Attractor Matching: A New Paradigm for Structural String Comparison

Simone Faro[§], Dominik Köppl[†] and Francesco Pio Marino[§,‡]
[§]University of Catania [†]University of Yamanashi [‡]University of Rouen

Abstract

We introduce *Attractor Matching*, a new framework for structural string comparison built upon the theory of string attractors. Given a pattern x of length m and one of its attractors Γ_x, the problem asks for all substrings $y[i..i+m-1]$ of a text y such that Γ_x is also an attractor of $y[i..i+m-1]$. Unlike classical notions of string matching, which rely on character equality or distance measures, attractor matching focuses on the structural properties that govern repetitiveness and compressibility. Our contribution is fourfold. First, we adapt the *IsAttractor* algorithm of Béal *et al.* by combining the DAWG with the sliding-window technique of Blumer, enabling online attractor verification as the window advances over the text. Second, we reformulate the verification procedure on the *Compressed DAWG* (CDAWG), obtaining a more compact representation that preserves correctness. Third, we employ the sliding-window CDAWG method of Inenaga *et al.*, which allows efficient attractor matching on sliding-window maintained CDAWGs with incremental updates. Finally, we introduce a relaxed variant, *Attractor Matching with Mismatches*, where the pattern attractor may be extended by at most ρ additional positions, enabling structurally tolerant matching. This paradigm bridges compression and similarity, opening new directions for structure-aware pattern matching.

1 Introduction

Classical string matching is one of the most fundamental problems in computer science: given a pattern x of size m and a text y of size n, the goal is to report all positions where x occurs as a contiguous substring in y [1]. Over time, many generalizations of this basic framework have been proposed to capture broader notions of similarity, reflecting both the symbolic content and the underlying structural patterns of strings. Most of these frameworks are based on *Substring-Consistent Equivalence Relations (SCERs)* [2]. These include *parameterized matching* [3], where two strings match under a consistent renaming of symbols; *order-preserving matching* [4], which compares the relative order of numeric symbols; and palindrome pattern matching [5] where the lengths of the maximal palindromes centered at corresponding positions in the strings are equal.

String attractors [6, 7] have recently emerged as a powerful and unifying concept in the theory of text compression and combinatorics on words. An attractor of a string w is a set of positions $\Gamma \subseteq [1..|w|]$ such that every distinct substring of w crosses at least one position of Γ. This notion captures the essence of repetitiveness: any substring must "touch" the attractor, and thus the attractor size provides a concise measure of the string's structural complexity. Since their introduction, attractors have been shown to characterize several measures of compressibility, including grammar compression, Lempel-Ziv parsing, and run-length encoded Burrows-Wheeler transforms, up to logarithmic factors.

Despite the increasing theoretical understanding of attractors, their algorithmic use has so far been mostly confined to representation rather than comparison: attractors are typically used to compress a single string or to bound the size of compressed indices. However, one may naturally ask a complementary and so far unexplored question: *can attractors be used to define a meaningful notion of similarity between strings?*

In this paper, we address this question by introducing the concept of *Attractor Matching*. Given a pattern x of size m and its attractor Γ_x, we aim to find all substrings $y[i..i+m-1]$ of a text y of size $n > m$ such that Γ_x is also an attractor of $y[i..i+m-1]$. In other words, we look for substrings of y that share the same structural coverage property as the pattern: every distinct substring of $y[i..i+m-1]$ must intersect Γ_x. This form of comparison ignores the specific alphabet symbols and instead focuses on the structural organization of repetition within the strings. It thus defines a new paradigm of string comparison based not on characters or distances, but on the *attractor structure* that governs compressibility.

From an algorithmic standpoint, we build upon the *directed acyclic word graph (DAWG)* of Blumer [8]. The DAWG is the minimal automaton recognizing all substrings of a word. Blumer [9] studied the maintainance of the DAWG under a sliding window of length m. Her work established that such a DAWG can be maintained in $O(nm)$ worst-case time and $O(n \log m)$ expected time under random text assumptions, using only $O(m)$ space, for constant-sized alphabets. Subsequently, Fujishige *et al.* [10] obtained the same $O(nm)$ worst-case time bounds for integer alphabets.

