Speeding-Up Graph-based Keyword Spotting by Quadtree Segmentations

Michael Stauffer[1,4], Andreas Fischer[2,3], and Kaspar Riesen[1]

[1] University of Applied Sciences and Arts Northwestern Switzerland,
Institute for Information Systems, Riggenbachstr. 16, 4600 Olten, Switzerland
{michael.stauffer,kaspar.riesen}@fhnw.ch

[2] University of Fribourg, Department of Informatics, 1700 Fribourg, Switzerland
andreas.fischer@unifr.ch

[3] University of Applied Sciences and Arts Western Switzerland,
Institute for Complex Systems, 1705 Fribourg, Switzerland

[4] University of Pretoria, Department of Informatics, Pretoria, South Africa

Abstract. Keyword Spotting (KWS) improves the accessibility to handwritten historical documents by unconstrained retrievals of keywords. The proposed KWS framework operates on segmented words that are in turn represented as graphs. The actual KWS process is based on matching graphs by means of a cubic-time graph matching algorithm. Although this matching algorithm is quite efficient, the polynomial time complexity might still be a limiting factor (especially in case of large documents). The present paper introduces a novel approach that aims at speeding up the retrieval process. The basic idea is to first segment individual graphs into smaller subgraphs by means of a quadtree procedure. Eventually, the graph matching procedure can be conducted on the resulting pairs of smaller subgraphs. In an experimental evaluation on two benchmark datasets we empirically confirm substantial speed-ups while the KWS accuracy is nearly not affected.

Keywords: Handwritten Keyword Spotting, Bipartite Graph Matching, Quadtree Graph Segmentation

1 Introduction

In the last decades, handwritten historical documents have been made increasingly digitally available around the world. Example documents are the *Barcelona marriage registry* [1], the *Saint Gall manuscript* [2] or the *George Washington letters* [3], to mention just a few. Yet, the accessibility to these documents with respect to browsing and searching is still an issue since automatic full transcriptions are often not feasible. Thus, *Keyword Spotting (KWS)* as an alternative to transcriptions has been proposed for this type of document [3–6]. KWS allows to retrieve any instances of a given keyword in a certain document. In case of historical documents, KWS is limited to document images only and is thus an *offline* task. Generally, offline KWS is regarded as the more difficult case when compared to *online* KWS, where temporal information about the writing process is available as well.

KWS approaches can be roughly distinguished into *template-based* or *learning-based* algorithms. In the case of template-based KWS, handwritten words are often represented by sequences of feature vectors used to store certain characteristics of the handwriting. A query word can then be retrieved in a set of document words by matching sequences of features vectors, for example by means of *dynamic time warping* [5, 7, 8]. Learning-based KWS on the other hand is based on a statistical model that is trained *a priori* on a relatively large set of training words [3, 6, 9]. Comparing both approaches with each other, we observe that learning-based approaches lead to higher accuracies in general, while template-based approaches are characterised by a higher flexibility (as no training is required). In the present paper, we focus on template-based KWS using graphs for the formal representation of words.

In various fields of pattern recognition, graphs have been employed as a versatile representation formalism [10–12]. Yet, for applications based on KWS, graphs are rarely used [13–18]. This is somehow surprising as graphs offer a natural and comprehensive representation for handwritten words. In particular, graphs are able to adapt both their size and structure to the complexity of the underlying handwritten words. Moreover, graphs are able to represent binary relationships that might exists between parts of the handwritten words.

In case of graph-based KWS, the actual spotting process includes matching a query graph with a set of document graphs (using some graph matching algorithm [11]). When large amounts of graph matchings are necessary, the complete KWS process might take too much time, even with fast approximate matching algorithms (e.g. [16]).

In the present paper, we focus on speeding up the graph-based KWS process by adapting the actual graph matching procedure. Graphs are first iteratively segmented into smaller subgraphs by means of a *quadtree* segmentation. Second, the graph matching is conducted on corresponding small subgraphs rather than on the large graphs representing the complete word. The rationale for this procedure is to substantially speed up the graph matching, as the matching time depends on the number of nodes of the involved graphs.

