium IV computer with 3GB of RAM running Sun's Java Virtual Machine. (Java's runtime optimization features was disabled.) The dataset is the same as the one used in Experiment 2, running the algorithm on 62,400 image/template pairs. The results are shown in Fig. 12. The values range from a factor of over 300 for low noise levels to slightly less than a factor of 10 at the highest noise levels tested (zero mean Gaussian noise with $\sigma = 50$).

6.7 Experiment 7

In this section we describe experimental comparison between the dual bound algorithm and three other exact template matching algorithms. We have implemented the following algorithms:

- Li and Salari (LS), as described in [13].
- The Gray Code Kernels method (GCK), as described in [2].
- The Incremental Dissimilarity Algorithm (IDA), as described in [23].
- The Dual Bound algorithm (DB).

The algorithms LS, GCK, IDA use only lower bounds, while the DB uses both lower and upper bounds. Another major difference is that the LS, GCK, and IDA require an estimate, or an exact pruning threshold as input, while the DB does not.

Unfortunately, we were unable to quantify the advantage of the DB over the other algorithms with regard to the pruning threshold, since the references describing these algorithms do not describe a "built in" method of computing an initial threshold. (A discussion of the importance of the initial threshold value and experiments

	$\sigma = 0$			$\sigma = 10$			$\sigma = 40$		
d_0	Inherent	q	Total	Inherent	q	Total	Inherent	q	Total
1	9.09	6.10	15.19	10.20	10.57	20.77	33.16	99.55	132.70
4	25.09	5.56	30.65	25.99	6.12	32.12	46.26	49.61	95.88
7	49.09	5.30	54.39	49.97	5.62	55.59	68.42	32.47	100.89
10	81.08	4.93	86.018	81.95	5.22	87.18	98.97	22.68	121.64

Fig. 11. The effect of d_0 on the cost.(Experiment 5.)

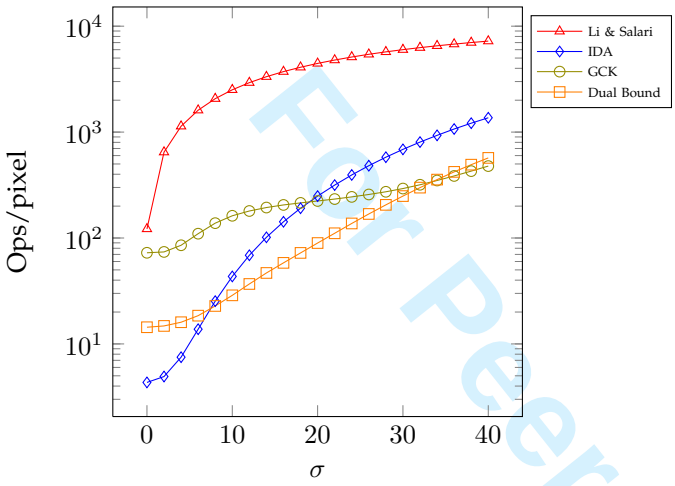

Fig. 13. Comparison of three algorithms with the Dual Bound algorithm

with different threshold values can be found in [23].) To enable comparison we provide the best possible pruning threshold (T_r in Section 5.2.1) to all algorithms. Observe that the Dual Bound algorithm does not use this information.

The algorithms were tested on the same dataset as in Experiment 2, with each algorithm applied to 62,400 image/template pairs. The results are shown in Fig. 13. They seem to suggest that the IDA beats the DB for low noise values, and that the GCK beats the DB for very high noise values. We discuss these results and argue that this would not have been the case if the cost of computing the pruning threshold was taken into account.

Reference [14] provides a formula for estimating the cost of an initial threshold (which is, in most cases inferior to the optimal threshold used in our experiments). According to that formula, with image size 512×512 and template size 64×64 the cost is 51 operations. The following table shows the results of adding this cost to the IDA at low noise levels:

σ	IDA	IDA + 51	Dual-Bound
0	4.3	55.3	14.4
8	25.4	76.4	22.8
10	43.4	94.4	28.8

At such low noise levels the cost of the DB algorithm falls significantly below the cost of estimating the initial threshold.