We leverage this DAWG model to realize an online version of attractor matching. In our setting, the window length corresponds to the pattern size, that is, $k = m$. As the window slides, the DAWG is updated to represent all substrings of the new window, and the IsAttractor procedure—originally defined by Béal *et al.* [11] to check whether a set of positions forms an attractor of a word—is applied to test whether the pattern's attractor Γ_x attracts the current substring. This integration of attractor theory with sliding-window automata enables attractor-based similarity search to be performed in streaming text.

Formally, the resulting algorithm solves the AttractorMatching problem with the following complexity: $T(n, m) = O(nm)$ time in the worst case and $O(m)$ space complexity. This establishes that attractor matching can be implemented online without asymptotic overhead compared to substring recognition.

Beyond its immediate algorithmic implications, attractor matching provides a conceptual bridge between two historically distinct areas: the combinatorics of compression and the theory of string matching. It invites a reinterpretation of similarity that is not based on symbol identity, but on the shared structural "skeleton" of repetition. We believe this perspective opens a new research direction in string algorithms, where compressibility itself becomes the matching criterion.

Beyond its theoretical appeal, attractor matching opens the way to a broad range of applications where structural, rather than symbolic, similarity is of interest. In compression and information theory, it enables the comparison of data streams based on their intrinsic repetitiveness, allowing the detection of regions that exhibit similar compressibility even when their symbols differ. This perspective may prove useful in compression-aware similarity search, where one aims to recognize equivalent structures across distinct encodings or to identify repetitive fragments in massive texts without explicit decompression. In this sense, attractor matching provides a new tool for analyzing the geometry of redundancy within data, complementing existing paradigms based on character identity or edit operations.

The same idea extends naturally to domains where structural regularity carries semantic meaning. In computational biology, attractor matching could identify genomic regions that share organizational patterns of repetition—such as tandem repeats or gene duplications—even in the absence of strong sequence conservation. In software analysis, it may reveal structurally similar code fragments that differ only by renaming or reordering, supporting clone detection and structural deduplication. More broadly, because the underlying algorithm operates in an online fashion, it lends itself to real-time analysis of streaming data, where monitoring changes in attractor structure can serve as a signal of evolving patterns or anomalies. These applications suggest that attractor matching is not merely a theoretical construct, but a new lens through which the structure of complex strings can be compared and understood.

The remainder of the paper is organized as follows. Section 2 recalls the main definitions related to string attractors and sliding DAWGs. Section 3 presents the attractor matching algorithm and its analysis. Section 4 reformulates the attractor verification procedure of Béal *et al.* on the Compressed DAWG (CDAWG), showing how the structure preserves correctness while

improving compactness. Section 5 extends this formulation to a sliding-window setting, exploiting the incremental maintenance of the CDAWG to obtain a faster attractor matching algorithm. In Section 6, we further extend this matching paradigm to efficiently handle mismatches, providing a relaxed version of the algorithm. Finally, in Section 7, we draw our conclusions.

2 Preliminaries

We consider a finite ordered alphabet Σ of integer size σ. For a string $w \in \Sigma^*$ of length $n = |w|$, we write $w[i]$ for its i-th character and $w[i..j]$ for the substring spanning positions i through j (inclusive), where $1 \leq i \leq j \leq n$. We denote by $\text{Substr}(w)$ the set of all distinct substrings of w. Positions are 1-indexed throughout this paper for convenience.

The *Directed Acyclic Word Graph* (DAWG) of a string w, also known as the *suffix automaton*, is the minimal deterministic finite automaton that recognizes $\text{Substr}(w)$. It has $O(n)$ states and transitions and can be built online in linear time by extending the automaton one character at a time. Each state of the DAWG corresponds to an *end-position equivalence class* of substrings, that is, the set of substrings sharing the same rightmost occurrence in w. We denote by $\text{DAWG}(w)$ the DAWG of string w.