The remainder of this paper is organised as follows. In Section 2, the basic KWS framework for graph-based word representations is reviewed. The actual speed-up procedure for graph-based KWS is introduced in Section 3 and evaluated in Section 4. Finally, Section 5 concludes the paper and outlines possible future research activities.

2 Graph-based Keyword Spotting

The present paper proposes a novel method for speeding up a framework for graph-based KWS [16]. The basic KWS framework consists of four different processing steps as illustrated in Fig. 1. In the following four subsections these four steps are briefly reviewed.

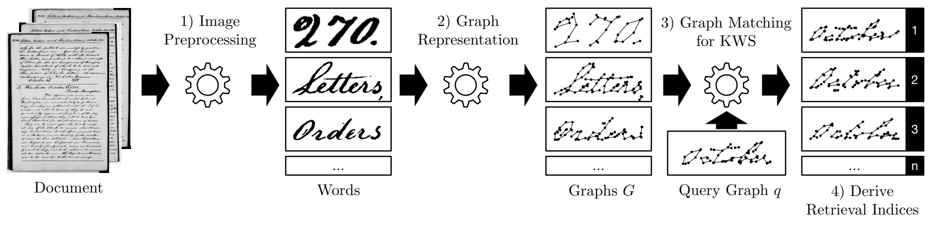


Fig. 1: Process of Graph-based Keyword Spotting of the Word "October"

2.1 Image Preprocessing

For the purpose of evaluation, we employ two historical documents, viz. the *George Washington letters (GW)* and the *Parzival manuscript (PAR)*. GW is based on twenty pages with a total of 4,894 handwritten words[5]. The letters are written in English by George Washington and his associates during the American Revolutionary War in 1755. Variations caused by both degradation and writing style are low. PAR consists of 45 pages with a total of 23,478 handwritten words[6]. The manuscript is written in Middle High German and originates in the 13th century. Variations caused by degradation are markable, while variations caused by writing style are low. Two exemplary words are given for both documents in the first row of Fig. 2.

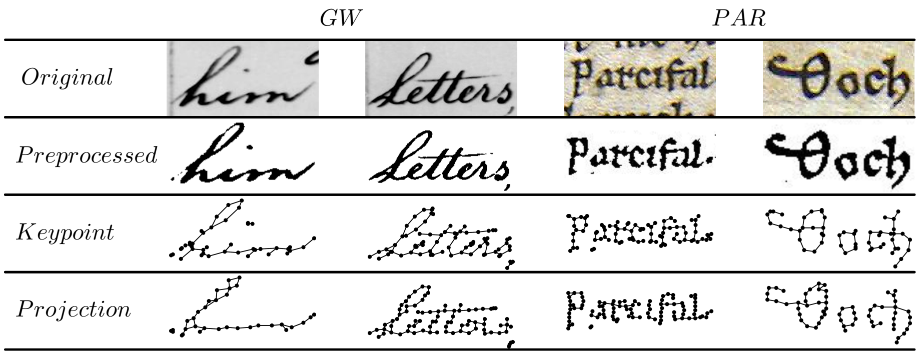


Fig. 2: Different representations of two sample words from both datasets.

The original document images are first preprocessed to reduce variations caused, for instance, by skew scanning, noisy background, and document degradation. On the basis of preprocessed document images, single word images are

[5] George Washington Papers at the Library of Congress, 1741-1799: Series 2, Letterbook 1, pp. 270-279 & 300-309, `http://memory.loc.gov/ammem/gwhtml/gwseries2.html`

[6] Parzival at IAM historical document database, `http://www.fki.inf.unibe.ch/databases/iam-historical-document-database/parzival-database`

automatically segmented by means of their projection profile (and if necessary manually corrected). In the second row of Fig. 2 we show the result of our preprocessing on some examples. For details regarding both image preprocessing and word segmentation we refer to [16].