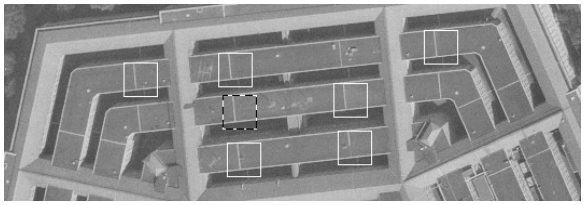

Fig. 14. Results for $k = 7$ on the "pentagon" image. The exact match is shown with a dashed outline. (Experiment 8.)

In evaluating the performance of the GCK we counted projections on Walsh kernels that must to be computed in order to estimate the bounds. The GCK computes them recursively, evaluating additional projections that may not be needed. At low noise levels where only few projections are needed, this overhead results in a mediocre performance. At high noise levels the overhead is much smaller since many more projections are needed. The superior cost per kernel of the GCK pays off and produces the best results in our experiments.

We observe, however, that the speed of the GCK as observed in our experiments is largely due to the fact that it is given the perfect pruning threshold. One would expect the cost of computing such threshold to increase with the amount of noise. Practical implementations of the GCK would most likely need to use some sort of binary search to determine the right threshold. At large noise levels the binary search would have to be performed over a bigger range of values. We expect such implementations to be significantly worse than the one we tested, and possibly inferior to the DB algorithm even for very high noise levels.

6.8 Experiment 8

We evaluated the performance of the algorithm with $k > 1$. Running the algorithm with small values of k typically produces matching candidates surrounding the best match. This is not the case when the image contains repeating patterns, or when k is moderately large. As an example we show results obtained with the 1024×1024 "pentagon" image from the USC database. The results for $k = 7$ are shown in Fig 14. Only a portion of the image containing the 7 matches is shown.

We evaluated the cost of running the algorithm with $k > 1$ on 10 images taken from the USC database. 5 templates were selected for each image, giving a total

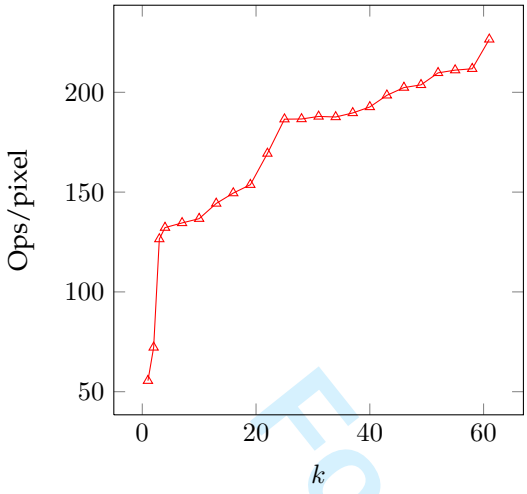

Fig. 15. Operations per pixel.(Experiment 8.)

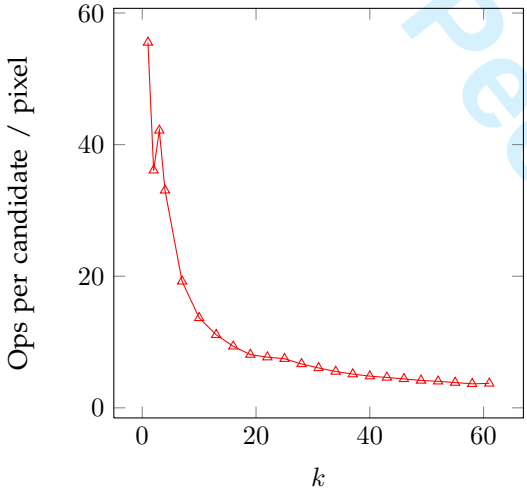

Fig. 16. Operations per candidate per pixel. (Experiment 8.)

of 1150 runs. The level of Gaussian noise was $\sigma = 5$. The results are shown in Fig. 15. We observe that the cost is monotonically increasing in k and grows very rapidly for small k values. But the cost per candidate decreases rapidly with k. As shown in Fig.16 for k values near 60 the cost per candidate goes down to less than 4 arithmetic operations per candidate per pixel.

