A Appendix

A.1 A tighter approximate k-motiflets algorithm

We discuss the pseudo-code of the proposed approximate local motiflet discovery algorithm. The algorithm uses a greedy search and evaluates candidate motiflets using extent, while MOTIFLETS uses a radius-based metric with leads to lower-quality results as illustrated in Fig. 4.

Algorithm. The procedure K_MOTIFLETS_HEAP is shown in Algorithm 2. We represent a motif $S_k = \{T_{i_1,l}, T_{i_2,l}, \ldots, T_{i_k,l}\}$ as $S_k = \{i_1, i_2, \ldots, i_k\}$ where each subsequence $T_{i,l}$ is represented by its starting index. The output $\mathcal{R}$ consists of at most $\hat{n} = n - l + 1$ motifs ranked by extent to enable *local* k-motiflet discovery (see Section 4.2). We start by iterating $\hat{n}$ times over each possible subsequence i (line 3). In each iteration, we aim to find the $k-1$ subsequences that result in the lowest maximal pairwise distance (lines 4–10). During the $k-1$ greedy search iterations, we select the next candidate subsequence that minimises extent and discard trivial-matching subsequences (lines 7-8). We filter out duplicate k-motiflets, i.e. different cores leading to an identical motif set (line 10). Finally, we return the list of k-motiflets sorted by extent.

Time complexity. The main loop is executed $\mathcal{O}(n)$ times. Greedy search optimising the *extent* requires $\mathcal{O}(k^2 \cdot n)$-time. The total complexity is $\mathcal{O}(k^2 \cdot n^2)$.

Algorithm 2: Discover Local k-Motiflets

Input: A time series T, distance matrix D, motif length l, motif size k
Result: List of at most $\hat{n}$ motifs sorted by extent.

```
1  procedure K_MOTIFLETS_HEAP(T, D, l, k)
2      R ← {}
3      for i ← 1 to n − l + 1 do
4          S ← {i}
5          C ← REMOVE_TRIVIAL_MATCHES({1, 2, ..., n − l + 1}, i)
6          for iter ← 2 to k do
7              j_k ← argmin_{j∈C} extent(S ∪ {j})
8              C ← REMOVE_TRIVIAL_MATCHES(C, j_k)
9              S ← S ∪ {j_k}
10         R ← R ∪ {S}
11     R ← sort R on extent
12     return R
```

A.2 An efficient exact k-motiflets algorithm

We present the pseudo-code of an exact algorithm for discovering k-motiflets. First, we introduce a second theorem.

Theorem 2. *For each candidate motif of size k, $S_k = \{T_{i,l}, \ldots, T_{k,l}\}$ with extent d the following holds: if $|\{T_{j,l} \mid T_{j,l} \in T : z\text{-}ED(T_{i,l}, T_{j,l}) < d\}| < k \iff extent(S_k) > d$.*

Proof: If fewer than k non-trivially matching subsequences have a distance to the core $T_{i,l}$ smaller than d, then at least one subsequence in the candidate motif S_k has a distance larger than d. Consequently, the radius to the core and the maximal pairwise distance, or extent, must have a distance greater than d.

The exact algorithm employs *depth-first search* to construct candidate motif sets incrementally from the bottom up, pruning supermotifs based on Theorem 1. Additionally, we apply Theorem 2 for initial pruning. For simplicity, we assume the algorithm discovers the top-1 k-motiflet.