2.2 Graph Representation

A graph g is generally defined as a four-tuple $g = (V, E, \mu, \nu)$ where V and E are finite sets of nodes and edges, and $\mu : V \rightarrow L_V$ as well as $\nu : E \rightarrow L_E$ are labelling functions for nodes and edges, respectively. Graphs can be divided into *undirected* and *directed* graphs, where pairs of nodes are either connected by undirected or directed edges, respectively. Additionally, graphs are often distinguished into *unlabelled* and *labelled* graphs. In the latter case, both nodes and edges can be labelled with an arbitrary numerical, vectorial, or symbolic label from L_v or L_e, respectively. In the former case we assume empty label alphabets, i.e. $L_v = L_e = \{\}$.

The following two graph extraction algorithms (originally presented in [17]) result in graphs where nodes are labelled with two-dimensional numerical labels, while the undirected edges remain unlabelled, i.e. $L_V = \mathbb{R}^2$ and $L_E = \{\}$.

- `Keypoint`: The first graph extraction algorithm makes use of keypoints in the word images such as start, end, and junction points. These keypoints are represented as nodes and labelled with the (x, y)-coordinates of the corresponding keypoint. Between pairs of keypoints further intermediate points (in equidistant intervals) are converted to nodes and added to the graph. Finally, undirected edges are inserted between pairs of nodes that are directly connected by a stroke.
- `Projection`: The second graph extraction algorithm is based on an adaptive segmentation of word images by means of horizontal and vertical projection profiles. A node is inserted into the graph for every segment and labelled by the (x, y)-coordinates of the centre of mass of the corresponding segment. Undirected edges are inserted into the graph for each pair of nodes that is directly connected by a stroke in the original word image.

In the third and fourth row of Fig. 2 we show the resulting graphs of `Keypoint` and `Projection`, respectively.

For both graph representations, the dynamic range of the (x, y)-coordinates of each node label $\mu(v)$ is normalised with a z-score. Formally,

$$\hat{x} = \frac{x - \mu_x}{\sigma_x} \quad \text{and} \quad \hat{y} = \frac{y - \mu_y}{\sigma_y} \qquad ,$$

where (μ_x, μ_y) and (σ_x, σ_y) represent the mean and standard deviation of all (x, y)-coordinates in the graph under consideration.

2.3 Graph Matching for Keyword Spotting

In our general KWS approach, a query graph q is individually matched with every graph g from a set of document graphs $G = \{g_1, \ldots, g_N\}$. For this particular task, we focus on inexact graph matching and employ the concept of *Graph Edit Distance (GED)* [19]. Note that any other graph matching algorithm could be used as well. Yet, GED is particularly interesting as it allows matchings of arbitrary graphs.

Given a query graph q and document graph $g \in G$, the basic principle of graph edit distance is to transform q into g using some edit operations (i.e. *insertions*, *deletions*, and *substitutions*) for both nodes and edges. A set $\{e_1, \ldots, e_k\}$ of k edit operations e_i that transform q completely into g is called an *edit path* $\lambda(q, g)$ between q and g.

To find the most suitable edit path, one commonly introduces a domain-specific cost function $c(e)$ for every edit operation e. This cost function is used to measure the degree of deformation of a given edit operation. Given an adequate cost model, the graph edit distance $d_{\text{GED}}(q, g)$, or d_{GED} for short, between q and g is defined by

$$d_{\text{GED}}(q, g) = \min_{\lambda \in \Upsilon(q,g)} \sum_{e_i \in \lambda} c(e_i) \quad ,$$

where $\Upsilon(q, g)$ denotes the set of all edit paths between q and g.

For the exact computation of d_{GED}, it is common to employ A*-based search techniques using some heuristics [20, 21]. However, these exhaustive search procedures are exponential with respect to the number of nodes of the involved graphs. Formally, GED belongs to the family of *Quadratic Assignment Problems (QAPs)* [22], which in turn belong to the class of $\mathcal{NP}$-complete problems.

