The skipping principle in string matching: a simplified analysis of Apostolico–Giancarlo

Simone Faro[1,†], Arianna Pavone[2,‡]

[1]Università di Catania, viale A. Doria n.6, 95125, Catania, Italy
[2]Università di Palermo, via Archirafi n.34, 90123, Palermo, Italy

Abstract

The Apostolico–Giancarlo string matching algorithm is a classical refinement of the Boyer–Moore paradigm and a cornerstone in the development of efficient skipping-based techniques, achieving one of the strongest known bounds on the number of character comparisons within this class. We revisit the analysis of the Apostolico–Giancarlo algorithm by extracting an explicit combinatorial framework from the structural ideas underlying its classical proof. We introduce an abstract notion of *certified positions* together with a semantic *skipping property* governing their reuse, and show that this property provides a simple and modular derivation of the classical $2n - m + 1$ bound on the number of character comparisons, where n and m are the lengths of the pattern and the text, respectively. This formulation separates algorithm-independent invariants from implementation-specific mechanisms, making explicit the structural basis of the analysis. We then refine the framework by distinguishing between fresh comparisons and re-comparisons and by introducing a group-based amortization scheme that captures how repeated comparisons are compensated by pattern shifts. This yields a concise reformulation of the tight $3n/2$ bound of Crochemore and Lecroq, exposing the local structural condition responsible for the improved analysis. Overall, the proposed framework provides a unified semantic abstraction for skipping-based string matching algorithms, separating structural invariants from implementation details and offering a reusable basis for the analysis of comparison complexity.

Keywords

Exact String Matching, Combinatorial Analysis, Amortized Complexity, Skipping Property, Algorithmic Frameworks

1. Introduction

Exact string matching is a fundamental problem in algorithmics, with applications ranging from text processing and information retrieval to computational biology [1, 2, 3, 4]. Among the many solutions proposed over the years, the Boyer–Moore family occupies a central position, combining excellent practical performance with a rich theoretical structure [5, 6, 7, 8]. A key step in its development was Galil's optimization [9], which established linear worst-case complexity by avoiding redundant examinations of previously verified text regions.

Building on this idea, Apostolico and Giancarlo introduced a refined Boyer–Moore variant that memorizes previously verified matches through an auxiliary array, allowing future alignments

ICTCS 2026: Italian Conference on Theoretical Computer Science, September 7–9, 2026, Udine, Italy
† Supported by University of Catania, progetto PIACERI 2024-2026 - Linea di Intervento 1
‡ Supported by Progetto Eurostart 2026
faro@dmi.unict.it (S. Faro); ariannamaria.pavone@unipa.it (A. Pavone)
0000-0001-5937-5796 (S. Faro); 0000-0002-8840-6157 (A. Pavone)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-4269/paper65.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-09-19

to skip redundant comparisons [10]. Their analysis established the classical upper bound of $2n - m + 1$ character comparisons, while Crochemore and Lecroq later refined this result to the tight bound of $3n/2$ through an amortized analysis of repeated successful comparisons [11]. These analyses represent some of the strongest comparison bounds known for skipping-based string matching algorithms.

Although the underlying combinatorial ideas already appear in the original analyses, they are presented as components of algorithm-specific proofs, closely tied to the operational behavior of the corresponding algorithms and their auxiliary data structures [10, 12]. In particular, the observation that a text position participates in at most one successful comparison is stated explicitly by Apostolico and Giancarlo, while Crochemore and Lecroq exploit a related non-redundancy property to obtain the refined $3n/2$ bound. However, these ideas are not formulated as abstract semantic invariants independent of the concrete implementation.

The goal of this paper is therefore not to establish new comparison bounds, but to extract these structural ideas into a unified semantic framework. We introduce the abstract notions of *certified positions* and the *skipping property*, which isolate the minimal semantic assumptions underlying the classical analysis independently of the implementation details of the Apostolico–Giancarlo algorithm. Within this framework, the bound $2n - m + 1$ follows from two independent ingredients: the skipping property bounds the number of successful comparisons, while the monotone progression of alignments bounds the number of mismatches. We then extend the framework with a group-based amortization scheme that naturally captures the refined $3n/2$ analysis of Crochemore and Lecroq within the same abstract setting.

