

Article

Parameterized String Matching by Text Sampling

Simone Faro ^{1,*}, Thierry Lecroq ², Francesco Pio Marino ^{1,2}, Arianna Pavone ³ and Antonio Scardace ¹¹ Department of Mathematics and Computer Science, University of Catania, 95131 Catania, Italy² LITIS UR 4108, University of Rouen Normandie, 76800 Rouen, France; thierry.lecroq@univ-rouen.fr³ Department of Mathematics and Computer Science, University of Palermo, 90123 Palermo, Italy

* Correspondence: faro@dmf.unict.it

Abstract

String matching is a foundational problem in computer science with applications that span data mining, text retrieval, bioinformatics, and software engineering. Traditional string matching algorithms often face limitations with space and time efficiency, particularly in large-scale data applications. Sampled string matching is a recent approach designed to improve efficiency by addressing the high space requirements for index construction and reducing search time for online solutions. Parameterized pattern matching, a generalized form of string matching, introduces flexible matching criteria, initially developed to detect duplicate code and plagiarism in software maintenance. Its utility has since expanded to various domains, enabling adaptable matching across diverse applications. In this paper, we propose a novel algorithm for parameterized string matching that takes advantage of character distance sampling. This technique samples the distances between consecutive occurrences of a specified pivot character, facilitating an efficient online search within the sampled data for pattern occurrences before performing verification in the full text. Through extensive experimentation with diverse datasets, we demonstrate that our algorithm outperforms existing parameterized string matching methods in terms of speed. These results suggest that character distance sampling is a promising approach to enhance the efficiency and scalability of parameterized string matching algorithms, opening new avenues for application in large-scale and complex data environments.

Keywords: string matching; parameterized matching; sampling algorithms; text processing**MSC:** 68W32

1. Introduction

String matching is a central problem in computer science [1], with applications spanning various fields such as bioinformatics, text processing, information retrieval, and data mining. For example, in bioinformatics, string matching is essential for comparing DNA sequences to identify genetic similarities and differences between organisms. In text retrieval, it enables efficient searches for documents relevant to user queries by locating specific patterns within large datasets.

This paper focuses on a particular type of approximate string matching known as parameterized string matching (PSM), originally developed to identify duplicate code segments in software [2].

In parameterized string matching, the alphabet is divided into two sets: the constant alphabet Σ_s and the parameter alphabet Σ_p . Thus, every string operates over the combined alphabet $\Sigma = \Sigma_s \cup \Sigma_p$, and such strings are termed parameterized strings, or p-strings [3].



Academic Editor: David N. Olivieri

Received: 25 June 2026

Revised: 22 July 2026

Accepted: 24 July 2026

Published: 2 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

To determine whether two p-strings x and y match, there must exist a bijective function g such that each character in $x \in \Sigma_p$ maps uniquely to a corresponding character in $y \in \Sigma_p$ via g , while every character in $x \in \Sigma_s$ matches exactly with the corresponding character in $y \in \Sigma_s$.

Initially designed to detect repetitive code segments in software [4], PSM has been extended to applications in bioinformatics, natural language processing, cybersecurity, and industrial automation, where flexible matching criteria are essential to handle varying data. PSM accommodates different string comparison requirements, making it versatile across applications.

To illustrate this, consider a program string x and another program string y where y is derived by renaming variables in x . Although exact characters may differ, parameterized matching captures their equivalence by mapping parameterized variables accordingly, which is particularly useful in detecting structurally similar code with variable renaming.

Over the years, several approaches have been developed to tackle the PSM problem. A key contribution in this field was made by Baker in 1993 [4], who introduced a method based on the Boyer–Moore algorithm [5], a well-known and efficient technique for exact string matching. Baker’s algorithm, along with the improved Turbo-Boyer–Moore variant, has significantly influenced further research. However, despite their importance, these methods come with a high computational cost, which can make them less practical for large-scale datasets. This challenge highlights the need for more efficient PSM algorithms capable of handling large amounts of data.

In this paper, we introduce a novel PSM algorithm that uses a character distance sampling technique [6] to significantly reduce computational costs in both search time and memory usage. Building on previous sampling-based approaches [7], our method strategically selects a subset of characters to minimize the number of required comparisons.

We chose character distance sampling because previous work has shown it to provide very competitive performance for string matching applications [8].

We evaluate our algorithm against Baker’s approaches based on Boyer–Moore and Turbo-Boyer–Moore, both recognized as effective solutions for PSM. Extensive experiments on diverse datasets demonstrate that our method consistently outperforms these traditional techniques in terms of speed, particularly when dealing with patterns containing a high proportion of parameter characters or large pattern sizes.

Our results underscore the effectiveness of character distance sampling in enhancing the efficiency of PSM algorithms. The proposed approach provides a faster solution for parameterized string matching, making it highly applicable in domains where efficient and scalable string matching is crucial.

2. Parameterized String Matching

A *parameterized string* (p-string) is a string defined over an alphabet $(\Sigma_s \cup \Sigma_p)$, where Σ_s and Σ_p are disjoint sets representing static symbols (s-symbols) and parameter symbols (p-symbols), respectively. Two p-strings x and y are said to *p-match* if and only if there exists a bijection mapping on Σ_p that transforms x into y while preserving the static symbols.