In order to overcome this limitation, we make use of an approximation algorithm for the computation of GED [23]. This method basically reduces the problem of GED computation to an instance of the *Linear Sum Assignment Problem* (*LSAP*). Both QAPs and LSAPs deal with the optimal alignment of entities of two sets. Yet, by encoding the GED problem as an LSAP, we have to neglect the global edge structures of the graphs. This actually leads to a general overestimation of the true GED. However, with this transformation, we benefit from the polynomial complexity of LSAPs (see [24] for an exhaustive survey on LSAP solving algorithms). For the remainder of this paper, we make use of this graph matching algorithm and name the corresponding suboptimal graph edit distance $d_{\text{BP}}(q, g)$, or d_{BP} for short[7].

2.4 Derive Retrieval Indices

Our approach for keyword spotting relies on retrieval indices which are based on the suboptimal graph edit distance d_{BP}. We define retrieval indices for both *local* and *global* threshold scenarios. In case of local thresholds, the KWS accuracy is

[7] BP stand for bipartite (LSAPs are also termed *bipartite matching problem*).

independently measured for every keyword, while in case of global thresholds, the KWS accuracy is measured for every keyword with one single threshold.

In both scenarios, d_{BP} is first normalised by the sum of the maximum cost edit path between q and g, i.e. the sum of the edit path that results from deleting all nodes and edges of q and inserting all nodes and edges in g. Formally,

$$\hat{d}_{\mathrm{BP}}(q,g) = \frac{d_{\mathrm{BP}}(q,g)}{(|V_q| + |V_k|)\,\tau_v + (|E_q| + |E_g|)\,\tau_e} \quad ,$$

where τ_v and τ_e denote the node and edge insertion/deletion costs. In case a query consists of a set of graphs $\{q_1, \ldots, q_t\}$ that represents the same keyword, the normalised graph edit distance $\hat{d}_{\mathrm{BP}}$ is given by the minimal distance achieved on all t query graphs. This normalised graph edit distance is used to derive a first retrieval index for local thresholds by

$$r_1(q,g) = -\hat{d}_{\mathrm{BP}}(q,g) \quad .$$

To derive a retrieval index for global thresholds, $\hat{d}_{\mathrm{BP}}$ is further normalised by using the average distance of a query graph q to its k nearest document graphs, i.e. the document graphs $\{g_{(1)}, \ldots, g_{(k)}\}$ with smallest distance values to q. Formally, we use

$$\bar{d}_k(q) = \frac{1}{k}\sum_{i=1}^{k} \hat{d}_{\mathrm{BP}}(q, g_{(i)}) \quad ,$$

to derive

$$\hat{\hat{d}}_{\mathrm{BP}}(q,g) = \frac{\hat{d}_{\mathrm{BP}}(q,g)}{\bar{d}_k(q)} \quad .$$

Finally, the distance $\hat{\hat{d}}_{\mathrm{BP}}$ is used to derive the retrieval index for global thresholds by

$$r_2(q,g) = -\hat{\hat{d}}_{\mathrm{BP}}(q,g) \quad .$$

Rather than defining k as a constant, we dynamically adapt k for every query graph q. In particular, k is defined such that the distance $d_{\mathrm{BP}}(q, g_{(k)})$ of q to its k-th nearest document graph $g_{(k)}$ is equal to

$$\bar{d}_m(q) \; + \; \theta\,(\bar{d}_N(q) - \bar{d}_m(q)) \quad ,$$

where m and θ are user defined parameters and N refers to the number of document graphs. The value of $\bar{d}_m(q)$ refers to the mean distance of q to its m nearest neighbours and $\bar{d}_N(q)$ refers to the mean distance to all document graphs available. This sum reflects the level of the dissimilarities of q to the graphs in its direct neighbourhood. If the sum is large, k is automatically defined large, too. This in turn increases $\bar{d}_k(q)$, which ultimately increases the scaling to $\hat{\hat{d}}_{\mathrm{BP}}$.

