Thinking Beyond the Block: Block Matching for Copy–Move Forgery Detection Revisited

Matthias Kirchner[a], Pascal Schöttle[b], and Christian Riess[c]

[a]Electrical Engineering and Computer Engineering Department, Binghamton University, USA;
[a]Department of Information Systems, University of Münster, Germany;
[b]Radiological Sciences Lab, Stanford, CA, USA

ABSTRACT

Detection of copy–move forgeries is one of the most actively researched topics in image forensics. It has been shown that so-called block-based methods give the highest pixelwise accuracy for detecting copy–move forgeries. However, matching of block-based features can be computationally extremely demanding. Hence, the current predominant line of thought is that block-based algorithms are too slow to be applicable in practice.

In this paper, we revisit the matching stage of block-based copy–move forgery detection methods. We propose an efficient approach for finding duplicate patterns of a given size in integer-valued input data. By design, we focus on the spatial relation of potentially duplicated elements. This allows us to locate copy–move forgeries via bit-wise operations, without expensive block comparisons in the feature space. Experimental investigation of different matching strategies shows that the proposed method has its benefits. However, on a broader scale, our experiments demonstrate that the performance of matching by lexicographic sorting might have been underestimated in previous work, despite its remarkable speed benefit on large images. In fact, in a practical setting, where accuracy and computational efficiency have to be balanced, lexicographic sorting may be considered the method of choice.

1. INTRODUCTION

Copy–move (CM) forgeries are a common form of local image processing, where parts of an image are copied and then re-inserted into another part of the same image. Localizing and detecting such manipulations has been among the most actively investigated fields in the image forensics community.[1] More than a decade[2] of research has led to a vast body of proposed methods, the majority of which can be categorized into block-based methods (for example, works by[3–6]) and methods based on local key point descriptors[7,8]. Recent literature concludes that the former tend to be more accurate, in particular when relatively smooth image regions are copied.[1,9] A major disadvantage of block-based approaches is their high computational load in the matching stage: in contrast to keypoint-based methods, the number of blocks essentially is the number of image pixels. For megapixel-sized images, comparing and matching all blocks of the image to a suitable number of candidate blocks can easily take hours. This is a particularly well-known issue[1] for the otherwise highly accurate method of kd-tree matching[4]. Particularly the (for large images) excessive runtime of kd-tree matching lead to the "common knowledge" in the community that keypoint-based CM detectors are the only approaches that are also feasible in the forensic practice. Recently, this assumption has been challenged by methods that are based on new ideas from the domain of approximate nearest neighbor search.[9,10] Besides, these approaches still leave considerable room for improvement. Another, very hands-on approach to reduce the computational time of block-based algorithms is to implement selected methods on a graphics processing unit (GPU).[11]

In this paper, we investigate different matching strategies for block-based algorithms. We propose a matching procedure that explicitly exploits the underlying structure of CM forgeries. We leverage the fact that a matching procedure not necessarily has to be decomposed into multiple independent block queries, when we really search for a duplicated patch of *multiple adjacent* elements. Our procedure focusses on this spatial relation by design, which allows us to find duplications by means of bit-wise operations—and thus potentially more efficiently than through direct block-wise comparisons. In fact, the method is considerably more runtime efficient than kd-tree matching. However, its runtime efficiency is outperformed by lexicographic sorting, a classic method proposed

Further author information: kirchner@binghamton.edu, pascal.schoettle@uni-muenster.de, riess@stanford.edu

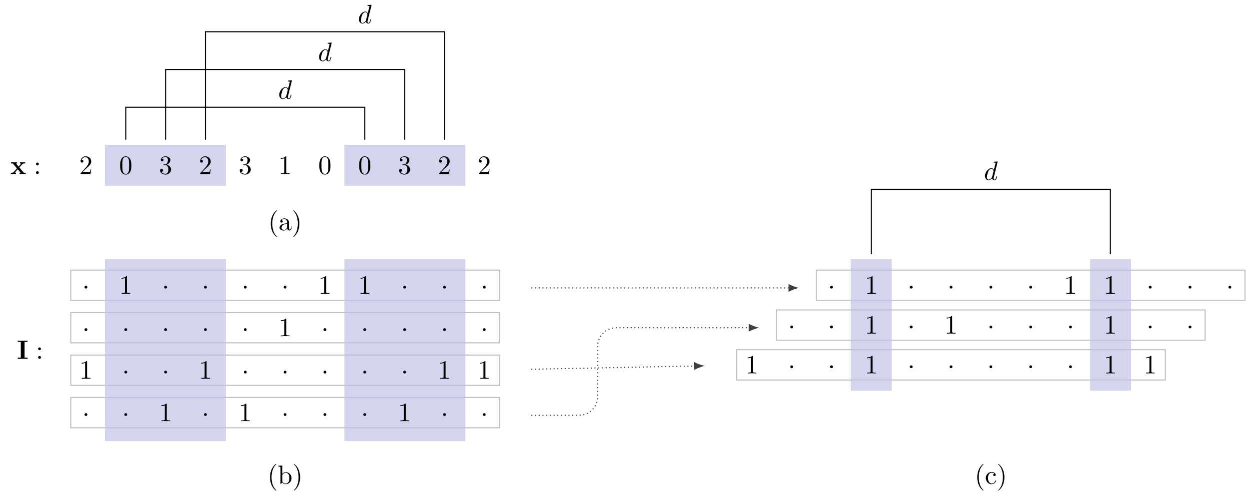


Figure 1. (a) Our approach exploits the fact that all elements within a copied sequence exhibit the same shift distance. (b) The matching problem is reformulated as identifying repeating sub-matrices in a binary index matrix. (c) Interpreting each matrix row as a binary sequence, matrix rows corresponding to consecutive input elements are horizontally realigned (i. e., shifted) and can then be compared through a simple bit-wise AND operation.

in the earliest work on CM detection,[2] and used in several follow-up works by other groups. Later analysis has shown that, for many feature sets, matching using kd-tree outperforms lexicographic sorting.[12] However, if the tremendous runtime benefit is taken into account, we demonstrate that lexicographic sorting may have considerable advantages in practical, real-world applications.