The *Compressed Directed Acyclic Word Graph* (CDAWG) [12] is a compacted version of the DAWG obtained by merging nodes that correspond to maximal repeats of the string. While the DAWG already provides a minimal deterministic automaton for all substrings, the CDAWG further compresses the structure by collapsing chains of non-branching nodes, yielding a representation proportional to the number of distinct right-maximal substrings of w. As a result, the CDAWG has size $O(e)$, where e denotes the number of distinct right-maximal substrings, and can be built in linear time for integer alphabets. This compactness makes it especially suitable for verifying attractor properties and maintaining substring information under sliding-window operations, as explored in Section 4 and Section 5.

Let $y[1..n]$ be a string of length n. For any m-length substring x of y, we define its *coverage interval* as $\text{cover}(x) = \{i \in [1..n] \mid \exists j : y[j..j+m-1] = x \wedge i \in [j..j+m-1]\}$. Given a set of posittions $\Gamma \subset [1..n]$, we say that x is *covered* by Γ if $\Gamma \cap \text{cover}(x) \neq \varnothing$. Γ is called a *string attractor* [6] of y if *every* substring of y is covered by Γ.

The size of a minimal attractor for y is denoted $\gamma(y)$ and serves as a measure of repetitiveness or compressibility.

Given a string w and a candidate set $\Gamma \subseteq [1..|w|]$, the IsAttractor problem asks whether Γ is indeed an attractor of w. Recent results by Béal *et al.* [11] show that IsAttractor can be solved in linear time using the suffix automaton $\text{DAWG}(w)$ by checking, for each equivalence class of substrings represented by a state, whether at least one occurrence intersects Γ. This result provides a direct connection between attractor theory and automata-based representations of strings.

We summarize here the main symbols used throughout the paper. The text is denoted by y and has length n, while x denotes the pattern of length m. The set Γ_x is an attractor of the pattern x. We let $W = y[i..i+m-1]$ denote the current window of length m in the text, and $\text{DAWG}(W)$ its corresponding suffix automaton. In the context of attractor matching, the goal is to determine, for each window $W = y[i..i+m-1]$, whether Γ_x is also an attractor of W. Throughout the paper, we denote by k the window size and assume $k = m$.

3 The Attractor Matching Algorithm

We now formalize the AttractorMatching problem. Given a text y of length n, a pattern x of length m, and a set $\Gamma_x \subseteq [1..m]$ which is an attractor of x, we seek all substrings $W = y[i..i+m-1]$ of length m such that Γ_x is also an attractor of W. Formally, for each window W, we must verify that for every substring $u \in \text{Substr}(W)$ there exists at least one occurrence of u crossing some position $p \in \Gamma_x$. This problem introduces a new form of structural comparison

between a pattern and substrings of a text: a substring $y[i..i+m-1]$ is considered structurally equivalent to the pattern x if it is attracted by the same set of positions Γ_x. We denote by AttractorMatching(y, Γ_x, m) the process of reporting all starting positions i such that Γ_x attracts the substring $y[i..i+m-1]$.

Example 1 *Let $x =$ `aabbabb` be the pattern and $\Gamma_x = \{2, 4\}$ one of its attractors, as shown in Table 1. Consider the text $y =$ `cdccddcddccd` and the window $W = x[4..10] =$ `cddcddc`. We verify that Γ_x is also an attractor of W; hence* AttractorMatching$(y, \Gamma_x, m) = \{4\}$.

Although x and W differ symbolically, they share the same structural coverage pattern, illustrating attractor-based rather than character-based similarity.

Table 1: Coverage of all substrings of $x =$ `aabbabb` by $\Gamma_x = \{2, 4\}$.