The contributions of this paper are summarized as follows:

- we introduce an abstract semantic framework for reasoning about skipping-based string matching algorithms through the notions of certified positions and the skipping property;
- we derive the classical $2n - m + 1$ comparison bound from a small set of algorithm-independent semantic invariants;
- we extend the framework with a group-based amortization model that provides a unified reformulation of the tight $3n/2$ analysis of Crochemore and Lecroq.

Overall, the proposed framework separates semantic invariants from implementation-specific mechanisms, providing a reusable abstraction for the analysis of comparison complexity in skipping-based string matching algorithms.

2. The Apostolico–Giancarlo algorithm

In this section we recall the Apostolico–Giancarlo string matching algorithm together with the structure of its analysis [10, 12, 7]. The goal is to provide a precise reference for the algorithm and to highlight the key ideas underlying its complexity bounds, which will motivate the abstract framework introduced in the next section.

Let $T[1..n]$ be a text and $P[1..m]$ be a pattern. An alignment is an integer $s \in [0, n - m]$, corresponding to aligning $P[1..m]$ with $T[s + 1..s + m]$ [1, 7]. As in Boyer–Moore type algorithms, comparisons are performed from right to left [5, 7].

For each alignment s, the algorithm compares positions $j = m, m-1, \ldots$ until a mismatch occurs or a full match is found.

The Apostolico–Giancarlo algorithm augments the basic Boyer–Moore strategy with additional information that allows it to reuse the results of previous successful comparisons [10, 12]. The main auxiliary structure is an array $A[1..n]$, where $A[t]$ stores the length of a previously verified block of comparisons that allows future alignments involving position t to skip explicit comparisons.

More precisely, if $A[t] = \ell > 0$, then during a previous alignment the algorithm verified ℓ consecutive successful comparisons involving position t, and this information can be reused to bypass these comparisons in subsequent alignments.

In addition, the algorithm uses standard Boyer–Moore shift rules, which we abstract as a function $\Delta(j, t)$, that determines the next alignment after a mismatch at position $(s + j)$ or after a full match [5, 7]. The exact definition of Δ depends on precomputed tables (e.g., good-suffix and bad-character functions), but for our purposes it suffices that it produces a strictly positive shift, ensuring that the sequence of alignments is strictly increasing. These tables are computed during the standard preprocessing phase of the Boyer-Moore algorithm and remain fixed throughout the search.

The array A acts as a *certification mechanism*: once a block of comparisons involving a text position has been verified, future alignments can use this information to bypass the corresponding comparisons and avoid re-examining that position explicitly [10, 12].

We now give an abstracted presentation of the algorithm that makes explicit the role of this memorization mechanism while suppressing lower-level implementation details. In particular, the pseudocode in Figure 1 should be understood as a semantic representation of the behavior of Apostolico–Giancarlo, rather than as its original line-by-line formulation [12, 7].

The key feature captured by this presentation is the test at line 6: when $A[s + j] > 0$, the algorithm bypasses a block of comparisons corresponding to previously verified information involving that position. This makes explicit the skipping mechanism that, in the original formulation, is implemented through the interaction of auxiliary structures such as `skip`, `suf`, and the associated case analysis.

The classical analysis of Apostolico–Giancarlo relies on two complementary observations. First, each successful comparison records persistent information in the array A, ensuring that the corresponding text position is never successfully compared again during subsequent alignments [10, 12]. Second, each alignment terminates after at most one mismatch and is followed by a strictly positive shift, implying that the total number of mismatches is at most $n - m + 1$ [10]. Together, these arguments yield the classical upper bound of $2n - m + 1$ character comparisons.

Crochemore and Lecroq later refined this analysis by distinguishing between first-time successful comparisons and successful re-comparisons [11]. Their key observation is that repeated successful comparisons can be amortized against the subsequent shifts, leading to the tighter bound of $3n/2$ comparisons. In the next section, we reformulate both analyses within a common abstract framework that isolates the semantic invariants underlying these arguments independently of the operational details of the algorithm.

