Simple Real-Time Constant-Space String Matching*

Dany Breslauer[1], Roberto Grossi[2], and Filippo Mignosi[3]

[1] Caesarea Rothchild Institute, University of Haifa, Haifa, Israel
[2] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
[3] Dipartimento di Informatica, Università dell'Aquila, L'Aquila, Italy

Abstract. We use a simple observation about the locations of *critical factorizations* to derive a real-time variation of the Crochemore-Perrin constant-space string matching algorithm. The real-time variation has a simple and efficient control structure.

1 Introduction

Numerous string matching algorithms have been published. The classical algorithm by Knuth, Morris and Pratt [24] takes $O(n + m)$ time and uses $O(m)$ auxiliary space, where n and m are the lengths of the text and the pattern, respectively. The algorithm is also *on-line* in the sense that it reports if an occurrence of the pattern ends at any given text location before moving on to examine the next text symbol. In comparison, the "naive" string matching algorithm works on-line and uses only constant auxiliary space, but takes up to $O(nm)$ time.

In some application, it is not sufficient that the output is produced on-line with an *average* amortized constant number of steps spent at each text location, but it is required that the output is produced in *real-time*, that is, worst case constant time spent at each text location (thus giving $O(m + n)$ time). The Knuth-Morris-Pratt algorithm has certain run-time "hiccups" that prevent real-time execution, spending up to $O(\log m)$ time at some text locations. Karp and Rabin's [23] linear-time randomized string matching algorithm requires only constant auxiliary space, but it comes in various randomized flavors and its real-time version may report false occurrences of the pattern with low probability.

In the early 1980s, two intriguing open question about the feasibility of string matching algorithms under certain conditions were settled when (a) Galil [15] derived a *real-time* variation of the Knuth-Morris-Pratt [24] algorithm and described a *predictability* condition allowing to transform a compliant on-line algorithms to real-time, and (b) Galil and Seiferas [18] discovered a linear-time

* Work partially supported by the European Research Council (ERC) project SFEROT, by the Israeli Science Foundation grant 347/09 and by Italian project PRIN AlgoDEEP (2008TFBWL4) of MIUR. Part of the work of the first author originated while visiting at the University of Palermo and of the second author originated while visiting the Caesarea Rothschild Institute.

R. Giancarlo and G. Manzini (Eds.): CPM 2011, LNCS 6661, pp. 173–183, 2011.
© Springer-Verlag Berlin Heidelberg 2011

string matching algorithms that requires only *constant auxiliary space* in addition to the read-only input storage; in earlier attempts Galil and Seiferas [16] first reduced the space requirements in variations of the Knuth-Morris-Pratt algorithm to $O(\log m)$ space and then further to constant space, but temporarily "borrowing" and later "restoring" parts of the writable input storage [17]. Galil and Seiferas [18] point out that using their constant-space string matching algorithm, their earlier work [15,29] on real-time Turing machine algorithms for string matching, for recognition of squares and palindromes, and for a number of generalization of these problems can be adapted to use constant-space and even to two-way multi-head finite automata in real-time. Jiang and Li [22] proved that one-way multi-head finite automata cannot solve the string matching problem.

Several other linear-time constant-space string matching algorithms were published [2,6,7,8,9,10,11,20,21,28] later using various combinatorial properties of words. The simplest and most elegant of these, perhaps, is the algorithm by Crochemore and Perrin [7] that relies on the *Critical Factorization Theorem* [5,25,26]. More recently, Gasieniec and Kolpakov [19] studied the auxiliary space utilization in real-time string matching algorithms and derived a real-time variation of the constant-space algorithm by Gasieniec, Plandowski and Rytter [20], that is based on the partial representation of the "next function," using only $O(m^\epsilon)$ auxiliary space, for any fixed constant $\epsilon > 0$.