Substring P	Covered by Γ_x	Substring P	Covered by Γ_x	Substring P	Covered by Γ_x
a	2	aab	2	bbab	4
b	4	abb	2, 4	aabba	2, 4
aa	2	bab	4	abbab	2, 4
ab	2	bba	4	bbabb	4
ba	4	aabb	2, 4	aabbab	2, 4
bb	4	abba	2, 4	abbabb	2, 4
		babb	4	aabbabb	2, 4

To solve this problem online, we rely on the *sliding-window DAWG* model introduced by Blumer [9]. This structure maintains the Directed Acyclic Word Graph corresponding to the most recent m symbols of the text, supporting incremental updates as the window slides: at each step, the leftmost character is removed and the next input symbol is appended. As shown by Blumer, this sliding-window algorithm preserves a compact representation of all substrings in the current window, achieving $O(nm)$ worst-case and $O(n \log m)$ expected time over a text of length n (for a full description and proofs, see [9]).

Let $W_t = y[t..t+m-1]$ denote the window of length m after processing t characters of the text. Initially, the automaton DAWG(W_1) is constructed in linear time from the first m symbols of y. As the window advances, for each subsequent step $t = 2, \ldots, n-m+1$, the leftmost character $y[t-1]$ is removed, the next character $y[t+m-1]$ is appended, and the automaton is updated from DAWG(W_{t-1}) to DAWG(W_t). After each update, the procedure IsAttractor is invoked to test whether Γ_x is an attractor of the new window W_t. Each state of the DAWG compactly represents an equivalence class of substrings of W_t, and the verification ensures that every such class has an occurrence intersecting Γ_x.

Given the automaton DAWG(W_t), attractor verification proceeds by traversing its states in topological order. For each state v, representing a set of substrings $\mathcal{S}(v)$, we determine whether at least one substring in $\mathcal{S}(v)$ has an occurrence crossing a position $p \in \Gamma_x$. This property can be efficiently propagated through the suffix links: whenever a substring crosses a position in Γ_x, all its suffixes inherit this property. Consequently, the verification phase runs in $O(m)$ time per window, which is proportional to the size of DAWG(W_t) [11].

The time and space complexity of maintaining the sliding DAWG are inherited from [9]. Updating the automaton as the window advances requires $O(m)$ space and $O(nm)$ time in the worst case, while under random input the expected time is $O(n \log m)$. The attractor verification step adds at most $O(m)$ time per window; therefore, the overall complexity of the AttractorMatching algorithm remains $O(nm)$ time. The corresponding pseudocode is shown in Algorithm 1.

In summary, given a text y of length n, a pattern x of length m, and an attractor Γ_x of x, the AttractorMatching problem can be solved in $O(nm)$ worst-case time using $O(m)$ space by maintaining a sliding-window DAWG and performing attractor verification through IsAttractor at each step. This shows that attractor matching can be executed online without increasing the asymptotic complexity of substring recognition. This connection between automata and attractors offers a new perspective on structure-aware pattern discovery in streaming texts, where compressibility rather than character identity becomes the guiding criterion.

Algorithm 1: AttractorMatching(y, Γ_x, m)

```
build DAWG(W_1) from the first m symbols of y        // O(m) time
t ← 1
repeat
    if IsAttractor(W_t, Γ_x)                         // O(m) time
     then
        report position t
    update DAWG(W_t) to DAWG(W_{t+1})                // O(m) time
    t ← t + 1
until t = n − m + 1                                  // O(n) time
```

4 On CDAWG-Based Attractor Verification

In the previous section we recalled the attractor verification procedure of Béal *et al.* [11], which operates on the Directed Acyclic Word Graph (DAWG). We now describe an equivalent formulation that relies on the *Compressed DAWG* (CDAWG), a more compact automaton obtained by merging every maximal chain of DAWG states connected through suffix links and single outgoing transitions. Each node of the CDAWG therefore represents an entire interval of substring lengths $[L_{\min}, L_{\max}]$ rather than a single end-position equivalence class. This compression preserves the structure of the DAWG while significantly reducing its number of nodes, especially for highly repetitive windows.

In the DAWG, each state v is characterized by its maximal length $L[v]$ and a suffix link $F[v]$ pointing to the state representing the longest proper suffix of the substrings in v. In the CDAWG these quantities become range-based: a node V stores the interval $[L_{\min}(V), L_{\max}(V)]$ corresponding to the consecutive DAWG states merged into V, and it maintains a single suffix link to the node representing the next shorter suffix interval. Thus, all suffix links along a linear chain of DAWG states collapse into one compressed link connecting two CDAWG nodes that summarize contiguous families of substrings.