2. METHODS

Block-based copy–move forgery detection (CMFD) computes for each image block a feature vector $\mathbf{f}_i$. We assume that copied regions are essentially a subset of feature vectors $\{\mathbf{f}_i\}$, extracted from a spatially connected region and with only little difference to another subset of feature vectors $\{\mathbf{f}_j\}$ from another spatially connected region. We further assume that CMFD feature vectors can be quantized to integer values, i.e., there exists a deterministic mapping $m(\mathbf{f}_i) \mapsto x_i$, $x_i \in \mathbb{N}$, that projects feature vectors onto scalars. The CMFD matching problem then reduces to identifying duplicates of spatially connected regions in a matrix of integers. Note that this is also known as the *exact matching problem* in the CMFD literature.[2] We describe a strategy that solves this task by sequentially manipulating and analyzing a binary index matrix. We first illustrate the idea for the one-dimensional case (e. g, CMFD along a single image row or column) before we extend the approach to full 2D matching.

2.1 Basic Setting: Matching Along One Dimension

Let $\mathbf{x} = (x_0, \dots, x_{N-1})$, $0 \le x_i \le M$, denote a vector of integers, which we want to examine for copied subsequences. We use subscript $\{a : b\}$ to refer to a subsequence with contiguous indices $[a \dots b]$, i. e., $\mathbf{x}_{\{a:b\}} \equiv (x_a, \dots, x_b)$. Further, define binary $M \times N$ index matrix $\mathbf{I}$ with elements

$$I_{j,i} = \begin{cases} 1 & \text{if } x_i = j \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

and let $\mathbf{I}_{\{\cdot\},i}$ $\mathbf{I}_{j,\{\cdot\}}$ denote the i-th column and the j-th row of the matrix. Each column $\mathbf{I}_{\{\cdot\},i}$ contains exactly one non-zero element set. A row-wise summation over $\mathbf{I}$ would result in the histogram of $\mathbf{x}$.

2.1.1 General Procedure

Figure 1(a) illustrates that a copied sub-sequence of length $w+1$, $\mathbf{x}_{\{i:i+w\}}$, is a sequence whose elements repeat themselves at an unknown shift distance d, i. e., $x_{i+k} = x_{i+k+d}$, $0 \le k \le w$. Identifying such duplications is

equivalent to finding identical submatrices $\mathbf{I}_{\{\cdot\},\{i\,:\,i+w\}}$ and $\mathbf{I}_{\{\cdot\},\{i+d\,:\,i+w+d\}}$ in matrix $\mathbf{I}$, cf. Figure 1(b). We exploit this by transforming the matching task into a sequence of binary shifts and AND operations over individual rows of $\mathbf{I}$. Specifically, the core idea of our backtracking-free algorithm for inspecting whether sequence $\mathbf{x}_{\{i\,:\,i+w\}}$ has duplicates is to horizontally shift rows $\mathbf{I}_{x_i,\{\cdot\}},\ldots,\mathbf{I}_{x_{i+w},\{\cdot\}}$ relatively to each other, such that all non-zero elements $I_{x_i,i},\ldots,I_{x_{i+w},i+w}$ are vertically aligned, cf. Figure 1(c). Because all elements in a respective row are shifted by the same amount, a duplication of $\mathbf{x}_{\{i\,:\,i+w\}}$ exists, iff the Hamming weight of the binary sequence obtained as the bit-wise AND over all shifted rows exceeds 1.

A straight-forward adaption of this procedure ignores duplications within a minimum spatial distance D to reflect that very close neighbors are usually not considered as copy–move candidates (so as to avoid false positives in homogenous regions of the signal).

2.1.2 Algorithmic Details

Without loss of generality, we work on the index matrix with a reversed column order in the remainder of this text. Denote this reversed matrix $\tilde{\mathbf{I}}$, with elements

$$\tilde{I}_{j,i} = I_{j,N-1-i}\,. \tag{2}$$

Interpreting rows $\tilde{\mathbf{I}}_{j,\{\cdot\}}$ as Big Endian bit strings (i. .e, the i-th bit is set when $x_i = j$), we use their decimal representation, b_j, for the sake of notational convenience,

$$b_j = \sum_{i=0}^{N-1} 2^{\tilde{I}_{j,i}}\,. \tag{3}$$

Shifting a matrix row r elements to the right is then equivalent to computing

$$b_j^{\overrightarrow{r}} \equiv 2^{-r}\cdot b_j\,. \tag{4}$$

For completeness, let us also define the equivalent of the bit-wise AND between two matrix rows b_u and b_v as

$$b_u \wedge b_v = \sum_{n=0}^{N} 2^n\cdot\left(\lfloor b_u^{\overrightarrow{n}}\rfloor \bmod 2\right)\cdot\left(\lfloor b_v^{\overrightarrow{n}}\rfloor \bmod 2\right)\,. \tag{5}$$

Above notation allows us to establish that $\mathbf{x}_{\{i\,:\,i+w\}} = \mathbf{x}_{\{i+d\,:\,i+w+d\}}$, $d>0$, iff

$$\left(\bigwedge_{k=0}^{w} b_{x_{i+k}}^{\overrightarrow{i+k}}\right)^{\overrightarrow{d}} = 1\,, \tag{6}$$

Evaluating Eq. (6) for a specific input element x_i thus implies identifying matrix row $b_{x_{i+k}}$ that corresponds to the k-th neighboring input element, shifting the k-th row $i+k$ elements to the right, taking the bit-wise AND over all $w+1$ shifted rows and testing wether the d-th bit of the resulting bit string is set.

This general procedure is repeated for all input elements x_i to find all duplicate subsequences of length $w+1$ in the input sequence. Because each column $\mathbf{I}_{\{\cdot\},i}$ represents exactly one element only, earlier shifts don't have to be repeated in later iterations. Instead, we can re-use already shifted rows in following iterations. The number of required shifts for a specific row $b_{x_{i+k}}$, $0\le k\le w$ in the i-th iteration is obtained as

$$\sigma_{i,k} = i + k - s_{x_{i+k}}\,, \tag{7}$$

where shift vector $\boldsymbol{s} = (s_0,\ldots,s_{M-1})$ is initialized with zeros and incremented by $\sigma_{i,k}$ according to Eq. (7) after every shift operation. This effectively reduces the length of the bit strings as we iterate over the input elements and right-shift previous elements "out of scope".