Algorithm. The main and auxiliary procedure are shown in Algorithm 3. First, we initialise the best motiflet and its extent d using the approximate k-motiflet solution (lines 2-3). Next, we initiate a recursive depth-first search over all motifs smaller or equal to size k (line 5). During recursion, we track the current candidate motif S, its extent, and the set $\mathcal{C}$ of candidate subsequences that can be added to S. In MOTIFLETS_DFS, we first check if the size of S is k and if its extent is smaller than the current best extent, thereby possibly improving on the approximate solution and decreasing the upper bound for pruning future candidates (lines 9-10). Next, we *prune* candidate motifs of size $\leqslant k$ if the current extent is greater or equal than d using Theorem 1 (line 12). For singleton motifs we update $\mathcal{C}$ by filtering candidate subsequences with a distance less than d and apply Theorem 2 for pruning (lines 14-17) Next, we get the next candidate subsequence i from $\mathcal{C}$ that does not trivially match any subsequence in S (line 18). We then compute

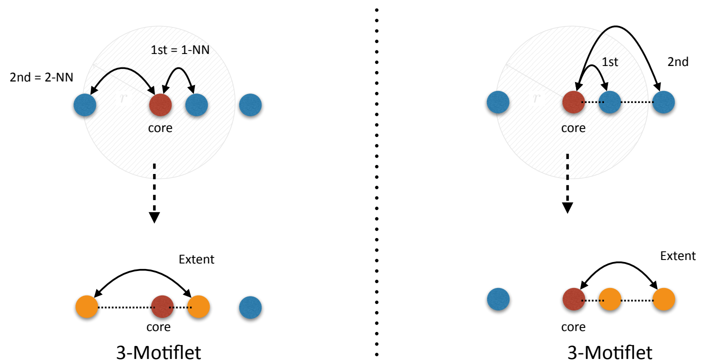

Fig. 4: Approximate k-motiflet discovery using the same 4 data points and either (left) radius to the core (right) extent of the set. Left: For a core subsequence (in red) the two nearest subsequences are retrieved [12]. Right: For a core subsequence we perform greedy search such that the maximal pairwise distance is minimal. Bottom: The resulting 3-motiflet on the right has a lower extent.

Algorithm 3: Discover Exact k-Motiflets

```
Input : A time series T, distance matrix D, motif length l, motif size k
Result: k-motiflet, extent
1  procedure K_MOTIFLETS_EXACT(T, D, l, k)
2      L ← K_MOTIFLETS_HEAP(T, D, l, k)
3      (motiflet, d) ← L[0]
4      C ← {1, 2, ..., n − l + 1}
5      MOTIFLETS_DFS({}, inf, C)
6      return (motiflet, d)
7  procedure MOTIFLETS_DFS(S, extent, C)
8      if |S| = k then
9          if extent < d then                // Update lower bound for extent
10             (motiflets, d) ← (S, extent)
11     else
12         if extent ⩾ d or |C| = 0 then         // Prune using theorem 1
13             return
14         if |S| = 1 then
15             C ← {T_{i,l} | T_{i,l} ∈ T : z-ED(T_{i,l}, S[0]) < d}
16             if |C| < k then                    // Prune using theorem 2
17                 continue
18         i ← NEXT_INDEX(C)
19         inc_extent ← max({D_{i,j} | ∀j ∈ S} ∪ {extent})
20         MOTIFLETS_DFS(S ∪ {i}, inc_extent, C\{i})
21         MOTIFLETS_DFS(S, extent, C\{i})
```

the extent of $S \cup \{i\}$ incrementally (line 19). Finally, we recursively search for all supermotifs with and without the next non-trivially matching subsequence in $\mathcal{C}$ (lines 20-21).

Time complexity. The worst-case time complexity is $\mathcal{O}(n^k)$, assuming trivial matches are ignored. The algorithm is memory-efficient by keeping at most k motifs in memory. In Section 6, we demonstrate that the algorithm is an order of magnitude faster.

A.3 An approximate algorithm for verifying self-sufficient motifs

To verify self-sufficient motifs, we employ an approximate solution based on the Minimum Spanning Tree (MST), thereby avoiding the costly generation of all possible subsets of a motif.