Abstracted presentation of AG

```
1.  s ← 0
2.  initialize A[1..n] ← 0
3.  while s ≤ n − m do
4.    j ← m
5.    while j ≥ 1 do
6.      if A[s + j] > 0 then
7.        j ← j − A[s + j]
8.      else
9.        compare P[j] with T[s + j]
10.       if mismatch then break
11.       A[s + j] ← m − j + 1
12.       j ← j − 1
13.   end while
14.   if j = 0 then report occurrence
15.   d ← Δ(j, s + j)
16.   s ← s + d
17. end while
```

Figure 1: Semantic abstraction of the comparison and memorization behavior of the Apostolico–Giancarlo algorithm. The figure is not intended as a complete algorithmic specification, but as an abstract representation that preserves only the execution semantics relevant to the comparison analysis developed in this paper.

3. A combinatorial framework for skipping-based matching

In this section we introduce an abstract framework for string matching algorithms that captures the semantic invariants governing the reuse of successful comparisons. Our goal is to formulate a minimal combinatorial condition under which a linear bound on the number of character comparisons follows, while keeping the model independent of implementation details.

Let $T[1..n]$ be a text and $P[1..m]$ be a pattern. An *alignment* is an integer $s \in [0, n-m]$, corresponding to aligning $P[1..m]$ with $T[s+1..s+m]$. Any string matching algorithm can be viewed as generating a sequence of alignments $s_1, s_2, \ldots, s_k$, where at each alignment the algorithm performs a sequence of character comparisons between P and the corresponding substring of T, terminating either with a mismatch or with a full match.

To make the model operational, we introduce the abstract algorithmic schema shown in Figure 2, which captures the essential behavior of skipping-based string matching algorithms without committing to any specific implementation. The abstraction introduced here is intentionally semantic rather than operational. Rather than modeling the internal data structures or execution logic of a particular algorithm, it captures only those aspects of the computation that are relevant to the analysis of character comparisons. The purpose of this abstraction is to identify the minimal invariants required by the comparison bounds, independently of how these invariants are realized by a concrete algorithm.

The algorithm maintains a set C of certified positions, representing text positions that have

Abstract skipping-based string matching algorithm

```
1.  s ← 0
2.  C ← ∅
3.  while s ≤ n − m do
4.    j ← m
5.    while j ≥ 1 do
6.      if s + j ∈ C then
7.        j ← j − 1
8.      else
9.        compare P[j] with T[s + j]
10.       if mismatch then break
11.       C ← C ∪ {s + j}
12.       j ← j − 1
13.   end while
14.   if j = 0 then report occurrence
15.   d ← Δ(s, C)
16.   s ← s + d
17. end while
```

Figure 2: Abstract skipping-based string matching algorithm.

already been involved in successful comparisons. Comparisons are performed only on positions not in C: positions in C are skipped (lines 6–7), while newly matched positions are added to C (line 11), yielding a monotone growth of certified information.

The function $\Delta(s, C)$ represents an abstract shift mechanism that determines the next alignment. We require only that $\Delta(s, C) > 0$ for all reachable states, ensuring that the sequence of alignments is strictly increasing. Our analysis does not depend on the specific definition of Δ, but only on the interaction between alignment progression and the reuse of previously established matches.

Formally, a *comparison* is a test of the form

$$T[t] \stackrel{?}{=} P[j],$$

for some $t \in [1, n]$ and $j \in [1, m]$. A comparison is *successful* if the characters are equal, and a *mismatch* otherwise.

In the following, we restrict attention to comparison schemes that terminate an alignment as soon as the first mismatch is encountered (as in standard right-to-left scanning strategies used in Boyer–Moore-type algorithms). Under this assumption, each alignment produces at most one mismatch.

A successful comparison is said to *involve* the text position t.

Definition 3.1 (Certified position)**.** A text position $t \in [1, n]$ is *certified* at time τ if, at that point in the execution, the algorithm has performed a successful comparison involving t.

We assume that certification is persistent: once a position becomes certified, it remains certified for the rest of the execution. This abstraction ignores how such information is represented internally, and focuses instead on its effect on future comparisons.