7 Comparison with other methods

7.1 Gharavi-Alkhansari's Algorithm

The algorithm presented in [8] performs pruning in a multi-resolution setting. It requires an expensive preprocessing step that according to the analysis in [8] takes approximately 24 operations per pixel for a 512x512 image. This dominates the cost of the algorithm at low noise levels, where the preprocessing cost alone is higher than the total cost of the dual-bound algorithm for the same image size.

According to the results reported in [8] the Gharavi-Alkhansari algorithm performs very well at high noise levels. There is not enough information to enable direct comparison with our implementation of the dual-bound algorithm. We expect that in some cases which are very suitable to multi-resolution analysis, Gharavi-Alkhansari's algorithm may outperform the dual-bound algorithm. In other cases the Walsh kernels that we use may provide better "energy compaction" than multi-resolution, and one would expect the dual-bound algorithm to outperform Gharavi-Alkhansari's algorithm.

7.2 Fast convolution techniques

Fast convolution techniques such as the FFT and the Number Theoretic Transform (see, e.g., [21]) can be used to compute template matching in $O(m \log m)$ time. This run time is independent of n, the template size, and the quality of the match, as measured by σ, the template noise level. Run time comparisons with fast pruning techniques were reported in [8], [10], [23]. They indicate that at low noise levels pruning techniques outperform fast convolution techniques for small template sizes.

The results of Experiment 2 show that the dual-bound algorithm performance deteriorates at high noise levels. One should expect fast convolution techniques to perform better in such cases. On the other hand, the results of Experiment 1 suggest that the cost of the dual-bound algorithm is not strongly affected by the template size. We expect that on hardware where the priority queue cost is cheap relative to the inherent cost (small r values), the dual-bound algorithm may provide an alternative to fast correlation techniques even for very large templates.

7.3 Inexact methods

Inexact methods do not guarantee a best possible match and can potentially run much faster than exact methods. Pele and Werman [15], [16] describe a very fast sampling based approach. They report performance of roughly 1.5% of brute force methods in [16], and an improved performance of 0.2% in [15]. The latter results which are based on sliding windows outperform the results reported here. Without sliding windows their results are comparable to ours.

8 Conclusions and Future Work

The main contribution of this paper is a new algorithm that uses both lower and upper bounds on the match measure to compute the k best matches. The algorithm uses a priority queue to decide which bounds need to be improved. Its run time cost has two components: the inherent cost which measures the arithmetic operations needed to evaluate the bounds, and the priority queue cost which measures the cost of creating and maintaining the priority queue.

The inherent cost of the algorithm is nearly optimal, in the sense that bounds for candidates that can be

pruned according to an optimal (and typically unknown) threshold value are not computed. Our results show that when good matches exist (as shown in experiments with low noise levels), the dominant cost is the inherent cost, so the overall performance of dual-bound algorithm is nearly optimal.

In cases where a good match does not exist, the queue cost dominates the overall cost. Still, as shown in our experiments, a classic binary heap implementation of priority queues compares favorably with the current state-of-the-art. We expect better implementations of the priority queue component to yield further improvements over the results reported here. There are many other possible implementations of priority queues [17], and hardware implementations were also investigated [11].

Our estimate of r, the ratio of memory swap cost to arithmetic operation cost (see Eq. (12) was $r = 5$. This value was measured on standard dual-core computers that come with highly optimized arithmetic logic units (ALU's) and floating point units (FPU's). We expect this ratio to be much smaller in implementations on simpler hardware such as the one used in hand-held devices. On such hardware we expect the performance of our algorithm to improve, when it is measured relative to the cost of arithmetic operations.

The technique of bounding match measures that was developed in Section 2 may be of independent interest. For example, one may use this technique to improve the pruning bounds used in [2], [7], [10].