The adaptation of the IsAttractor algorithm to the CDAWG follows directly from this structural correspondence. Instead of propagating coverage information across individual states, the CDAWG-based variant operates over intervals. For each node V, we check whether any substring of length $\ell \in [L_{\min}(V), L_{\max}(V)]$ intersects a position of the candidate attractor Γ_x. If such a substring exists, V is marked as covered, and this information is propagated through its compressed suffix link, ensuring that all shorter suffix intervals are also marked. The correctness argument remains identical to the original formulation: every distinct substring of the window must intersect at least one position in Γ_x.

Although the CDAWG provides a more succinct representation of the DAWG, the verification still requires inspecting each character of the window W. Consequently, the total running time of the CDAWG-based IsAttractor remains $O(m)$, matching the complexity of the DAWG-based algorithm, where $m = |W|$. The improvement lies in space: if e denotes the number of nodes in the CDAWG, corresponding to the number of distinct right contexts of W, then the algorithm requires only $O(e)$ space, where e is the number of nodes in the CDAWG, with $e \leq 2m - 1$ and typically $e \ll m$ for repetitive inputs. Thus, the CDAWG formulation maintains the same asymptotic time guarantees while reducing the memory footprint and constant factors. This reformulation emphasizes that attractor verification depends solely on the equivalence classes of right contexts, not on the explicit enumeration of substrings, and that the CDAWG offers a more compact and conceptually elegant structure for attractor-based reasoning within the attractor matching framework.

5 Sliding-Window Attractor Matching on the CDAWG

The use of a CDAWG provides not only a more compact representation for attractor verification, but also enables an efficient *sliding-window* version of the procedure. In the sliding-window DAWG model, each verification step is performed on an automaton that must be updated as the window slides, requiring up to $O(m)$ time per shift and thus a total cost of $O(nm)$ over a text of length n and window size m. In contrast, Inenaga *et al.* [13] introduced a linear-time algorithm to maintain the *CDAWG* of a text under a sliding window of width k, allowing the structure to be updated incrementally in overall $O(n)$ time. This improvement reduces the complexity to linear, making CDAWG-based attractor matching feasible for online processing.

The incremental sliding of the CDAWG over a text can be decomposed into two local operations: the deletion of the leftmost character of the current window and the insertion of a new character at its right boundary. The latter corresponds to extending the active point, following Ukkonen's online construction [14, 15], and can be performed in $O(\lg \sigma)$ amortized time by using binary search trees for storing the pointers of a node's outgoing edges. The former requires deleting or shortening the longest suffix that becomes obsolete; this step is more intricate, and the best known solution is the approximate maintenance algorithm of Inenaga *et al.* [13], which achieves amortized constant time by occasionally removing a larger portion of the window. Overall, these two operations allow the CDAWG to be updated incrementally in total $O(n \cdot \lg n)$ time and $O(e)$ space, where e denotes the number of edges in the automaton (with $e = O(m)$ for a window of length m).

Integrating this maintenance mechanism into the attractor-matching framework yields a *sliding-window* attractor verification algorithm that avoids rebuilding the automaton for each window. At every step, the IsAttractor procedure is executed directly on the current CDAWG, ensuring that attractor verification operates online. Since the CDAWG updates require $O(n \cdot \lg n)$ total time while each verification step costs $O(m)$, the overall complexity of the attractor-matching process remains $O(nm \cdot \lg n)$ time and $O(\max_i e(W_i))$ space, where $e(W_i)$ denotes the number of edges in the CDAWG of the i-th window.

This integration yields a compact and efficient framework for online attractor matching. The CDAWG provides the structural abstraction required to group substrings with identical right contexts, which directly supports attractor verification. The sliding-window maintenance algorithm ensures that this structure is updated incrementally as new characters arrive and old ones expire. The resulting *sliding-window CDAWG attractor matcher* preserves the correctness of the static formulation while maintaining the same asymptotic bounds as the DAWG-based approach. In practice, it performs fewer structural updates and exhibits smaller constant factors, especially on highly repetitive inputs.