Definition 3.2 (Skipping property)**.** An algorithm satisfies the *skipping property* if each text position $t \in [1, n]$ is involved in at most one explicit character comparison during the entire execution.

Equivalently, once a successful comparison involving t has been performed, no subsequent successful comparison involves t. While the abstract schema in Figure 2 enforces this condition explicitly, the definition itself is purely semantic and applies to any algorithm whose observable behavior satisfies this constraint.

The skipping property captures, in a combinatorial way, the effect of memorization mechanisms used in concrete algorithms. It directly constrains the reuse of successful comparisons, isolating the structural invariant responsible for eliminating redundancy, while remaining independent of specific data structures, shift rules, or traversal strategies.[1]

In the next section, we show that this property alone suffices to bound the number of successful comparisons, and, combined with a standard argument on mismatches, yields the classical $2n - m + 1$ upper bound.

3.1. Bounding comparisons via the skipping property

We now derive an upper bound on the total number of character comparisons performed by any algorithm satisfying the skipping property (Definition 3.2). We assume that the sequence of alignments is strictly increasing, i.e., $s_{i+1} > s_i$ for all i, which corresponds to requiring that the shift function $\Delta(s, C)$ always produces a strictly positive displacement.

We begin by bounding the number of successful comparisons.

Lemma 3.3. *The number of successful comparisons is at most* n*.*

Proof. Each successful comparison involves a text position $t \in [1, n]$. By the skipping property (Definition 3.2), each position can be involved in at most one successful comparison during the entire execution. Since the text contains n positions, the claim follows. □

We next bound the number of mismatches.

Lemma 3.4. *The number of mismatches is at most* $n - m + 1$*.*

Proof. By assumption, each alignment produces at most one mismatch. Since the sequence of alignments is strictly increasing, the total number of alignments is at most $n - m + 1$. Therefore, the number of mismatches is also at most $n - m + 1$. □

We can now combine the two bounds.

[1] The skipping property is not intended as a new property of the Apostolico–Giancarlo algorithm. Indeed, the underlying observation that a text position involved in a successful comparison is never successfully compared again already appears in the original proof of Apostolico and Giancarlo. The contribution of the present framework is to isolate this observation as an abstract semantic invariant, independent of the particular data structures and execution strategy of the algorithm, so that it can be used as the basis of a general comparison analysis.

Theorem 3.5. *Any string matching algorithm satisfying the skipping property (Definition 3.2) performs at most* $2n - m + 1$ *character comparisons.*

Proof. The total number of comparisons is the sum of successful comparisons and mismatches. By Lemmas 3.3 and 3.4, the number of successful comparisons is at most n, and the number of mismatches is at most $n - m + 1$. Hence, the total number of comparisons is at most $n + (n - m + 1) = 2n - m + 1$. □

The skipping property (Definition 3.2) alone does not directly bound the total number of comparisons; rather, it limits the reuse of successful comparisons by ensuring that each text position contributes to at most one such comparison. The overall bound follows by combining this constraint with the monotonic progression of alignments.

3.2. A refined group-amortized model

The skipping property (Definition 3.2) isolates the contribution of successful comparisons that certify a text position for the first time (Definition 3.1), but it does not by itself control the cost of successful comparisons performed on text positions that have already been examined in previous alignments. To obtain tighter bounds, we refine the model by distinguishing these two types of successful comparisons and relating the latter to the progression of alignments.

For each alignment i, let s_i be its starting position. For all but the last alignment, let $d_i = s_{i+1} - s_i$ denote the subsequent shift length; for the last alignment, we define $d_i = 0$. Let

$$c_i = r_i^{\text{new}} + r_i^{\text{old}} + f_i \tag{1}$$

be the total number of comparisons performed at alignment i, where:

- r_i^{new} is the number of successful comparisons involving text positions that were not certified before alignment i;
- r_i^{old} is the number of successful comparisons involving text positions that were examined in some earlier alignment, but were not certified before alignment i;
- $f_i \in \{0, 1\}$ indicates whether a mismatch occurs.