The dual-bound algorithm itself is general, and can be used in all cases where it is possible to incrementally compute lower and upper bounds on the matching values. It may be possible to use the algorithm with alternative kernels, or other methods of computing the bounds. We expect the same framework may lend itself to many extensions. In particular, we have preliminary results which show that a similar method can be used to detect arbitrarily shaped templates, as well as handling the case of multiple templates. We also expect that a version of the algorithm using the fast Walsh kernels technique, as described in [2], would improve on the results described here.

REFERENCES

[1] K. Beauchamp. *Applications of Walsh and Related Functions*. Academic Press, 1984.

[2] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or. The gray-code filter kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:382–393, 2007.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 2001.

[4] J. M. de Graaf and W. A. Kosters. Expected heights in heaps. *BIT*, 32(4):570–579, 1992.

[5] R. Deng, R. F. Anderson, and H. Schweitzer. A corner-based method for computing walsh transform at multiple locations in an image. Technical Report UTDCS-17-09, Department of Computer Science, The University of Texas at Dallas, July 2009.

[6] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

[7] M. Gharavi-Alkhansari. A fast full-search equivalent algorithm using energy compacting transforms. In *Proceedings. 2001 International Conference on Image Processing*, volume 2, pages 713–716, 2001.

[8] M. Gharavi-Alkhansari. A fast globally optimal algorithm for template matching using low-resolution pruning. *IEEE Transactions on Image Processing*, 10:526–533, 2001.

[9] A. Goshtasby, S. H. Gage, and J. F. Bartholic. A two-stage cross correlation approach to template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:374–378, 1984.

[10] Y. Hel-Or and H. Hel-Or. Real-time pattern matching using projection kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1430–1445, 2005.

[11] F. Hoeg, N. Mellergaard, and J. Staunstrup. The priority queue as an example of hardware/software codesign. In *Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 81 – 88, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[12] T. Kawanishi, T. Kurozumi, K. Kashino, and S. Tagagi. A fast template matching algorithm with adaptive skipping using inner-subtemplates' distances. In *Proceedings of the 17th International Conference on Pattern Recognition*. IEEE Computer Society Press, 2004.

[13] W. Li and E. Salari. Successive elimination algorithm for motion estimation. *IEEE Transactions on Image Processing*, 4:105–107, 1995.

[14] S. Mattoccia, F. Tombari, and L. Di Stefano. Fast full-search equivalent template matching by enhanced bounded correlation. *IEEE Transactions on Image Processing*, 17:528–538, 2008.

[15] O. Pele and M. Werman. Accelerated pattern matching or how much can you slide? In *ACCV 2007*, number 4844 in Lecture Notes in Computer Science, pages 435–446. Springer-Verlag, 2007.

[16] O. Pele and M. Werman. Robust real-time pattern matching using bayesian sequential hypothesis testing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1427–1443, 2008.

[17] R. Ronngren and R. Ayani. A comparative study of parallel and sequential priority queue algorithms. *ACM Transactions on Modeling and Computer Simulation*, 7:157–209, 1997.

[18] A. Rosenfeld and G. J. Vanderburg. Coarse-fine template matching. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 104–107, February 1977.

[19] B. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman. Labelme: a database and web-based tool for image annotation. Technical Report AIM-2005-025, MIT, AI Lab, September 2005.

[20] H. Schweitzer, J. W. Bell, and F. Wu. Very fast template matching. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *Computer Vision - ECCV 2002*, number 2353 in Lecture Notes in Computer Science, pages 358–372. Springer-Verlag, 2002.

[21] J. O. Smith. *Mathematics of the Discrete Fourier Transform (DFT), with Audio Applications*. W3K Publishing, 2007.

[22] G. W. Stewart and J. Sun. *Matrix Perturbation Theory*. Academic Press, 1990.

[23] F. Tombari, S. Mattoccia, and L. D. Stefano. Full search-equivalent pattern matching with incremental dissimilarity approximations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):129–141, January 2009.

[24] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004.