Theorem 1 *Let y be a string of length n and let m be the window size. Using the incremental maintenance algorithm of Inenaga* et al. *[13], the CDAWG of each window can be updated online in total $O(n \cdot \log n)$ time. Combined with attractor verification over each window in $O(m)$ time, this yields a sliding-window attractor matcher running in $O(nm \cdot \log n)$ total time and $O(e)$ space, where $e = \max_i e(W_i)$ denotes the maximum number of edges in the CDAWG among all windows W_i.*

6 Attractor Matching with Mismatches

The strict formulation of attractor matching requires that the pattern's attractor Γ_x exactly attracts every substring of the current window $W = x[i..i + m - 1]$ extracted from the text y. We now extend this notion by introducing a structural tolerance parameter ρ, which measures the number of additional coverage positions required for the attractor of the pattern x to become a valid attractor of the corresponding window W of the text.

Definition 1 (ρ-mismatch attractor) *Given a non-negative integer ρ, we say that $\Gamma \subseteq [1..n]$ is a* ρ-mismatch attractor *of y if there exists a set $\Gamma' \subseteq [1..n]$ such that $\Gamma \subseteq \Gamma'$ and $|\Gamma' \setminus \Gamma| \leq \rho$, and Γ' is a string attractor of y.*

Example 2 *Let $x =$ `aabbabb` be the pattern and $\Gamma_x = \{2, 4\}$ one of its attractor, with $\rho = 2$. Consider the text $y =$ `cdccddcdddcd` and the window $W = y[4..10] =$ `cddcddd`. We verify that Γ_x is not an attractor of W, since the substring $W[5..7] =$ `ddd` is not covered by any position in Γ_x. Adding a single position $p \in [5..7]$ restores full coverage; for instance, $\Gamma_W = \{2, 4, 5\}$ is a valid attractor of W.*

Because only one additional position is required to extend the attractor, we have $\Gamma_x \subseteq \Gamma_W$ and $|\Gamma_W \setminus \Gamma_x| \leq \rho$, thus x and W are ρ-mismatch attractors.

Intuitively, Γ is an *almost-attractor* of y whose structural coverage can be completed by adding at most ρ positions. This relaxed definition preserves the attractor property up to a bounded structural deviation: if Γ fails to cover a small number of equivalence classes of substrings, we may extend it minimally by introducing additional coverage points. The resulting set Γ' constitutes a valid attractor of y, and the quantity ρ expresses the distance between the two attractors.

Given the set of uncovered substrings of y, that is, those for which $\Gamma \cap \text{cover}(x) = \varnothing$, the extension Γ' can be obtained by applying any known algorithm for string-attractor construction on the uncovered portion of the text. In particular, the optimal completion can be reduced to the NP-hard *hitting-set* problem over substring equivalence classes [7], while approximate solutions may rely on greedy or other heuristic approaches [11]. This formulation allows the ρ-extension to be computed within a bounded structural tolerance, preserving the attractor property under controlled deviation.

Lemma 1 *For any constant ρ, the ρ-mismatch attractor problem can be solved in polynomial time. In particular, one can brute–force all subsets of $[1..n] \setminus \Gamma$ of size ρ and test whether they extend Γ to a string attractor.*

Algorithmically, the ρ-mismatch formulation can be realized by analyzing the intersection between the pattern attractor Γ_x and the coverage set of the current window $\text{cover}(W)$. If $\Gamma_x \cap \text{cover}(W) \neq \varnothing$, the remaining uncovered substrings of W can be handled by introducing a minimal set of additional positions—at most ρ—so that the extended set $\Gamma'_x = \Gamma_x \cup \Gamma_\rho$ becomes a valid attractor of W. This extension step follows the same optimization or approximation strategy outlined above. Hence, the parameter ρ quantifies the structural distance between the pattern and window attractors, defining a bounded notion of structural similarity between y and x.