Thus, r_i^{new} counts successful comparisons that certify a text position for the first time (Definition 3.1), while r_i^{old} counts successful comparisons on positions that are not new to the execution, but are also not yet excluded by the skipping property (Definition 3.2).

By the skipping property (Definition 3.2), we have

$$\sum_i r_i^{\text{new}} \leq n, \tag{2}$$

since each text position can become certified at most once. The remaining task is to bound the total contribution of the r_i^{old} term.

To capture how these repeated successful comparisons propagate across alignments, we group together alignments that are linked by the reuse of previously examined text positions. Intuitively, two alignments belong to the same group if they are connected through a chain of shared old successful comparisons, so that their costs are not independent and should be amortized jointly.

Definition 3.6 (Groups)**.** Define a relation $\sim$ on alignments as follows: $i \sim j$ if there exists a sequence of alignments

$$i = i_0, i_1, \ldots, i_k = j$$

such that for each ℓ, alignments i_ℓ and $i_{\ell+1}$ perform a successful comparison on a common text position that contributes to the corresponding r^{old} term.

The relation $\sim$ is an equivalence relation: reflexivity and symmetry are immediate, and transitivity follows by concatenating such sequences. Therefore, it partitions the set of alignments into disjoint groups. In particular, the sets G form a partition, and we can decompose sums over alignments as sums over groups.

For a group G, we define

$$R(G) = \sum_{i \in G} r_i^{\text{old}}, \qquad D(G) = \sum_{i \in G} d_i. \tag{3}$$

We now introduce a structural condition relating these repeated successful comparisons to shift lengths.

Definition 3.7 (Group-amortized skipping property)**.** An algorithm satisfies the ρ-group-amortized skipping property if, for every group G (Definition 3.6),

$$R(G) \leq \rho\, D(G). \tag{4}$$

The following lemma shows how a local structural condition implies this global property.

Lemma 3.8 (Local-to-global amortization)**.** *Suppose that there exists a constant $\alpha > 0$ such that for every alignment i with $r_i^{\text{old}} > 0$,*

$$d_i \geq \alpha\, r_i^{\text{old}}. \tag{5}$$

Then the algorithm satisfies the ρ-group-amortized skipping property (Definition 3.7) with $\rho = 1/\alpha$.

Proof. Let G be a group (Definition 3.6), and let $G' \subseteq G$ be the set of alignments with $r_i^{\text{old}} > 0$. Summing the inequality (5) over all $i \in G'$, we obtain

$$\sum_{i \in G'} d_i \geq \alpha \sum_{i \in G'} r_i^{\text{old}} = \alpha R(G).$$

Since $d_i \geq 0$ for all i, we have $D(G) = \sum_{i \in G} d_i \geq \sum_{i \in G'} d_i$, and therefore $D(G) \geq \alpha R(G)$, which implies $R(G) \leq \frac{1}{\alpha} D(G)$. □

The purpose of this grouping is to localize the effect of repeated successful comparisons: within each group, these comparisons are linked through shared text positions and can be amortized against the cumulative shift of that group.

We can now derive a global bound on the number of comparisons.

Theorem 3.9. *Any string matching algorithm satisfying the skipping property (Definition 3.2) and the ρ-group-amortized skipping property (Definition 3.7) performs at most $n + \rho \sum_i d_i + \sum_i f_i$ character comparisons.*

Proof. By equation (1),

$$\sum_i c_i = \sum_i r_i^{\text{new}} + \sum_i r_i^{\text{old}} + \sum_i f_i.$$

Since r_i^{new} counts successful comparisons that certify a text position for the first time (Definition 3.1), and each text position can be certified at most once, we have (2): $\sum_i r_i^{\text{new}} \leq n$.

By summing over groups (Definition 3.6) and applying the ρ-group-amortized skipping property (Definition 3.7), we obtain

$$\sum_i r_i^{\text{old}} = \sum_G R(G) \leq \sum_G \rho D(G) = \rho \sum_i d_i.$$

Substituting these bounds into the expression for $\sum_i c_i$ yields $\sum_i c_i \leq n + \rho \sum_i d_i + \sum_i f_i$, as claimed. □

In particular, since the sequence of alignments is strictly increasing, we have $\sum_i d_i \leq n-m+1$ and $\sum_i f_i \leq n - m + 1$, so that the total number of comparisons is bounded by

$$n + \rho(n - m + 1) + (n - m + 1).$$

This formulation separates the universal contribution of first certifications from the algorithm-specific contribution of repeated successful comparisons, and shows that tighter bounds follow from local structural conditions relating repeated successful comparisons to shift lengths.