Algorithm. We begin by constructing an MST using Prim's algorithm, which adds edges between nodes one by one while avoiding cycles. In this MST, each node represents a subsequence, and each edge corresponds to the z-normalized Euclidean distance between subsequences. Next, we generate the k-1 possible partitions of the MST by removing each edge in turn, ensuring that each partition contains at least two nodes (or subsequences). Finally, we verify whether the sum of the extents of each partition is smaller than the extent of the candidate motif S_k.

Time complexity. The time complexity for generating the MST using Prim's algorithm is $\mathcal{O}(k \log k)$. The approximate solution for verifying self-sufficiency has a time complexity of $\mathcal{O}(k^3)$. In contrast, an exact solution requires $\mathcal{O}(2^k)$ time.

A.4 MotiSet algorithm for discovering the best set of motifs

We present the pseudo-code of the proposed MotiSet algorithm, which uses A* search to enumerate all sets of motifs and select the set with the lowest value.

Algorithm. Algorithm 4 presents the MotiSet algorithm. We begin by computing the distance matrix and identifying local motiflets using the algorithm described in Section 4.3. Specifically, we call k_motiflets_heap once with the maximum motif size k_{max} and obtain an index containing local motiflets of varying sizes. We iterate over motif sizes from 2 to k_{max} and remove vertically redundant motifs, i.e., motifs of the same size that are redundant (lines 7–9). The final set of candidate motifs is used by AStarMotif. AStarMotif intializes an empty queue with each candidate motif as a singleton set. We prune candidate motifs before adding them to this singleton set by removing all horizontally redundant candidates, i.e., motifs of different sizes that are redundant. Additionally, we enforce an ordering among motifs, denoted by $\prec$, to prevent generating motif sequences instead of sets (line 15). We determine a global ordering by sorting all motifs based on the index of their first subsequence. Next, we start the A* search (lines 18–27). In the main loop, we take the candidate set of motifs $\mathbf{S}$ with the lowest value $f(\mathbf{S})$ (see Def. 12) and the set of possible motifs for expansion, $\mathcal{C}_{\mathbf{S}}$. Next, we check boundary conditions and terminate the current branch if there are no more candidate motifs to add. We also stop if the current set contains m motifs if the optional parameter m is defined (line 20-22). If no stopping condition apply, we continue the search. We expand the current set of motifs by adding the first motif from the remaining candidate motifs and adding the new set to the priority queue. As before, we also prune possible future candidate motifs on horizontal redundancy and using the global ordering (line 25). Finally, we return the set of motifs with the overall lowest value (28).

Time complexity. The computational complexity of constructing the distance matrix and identifying motiflets is the same as in MotiPlus. The complexity of A* search depends on the cardinality of $\mathcal{C}$ and the effectiveness of vertical and horizontal pruning. In the worst case, there are $O(k \cdot n)$ candidate motifs in $\mathcal{C}$. Checking vertically and horizontal non-redundant motifs takes linear time for each motif. Evaluating the value function takes $\mathcal{O}(m \cdot (k^2 + n))$-time but can be trivially cached for each motif. The theoretical worst-case time complexity of A* is $O((k \cdot n)^m)$. We note that in practice, we suggest to stop after n iterations, e.g. $n = 10\,000$.

Algorithm 4: MotiSet: Discovering the best set of m motifs

Input: A time series T, motif length l, number of motifs m (optional), the maximum motif size k_{max}

Result: Set $\mathbf{S}$ with up to m motifs

procedure MotiSet(T, l, m, k_{max})
    $D \leftarrow$ calc_distance_matrix(T, l)
    index $\leftarrow$ k_motiflets_heap(T, D, l, k_{max})
    $\mathcal{C} \leftarrow \{\}$
    for $k \leftarrow 2$ **to** k_{max} **do**
        $\mathcal{C}_k \leftarrow \{\}$
        for $S_k \in index[k]$ *sorted on extent* **do**
            if *not-vert-redundant*($S_k, \mathcal{C}_k$) **then**
                $\mathcal{C}_k \leftarrow \mathcal{C}_k \cup \{S_k\}$
        $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_k$
    return AStarMotifs($T, \mathcal{C}, m$)