Originally introduced for matching strings of equal length, *parameterized matching* has since been extended to accommodate text-pattern matching scenarios, akin to classical string matching. In this paper, we focus on this broader class of problems.

Example 1. Consider the alphabets $\Sigma_s = \{A, B, C\}$ and $\Sigma_p = \{u, v, w, z\}$. The pattern $P = AuBvCu$ *p-matches* within the text

$$T[1..20] = AvBuCvAwBuCzu vAzBwCz$$

at positions 1 and 15. At position 1, the mapping is $u \rightarrow v$ and $v \rightarrow u$, while at position 15, the mapping is $u \rightarrow z$ and $v \rightarrow w$. However, P does not match at position 7, since u would be required to map to both w and z , violating the bijectivity condition.

A naive method for determining whether a string $P[1..m]$ p-matches at position s in text T involves a left-to-right traversal of both P and $T[s..s + m - 1]$, dynamically constructing a mapping table for p-symbols. A mismatch occurs if:

1. One character is a parameter while the other is not;
2. Both characters are static but different; or
3. Both are parameters but have conflicting mappings.

If no mismatches occur, a p-match is found. This procedure, depicted in Figure 1, runs in $O(m)$ time with $O(|\Sigma_p|)$ space complexity.

```

CHECK-P-MATCH( $P, T, m$ )
1.   $map_P \leftarrow \{\}$ 
2.   $map_T \leftarrow \{\}$ 
3.  for  $i = 1$  to  $m$  do
4.    if  $P[i] \in \Sigma_s$  or  $T[i] \in \Sigma_s$  then
5.      if  $P[i] \neq T[i]$  then
6.        return FALSE
7.    else
8.      if  $P[i] \in map_P$  then
9.        if  $map_P[P[i]] \neq T[i]$  or
            $map_T[T[i]] \neq P[i]$  then
10.         return FALSE
11.     else
12.        $map_P[P[i]] = T[i]$ 
13.        $map_T[T[i]] = P[i]$ 
14.   return TRUE

```

Figure 1. Pseudocode for CHECK-P-MATCH, which verifies whether a string x of length m p-matches a equal length string y . The procedure checks for the required isomorphism.

The parameterized string matching problem was first explored by Baker [4] in the context of software maintenance, particularly for detecting duplicate code segments. Another early application was in plagiarism detection [9]. Baker's approach introduced the concept of *predecessor strings* as a preprocessing step, allowing parameterized matching to be reduced to classical exact string matching techniques [10].

Definition 1 (Prev Transformation). Let $x[1..m]$ be a p-string over an alphabet $\Sigma = \Sigma_s \cup \Sigma_p$, where Σ_s and Σ_p are disjoint. The transformation *prev* of x produces another string $x'[1..m]$ over the alphabet $\Sigma' = \Sigma_s \cup \mathbb{N}$, where:

$$x'[i] = \begin{cases} x[i] & \text{if } x[i] \in \Sigma_s \\ 0 & \text{if } x[i] \in \Sigma_p \text{ and } i \text{ is the position} \\ & \text{of the first occ. of } x[i] \text{ in } x \\ i - j & \text{if } x[i] \in \Sigma_p \text{ and } j = \max\{k \mid \\ & 1 \leq k \leq i - 1 \text{ and } x[k] = x[i]\} \\ & \text{otherwise.} \end{cases}$$

Computing $prev(x)$ requires $\Theta(m)$ time and $O(|\Sigma_p|)$ space, using a lookup table to track parameter occurrences. The key property of $prev(x)$ is that it abstracts away the actual

parameter values, retaining only their relative order, thereby enabling direct comparison of two p-strings: x and y p-match if and only if $prev(Xx) = prev(y)$.

In 1997, Amir et al. [11] extended parameterized matching to two-dimensional settings, enabling applications in programming languages, computational biology, and image processing, where object recognition must remain invariant to altered color mappings. This was further developed by Hazay et al. [12] and refined by Cole et al. [13].

Salmela and Tarhio [10] proposed an alternative approach using q -grams, which enables sublinear time complexity for moderately repetitive patterns. Other notable developments include multi-pattern parameterized matching [14], approximate matching [15], and matching with mismatches [16].

Baker's pioneering suffix-tree-based algorithm [4] preprocesses the text into a predecessor representation, allowing for exact matching algorithms to be applied. However, managing windowed text segments posed a challenge, as predecessor references could extend beyond a given window. To address this, Baker modified suffix-tree construction to achieve $O(n \log n)$ time complexity. Subsequent refinements by Kosaraju [17] and Cole and Hariharan [18] led to more efficient suffix tree construction techniques.

Amir et al. [19] developed a parameterized Knuth–Morris–Pratt (KMP) variant with a worst-case complexity of $O(n \log(\sigma))$, where σ is the alphabet size. Fredriksson and Mozgovoy [9] later introduced sublinear-time methods leveraging the shift-or technique [20] and backward DAWG matching (BDM) [21], further optimizing parameterized matching.