4. The Apostolico–Giancarlo algorithm in the abstract model

In this section we show that the Apostolico–Giancarlo algorithm satisfies the skipping property (Definition 3.2) introduced in Section 3, and therefore falls within the scope of our abstract framework.

We rely on the abstracted presentation given in Figure 1, which captures the essential behavior of the algorithm in terms of its memorization and skipping mechanism.

4.1. The skipping property for Apostolico–Giancarlo

The key feature of the Apostolico–Giancarlo algorithm is that it records the outcome of successful comparisons and uses this information to avoid re-examining the same text positions in subsequent alignments. In particular, once a text position has been successfully matched, it is never re-compared character by character.

We formalize this property in the language of our model (Definition 3.2), referring to the abstracted presentation in Figure 1.

Lemma 4.1. *In the abstracted presentation of the Apostolico–Giancarlo algorithm, each text position $t \in [1, n]$ is involved in at most one successful comparison.*

Proof. Consider a text position $t \in [1, n]$, and suppose that at some alignment s the algorithm performs a successful comparison between $P[j]$ and $T[s + j]$ with $t = s + j$ (line 9). At

this point, the algorithm sets $A[t] > 0$ (line 11), recording that a previously verified block of successful comparisons involving position t can be reused in subsequent alignments.

We claim that, from this moment on, no subsequent alignment performs an explicit comparison involving position t. Let $s' > s$ be any later alignment such that $t \in [s' + 1, s' + m]$, and let j' be such that $t = s' + j'$.

If the inner loop reaches position j', then the condition at line 6 is satisfied, since $A[t] > 0$. Therefore, the algorithm does not perform the comparison at position t, but instead skips over a block of positions by setting $j' \leftarrow j' - A[t]$. Otherwise, the position t is skipped without being explicitly compared.

In either case, no explicit comparison is performed at position t in alignment s'.

It follows that, after the first successful comparison involving t, no later alignment performs an explicit comparison at that position. In particular, no further successful comparison can involve t.

Since this holds for every text position $t \in [1, n]$, the claim follows. □

We make explicit the correspondence between the abstract model of Section 3 and the abstracted presentation of Apostolico–Giancarlo.

Lemma 4.2. *Consider the abstracted presentation in Figure 1. Let C be the set of text positions t such that $A[t] > 0$. Then, throughout the execution:*

1. *C is monotonically non-decreasing;*
2. *a position t is added to C exactly when a successful comparison involving t is performed (cf. Definition 3.1);*
3. *if $t \in C$, then no subsequent alignment performs an explicit comparison at position t.*

Proof. (1) The array A is initialized to zero and entries are only updated at line 11 by assigning positive values. Hence, once $A[t] > 0$, it remains so.

(2) A position $t = s + j$ is assigned a positive value in $A[t]$ only after a successful comparison at line 9, and no other updates introduce positive values.

(3) This follows from the argument in Lemma 4.1: whenever $A[t] > 0$, any later alignment containing t skips that position via the test at line 6. □

Corollary 4.3. *The abstracted presentation of the Apostolico–Giancarlo algorithm satisfies the skipping property (Definition 3.2).*

Proof. The claim follows directly from Lemma 4.1 and Definition 3.2. □

This argument shows that the abstract skipping property provides a semantic formulation of the structural invariant exploited by the Apostolico–Giancarlo algorithm.: once a text position has been successfully compared, it is never re-examined through explicit comparisons in subsequent alignments. Importantly, the proof relies only on the persistence of the information stored in A, and not on the specific implementation details of the underlying data structures.

This shows that the skipping property (Definition 3.2) captures the core mechanism responsible for eliminating redundant successful comparisons in the Apostolico–Giancarlo algorithm. Similar reasoning applies to other algorithms incorporating the Galil rule or related memorization techniques, where previously matched regions are protected from re-examination.