3 Speeding-Up the Graph Matching

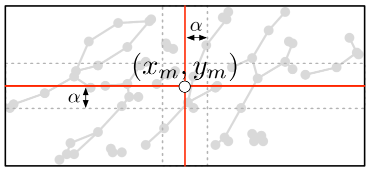
The contribution of the present paper is a novel method that aims at faster computations of pairwise graph dissimilarities. That is, we focus on reducing the time for computing d_{BP}. Basically, rather than matching complete graphs, we first apply a *quadtree* segmentation to individual graphs. Next, we match the small subgraphs (corresponding to each other w.r.t. the segmentation) and sum up the individual matching costs. This procedure might substantially speed up the graph matching procedure as the complexity of the graph matching algorithm is a cubic function of the number of nodes of the involved graphs.

The graph segmentation is carried out as follows. First, the bounding box surrounding a graph g is segmented at the *Centre of Mass* (x_m, y_m) into four segments as illustrated in Fig. 3a. To make this segmentation more robust against variations in the underlying graphs, we overlap each segment depending on a user defined factor $\alpha \in]0, 1[$. Parameter α defines the overlap of a segment to its neighbouring segments with respect to width and height of the corresponding segment. That is, for $\alpha = 0.10$, for instance, the overlapping region is 10% of the size of the corresponding segment. For each of the resulting segments one subgraph is created that includes all nodes (and edges) of the the corresponding segment. Hence, we obtain four (not necessarily disjoint) subgraphs. These subgraphs are iteratively segmented at their corresponding centre of mass into further subgraphs until the recursion level l is equal to a maximum recursion depth $r > 0$ (defined by the user). This procedure is illustrated in Fig. 3b and 3c.

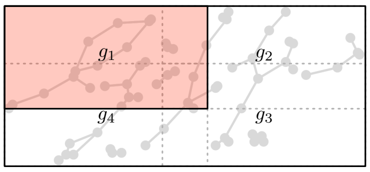
The actual procedure for computing a dissimilarity between two graphs g and g' using the proposed segmentation is formalised in Algorithm 1 (termed *Quadtree Graph Matching*). The proposed procedure is initialised by an external call with recursion level $l = 1$, i.e. $BPQ(1, g, g')$. First, both graphs g and g' are segmented into four subgraphs with the procedure described above. Each of these subgraphs represent the nodes and edges in one of the four segments under consideration (with a relative overlap of α) (see line 2 of Algorithm 1). Eventually, the sum of the four bipartite graph edit distances computed on the corresponding four subgraphs is built (see line 3). Finally, the subgraph pairs are further segmented by means of a recursive function call of BPQ (see line 6). This procedure is repeated until the current recursion level l is equal to the user-defined maximum depth r (see line 4 and 5).

Algorithm 1 Quadtree Graph Matching

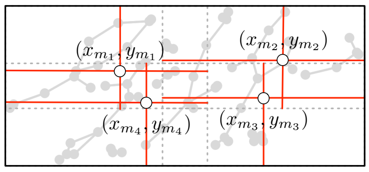
```
Input: Graphs $g$ and $g'$, overlap factor $\alpha$, maximum recursion depth $r > 0$
Output: Graph distance $d_{\text{BP}_\text{Q}}$ between graph $g$ and $g'$
1: function BPQ($l$, $g$, $g'$)
2:     Quadtree segment $g$ and $g'$ to $g_1$, $g_2$, $g_3$, $g_4$ and $g'_1$, $g'_2$, $g'_3$, $g'_4$
3:     $d_{\text{BP}_\text{Q}} = \sum_{i=1}^{4} d_{\text{BP}(g_i, g'_i)}$
4:     if $l$ equal $r$ then
5:         return $d_{\text{BP}_\text{Q}}$
6:     return $d_{\text{BP}_\text{Q}} + \left( \sum_{i=1}^{4} \text{BPQ}(l+1, g_i, g'_i) \right)$
```



(a) Recursion Level $l = 1$ with Centre of Mass (x_m, y_m) and Overlap Factor α



(b) Recursion Level $l = 1$ with Subgraphs g_1 (highlighted), g_2, g_3, and g_4



(c) Recursion Level $l = 2$ with Centres of Mass (x_{m_1}, y_{m_1}) , . . . , (x_{m_4}, y_{m_4})

Fig. 3: Quadtree Graph Segmentation.