Constantly updating rows by their shifted versions requires special treatment of identical elements $x_u = x_v$, $|v-u|\le w$. This situation would impose multiple right-shifts of the same row b_{x_u} within the same iteration,

Algorithm 1 BitMatch

```
1: DONE ← ∅, C ← ∅, s = 0
2: for every element x_i do
3:     if i ∉ DONE then
4:         σ_{i,0} ← i − s_{x_i} // how many bits do we need to shift?
5:         b̃_i ← b̃_i^{→σ_{i,0}} // shift
6:         s_{x_i} ← s_{x_i} + σ_{i,0} // update shift vector
7:         z ← b̃_i
8:         for k = 1; k ≤ w; k++ do
9:             σ_{i,k} ← i + k − s_{x_{i+k}} // how many bits do we need to shift?
10:            b̃_{i+k} ← b̃_{i+k}^{→σ_{i,k}} // shift
11:            s_{x_{i+k}} ← s_{x_{i+k}} + σ_{i,k} // update shift vector
12:            z ← z ∧ b̃_{i+k} // AND
13:            if z < 2^{w+1} break
14:            if k == w then
15:                d = () // empty vector of offsets
16:                scan z for set bits // at least the w-th bit and one additional bit are set
17:                if z[d+w] is set then add d + i to vector of possible offsets d
18:                for j = 0; j < length(d)−1; j++ do
19:                    DONE ← DONE ∪ d_j
20:                    for all admissible correspondences (d_j, d_j + n) do C ← C ∪ (d_j, d_j + n)
21:                end for
22:            end if
23:        end for
24:    end if
25: end for
```

setting the index matrix into an inconsistent state at the beginning of the next iteration. We circumvent this by initially augmenting $\tilde{\mathbf{I}}$ by w additional buffer columns, $\tilde{b}_j \equiv b_j^{\overrightarrow{-w}}$, so that right-shifts from the previous iteration can be reversed without information loss if necessary. Equation (6) can then be decomposed into up to $w+1$ evaluations of the following steps ($0 \leq k \leq w$):

$$\tilde{b}_{i+k} \leftarrow \tilde{b}_{i+k}^{\overrightarrow{\sigma_{i,k}}} \tag{8}$$

$$s_{x_{i+k}} \leftarrow s_{x_{i+k}} + \sigma_{i,k} \tag{9}$$

$$z_k \leftarrow \left(\tilde{b}_{i+k} \wedge z_{k-1}\right) \overset{?}{\geq} 2^{w+1} \qquad (z_{-1} = 2^{N+w} - 1) \tag{10}$$

This test procedure moves on to iteration $i+1$ if Eq. (10) does not hold (no duplications found).

If Eq. (10) holds after all w neighbors have been inspected, we declare a duplication. The resulting bit string z_{w+1} is then scanned for set bits. Their offset to the w-th bit reflects the distance d of the duplication (we start scanning at offset w and ignore buffer bits). The result is a vector of absolute offsets $\mathbf{d} = (i, i{+}d_1, i{+}d_2, \ldots, i{+}d_K)$, which is transformed into correspondence pairs $\{(i, i+d_1), (i, i+d_2) \ldots, (i, i+d_K)\}$. After removing the first element of vector $\mathbf{d}$ and adding it to the set of inspected indices, this procedure is continuously repeated, resulting in correspondence pairs $\{(i+d_1, i+d_2), \ldots, (i+d_1, i+d_K)\}$ etc. All correspondence pairs have been found when there is only one element in $\mathbf{d}$ left, and the next iteration resumes. Note that correspondence pairs can be checked for their admissibility (e. g., whether the two indices are at least D elements apart) before storing them for later inspection. We also note that already inspected indices can be ignored in all following iterations of the algorithm. Algorithm 1 outlines the general procedure of the *BitMatch* Algorithm. Figure 2 illustrates an indicative example of the first few iterations of the matching procedure with $w+1=3$.