In this paper we revisit the Crochemore-Perrin [7] *two-way* constant-space string matching algorithm and its use of critical factorizations. We observe that if instead of verifying the pattern suffix in the algorithm's *forward scan* and only then verifying the remaining pattern prefix in the algorithm's *back fill*, we embark on the back fill simultaneously and at the same rate as the forward scan, then for most patterns the algorithm completes the back fill by the time the forward scan is done. We then prove that by deploying a *second instance* of the algorithm to match some pattern substring and by carefully choosing which critical factorizations are used in both instances, the two instances together can verify complementary parts of the pattern, and therefore, we can match the whole pattern in real-time. Thus, we derive a *real-time* variation of the Crochemore-Perrin algorithm circumventing the authors' conclusion that the very nature of their two-way algorithm would not allow it to operate in real-time. The new real-time algorithm has a very simple and efficient control structure, maintaining only three synchronized text pointers that move one way in tandem and induce homologous positions on the pattern, making the variation a good candidate for efficient hardware implementation in deep packet inspection network intrusion detection systems (e.g. [1]) and in other applications. Our observations about the choices of critical factorizations might be of independent interest. Breslauer, Jiang and Jiang [4] also use critical factorizations in the first half of the rotation of a periodic string to obtain better approximation algorithms for the *shortest superstring* problem. It is perhaps worthwhile to emphasis that this paper is concerned with the *auxiliary space* utilization where the input pattern and text are accessible in read-only memory, which is very different from the *streaming model* where space is too scarce to even store the whole inputs. In the streaming model

Breslauer and Galil [3] recently described a randomized real-time string matching algorithm using overall $O(\log m)$ space, improving on previous streaming algorithms [14,27] that take $O(n \log m)$ time.

We start by reviewing critical factorizations and give the basic real-time variation of the Crochemore-Perrin algorithm in Section 2. We then show in Section 3 how to use two instances of the basic algorithm to match any pattern and describe the pattern preprocessing in Section 4. We conclude with some remarks and open questions in Section 5.

2 Basic Real-Time Algorithm

We need the following definitions. A string u is *a period* of a string x if x is a prefix of u^k for some integer k, or equivalently if x is a prefix of ux. The shortest period of x is called *the period* of x and its length is denoted by $\pi(x)$. A *substring* or a *factor* of a string x is a contiguous block of symbols u, such that $x = x'ux''$ for two strings x' and x''. A *factorization* of x is a way to break x into a number of factors. We consider factorizations (u, v) of a string $x = uv$ into two factors: a *prefix* u and a *suffix* v. Such a factorization can be represented by a single integer and is *non-trivial* if neither of the two factors is equal to the empty string.

Given a factorization (u, v), a *local period* of the factorization is defined as a non-empty string z that is consistent with both sides u and v. Namely, (i) z is a suffix of u or u is a suffix of z, and (ii) z is a prefix of v or v is a prefix of z. The shortest local period of a factorization is called *the local period* and its length is denoted by $\mu(u, v)$. A non-trivial factorization (u, v) of a string $x = uv$ is called a *critical factorization* if the local period of the factorization is of the same length as the period of x, i.e. $\mu(u, v) = \pi(uv)$. See Figure 1.

After the preprocessing to find a critical factorization (u, v) of the pattern x, the Crochemore-Perrin algorithm [7] first verifies the suffix v in the what we call the *forward scan* and then verifies the prefix u in what we call the *back fill*. The key to the algorithm is that the algorithm always advances in the forward scan while verifying the suffix v and never needs to back up. The celebrated *Critical Factorization Theorem* is the basis for the Crochemore-Perrin two-way constant-space string matching algorithm.

Theorem 1. *(Critical Factorization Theorem, Cesari and Vincent [5,25]) Given any $|\pi(x)| - 1$ consecutive non-trivial factorizations of a string x, at least one is a critical factorization.*

```
  a | b a a a b a        a b | a a a b a        a b a | a a b a
b a   b a              a a a b   a a a b            a   a
```

(a) (b) (c)

Fig. 1. The local periods at the first three non-trivial factorizations of string abaaaba. In some cases the local period overflows on either side; this happens when the local period is longer than either of the two factors. The factorization (b) is a critical factorization with (local) period aaab.

We introduce the basic algorithm and assume throughout this section that all the critical factorizations and period lengths that are used were produced in the pattern preprocessing step. The basic idea is to find candidate occurrences of the pattern in real-time by repeatedly interleaving the comparisons in the (Crochemore-Perrin) back fill simultaneously with the comparisons in the forward scan. Note that the basic algorithm might interrupt some of these back fills before they come to completion (while the forward scans are always completed). As a result, the basic algorithm never misses an occurrence of the pattern, but might not fully verify that the produced candidates are real occurrences of the whole pattern, as stated below.