Changes

The paper went through a major revision. The main changes are:

- Largely expanded experimental section.
- Text was trimmed whenever possible.
- Arguments and proofs were simplified and clarified.
- Section 4 was rewritten.
- Changes were made to address reviewers comments.

Detailed changes

In response to Reviewer 1

Reviewer 1 wanted the experimental section expanded. This was done. It now contains results obtained from approximately half a million runs. This was obtained from several thousand images and templates, changing various parameters. Explicit comparison to other methods is also included.

In response to Reviewer 2

- Text was trimmed.
- Additional experiments were added.
- The revised section 6.8 gives more information and provides more experimental data for the $k > 1$ case.

1) Vector norm are now properly displayed.
2) Detailed explanation for Fig.1 has been added to the text.
3) Subsection numbering has been fixed.

In response to Reviewer 3

The authors would like to thank Reviewer 3 for the many helpful suggestions and for pointing out some of our inaccuracies.

1. Text was trimmed, and material from several sections was combined. Section 1.2 from previous version of paper was deleted.

2. Notation was improved. Vectors are boldface, vector norms are properly displayed, transpose is denoted by T, template is not denoted by "lambda".
The orthogonal subspaces are now denoted by P, Q and not by A, B.
Projections are now defined in terms of orthonormal vectors.
The dependence of the projection on 'd' is now explicit.
The programming-like notation in the old eq. 8 was fixed. As suggested, it is now expressed in terms of d.
Equations are no longer framed in boxes.

3. The orthogonal projection is now defined in terms of orthonormal vectors.
Regarding comments about projections: The proper mathematical term defines projections on (onto) a subspace and not on vectors. We applied these changes and use projections on a subspace.

4. A definition of "energy compaction" is now given.

6. α_{00} and R_w are now clearly defined.
Walsh kernels are defined.

7. The introductory text in Section 3 was shortened.

8,9,10,11. Section 4 was completely rewritten following the criticism and the suggestions by the referees. Unlike the earlier version, the set A in Section 4 is an arbitrary set of k candidates. The constraint that it always contains the k candidates with smallest lower bounds is added in Section 5.

12. The flowchart was deleted.

13,18. The algorithm was cleaned and (hopefully) made clearer. This includes the suggested "if-then-else" constructs. Steps are now expressed by explicit formulas.

We did not follow the suggestion of introducing the priority queues inside the algorithm description. We believe it is better to describe the algorithm in more general terms and introduce the priority queues later. We have in mind situations in which only the inherent cost is meaningful. In such cases it may not be necessary to implement priority queues at all.

We did not follow the suggestion of explicitly specifying d everywhere in the algorithm, since different candidates may have different degrees of accuracy. Instead we explain in the text that our lower and upper bound notation always refers to the most accurate values available to the algorithm.

14. The number N is now introduced in the introduction. (Old version N = new version $N + 1$.)

15,16. The terms needed for stating and proving the theorem in Section 5 are now carefully defined.

17. Experiment 5 was conducted to evaluate the effect of d0 on the run time of the algorithm.

19. We followed the suggestion to move what was previously the complexity analysis to the beginning of Section 6.

20. Regarding our choice of skipping pruned subtrees instead of removing them. We couldn't find an alternative O(log m) algorithm that performs a delete operation. Such algorithms exist for deleting to top element, but not for deleting other elements. We discussed it with some experts in the theory of algorithms and they couldn't help either. The skipping idea is not really a deletion since the heap cannot be squeezed back to logarithmic depth. It is turned into a binary tree.

As mentioned in the paper this idea also doesn't seem to be very effective. We measured an improved performance of about 5% which may not be worth much.

21. The introduction to Section 6 was shortened and combined with material from other sections.

22. We added Experiment 6 which reports run time results.

Operation count was reduced to 3 criteria.

We added Experiment 8 to evaluate the performance with varying k.

We added Experiment 4 which evaluates p_d for different levels of noise.

We added Experiment 5 to evaluate the effect of d_0.

23. We expanded Experiment 7 to include comparison with two of the current state-of-the-art methods.

Verbal comparison to an approximate method is provided.