For two-dimensional parameterized matching, Amir et al. [11] initially proposed function-based preprocessing, achieving a complexity of $O(n^2 \log^2 m)$. Hazay et al. [12] improved this using the *duel-and-sweep* paradigm, reducing complexity to $O(n^2 + m^{2.5} \text{polylog}(m))$. Later approaches, such as those by Salmela et al. [10], utilized q -grams for efficiency.

Existing parameterized matching approaches exhibit different trade-offs between preprocessing cost, query efficiency, and memory consumption. Algorithms based on suffix trees and related indexing structures provide fast query capabilities after preprocessing, but they require significant additional space and construction time, making them less suitable for large-scale or dynamic text collections. Conversely, online approaches avoid expensive indexing structures and require limited memory, but may provide weaker performance for repeated searches over the same text. Methods based on predecessor representations and q -grams offer a balance between these extremes, although their efficiency depends on pattern characteristics and alphabet properties. These limitations motivate the investigation of intermediate solutions that exploit partial information from the text while avoiding the overhead of full indexing structures.

The CDS-based approach considered in this work follows this direction by introducing a partial-index representation of the text. Unlike full-text indexes, it stores only sampled information while allowing efficient matching operations on the compressed representation. To the best of our knowledge, this represents the first approach that applies partial indexing principles to parameterized string matching, providing a trade-off between the memory requirements of full indexes and the computational cost of purely online methods.

3. A Text Sampling Solution to Parameterized Matching

Text sampling [22,23] is a technique designed to improve the efficiency of string matching by reducing both the search time and the space required for indexing. Instead of processing the entire text character by character, sampling-based approaches extract a subset of representative positions or features, allowing for a faster and more space-efficient search. This method is particularly useful for large-scale text processing, where traditional exact matching techniques may be computationally expensive.

Over the past few years, several text sampling strategies have been proposed, each aiming to optimize different aspects of the matching process. However, among these, the Character Distance Sampling [22] (CDS) approach has emerged as the most efficient solution. CDS makes use of the distances between selected pivot characters to construct a compact yet highly effective representation of the text, significantly accelerating the matching process while maintaining a minimal memory footprint.

Specifically, the CDS-based approach, in the classical case, has demonstrated remarkable efficiency in practical applications, achieving search times up to 40 times faster than standard exact matching while requiring only 2% of the text size for indexing. Sampled string matching has also proven to be highly flexible, showing effectiveness even in SCER-matchings, where $\approx$ is defined as a *substring consistent equivalence relation* (SCER) on strings such that for any two strings x and y , the condition $x \approx y$ implies that $|x| = |y|$ and $x[i..j] \approx y[i..j]$ for $1 \leq i \leq j \leq |x|$ [24]. This includes notable applications such as *Order Preserving Pattern Matching* [25].

In this section, we introduce an adaptation of text sampling techniques to another class of SCER-matching: the parameterized string matching (PSM) problem. Specifically, we modify the CDS approach [22], originally designed for exact string matching, to accommodate the unique constraints of parameterized matching.

Let T be the input text of length n , and let P be the input pattern of length m , both defined over an alphabet $\Sigma = \Sigma_s \cup \Sigma_p$ of size $\sigma = \sigma_s + \sigma_p$. We assume that all strings are treated as vectors indexed from position 1, where $P[i]$ denotes the i -th character of string P , for $1 \leq i \leq m$.

Our approach begins by selecting a sub-alphabet $C \subseteq \Sigma_s$, referred to as the set of pivot characters. These pivots serve as anchors to sample the text T by computing the distances between consecutive occurrences of any pivot character $c \in C$. Formally, this sampling methodology is based on position sampling. Given $\delta : \{1, \dots, n_c\} \rightarrow \{1, \dots, n\}$ where $\delta(i)$ denotes the position of the i -th occurrence of a pivot character c in T , the position-sampled version of T , denoted as $\dot{T}$, is a numeric sequence of length n_c : $\dot{T} = \langle \delta(1), \delta(2), \dots, \delta(n_c) \rangle$.

Example 2. Consider the text sequence $T = \text{“AgAACgCAgtAtA”}$ of length 13 over the alphabet $\Sigma_s = \{A, C\}$ and $\Sigma_p = \{g, t\}$. If we select $C = \{A\}$ as the set of pivot characters, the position-sampled representation of T is $\dot{T} = \langle 1, 3, 4, 8, 11, 13 \rangle$.

Here, the first occurrence of A is at position 1 ($T[1] = \text{“A”}$), the second at position 3 ($T[3] = \text{“A”}$), and so forth.

Following this representation, we define the character distance function as $\Delta(i) = \delta(i+1) - \delta(i)$, $1 \leq i \leq n_c - 1$.

The character-distance sampled version of T , denoted as $\bar{T}$, is a numeric sequence of length $n_c - 1$:

$$\begin{aligned}\bar{T} &= \langle \Delta(1), \Delta(2), \dots, \Delta(n_c - 1) \rangle \\ &= \langle \delta(2) - \delta(1), \delta(3) - \delta(2), \dots, \delta(n_c) - \delta(n_c - 1) \rangle.\end{aligned}$$

For instance, given the same text $T = \text{“AgAACgCAgtAtA”}$ with $C = \{A\}$, the character-distance sampled version of T is $\bar{T} = \langle 2, 1, 4, 3, 2 \rangle$.