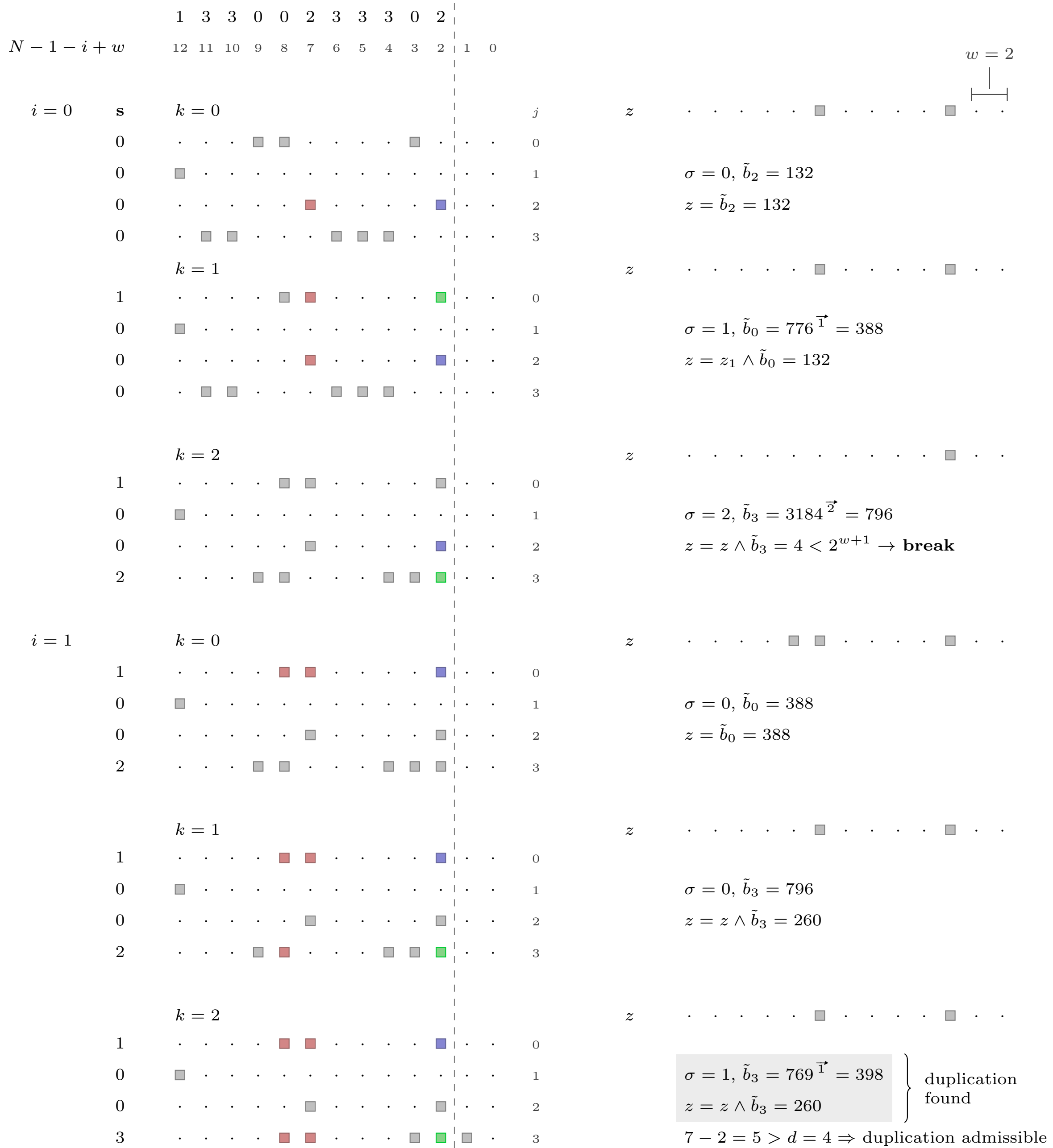


Figure 2. First two iterations of the matching procedure for input $\mathbf{x} = (2, 0, 3, 3, 3, 2, 0, 0, 3, 3, 1)$, $w = 2$ and $d = 4$. Reversed index matrix $\tilde{\mathbf{I}}$ is shown at the top ($i = 0$, $k = 0$). Squares symbolize a '1', small dots a '0'. A blue square denotes the current element x_i, a green square is the k-th neighbor, and a red square indicates potentially duplicate elements after the k-th iteration.

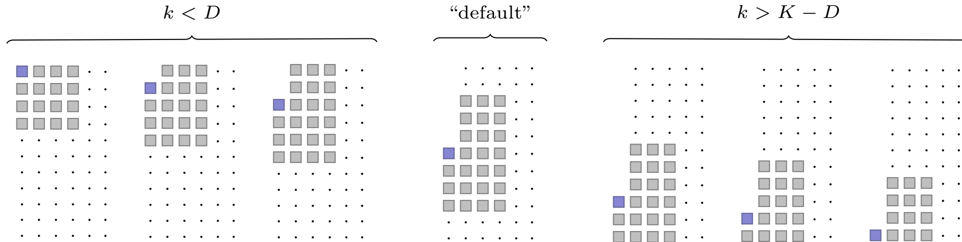


Figure 3. Inadmissible relative offsets for 2D input with $K = 11$ rows, assuming column-major vectorization, $D = 3$, and a L_∞-based distance measure. The blue square indicates the corresponding spatial position $0 \leq k < K$ of the i-th vectorized element within it's column.

2.2 Extension to Two-Dimensional Data

An extension of the above procedure to the analysis of rigid translations in two-dimensional signals is straightforward, after vectorization of the input. We assume column-major order without loss of generality, i. e., two-dimensional matrix indices (k, l) of input elements $x_{k,l}$, $0 \leq k < K$, $0 \leq l < L$, are mapped to vector indices as $(k, l) \mapsto i \equiv k + lK$, $0 \leq i < KL$. All elements of a duplicated $(w + 1) \times 1$ patch will still exhibit the same offsets with respect to the original elements after vectorization. The result is a matching procedure to find duplications of $(w + 1) \times 1$ patches. A $(w + 1) \times (w + 1)$ block consists of $w + 1$ horizontally adjacent patches.*

A mandatory adaption is the special treatment of elements close to image borders to avoid matching across column jumps. This can be achieved by excluding the last w rows from the set of indices that should be analyzed (to avoid erroneous "original" patches), in combination with a corresponding test when checking for the admissibility of a match (to eliminate cases where the original patch is located elsewhere in the image). The test for admissible correspondences requires further careful adjustments when only duplications outside a certain spatial distance D are to be considered. In contrast to the one-dimensional case, the relative offset of vectorized indices that fall into this category here depends on the row index of the input element, cf. Fig. 3. Testing whether a found relative offset d corresponds to an element x_{i+d} within spatial distance D can be implemented via a lookup table that returns the set of inadmissible relative offsets for a given row index $k = i - K\lfloor i/K \rfloor$. Assuming column-major vectorization of an input matrix with K rows, such lookup table contains a total of $\min(D + \max(0, D - w + 1), K - w)$ distinct masks.

2.3 Implementation

We have implemented a prototype of BitMatch using the GNU Multiple Precision Arithmetic Library† to handle and manipulate bit strings of arbitrary length. Source code will be available from the first author's website.