Lemma 1. *Given a pattern x with critical factorization (u, v), there exists a real-time constant-space algorithm that finds candidate occurrences of $x = uv$ such that:*

1. *The actual occurrences of the pattern x are never missed.*
2. *Candidate occurrences end with an occurrence of the pattern suffix v.*
3. *Candidate occurrences contain a specified substring of the pattern prefix u of length up to $|v|$.*

We now prove Lemma 1. Let $x = uv$ be the pattern with period of length $\pi(x)$ and critical factorization (u, v), such that $|u| < \pi(x)$. Such critical factorization always exists by Theorem 1. The basic algorithm aligns the pattern starting at the left end of the text and tries to verify the pattern suffix v (forward scan) and *simultaneously* also some other specified part of the skipped prefix u (back fill), advancing one location in each step.

Suppose that the algorithm has successfully verified some prefix z of v and some part of u, but failed to verify the next symbols, either in the suffix v or in the substring of u. The key observation in the Crochemore-Perrin algorithm is that the pattern may be shifted ahead by $|z| + 1$ text location, thus always moving forward in the text past the prefix of v that has been compared so far. To see this we must convince ourselves that any shorter shift can be ruled out as an occurrence of the pattern. We need the following lemma in our proof.

Lemma 2. *(Crochemore and Perrin [7]) Let (u, v) be a critical factorization of the pattern $x = uv$ and let z be any local period at this factorization, such that $|z| \leq \max(|u|, |v|)$. Then $|z|$ is a multiple of $\pi(x)$, the period length of the pattern.*

Since (u, v) is a critical factorization, by Lemma 2, any shift by $|z|$ or fewer symbols cannot align the pattern with occurrence in the text unless the shift is by a multiple of the period length $\pi(x)$. If the comparison that failed was verifying the suffix v, then shifts by the period length $\pi(x)$ can be ruled are since any pattern symbol $\pi(x)$ locations apart are identical. If the comparison that failed was verifying the prefix u, recalling that $|u| < \pi(x)$, there can be no such multiples of $\pi(x)$. In either case, the pattern may be shifted ahead with respect to the text by $|z| + 1$ locations without missing any occurrences. See Figure 2.

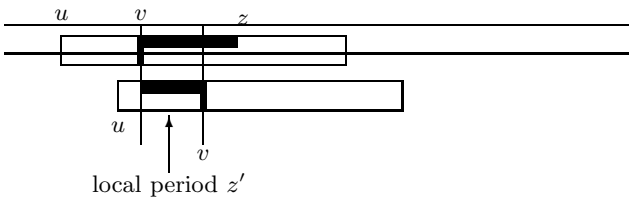

Fig. 2. If z is a prefix of v verified in the forward scan, and z' is a prefix of z and a suffix of u, then the critical factorization (u, v) should have a local period z' shorter than the period $\pi(x)$, which gives a contradiction

If the algorithm has finished verifying the whole suffix v, then it has also verified a substring of u of up to the same length, and therefore, if $|u| \leq |v|$ and the algorithm has started at the beginning of u, it has matched the whole pattern $x = uv$. However, if $|u| > |v|$, then the algorithm has only verified the some substring of u of length at most $|v|$, but has not verified the remainder of the pattern (unfortunately, some patterns, e.g. $aa \cdots aaab$, only have critical factorizations close to their right end).

By Lemma 2, subsequent overlapping occurrences of the pattern must be at locations that are multiple of $\pi(x)$ locations apart, allowing the algorithm to shift the pattern ahead by $\pi(x)$ locations using the pattern periodicity to avoid going back. Observe that since $|u| < \pi(x)$, there is no need to go back to verify the substring of the prefix u after shifting the pattern ahead by $\pi(x)$ locations. To simplify the exposition, we give up on this optimization that was discussed by Crochemore and Perrin [7] and further optimized in [2].

We have therefore demonstrated that the algorithm has not missed occurrences of the pattern, and has verified that the candidate occurrences it produced end with v and contained an occurrence of a specified substring of u of length up to $|v|$. Also, after the last symbol of a pattern occurrence is read, it takes just $O(1)$ time to report an occurrence candidate, so the basic algorithm is real-time; the fact it uses $O(1)$ additional space, derives from [7]. The reason we specified a substring of u rather then a prefix of u will become apparent later, where two instances of the algorithm will be used to cover complementary parts of u and one of the instances does not even require the substring of u. This completes the proof of Lemma 1.