Specifically, $\bar{T}[1] = \Delta(1) = \delta(2) - \delta(1) = 3 - 1 = 2$ and $\bar{T}[3] = \Delta(3) = \delta(4) - \delta(3) = 8 - 4 = 4$.

In practice, particularly for large alphabets, the set of pivot characters is often restricted to a single character. Consequently, we frequently refer to the pivot character in the singular rather than explicitly mentioning the entire set. By restricting the selection of pivots to characters from the static alphabet Σ_s , the CDS algorithm reduces the text to a simplified

representation requiring only exact matching. Given this assumption, the preprocessing and searching phases follow a straightforward pipeline: first, the position-sampled versions of both the text and the pattern are computed, and second, all occurrences of the sampled pattern within the sampled text are identified.

However, like other sampling-based string matching approaches, an additional verification phase is required once a potential match is detected. In the case of parameterized string matching, the verification process consists of two distinct steps. The first step ensures that all characters from Σ_s in the pattern match exactly with those in the corresponding positions of the text. Formally, given a candidate match at position j , we define

$$\text{VERIFY}(P, T, j) = \begin{cases} 1, & \text{if } T[i+j] = P[i], \quad \forall i \mid 1 \leq i \leq m, \\ & T[i+j] \in \Sigma_s, P[i] \in \Sigma_s, \\ 0, & \text{otherwise.} \end{cases}$$

An additional step is introduced to handle characters from Σ_p . This second phase ensures that occurrences identified by the algorithm also satisfy the bijective mapping conditions of parameterized string matching, described above. By incorporating this two-phase verification process, our adaptation maintains the efficiency of CDS while ensuring correctness in the parameterized setting. The resulting approach improves computational scalability, making it well-suited for large-scale parameterized string matching tasks.

Before presenting the algorithm, we introduce two auxiliary components used during the search phase.

The parameter γ denotes the offset between the first sampled position and the starting position of a candidate occurrence in the original text. More precisely, if $\tilde{T}[\text{shift}]$ is the position of a pivot character in T , then the corresponding candidate occurrence of the pattern begins at position $\tilde{T}[\text{shift}] - \gamma$. The value of γ depends on the position of the first pivot character in the pattern P and is computed during preprocessing.

The array hbc (Horspool Bad Character table) is a shift table computed on the sampled pattern $\bar{P}$. For each possible distance value d , hbc[d] stores the shift to be applied when a mismatch occurs, following the Horspool strategy adapted to the sampled domain. This table is constructed in $O(|\bar{P}|)$ time during preprocessing and allows the algorithm to skip unnecessary comparisons efficiently.

The pseudocode of the proposed searching procedure is presented in Algorithm 1.

The CDS-based sampled string matching approach maintains a partial index, represented by the position-sampled version of T . This index requires $32n_c$ bits in memory, assuming 32-bit storage for each sampled position, making it lightweight and efficient for search operations.

More recent advances have further explored ways to reduce memory usage while preserving query efficiency. In particular, approaches such as [26] demonstrate how to obtain optimal sampling via a set cover formulation. We refer the reader to [26] for additional details and a comprehensive description of the construction and implementation techniques.

To search for a pattern P of length m in T , a preprocessing step computes its sampled version $\bar{P}$. An occurrence of P in T corresponds to an occurrence of $\bar{P}$ in $\tilde{T}$, meaning that standard string matching algorithms can locate $\bar{P}$ in $\tilde{T}$. However, the reverse does not necessarily hold, requiring a validation step in T .

Example 3. The sampled representation may identify candidate occurrences that are not valid parameterized matches. Consider: $P = \mathbf{A}u\mathbf{B}u$ and a candidate text window: $T[i..i+m-1] = \mathbf{A}x\mathbf{B}y$.

The pivot positions may generate the same sampled representation as the pattern, but the verification phase rejects the candidate because the parameter symbol u would need to map both to x and y , violating the bijective mapping requirement.

Since the proposed algorithm adopts exactly the same Character Distance Sampling (CDS) scheme as the original approach [27], the completeness of the sampling phase follows directly from the correctness proof given there. In particular, every occurrence of the original pattern induces a corresponding occurrence in the sampled representation, while the converse is not necessarily true. Consequently, the verification phase is only required to eliminate false positives and cannot discard any valid occurrence. The main contribution of this work lies exclusively in the verification stage, which has been adapted to enforce the bijective mapping required by parameterized string matching, without altering the underlying sampling framework.

Since validation requires $O(m)$ time, the total search complexity is $O(mn)$. However, optimizations ensure that the search remains linear in practice, as shown in [8], where several online string matching algorithms are adapted to operate on the CDS-sampled representation of the text. Notably, the CDS approach does not explicitly maintain $\tilde{T}$, instead keeping $\hat{T}$, which holds the positions. The sequence $\tilde{T}$ is computed dynamically during the search, with each element given by: $\tilde{T}[i] = \hat{T}[i + 1] - \hat{T}[i]$.