3. RESULTS

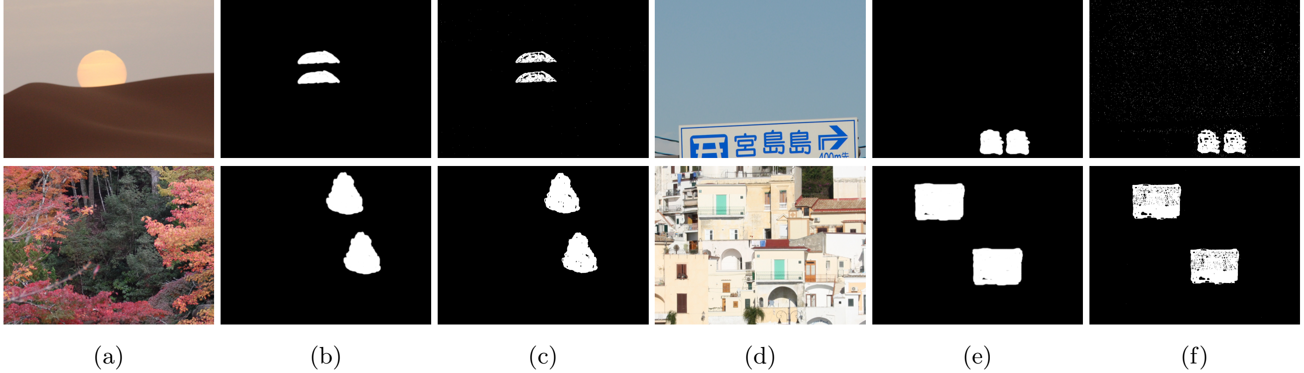
3.1 Baseline Experiment

We test the viability of the BitMatch algorithm on 80 images of size 768×1024 from the GRIP-UNINA CMFD database.[10] These uncompressed images of varying content were generated with the Erlangen CMFD toolbox,[1] each of them contains exactly one pair of duplicated regions of varying size. We set the matching procedure to work on patches of size 5×1, excluding duplications within a spatial Euclidian distance of $D = 50$. We also ignore constant patches as a rough measure against artifacts due to saturated or flat regions. Images are converted to grayscale before analysis. We inspect the shift vectors of all found correspondences and declare the largest subset with a common shift vector a manipulation.‡ The i-th and the $(i + d)$-th element in a binary correspondence

*The existence of duplicated blocks could be checked for in the post-processing phase, if needed. In theory, the described matching procedure could also be adapted to find blocks directly by extending the definition of neighboring elements and adding buffer bits correspondingly.

†`https://gmplib.org`

‡We obtained similar results when reporting all subsets with at least 3000 entries, except for a few images with very small copied regions.



(a) (b) (c) (d) (e) (f)

Figure 4. BitMatch CMFD results. (a),(d) Four images from the GRIP-UNINA CMFD database. (b),(e) Ground-truth maps. (c),(f) Obtained correspondence maps.

map are set to 1, if tuple $(i, i+d)$ is an element of this set. Correspondence maps are not subject to further post-processing. Figure 4 depicts a selection of representative results.

Denoting TP (true positives), FP (false positives) and FN (false negatives), respectively, the number of correctly marked, incorrectly marked and missed indices, the resulting pixel-level F-measure,

$$F = \frac{2\mathrm{TP}}{2\mathrm{TP} + FP + FN}\,, \tag{11}$$

averages to 0.95 over all images, with a standard deviation of 0.03. In line with the literature, we exclude ground-truth indices at semi-transparent boundaries between forgery and background. The average runtime for analyzing those images is 11.7 s on a 2.2 GHz i7 MacBook Pro (single thread). The results indicate an overall very reliable localization at an acceptable runtime.

3.2 The Rise of Lexicographic Matching

As we assume a setting where exact matching is viable, lexicographic matching[2] is a natural benchmark for our approach. The general idea of this matching procedure is to row-wise arrange feature values of overlapping blocks into a feature matrix, and to establish a lexicographic order over these rows. Duplicate blocks are found by inspecting for each row index up to R neighboring rows in the sorted matrix. In the exact matching scenario, the search for the current row index is stopped once a non-duplicate row is found. For a fair comparison, we have implemented lexicographic matching using the same general software framework as used for implementing BitMatch.

Not surprisingly, the localization accuracy of lexicographic matching increases with the number of inspected rows, R. Table 1 indicates that about 500 rows are necessary to achieve the same average accuracy as BitMatch, whereas the F-measure drops to 0.86 for $R = 10$. The strength of lexicographic matching is runtime, however. Even for $R = 500$, lexicographic matching finishes on average in only 5.7 s, although the high standard deviation indicates outliers. Figure 5(a) depicts a per-image comparison of runtimes. The graph suggests that BitMatch outperforms lexicographic matching for a few selected images, while the latter is considerably faster in the majority of cases. A closer inspection of the dataset suggests that very homogenous images are usually the ones that take longest with lexicographic matching, for instance the upper image in Fig. 4(d). At the same time, those images often result in relatively short analysis times with BitMatch. This can be expected as BitMatch is designed to find all occurrences of a specific patch at once.

3.3 The Fall of BitMatch

For a better understanding of the runtime characteristics of BitMatch and lexicographic matching, we cropped 50 randomly chosen uncompressed images from the Dresden Image Database[13] (camera model Nikon D70, demosaiced with Adobe Lightroom) to different sizes and fed them into the matching procedures,[§] keeping all other parameters

[§]Runtime does not depend on the presence of copied regions.

Table 1. BitMatch and lexicographic matching average F-measures and runtimes. 80 GRIP-UNINA CMFD database images, size 768×1024.

	F-measure	runtime	(sd)
BitMatch	0.9487	11.71598	(3.2)
Lexi10	0.8617	2.43631	(0.3)
Lexi100	0.9343	3.78639	(1.9)
Lexi500	0.9477	5.73957	(5.8)

as before. Figure 5(b) reports average runtimes for images of size 256×256, 384×384, 512×512, 768×768, 1024×1024, and 1536×1536. The graphs clearly suggest the superiority of lexicographic sorting (we set $R = 100$). While BitMatch is slightly faster for the smallest tested image size of 256×256 (0.16 s vs. 0.22 s), increasing the number of pixels by a factor of four from 768^2 to 1536^2, for instance, increases the average runtime from 6.3 s to 87.9 s (BitMatch) vs. 2.4 s to 16.1 s (lexicographic matching).