Intuitively, ρ quantifies how much the attractor of the pattern must be extended to achieve full coverage of the text window, providing a natural measure of structural difference between the two strings.

7 Conclusion and Future Work

We have introduced the paradigm of *Attractor Matching*, establishing its algorithmic realization on dynamic DAWGs and CDAWGs and extending it to handle bounded structural mismatches. This framework bridges compressibility and similarity, redefining string comparison through structural coverage rather than symbol identity.

Future directions include optimizing minimal ρ-extensions, extending the model to edit-based attractors, and exploring its use in compression-aware indexing and biological sequence analysis. An interesting open question concerns the *stability* of the ρ-mismatch measure across the sliding windows of the text. In particular, one may study whether similar values of ρ' recur among consecutive windows, or equivalently, determine the maximal number of additional positions required—over all windows—to preserve attractor coverage. Such an analysis could reveal how local structural deviations evolve along the text, providing a quantitative view of the dynamics of repetitiveness and the robustness of attractor-based similarity under streaming conditions.

Acknowledgements This research was supported by JSPS KAKENHI with grant numbers JP25K21150 and JP23H04378 and "Gruppo Nazionale Calcolo ScientificoIstituto Nazionale di Alta Matematica Francesco Severi (GNCS-INdAM)" and by "Piano di incentivi per la ricerca di Ateneo 2024/2026 (Pia.ce.ri.)" of the University of Catania.

References

[1] S. Faro and T. Lecroq, "The exact online string matching problem: A review of the most recent results," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 13:1–13:42, 2013.

[2] Y. Matsuoka, T. Aoki, S. Inenaga, H. Bannai, and M. Takeda, "Generalized pattern matching and periodicity under substring consistent equivalence relations," *Theoretical Computer Science*, vol. 656, pp. 225–233, 2016.

[3] B. S. Baker, "A theory of parameterized pattern matching: Algorithms and applications," in *Proc. STOC*, 1993, p. 71–80.

[4] S. Faro, F. P. Marino, A. Pavone, and A. Scardace, "Towards an efficient text sampling approach for exact and approximate matching," in *Proc. PSC*, 2021, pp. 75–89.

[5] T. I, S. Inenaga, and M. Takeda, "Palindrome pattern matching," *Theoretical Computer Science*, vol. 483, pp. 162–170, 2013.

[6] D. Kempa and N. Prezza, "At the roots of dictionary compression: string attractors," in *Proc. STOC*. ACM, 2018, pp. 827–840.

[7] D. Kempa, A. Policriti, N. Prezza, and E. Rotenberg, "String attractors: Verification and optimization," in *Proc. ESA*, 2018.

[8] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. T. Chen, and J. I. Seiferas, "The smallest automaton recognizing the subwords of a text," *Theor. Comput. Sci.*, vol. 40, pp. 31–55, 1985.

[9] J. A. Blumer, "How much is that DAWG in the window? A moving window algorithm for the directed acyclic word graph," *J. Algorithms*, vol. 8, no. 4, pp. 451–469, 1987.

[10] Y. Fujishige, Y. Tsujimaru, S. Inenaga, H. Bannai, and M. Takeda, "Linear-time computation of DAWGs, symmetric indexing structures, and MAWs for integer alphabets," *Theoretical Computer Science*, vol. 973, p. 114093, 2023.

[11] M.-P. Béal, M. Crochemore, and G. Romana, "Checking and producing word attractors," 2025.

[12] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht, "Complete inverted files for efficient text retrieval and analysis," *J. ACM*, 1987.

[13] S. Inenaga, A. Shinohara, M. Takeda, and S. Arikawa, "Compact directed acyclic word graphs for a sliding window," *J. Discrete Algorithms*, vol. 2, no. 1, pp. 33–51, 2004.

[14] S. Inenaga, H. Hoshino, A. Shinohara, M. Takeda, S. Arikawa, G. Mauri, and G. Pavesi, "On-line construction of compact directed acyclic word graphs," *Discrete Applied Mathematics*, vol. 146, no. 2, pp. 156–179, 2005.

[15] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995.