Algorithm 1: Parameterized Search with Character Distance Sampling

```

1 Function pCDS( $\bar{P}$ ,  $|\bar{P}|$ ,  $\hat{T}$ ,  $|\hat{T}|$ ,  $T$ ,  $n$ ,  $P$ ,  $m$ ,  $\gamma$ ):
2    $occurrences \leftarrow 0$ 
3    $shift \leftarrow 1$ 
4    $hbc \leftarrow \text{PreprocessHBC}(\bar{P}, |\bar{P}|)$ 
5   while  $shift \leq |\hat{T}| - |\bar{P}|$  do
6      $i \leftarrow 2$ 
7     while  $i \leq |\bar{P}|$  and  $\bar{P}[i - 1] = \hat{T}[shift + i] - \hat{T}[shift + i - 1]$  do
8        $i \leftarrow i + 1$ 
9     if  $i = |\bar{P}| + 1$  then
10       $start \leftarrow \hat{T}[shift] - \gamma$ 
11       $i \leftarrow 1$ 
12      while  $i \leq m$  and ( $P[i] = T[start + i]$  or  $P[i] \in \Sigma_p$ ) do
13         $i \leftarrow i + 1$ 
14      if  $i = m + 1$  and  $\text{CHECK-P-MATCH}(P, T[start..start + m - 1], m)$  then
15         $occurrences \leftarrow occurrences + 1$ 
16       $shift \leftarrow shift + hbc[\bar{P}[1]]$ 
17      continue
18       $k \leftarrow \hat{T}[shift + |\bar{P}|] - \hat{T}[shift + |\bar{P}| - 1]$ 
19       $shift \leftarrow shift + hbc[k]$ 
20  return  $occurrences$ 

```

Despite these advantages, CDS has limitations. Its performance depends on pivot selection, which must balance index size and execution time. Studies indicate that, in English text, the pivot character ranked eighth in frequency offers optimal results. Additionally, for very short patterns without pivot occurrences, standard string search methods are required. In small-alphabet texts, CDS may not provide space efficiency benefits, though the use of condensed alphabets [28] can mitigate this issue. Recently, further improvements in space and time efficiency have been introduced using the *fake distance representation* [29].

4. Experimental Results

This section presents the evaluation of our parameterized string matching approach, Character Distance Sampling (pCDS). We compare pCDS against the parameterized Boyer–Moore (pBM), parameterized Turbo-Boyer–Moore (pTBM), and our adapted parameterized Horspool (pHor) algorithm.

The parameterized Horspool (pHor) algorithm is an adaptation of the classical Horspool string matching algorithm to the parameterized setting. In this approach, characters belonging to the parameter alphabet Σ_p are treated as *don't-care* symbols during the searching phase, allowing them to match any character in the text. The algorithm applies the standard Horspool bad-character heuristic on a transformed version of the pattern, computing shifts based only on the relevant symbols. As a consequence of the relaxed matching criterion, the algorithm may produce false positives. Therefore, each candidate occurrence must be verified using a parameterized matching check that enforces the bijective mapping between parameter symbols. This approach provides a simple and efficient baseline, combining fast shift-based filtering with a verification phase similar to that used in more advanced parameterized matching algorithms.

The algorithms were implemented in C and tested using a modified Smart tool [30], optimized for parameterized text sampling. Experiments were conducted on a MacBook Pro with a 2 GHz Intel Core i7 processor, 4 cores, 16 GB RAM, and 6 MB L3 Cache.

We evaluated performance across multiple randomly generated benchmark datasets of size 5 MB, each characterized by a different parameter probability (φ). For each value of φ , parameterized texts were generated by independently replacing each fixed character with a parameter symbol with probability φ . We then extracted 1000 distinct patterns of the desired length uniformly at random from each text, ensuring that every pattern occurred at least once in the corresponding dataset. The primary performance metric is the average execution time required to match a pattern.

Notably, for CDS-based methods, the additional overhead required for constructing auxiliary structures (e.g., partial indices) remains minimal, ensuring that execution performance is not adversely affected.

Experimental results are presented in Table 1. They clearly highlight significant differences in the performance of the evaluated algorithms.

Among the tested approaches, pCDS consistently achieves the best execution times. Its performance is significantly superior across all values of φ and pattern lengths m , demonstrating its efficiency in parameterized string matching. In contrast, the pBM and pTBM algorithms exhibit higher execution times, with pTBM generally performing slightly better than pBM, although with some fluctuations. The pHor algorithm offers intermediate performance, outperforming pBM but remaining less efficient than pCDS.

As φ increases, meaning a larger proportion of parameterized characters in the text, the performance of all algorithms deteriorates. This effect is particularly evident in the case of pHor, which experiences a notable increase in execution time as φ grows from 10% to 50%. The pBM and pTBM methods show similar trends, with execution times increasing gradually. Although pCDS remains the most efficient solution, it is also affected by the rise in φ , with its execution time nearly doubling when φ increases from 10% to 50%.