procedure AStarMotifs($T, \mathcal{C}, m$)
    $Q \leftarrow$ *create empty priority queue*
    for $S \in \mathcal{C}$ **do**
        $\mathcal{C}_{\mathbf{S}} \leftarrow \{S' \in \mathcal{C} \mid S \prec S' \wedge \textit{not-hor-redundant}(S', S)\}$
        push($Q, \langle \{S\}, f(S), \mathcal{C}_{\mathbf{S}} \rangle$)
    $\mathcal{R} \leftarrow \{\}$
    while $Q \neq \varnothing$ **do**
        $\langle \mathbf{S}, f(\mathbf{S}), \mathcal{C}_{\mathbf{S}} \rangle \leftarrow$ pop(Q)
        if $\mathcal{C}_{\mathbf{S}} = \varnothing$ **or** $|\mathbf{S}| = m$ **then**
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathbf{S}\}$
            continue
        $\hat{S} \leftarrow$ first($\mathcal{C}_{\mathbf{S}}$)
        $\hat{\mathbf{S}} \leftarrow \mathbf{S} \cup \{\hat{S}\}$
        $\hat{\mathcal{C}}_{\mathbf{S}} \leftarrow \{S' \in \mathcal{C}_{\mathbf{S}} \mid \hat{S} \prec S' \wedge \textit{not-hor-redundant}(S', \hat{S})\}$
        push($Q, \langle \mathbf{S}, f(\mathbf{S}), \mathcal{C}_{\mathbf{S}} \backslash \{\hat{S}\} \rangle$)
        push($Q, \langle \hat{\mathbf{S}}, f(\hat{\mathbf{S}}), \hat{\mathcal{C}}_{\mathbf{S}} \rangle$)
    return $\underset{\mathbf{S} \in \mathcal{R}}{argmax}\ f(\mathbf{S})$

A.5 The runtime and the quality for discovering motiflets using the exact and approximate algorithm

In this section, we report detailed results comparing the approximate and exact algorithms for discovering motiflets. The results concerning quality are reported in Fig. 5. We note that on these smaller datasets ($n = 1000$), the difference in runtime is negligible, with both methods completing in under one second. We conclude that the proposed approximate algorithm closely approximates the exact solution for discovering k-motiflets, which becomes intractable for larger values of k and n.

In a second experiment, we evaluate the runtime of the exact algorithm for discovering motiflets. We ran this experiment on a laptop with a 8-Core Intel CPU and 16 GB of RAM. The runtime in seconds for varying k of the first three datasets is shown in Fig. 6. The original exact algorithm enumerates all possible n^k subsets if pruning fails using Theorem 2 [12]. As a result, the original algorithm takes several days to complete and runs out of memory for $k = 15$. We find that the proposed exact k-motiflet algorithm completes within a reasonable time for k up to 15. We note that in real-world time series, the maximal cardinality of

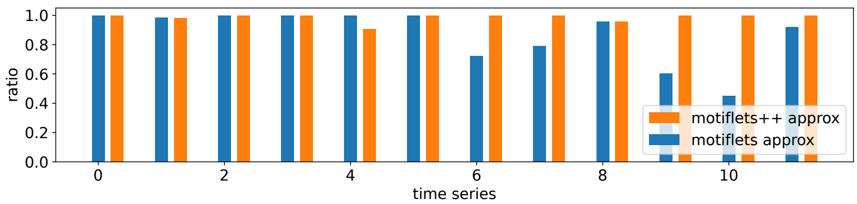

Fig. 5: Comparing the quality of the approximate k-motiflet discovery algorithms on 12 benchmark TS [7]. We report the ratio of the extent of the top-1 motif found by the approximate algorithm to that of the exact algorithm. We find that the proposed algorithm discovers motiflets that closely match those found by the exact algorithm in all TS instances.