4 Experimental Evaluation

The proposed speed-up procedure of quadtree segmentation (termed BP-Q from now on) is compared with two reference systems. First, we use the original KWS framework presented in [16] (termed BP from now on). Second, we use BP in conjunction with a fast rejection procedure recently proposed in [18] (termed BP-FR from now on). BP-FR also aims at speeding up the KWS process. Yet, this approach is based on a reduction of the number of graph matchings. In particular, pairs of query and document graphs are first compared with respect to their node distributions in a polar coordinate system. If these distributions are similar enough, the graph matching is actually carried out (otherwise the document graph is rejected without further computations).

In the following subsection, the optimisation of the proposed KWS system is described. Eventually, the results are presented and discussed in Subsection 4.2.

4.1 Optimisation of the Parameters

For the optimisation of the KWS framework, we manually select ten different keywords (with different word lengths) on both datasets (GW and PAR). Moreover, a validation set is defined consisting of 1,000 different random words including at least ten instances of all ten keyword instances. The KWS experiments are finally conducted with optimised parameter settings on the same training and test sets as proposed in [3]. In Table 1, the number of keywords, as well as the size of the training- and test set are shown for both datasets.

Table 1: Number of keywords as well as the size of the training and test set for both benchmark datasets.

Dataset	Keywords	Train	Test
GW	105	2,447	1,224
PAR	1,217	11,468	6,869

In case of global thresholds, the accuracy of KWS systems is often measured by the *Average Precision (AP)*, which is the area under the *Recall-Precision (RP)* curve for all keywords given one single global threshold. In case of local thresholds, the KWS accuracy is commonly measured by the *Mean Average Precision (MAP)*, that is the mean over the AP of each individual keyword query. In a real-world scenario, global thresholds are regarded as the more realistic but also more difficult case.

The optimal parameters for the KWS system BP, the fast rejection method BP-FR, as well as the two graph extraction methods `Keypoint` and `Projection` are adopted from previous works [16–18]. The parameters of the quadtree segmentation, i.e. the maximum recursion depth l and the overlap factor α, are optimised as follows. We evaluate five maximum recursion depths $r \in \{1, 2, 3, 4, 5\}$ in combination with 20 overlap factors $\alpha \in \{0.01, 0.02, \ldots, 0.20\}$. On both datasets and for both extraction methods a maximum recursion depth of 1 turns out to be optimal. On PAR an overlap factor α of 0.01 is optimal for both graph representations, while on GW $\alpha = 0.01$ and $\alpha = 0.02$ is optimal for `Keypoint` and `Projection`, respectively.

The retrieval index r_2 is optimised for the scenario with global thresholds. In particular, parameter m and threshold scaling factor θ are optimised. To this end, we evaluate 2,000 parameter pairs (m, θ) with $m = \{10, 20, \ldots, 990, 1000\}$ and $\theta = \{0.01, 0.02, \ldots, 0.19, 0.20\}$. In Table 2, the optimal parameter settings for r_2 are given for both graph extraction methods and benchmark datasets.

Table 2: Optimal parameters m and θ for retrieval index r_2 for both graph extraction methods and benchmark datasets.

	GW		PAR	
Method	m	θ	m	θ
`Keypoint`	70	0.01	950	0.20
`Projection`	60	0.02	1,000	0.20

4.2 Results and Discussion

In Table 3, the MAP and AP for local and global threshold scenarios are given for all three KWS systems, i.e. the original framework BP [16, 17], the BP framework with fast rejection BP-FR [18], as well as our novel procedure BP-Q. Additionally, we indicate the speed-up factor[8] as well as the relative gain or loss of the KWS accuracy of both speed-up approaches when compared with the original system BP.