The impact of the pattern length m is also significant. As m increases, all algorithms benefit from reduced execution times. However, this improvement is particularly pronounced for pCDS, which scales exceptionally well. For instance, while pBM, pTBM, and pHor stabilize at relatively high execution times for large m , pCDS achieves a dramatic reduction, dropping from approximately 40 ms for $m = 32$ to only 3.8 ms for $m = 1024$.

This confirms the superior scalability of the CDS-based approach in handling long patterns efficiently.

Table 1. Comparative performance of parameterized string matching algorithms across different values of φ and pattern lengths m . Execution time values are in milliseconds. Bold values indicate best performance, while the second-best results are highlighted in gray.

φ	m	pBM	pTBM	pHor	pCDS
10	32	73.55	66.48	57.46	39.77
	64	55.84	53.50	46.93	23.52
	128	52.25	49.49	43.83	13.20
	256	51.57	58.37	44.25	06.42
	512	50.51	46.98	42.32	04.73
	1024	49.45	48.51	42.79	03.80
20	32	75.86	78.64	63.45	46.01
	64	60.53	60.52	51.40	27.47
	128	54.00	52.18	47.76	13.35
	256	53.50	60.92	45.46	06.76
	512	51.80	51.61	47.05	05.02
	1024	53.81	48.83	47.26	03.92
30	32	82.16	74.93	71.25	54.83
	64	61.56	63.91	53.70	30.18
	128	55.31	54.02	50.47	14.77
	256	55.97	63.56	51.94	06.56
	512	53.24	52.59	50.76	04.85
	1024	53.61	51.03	50.07	03.87
40	32	93.01	92.08	88.81	70.99
	64	68.03	62.68	65.55	37.19
	128	59.75	59.95	59.04	17.51
	256	57.85	67.97	59.62	06.52
	512	53.29	53.82	55.65	04.45
	1024	55.55	55.01	55.49	03.63
50	32	103.83	97.49	112.45	96.46
	64	73.97	79.07	76.10	52.46
	128	61.82	54.65	64.32	23.02
	256	54.32	67.90	63.32	07.42
	512	54.12	55.52	61.80	04.47
	1024	56.11	55.17	62.86	03.43

Each of the evaluated algorithms presents distinct strengths and weaknesses. The pBM algorithm is a well-established method with solid theoretical foundations. It performs well for long patterns but suffers from high computational costs when applied to short patterns or when φ is high. Its optimized variant, pTBM, aims to improve upon pBM by introducing more efficient pattern shifts. However, its performance varies across different settings, and it does not consistently surpass pBM in all cases.

The pHor algorithm provides a balanced trade-off between efficiency and implementation complexity. It generally outperforms pBM and pTBM but exhibits a noticeable performance degradation for longer patterns and higher values of φ . This makes it a reasonable choice when seeking a simple yet effective approach, though it does not achieve the same level of efficiency as pCDS.

Among all tested methods, pCDS emerges as the fastest and most scalable solution. It significantly reduces execution times across all scenarios and is particularly well-suited for large patterns. However, its efficiency declines more rapidly as φ increases, indicating that its advantages are more pronounced in cases where the proportion of parameterized characters is moderate.

The results demonstrate that pCDS is the most efficient algorithm for parameterized string matching, achieving the lowest execution times across all tested scenarios. When φ is low, all algorithms perform relatively well, but pCDS consistently maintains an advantage. The benefits of pCDS become even more pronounced for longer patterns, where its execution time is significantly lower than that of the other competing methods.

However, for higher values of φ ($\geq 40\%$), pCDS experiences a more noticeable performance drop compared to pBM and pTBM. Despite this, it remains the best-performing algorithm overall.

In conclusion, pCDS represents the most effective solution for parameterized string matching, particularly when the percentage of parameterized characters is moderate. Its combination of speed and scalability makes it highly suitable for real-world applications.

5. Conclusions and Future Works

This paper introduced pCDS, a novel approach for parameterized string matching based on Character Distance Sampling. The proposed method extends sampling-based string matching techniques to the parameterized setting by combining a compact partial representation of the text with a verification phase tailored to the bijective mapping constraints of parameterized matching. Unlike traditional online approaches, pCDS exploits partial indexing information to reduce the amount of text that must be inspected while avoiding the memory overhead required by full-text indexing structures.

The experimental results show that pCDS consistently achieves lower execution times than established parameterized string matching approaches, such as pBM, pTBM, and pHor, across the tested datasets and pattern lengths. In particular, pCDS provides significant benefits for large patterns, where the sampled representation allows effective filtering of candidate occurrences and improved scalability. These results demonstrate that character distance sampling is a promising strategy for reducing the computational cost of parameterized string matching while maintaining limited memory requirements.

However, the effectiveness of pCDS depends on the characteristics of the input data. In particular, the proportion of parameterized characters (φ) influences the efficiency of the sampling phase, and the performance advantage becomes less pronounced for datasets containing a high percentage of parameter symbols. Furthermore, the choice of pivot characters remains an important factor, as unsuitable sampling configurations may reduce the effectiveness of the partial index. These aspects represent limitations of the current approach and motivate further investigation.