A more detailed analysis indicated that most time of a typical BitMatch call is spent on the AND operations in the nested for-loop. Up to w ANDs are necessary for each unseen index i, where the number of inspected indices depends mainly on the image size, but also on the image's smoothness. At the same time, longer signals imply longer bit strings (the index matrix has N columns), i. e., a single AND operation takes longer. All three effects are also apparent from Fig. 7(a), which depicts total runtimes of BitMatch as a function of the number of AND operations performed when analyzing each 50 images of sizes 768×768, 1024×1024, and 1536×1536, respectively. Observe that the total number of AND operations generally increases with the image size, while there is a roughly linear relationship between the number of ANDs and runtime for a fixed image size. The same number of ANDs for a larger image size is more expensive, however. In combination, the time spent for bitwise operations (shifts and ANDs) almost completely determines the time needed to finish a BitMatch call. A similar behavior can be observed for the runtime of lexicographic matching, which is mostly driven by the time needed to inspect the sorted rows for duplicates, cf. Fig. 7(d). Sorting the feature matrix adds a size-dependent, but relatively constant offset, which can be expected to be in the order of $\mathcal{O}(n \log(n))$.

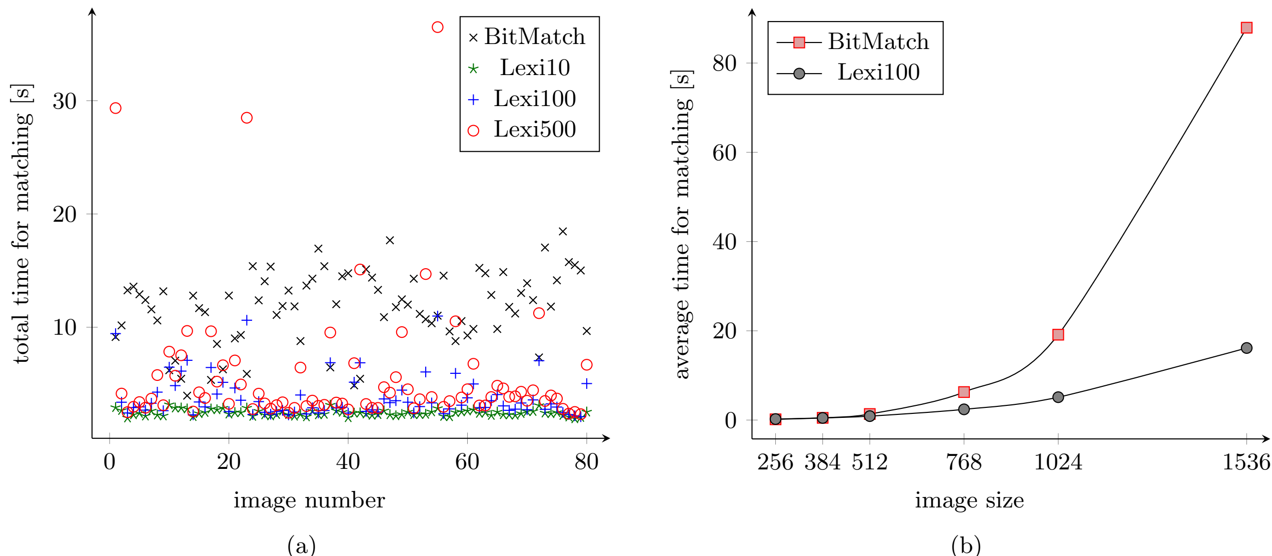


Figure 5. Runtimes of BitMatch and lexicographic matching. (a) 80 GRIP-UNINA CMFD database images, size 768×1024. (b) Average over 50 DDImgDB images of varying size.

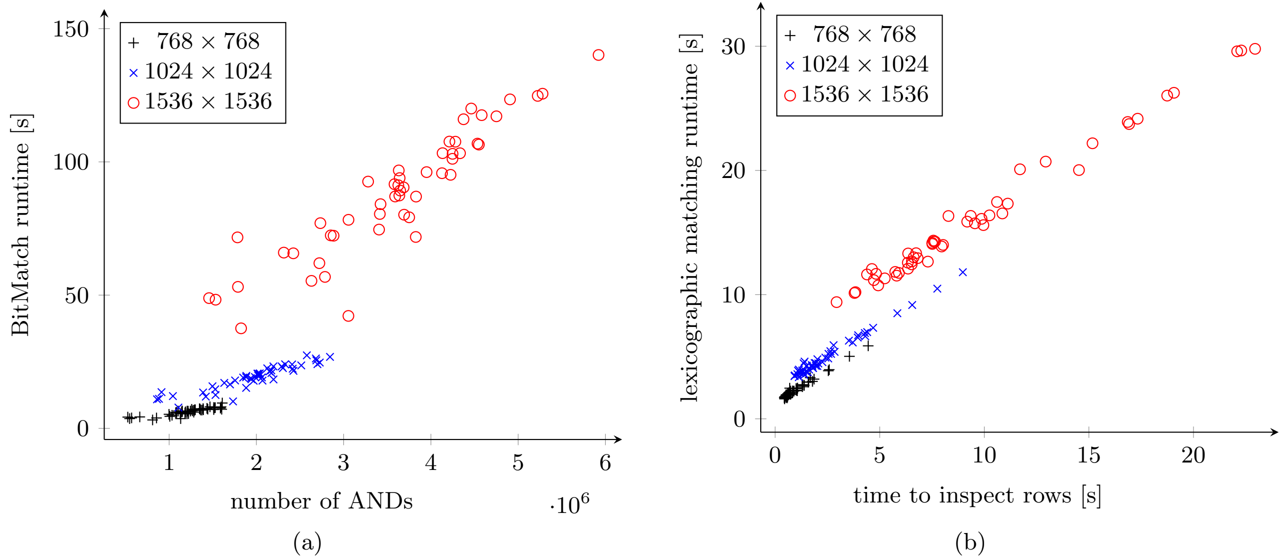


Figure 6. Runtimes of BitchMatch (a) and lexicographic matching (b) as functions of the number of AND operations and the time needed to inspect sorted rows, respectively, for different image sizes.