Table 3: Mean average precision (MAP) using local thresholds, average precision (AP) using a global threshold, and speed-up factor (SF) for KWS using the original bipartite graph matching without rejection (BP), with fast rejection (BP-FR), and with quadtree segmentation (BP-Q). With ± we indicate the relative percental gain or loss in the accuracy of BP-FR and BP-Q when compared with BP.

		GW					PAR				
	Method	MAP	±	AP	±	SF	MAP	±	AP	±	SF
BP	`Keypoint`	66.08		55.22			62.04		60.76		
	`Projection`	61.43		49.34			66.23		62.38		
BP-FR	`Keypoint`	68.81	+4.1	54.10	−2.0	3.2	67.70	+9.1	63.01	+3.7	2.4
	`Projection`	64.65	+5.2	48.94	−0.8	2.6	72.02	+8.7	63.49	+1.8	2.3
BP-Q	`Keypoint`	65.92	−0.2	54.91	−0.6	17.1	56.83	−8.4	54.66	−10.0	21.2
	`Projection`	59.57	−3.0	48.13	−2.5	15.0	64.62	−2.4	61.72	−1.1	21.5

When compared to BP, the proposed method BP-Q achieves speed-up factors of about 15-17 and 21 on GW and PAR, respectively. This refers to a substantial improvement of the performance, especially as the previous method for speeding up the KWS process (BP-FR) leads to speed-up factors of about 2 to 3 only. However, for both datasets and both threshold scenarios an accuracy loss has to be taken into account with BP-Q, while BP-FR outperforms BP in three out of four cases. Yet, this deterioration of BP-Q in the KWS accuracy is negligible. In particular, when we consider the results of `Keypoint` on GW and `Projection` on PAR (where the relative loss of accuracy is lower than 1% and 2.5%, respectively). Hence, we can summarise that BP-Q achieves comparable results as BP but needs about 20 times less computation time for KWS.

[8] We carry out our experiments on a high performance computing cluster with dozens of 2.2GHz CPU nodes. Hence, these readings refer to the average matching time per keyword measured in a sequential scenario.

5 Conclusion and Outlook

In the present paper a procedure for speeding up graph-based keyword spotting is presented. The basic idea is to iteratively segment graphs into smaller subgraphs by means of a quadtree segmentation. These small subgraphs, rather than complete graphs, are eventually matched during the KWS process. The motivation for this procedure is to decrease the runtime of the KWS process. This is actually reasonable as the time complexity of the employed graph matching algorithm is a cubic function of the number of nodes of the involved graphs.

We compare the proposed speed-up procedure BP-Q with the original framework BP and a recent fast rejection method BP-FR on two different benchmark datasets. On both datasets, BP-Q achieves remarkable speed-up factors of 15 to 21 when compared with BP (BP-FR leads to substantially smaller speed-up factors of 2 to 3). However, these performance improvements are accomplished with a marginal loss in accuracy when compared with BP.

In future work we aim at combining both speed-up approaches BP-Q and BP-FR to further speed up the KWS process. That is, graphs might be first filtered by the fast rejection method [18] and eventually segmented and matched by means of the quadtree graph matching procedure. Moreover, we see great potential in applying our fast matching procedure in other fields of graph-based pattern recognition. Last but not least, it would be interesting to employ our general method in a parallelised computation scenario.

Acknowledgments. This work has been supported by the Hasler Foundation Switzerland.