Future work will focus on a more extensive analysis of the relationship between pivot selection, alphabet distribution, and the value of φ . Additional experiments on larger and more diverse real-world datasets will be conducted to further validate the applicability of pCDS. Moreover, investigating adaptive pivot selection strategies, improved sampling schemes, and offline or indexed parameterized matching solutions based on CDS may further enhance the flexibility and scalability of the proposed approach.

Overall, pCDS represents a step toward bridging the gap between fully online parameterized matching algorithms and memory-intensive indexed solutions. Its ability to exploit compact sampled representations while preserving the correctness guarantees of parameterized matching makes it a promising framework for future developments in large-scale string processing applications.

Author Contributions: Conceptualization, S.F., T.L., F.P.M., A.P. and A.S.; Methodology, S.F., T.L., F.P.M., A.P. and A.S.; Software, S.F., T.L., F.P.M., A.P. and A.S.; Validation, S.F., T.L., F.P.M., A.P. and A.S.; Formal analysis, S.F., T.L., F.P.M., A.P. and A.S.; Investigation, S.F., T.L., F.P.M., A.P. and A.S.; Resources, S.F., T.L., F.P.M., A.P. and A.S.; Data curation, S.F., T.L., F.P.M., A.P. and A.S.; Writing—original draft, S.F., T.L., F.P.M., A.P. and A.S.; Writing—review & editing, S.F., T.L., F.P.M., A.P. and A.S.; Visualization, S.F., T.L., F.P.M., A.P. and A.S.; Supervision, S.F., T.L., F.P.M., A.P. and A.S.; Project administration, S.F., T.L., F.P.M., A.P. and A.S.; Funding acquisition, S.F., T.L., F.P.M., A.P. and A.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Faro, S.; Lecroq, T. The exact online string matching problem: A review of the most recent results. *ACM Comput. Surv.* **2013**, *45*, 13:1–13:42. [[CrossRef](#)]
2. Mendivelso, J.; Thankachan, S.V.; Pinzón, Y.J. A brief history of parameterized matching problems. *Discret. Appl. Math.* **2020**, *274*, 103–115. [[CrossRef](#)]
3. Ideguchi, H.; Hendrian, D.; Yoshinaka, R.; Shinohara, A. Efficient Parameterized Pattern Matching in Sublinear Space. In *Proceedings of the String Processing and Information Retrieval*; Nardini, F.M., Pisanti, N., Venturini, R., Eds.; Springer: Cham, Switzerland, 2023; pp. 271–283.
4. Baker, B.S. A Theory of Parameterized Pattern Matching: Algorithms and Applications. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, New York, NY, USA, 16–18 May 1993. [[CrossRef](#)]
5. Boyer, R.S.; Moore, J.S. A Fast String Searching Algorithm. *Commun. ACM* **1977**, *20*, 762–772. [[CrossRef](#)]
6. Faro, S.; Marino, F.P.; Pavone, A. Improved characters distance sampling for online and offline text searching. *Theor. Comput. Sci.* **2023**, *946*, 113684. [[CrossRef](#)]
7. Faro, S.; Marino, F.P.; Pavone, A.; Scardace, A. Towards an Efficient Text Sampling Approach for Exact and Approximate Matching. In *Proceedings of the Prague Stringology Conference 2021, Prague, Czech Republic, 30–31 August 2021*; Holub, J., Zdárek, J., Eds.; Czech Technical University in Prague: Prague, Czechia, 2021; pp. 75–89.
8. Faro, S.; Marino, F.P.; Moschetto, A. Beyond Horspool: A Comparative Analysis in Sampled Matching. In *Proceedings of the Prague Stringology Conference 2024, Prague, Czech Republic, 26–27 August 2024*; Holub, J., Zdárek, J., Eds.; Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science: Prague, Czechia, 2024; pp. 16–26.
9. Fredriksson, K.; Mozgovoy, M. Efficient parameterized string matching. *Inf. Process. Lett.* **2006**, *100*, 91–96. [[CrossRef](#)]
10. Salmela, L.; Tarhio, J. Fast parameterized matching with q-grams. *J. Discrete Algorithms* **2008**, *6*, 408–419. [[CrossRef](#)]
11. Amir, A.; Aumann, Y.; Cole, R.; Lewenstein, M.; Porat, E. Function Matching: Algorithms, Applications, and a Lower Bound. In *Proceedings of the Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, 30 June–4 July 2003*; Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2719, pp. 929–942. [[CrossRef](#)] [[PubMed](#)]
12. Hazay, C.; Lewenstein, M.; Tsur, D. Two Dimensional Parameterized Matching. In *Proceedings of the Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, 19–22 June 2005*; Apostolico, A., Crochemore, M., Park, K., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3537, pp. 266–279. [[CrossRef](#)]
13. Cole, R.; Hazay, C.; Lewenstein, M.; Tsur, D. Two-Dimensional Parameterized Matching. *ACM Trans. Algorithms* **2014**, *11*, 12:1–12:30. [[CrossRef](#)]
14. Idury, R.M.; Schäffer, A.A. Multiple Matching of Parameterized Patterns. In *Proceedings of the Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, CA, USA, 5–8 June 1994*; Crochemore, M., Gusfield, D., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1994; Volume 807, pp. 226–239. [[CrossRef](#)] [[PubMed](#)]