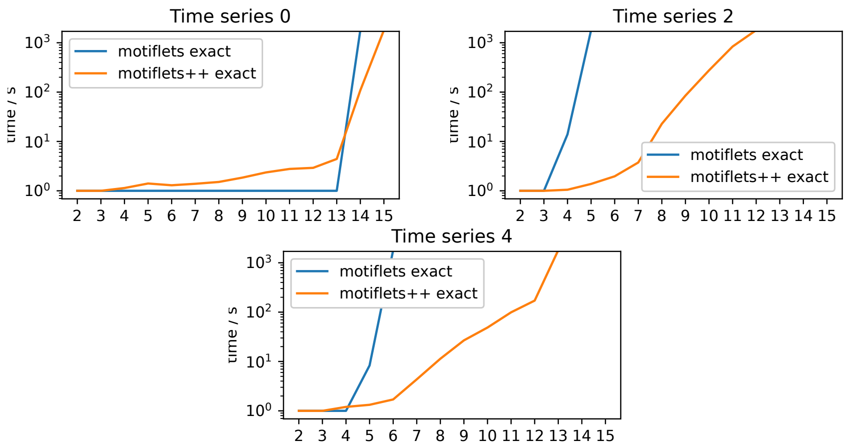

Fig. 6: Comparison of the runtime for the discovery of k-motiflets for increasing values of k using the proposed exact method and the brute-force baseline on the first three datasets from [7]. The baseline method takes more than 30 minutes for $k = 6$ on TS 2 and 4, whereas the proposed exact method completes for all values of k.

motifs is often limited. For example, in the *tsdm-benchmark*, a maximum value of $k = 10$ is sufficient.

A.6 Selecting hyper-parameters on the TSMD-benchmark

In Table 3, we summarize the parameter settings for each motif discovery method, optimized via grid search. We use l_{gt}, k_{gt}, and m_{gt} to denote the fixed motif length, maximum motif size, and maximum motif count from the ground truth, respectively. Default parameters were chosen based on preliminary experiments to maximize accuracy. For each dataset, we use the same free parameter settings across all time series instances, identified through grid search on validation data following [15].

LoCoMotif and MMotif depend on sensitive distance thresholds, while Grammarviz requires selecting an optimal alphabet and word size for discretization—both of which require careful tuning per dataset. In contrast, MotiPlus and MotiSet need only three parameters: motif length, (optional) number of motifs, and maximum motif size. The window length is selected automatically by maximizing the area under the curve, and the maximum size mainly impacts runtime. A default value of $k_{max} = 10$ performs well across datasets. Finally, we note that exhaustive tuning is generally infeasible in unsupervised motif discovery.

`Dataset`	Free parameters	Default parameters	Fixed parameters
Grammarviz	$a \in \{2, 4, 8, 16\}$, $w \in 2, \dots, \frac{l}{2}$		$l = l_{gt}$
MMotif	$r \in \{1, 2, 3, 4, 5\}$		$m = m_{gt}$, $l = l_{gt}$
LocoMotif	$\rho \in \{0.1, 0.2, \dots, 0.9\}$	*warping* = *True*	$m = m_{gt}$, $l = l_{gt}$
Motiflets	$k_{max} \in [k_{gt} - 3, k_{gt} + 3]$		$m = m_{gt}$, $l = l_{gt}$
MotiPlus	$k_{max} \in [k_{gt} - 3, k_{gt} + 3]$	*self_sufficient* = *True*, *k_closed* = *True*	$m = m_{gt}$, $l = l_{gt}$
MotiSet	$k_{max} \in [k_{gt} - 3, k_{gt} + 3]$,	$n = 10\,000$, $\gamma = 0.2$	$m = m_{gt}$, $l = l_{gt}$

Table 3: Parameter ranges for grid search in experiments.