3.4 Back to the Roots

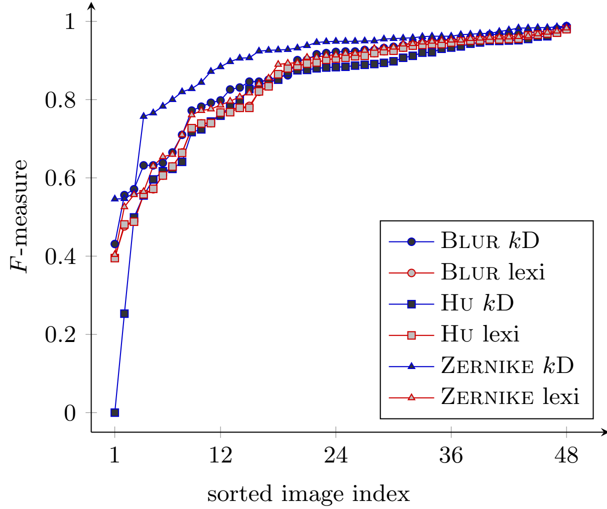
The generally very favorable runtime performance of lexicographic matching appears to be in stark contrast to the common dismissal of block-based CMFD methods as being too slow for practical analyses. Typical benchmarks choose kD-tree matching[4] for its accuracy, accepting inferior (and admittedly impractical) runtimes in the range of hours for megapixel-sized images. However, our observations above left us wondering whether lexicographic matching might be the better trade-off in practice, when taking both accuracy and runtime into account.

Figure 7 compares lexicographic matching and kD-tree matching with 13 different feature representations (cf. Tab. 2), computed from overlapping 16×16 blocks of the 48 uncompressed images in the Erlangen CMFD database.[1] These images are substantially larger than the images in the GRIP-UNINA CMFD database, with sizes assuming up to 10 megapixel. The four panels organize the different feature types into moment-based, dimensionality-reduction-based, intensity-based and frequency-based features. The graphs report per-image F-measures, for each combination of feature type and matching approach sorted in increasing order. We used the same-shift-vector approach for determining the final matches, and set the minimum number of same shifts to 800.

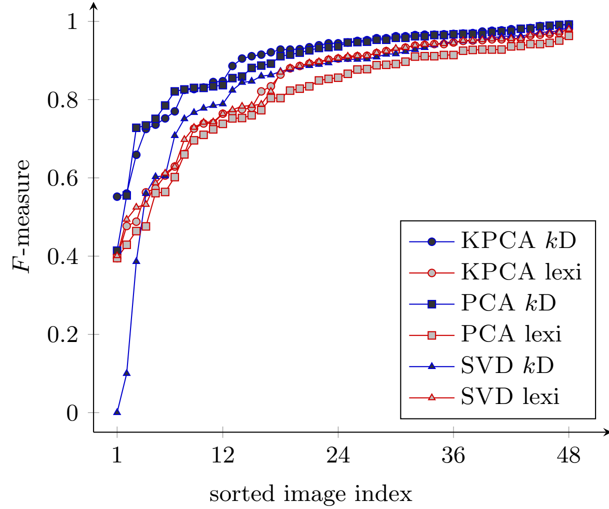
Overall, the graphs justify the use of kD-trees over lexicographic matching, when accuracy is the only concern. For the large majority of feature sets, sorting by approximate nearest neighbors outperforms lexicographic sorting. However, for most moment-based, intensity-based, and frequency-based features, the relative difference between the two matching approaches is rather small. A number of feature representations even work better with lexicographic matching (most evidently the Wang features, for which the average lexicographic matching F-measure is about 3 percentage points higher). Taking into consideration that lexicographic matching reduces the matching time from hours to minutes, a practical trade-off would most likely always opt for speed over a slightly higher accuracy, in particular when the resulting correspondence maps undergo manual inspection.

4. LESSONS LEARNED AND FUTURE WORK

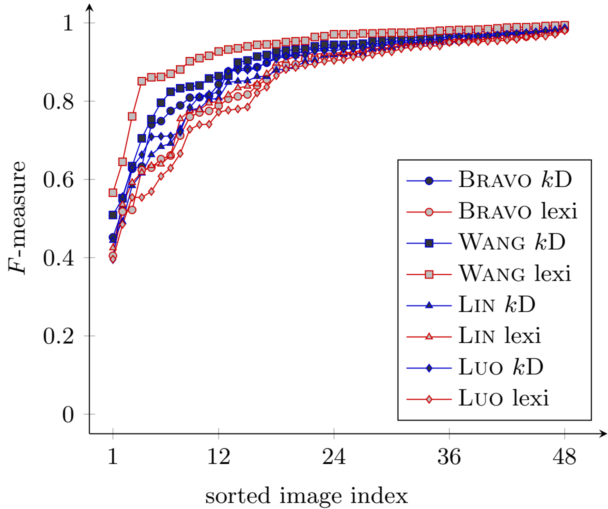
Copy–move forgery detection (CMFD)—despite being one of the most established problems in image forensics[1,2]—is still among the most actively researched topics in this community. While most efforts have traditionally focussed on suitable feature representations, more recent works have started to question the frequently postulated believe that only methods based on key points are suitable for practical analyses where runtime can be a crucial factor.[10] This paper has reported findings from an attempt to devise a novel matching procedure for a setting where the CMFD problem can be formulated as an instance of the exact matching problem.[2] Driven by the belief that



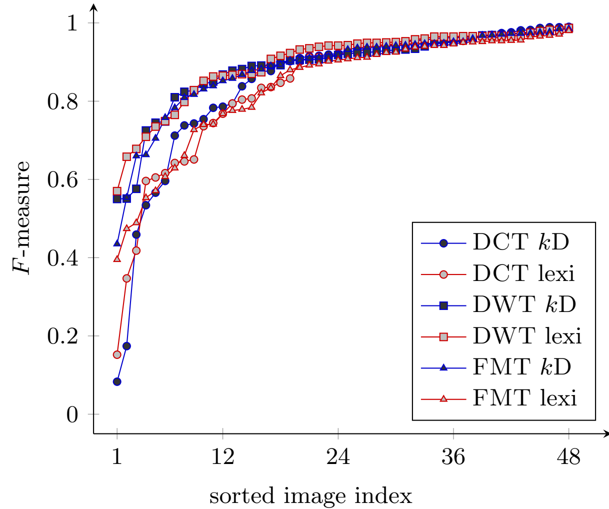
(a) moments



(b) dimensionality reduction



(c) intensity



(d) frequency

Figure 7. Comparison of F-measures for lexicographic matching (red) and kD-tree matching (blue). Erlangen database.

bitwise operations are computationally cheaper than feature space comparisons we perform the matching by manipulating and analyzing bit strings derived from the spatial relation of input elements.