4.2. A refined analysis of Apostolico–Giancarlo

We now refine the analysis of the Apostolico–Giancarlo algorithm within the framework introduced in Section 3, deriving the tight upper bound on the number of character comparisons established by Crochemore and Lecroq.

As in the abstract model, we distinguish between *fresh* successful comparisons, involving text positions that are certified for the first time, and *repeated successful comparisons*, involving text positions that have already been examined in previous alignments but were not certified before the current alignment. By the skipping property, each text position can be certified at most once, and therefore the total number of fresh successful comparisons is at most n.

It remains to bound the total number of repeated successful comparisons. To this end, we rely on a structural property established by Crochemore and Lecroq [11], which relates the number of such comparisons performed at an alignment to the subsequent shift.

Lemma 4.4 (Crochemore–Lecroq)**.** *Let i be an alignment of the Apostolico–Giancarlo algorithm. If alignment i performs k repeated successful comparisons, then the subsequent shift satisfies $d_i \geq k + 1$.*

Proof. This is a restatement of Lemma 2 in [11]. The result follows from the combinatorial structure of the Apostolico–Giancarlo algorithm, which ensures that distinct repeated comparisons impose independent constraints on the next valid alignment. Any shift smaller than $k + 1$ would violate at least one of these constraints. □

We can now apply the general framework of Section 3. Lemma 4.4 implies that for every alignment i, $d_i \geq r_i^{\text{old}} + 1$, and in particular $d_i \geq r_i^{\text{old}}$.

Thus the hypothesis of Lemma 3.8 holds with $\alpha = 1$, and therefore the algorithm satisfies the ρ-group-amortized skipping property with $\rho = 1$.

Applying the general bound of Section 3, we obtain $\sum_i r_i^{\text{old}} \leq \sum_i d_i$.

We emphasize that this argument directly instantiates the general group-amortized framework. However, the stronger inequality $d_i \geq r_i^{\text{old}} + 1$ allows a tighter amortization specific to the Apostolico–Giancarlo algorithm.

Summing over all alignments with $r_i^{\text{old}} > 0$, we obtain

$$\sum_{i : r_i^{\text{old}} > 0} d_i \geq \sum_{i : r_i^{\text{old}} > 0} (r_i^{\text{old}} + 1) = \sum_i r_i^{\text{old}} + |G'|,$$

where G' is the set of alignments performing at least one repeated successful comparison.

Since each such alignment contributes at least one unit of shift, we have $|G'| \leq \sum_i d_i$. Therefore,

$$\sum_i d_i \geq \sum_i r_i^{\text{old}} + |G'| \geq \sum_i r_i^{\text{old}} + \frac{1}{2} \sum_i d_i,$$

which implies $\sum_i r_i^{\text{old}} \leq \frac{1}{2} \sum_i d_i$.

Since the sequence of alignments is strictly increasing, we have $\sum_i d_i \leq n$, and therefore $\sum_i r_i^{\text{old}} \leq \frac{n}{2}$.

We can now bound the total number of comparisons.

Theorem 4.5. *The Apostolico–Giancarlo algorithm performs at most $\frac{3}{2}n$ character comparisons.*

Proof. By the refined analysis above, the total number of repeated successful comparisons satisfies

$$\sum_i r_i^{\text{old}} \le \frac{1}{2} \sum_i d_i \le \frac{n}{2}.$$

Moreover, by the skipping property, each text position contributes to at most one fresh successful comparison, and therefore $\sum_i r_i^{\text{new}} \le n$.

The bound $\frac{3}{2}n$ on the total number of character comparisons now follows from the tight analysis of Crochemore and Lecroq [11], of which the argument above makes explicit the amortized contribution of repeated successful comparisons. In particular, Lemma 4.4 is precisely the local structural property underlying their proof, and the estimate

$$\sum_i r_i^{\text{old}} \le \frac{n}{2}$$

isolates the source of the improvement over the classical $2n - m + 1$ bound. □

This bound is tight: as shown in [11], there exist families of inputs for which the algorithm performs $\frac{3}{2}n - \Theta(m)$ comparisons.