References

1. Fernandez-Mota, D., Almazan, J., Cirera, N., Fornes, A., Llados, J.: BH2M: The Barcelona Historical, Handwritten Marriages Database. In: International Conference on Pattern Recognition. (2014) 256–261
2. Fischer, A., Frinken, V., Fornés, A., Bunke, H.: Transcription alignment of Latin manuscripts using hidden Markov models. In: Workshop on Historical Document Imaging and Processing, New York, New York, USA (2011) 29
3. Fischer, A., Keller, A., Frinken, V., Bunke, H.: Lexicon-free handwritten word spotting using character HMMs. Pattern Recognition Letters **33**(7) (2012) 934–942
4. Manmatha, R., Chengfeng Han, Riseman, E.: Word spotting: a new approach to indexing handwriting. In: Computer Vision and Pattern Recognition. (1996) 631–637
5. Rath, T., Manmatha, R.: Word image matching using dynamic time warping. In: Computer Vision and Pattern Recognition. Volume 2. (2003) II–521–II–527
6. Rodríguez-Serrano, J.A., Perronnin, F.: Handwritten word-spotting using hidden Markov models and universal vocabularies. Pattern Recognition **42**(9) (2009) 2106–2116
7. Rodriguez, J.A., Perronnin, F.: Local gradient histogram features for word spotting in unconstrained handwritten documents. In: International Conference on Frontiers in Handwriting Recognition. (2008) 7–12

8. Rodríguez-Serrano, J.A., Perronnin, F.: A model-based sequence similarity with application to handwritten word spotting. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(11) (2012) 2108–20
9. Perronnin, F., Rodriguez-Serrano, J.A.: Fisher Kernels for Handwritten Word-spotting. In: International Conference on Document Analysis and Recognition. (2009) 106–110
10. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty Years Of Graph Matching In Pattern Recognition. International Journal of Pattern Recognition and Artificial Intelligence **18**(03) (2004) 265–298
11. Riesen, K.: Structural Pattern Recognition with Graph Edit Distance. Advances in Computer Vision and Pattern Recognition, Cham (2015)
12. Stauffer, M., Tschachtli, T., Fischer, A., Riesen, K.: A Survey on Applications of Bipartite Graph Edit Distance. In: Graph-Based Representations in Pattern Recognition. (2017)
13. Wang, P., Eglin, V., Garcia, C., Largeron, C., Llados, J., Fornes, A.: A Novel Learning-Free Word Spotting Approach Based on Graph Representation. In: International Workshop on Document Analysis Systems. (2014) 207–211
14. Bui, Q.A., Visani, M., Mullot, R.: Unsupervised word spotting using a graph representation based on invariants. In: International Conference on Document Analysis and Recognition. (2015) 616–620
15. Riba, P., Llados, J., Fornes, A.: Handwritten word spotting by inexact matching of grapheme graphs. In: International Conference on Document Analysis and Recognition. (2015) 781–785
16. Stauffer, M., Fischer, A., Riesen, K.: Graph-based Keyword Spotting in Historical Handwritten Documents. In: International Workshop on Structural, Syntactic, and Statistical Pattern Recognition. (2016)
17. Stauffer, M., Fischer, A., Riesen, K.: A Novel Graph Database for Handwritten Word Images. In: International Workshop on Structural, Syntactic, and Statistical Pattern Recognition. (2016)
18. Stauffer, M., Fischer, A., Riesen, K.: Speeding-Up Graph-based Keyword Spotting in Historical Handwritten Documents. In: Graph-Based Representations in Pattern Recognition. (2017)
19. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. Pattern Recognition Letters **1**(4) (1983) 245–253
20. Berretti, S., Del Bimbo, A., Vicario, E.: Efficient matching and indexing of graph models in content-based retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence **23**(10) (2001) 1089–1105
21. Fankhauser, S., Riesen, K., Bunke, H.: Speeding Up Graph Edit Distance Computation through Fast Bipartite Matching. In: Graph-Based Representations in Pattern Recognition. (2011) 102–111
22. Koopmans, T.C., Beckmann, M.: Assignment Problems and the Location of Economic Activities. Econometrica **25**(1) (1957) 53
23. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing **27**(7) (2009) 950–959
24. Burkard, R., Dell'Amico, M., Martello, S.: Assignment Problems. (2009)