Experimental results indicate that our implementation of BitMatch *can* be advantageous by a small margin, however, only under very specific circumstances (smooth images of rather small size), and only when finding *all* duplicate patches is of concern. For the majority of cases—including also the practically relevant ones—matching via lexicographic sorting of feature values turned out to be computationally much more efficient. We can only speculate that a more efficient handling of (very) long bit strings might reduce the runtime of BitMatch further, and we welcome any suggestion from the community. The admittedly high speed of lexicographic matching came much to our surprise, and it also appears to be a somewhat disregarded fact in the current literature. Except for some recent alternative works,[9,10] kD-tree matching is the method of choice. This nearest neighbor search strategy can easily consume time in the order of an hour or more for large images,[1] compared to less than a few minutes for lexicographic matching. We suspect that this latent unawareness in the recent literature (including our own work) can be explained to some degree by the "age" of lexicographic matching. When it was first

Table 2. Grouping of evaluated feature sets for copy-move forgery detection.

Group	Methods
Moments	Blur[4] Hu[14] Zernike[5]
Dimensionality reduction	PCA[3] SVD[15] KPCA[6]
Intensity	Luo[16] Bravo[17] Lin[18] Wang[19]
Frequency	DCT[2] DWT[6] FMT[20]

proposed in 2003,[2] typical computers were most likely too slow to analyze larger images in reasonable time *per se*. Over the years, lexicographic matching went out of age and got replaced by more sophisticated, but relatively much slower, methods. Considering that some of the best feature representations may indeed work better with lexicographic matching, and many others will still give reasonably good results in practice, we believe that it is important to carefully (re)evaluate what the best trade-off is, in particular also when benchmarks of new schemes claim runtime efficiency.

REFERENCES

[1] Christlein, V., Riess, C., Jordan, J., Riess, C., and Angelopoulou, E., "An evaluation of popular copy-move forgery detection approaches," *IEEE Transactions on Information Forensics and Security* **7**(6), 1841–1854 (2012).

[2] Fridrich, J., Soukal, D., and Lukáš, J., "Detection of copy-move forgery in digital images," in [*Digital Forensic Research Workshop*], (2003).

[3] Popescu, A. C. and Farid, H., "Exposing digital forgeries by detecting duplicated image regions," Technical Report TR2004-515, Department of Computer Science, Dartmouth College (2004).

[4] Mahdian, B. and Saic, S., "Detection of copy-move forgery using a method based on blur moment invariants," *Forensic Science International* **171**, 180–189 (2007).

[5] Ryu, S.-J., Lee, M.-J., and Lee, H.-K., "Detection of copy-rotate-move forgery using Zernike moments," in [*Information Hiding*], Böhme, R., Fong, P., and Safavi-Naini, R., eds., *Lecture Notes in Computer Science* **6387**, 51–65, Springer-Verlag, Berlin, Heidelberg (2010).

[6] Bashar, M., Noda, K., Ohnishi, N., and Mori, K., "Exploring duplicated regions in natural images," *IEEE Transactions on Image Processing* (Mar. 2010). Accepted for publication.

[7] Pan, X. and Lyu, S., "Region duplication detection using image feature matching," *IEEE Transactions on Information Forensics and Security* **5**(4), 857–867 (2010).

[8] Amerini, I., Ballan, L., Caldelli, R., Bimbo, A. D., and Serra, G., "A SIFT–based forensic method for copy-move attack detection and transformation recovery," *IEEE Transactions on Information Forensics and Security* **6**(3), 1099–1110 (2011).

[9] Ryu, S.-J., Kirchner, M., Lee, M.-J., and Lee, H.-K., "Rotation invariant localization of duplicated image regions based on Zernike moments," *IEEE Transactions on Information Forensics and Security* **8**(8), 1355–1370 (2013).

[10] Cozzolino, D., Poggi, G., and Verdoliva, L., "Copy-move forgery detection based on PatchMatch," in [*IEEE International Conference on Image Processing (ICIP)*], (2014).

[11] Singh, J. and Raman, B., "A high performance copy-move image forgery detection scheme on GPU," in [*Advances in Intelligent and Soft Computing*], **131**, 239–246 (Dec. 2011).

[12] Christlein, V., Riess, C., and Angelopoulou, E., "A study on features for the detection of copy-move forgeries," in [*GI SICHERHEIT*], (Oct. 2010).

[13] Gloe, T. and Böhme, R., "The Dresden image database for benchmarking digital image forensics," *Journal of Digital Forensic Practice* **3**, 150–159 (2010).

[14] Wang, J., Liu, G., Zhang, Z., Dai, Y., and Wang, Z., "Fast and robust forensics for image region-duplication forgery," *Acta Automatica Sinica* **35**, 1488—1495 (Dec. 2009).

[15] Kang, X. and Wei, S., "Identifying tampered regions using Singular Value Decomposition in digital image forensics," in [*International Conference on Computer Science and Software Engineering*], **3**, 926–930 (2008).

[16] Luo, W., Huang, J., and Qiu, G., "Robust detection of region-duplication forgery in digital images," in [*International Conference on Pattern Recognition*], **4**, 746–749 (Aug. 2006).

[17] Bravo-Solorio, S. and Nandi, A. K., "Exposing duplicated regions affected by reflection, rotation and scaling," in [*International Conference on Acoustics, Speech and Signal Processing*], 1880–1883 (May 2011).

[18] Lin, H., Wang, C., and Kao, Y., "Fast copy-move forgery detection," *WSEAS Transactions on Signal Processing* **5**(5), 188–197 (2009).

[19] Wang, J., Liu, G., Li, H., Dai, Y., and Wang, Z., "Detection of image region duplication forgery using model with circle block," in [*International Conference on Multimedia Information Networking and Security*], 25–29 (June 2009).

[20] Bayram, S., Sencar, H., and Memon, N., "An efficient and robust method for detecting copy-move forgery," in [*IEEE International Conference on Acoustics, Speech, and Signal Processing*], 1053–1056 (Apr. 2009).