15. Baker, B.S. Parameterized diff. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, 17–19 January 1999*; Tarjan, R.E., Warnow, T.J., Eds.; ACM/SIAM: New York, NY, USA, 1999; pp. 854–855.
16. Hazay, C.; Lewenstein, M.; Sokol, D. Approximate Parameterized Matching. In *Proceedings of the Algorithms—ESA 2004, 12th Annual European Symposium, Bergen, Norway, 14–17 September 2004*; Albers, S., Radzik, T., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3221, pp. 414–425. [\[CrossRef\]](#)
17. Kosaraju, S.R. Faster Algorithms for the Construction of Parameterized Suffix Trees (Preliminary Version). In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, USA, 23–25 October 1995*; IEEE Computer Society: Washington, DC, USA, 1995; pp. 631–637. [\[CrossRef\]](#)
18. Cole, R.; Hariharan, R. Faster suffix tree construction with missing suffix links. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, Portland, OR, USA, 21–23 May 2000*; Yao, F.F., Luks, E.M., Eds.; ACM: New York, NY, USA, 2000; pp. 407–415. [\[CrossRef\]](#)
19. Amir, A.; Farach, M.; Muthukrishnan, S. Alphabet dependence in parameterized matching. *Inf. Process. Lett.* **1994**, *49*, 111–115. [\[CrossRef\]](#)
20. Baeza-Yates, R.A.; Gonnet, G.H. A New Approach to Text Searching. In *Proceedings of the SIGIR'89, 12th International Conference on Research and Development in Information Retrieval, Cambridge, MA, USA, 25–28 June 1989*; Belkin, N.J., van Rijsbergen, C.J., Eds.; ACM: New York, NY, USA, 1989; pp. 168–175. [\[CrossRef\]](#)
21. Crochemore, M.; Lecroq, T.; Czumaj, A.; Gasieniec, L.; Jarominek, S.; Plandowski, W.; Rytter, W. Speeding Up Two String-Matching Algorithms. In *Proceedings of the STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science, Cachan, France, 13–15 February 1992*; Finkel, A., Jantzen, M., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1992; Volume 577, pp. 589–600. [\[CrossRef\]](#) [\[PubMed\]](#)
22. Faro, S.; Marino, F.P.; Pavone, A. Efficient Online String Matching Based on Characters Distance Text Sampling. *Algorithmica* **2020**, *82*, 3390–3412. [\[CrossRef\]](#)
23. Claude, F.; Navarro, G.; Peltola, H.; Salmela, L.; Tarhio, J. String matching with alphabet sampling. *J. Discret. Algorithms* **2012**, *11*, 37–50. [\[CrossRef\]](#)
24. Matsuoka, Y.; Aoki, T.; Inenaga, S.; Bannai, H.; Takeda, M. Generalized pattern matching and periodicity under substring consistent equivalence relations. *Theor. Comput. Sci.* **2016**, *656*, 225–233. [\[CrossRef\]](#)
25. Kim, J.; Eades, P.; Fleischer, R.; Hong, S.H.; Iliopoulos, C.S.; Park, K.; Puglisi, S.J.; Tokuyama, T. Order-preserving matching. *Theor. Comput. Sci.* **2014**, *525*, 68–79. [\[CrossRef\]](#)
26. Faro, S.; Lecroq, T.; Marino, F.P. Optimal Character Distance Sampling for Exact String Matching Through Set Cover Reformulation. *IEEE Access* **2026**, *14*, 13910–13920. [\[CrossRef\]](#)
27. Faro, S.; Marino, F.P. Reducing Time and Space in Indexed String Matching by Characters Distance Text Sampling. In *Proceedings of the Prague Stringology Conference 2020, Prague, Czech Republic, 31 August–2 September 2020*; Holub, J., Zdárek, J., Eds.; Czech Technical University: Prague, Czech Republic, 2020.
28. Faro, S.; Marino, F.P.; Pavone, A. Enhancing Characters Distance Text Sampling by Condensed Alphabets. In *Proceedings of the 22nd Italian Conference on Theoretical Computer Science, Bologna, Italy, 13–15 September 2021*; Coen, C.S., Salvo, I., Eds.; CEUR Workshop Proceedings: Aachen, Germany, 2021; Volume 3072.
29. Faro, S.; Marino, F.P.; Moschetto, A.; Pavone, A.; Scardace, A. The Great Textual Hoax: Boosting Sampled String Matching with Fake Samples. In *Proceedings of the 12th International Conference on Fun with Algorithms (FUN 2024)*; Broder, A.Z., Tamir, T., Eds.; Schloss Dagstuhl–Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2024; Volume 291, pp. 13:1–13:17. [\[CrossRef\]](#)
30. Faro, S.; Lecroq, T.; Borzi, S.; Mauro, S.D.; Maggio, A. The String Matching Algorithms Research Tool. In *Proceedings of the Prague Stringology Conference 2016, Prague, Czech Republic, 29–31 August 2016*; Holub, J., Zdárek, J., Eds.; Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague: Prague, Czech Republic, 2016; pp. 99–111.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.