5. Discussion and concluding remarks

We have presented an abstract semantic framework for the analysis of skipping-based string matching algorithms, motivated by the classical Apostolico–Giancarlo algorithm. Rather than introducing new comparison bounds, the framework isolates the algorithm-independent invariants underlying the classical analyses and separates them from the implementation mechanisms used to realize them.

Within this abstraction, the classical $2n-m+1$ bound follows directly from the persistence of certified positions and the monotone progression of alignments. The same framework naturally extends to the refined analysis of Crochemore and Lecroq through a group-based amortization scheme, providing a unified formulation of the tight $3n/2$ comparison bound.

As discussed in Appendix **??**, the framework is intended to capture analyses based on persistent certification of previously verified text positions, and therefore does not encompass algorithms whose efficiency relies on fundamentally different principles, such as shift heuristics or automata-based search. This distinction helps clarify both the scope of the proposed abstraction and the class of algorithms for which it is most naturally applicable.

We believe that the proposed framework provides a useful conceptual tool for reasoning about comparison complexity in memorization-based string matching algorithms. Beyond the specific case of Apostolico–Giancarlo, it offers a common language for expressing algorithm-independent invariants, comparing existing analyses, and guiding the study of future skipping-based variants.

Declaration on generative AI

Generative AI was used exclusively to assist with the linguistic refinement of the manuscript, including the rephrasing of selected sentences and paragraphs to improve clarity, conciseness, and readability.

References

[1] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

[2] S. Faro, T. Lecroq, The exact online string matching problem: A review of the most recent results, ACM Computing Surveys 45 (2013) 13:1–13:42.

[3] D. Cantone, S. Faro, E. Giaquinta, Approximate string matching allowing for inversions and translocations, in: J. Holub, J. Zdárek (Eds.), Proceedings of the Prague Stringology Conference 2010, Prague, Czech Republic, August 30 - September 1, 2010, Prague Stringology Club, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2010, pp. 37–51. URL: http://www.stringology.org/event/2010/p04.html.

[4] S. Faro, T. Lecroq, An efficient matching algorithm for encoded DNA sequences and binary strings, in: G. Kucherov, E. Ukkonen (Eds.), Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Lille, France, June 22-24, 2009, Proceedings, volume 5577 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 106–115. URL: https://doi.org/10.1007/978-3-642-02441-2_10. doi:10.1007/978-3-642-02441-2_10.

[5] R. S. Boyer, J. S. Moore, A fast string searching algorithm, Communications of the ACM 20 (1977) 762–772. doi:10.1145/359842.359859.

[6] L. J. Guibas, A. M. Odlyzko, A new proof of the linearity of the boyer–moore string searching algorithm, SIAM Journal on Computing 9 (1980) 672–682. doi:10.1137/0209051.

[7] C. Charras, T. Lecroq, Handbook of Exact String-Matching Algorithms, College Publications, 2004.

[8] D. Cantone, S. Faro, Pattern matching with swaps for short patterns in linear time, in: M. Nielsen, A. Kucera, P. B. Miltersen, C. Palamidessi, P. Tuma, F. D. Valencia (Eds.), SOFSEM 2009: Theory and Practice of Computer Science, 35th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24-30, 2009. Proceedings, volume 5404 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 255–266. URL: https://doi.org/10.1007/978-3-540-95891-8_25. doi:10.1007/978-3-540-95891-8_25.

[9] Z. Galil, On improving the worst case running time of the boyer-moore string matching algorithm, Communications of the ACM 22 (1979) 505–508.

[10] A. Apostolico, R. Giancarlo, The boyer-moore-galil string searching strategies revisited, SIAM Journal on Computing 15 (1986) 98–105.

[11] M. Crochemore, T. Lecroq, Tight bounds on the complexity of the apostolico–giancarlo algorithm, Information Processing Letters 64 (1997) 119–123.

[12] M. Crochemore, C. Hancart, T. Lecroq, A unifying look at the apostolico-giancarlo string matching algorithm, Theoretical Computer Science 292 (2003) 73–94.