3 Real-Time Variation

Consider the basic algorithm from Lemma 1 that finds candidate occurrences of the pattern $x = uv$ with critical factorization (u, v). As mentioned in the proof of Lemma 1, when $|u| \leq |v|$, the basic algorithm is able to verify all the actual occurrences of x. What if $|u| > |v|$? We have to refine the basic algorithm and consider the pattern as $x = uvw$: the main idea to obtain the real-time constant-space variation of the Crochemore-Perrin algorithm is to use

two aligned instances (see case (2) of Lemma 3 below). When taken together, the complementary parts in x that are verified by the two instances *cover* the whole pattern occurrences, thus identifying the actual occurrences of x. The variation relies on the following observation about the locations of critical factorizations.

Lemma 3. *Given any string $x = uvw$, there either exists:*

1. *a critical factorization (uv, w) such that $|uv| \leq |w|$, or*
2. *a critical factorization (uv, w) and another critical factorization (u, vv') of a prefix uvv' of x, such that $|u| \leq |vv'|$.*

Proof. Let (uv, w) be the leftmost critical factorization of $x = uvw$. If $|uv| \leq |w|$, then we have proved case (1). Otherwise, consider all factorizations (u, vw) to the left of (uv, w) that further satisfy the condition that $|u| \leq \mu(u, vw)$. If also $|vw| \leq \mu(u, vw)$, then the local period covers both extremes and (u, vw) is a critical factorization, contradicting that (uv, w) was the leftmost critical factorization. Therefore, $\mu(u, vw) < |vw|$. By Theorem 1, let (u, v) be a critical factorization of uv, such that $|u| \leq \pi(uv) = \mu(u, v)$, and let vv' be the prefix of vw whose length is $\max(|v|, \mu(u, vw)) < |vw|$ (in fact, vv' can be chosen such that uvv' is the longest prefix with period length $\mu(u, vw)$). But then (u, vv') is a critical factorization of uvv' and $|u| \leq \mu(u, v) \leq \mu(u, vv') = \mu(u, vw) \leq |vv'|$. □

Note that case (1) of Lemma 3 holds for most patterns strings: as previously mentioned, one instance of the basic real-time algorithm in Lemma 1 solves the string matching problem. The second instance of the basic algorithm is only required in case (2). There are many different ways to divide up the work between the two instances. We discuss a simple one.

Theorem 2. *There exists a real-time constant-space string matching algorithm that identifies all the occurrences of a pattern x, and uses only two instances of the basic algorithm mentioned in Lemma 1.*

To prove Theorem 2, let the first instance use the critical factorization (uv, w) of the whole pattern $x = uvw$ (see Lemma 3) and verify only the suffix w in the forward scan (no back fill). Let the second instance match occurrences of the pattern prefix uvv' (case (2) of Lemma 3) using the critical factorization (u, vv'). The correctness of the algorithm follows from the observation that aligned occurrences of w (partially) overlapping with occurrences of uvv' ending $|w| - |v'|$ text locations earlier, identify occurrences of the whole pattern $x = uvw$. The complexity is twice that of the basic algorithm, and the resulting algorithm is real-time since we repeatedly interleave $O(1)$ steps of the two instances. The variation makes fewer than $3n - m$ symbol comparisons.

We give the pseudocode of the real-time constant-space algorithm in Figure 3, where we focus on the case of the two factorizations mentioned in case (2) of Lemma 3 (since one factorization is straigthforward). We denote the pattern by $x \equiv x[0..m-1]$ and its two factorizations by $(u, vv') \equiv (x[0..a-1], x[a, c-1])$ and

```
for (t = 0, p = t-m+c-a, q = t-b; t < n; t++) {
   s = t-m+c;
   is_pref = false;     /* prefix instance */
   if (T[s]==x[s-p] && T[s-a]==x[s-p-a]) {
     if (s==p+c-1) {
        is_pref = true;
        p += pi_pref;
     }
  } else
     p = s-a+1;
  is_pat = false;       /* pattern instance */
  if (T[t]==x[t-q]) {
     if (t==q+m-1) {
        is_pat = true;
        q += pi_pat;
     }
  } else
     q = t-b+1;
  if (is_pref && is_pat)
        report an occurrence ending at text position t;
}
```

Fig. 3. The real-time constant-space variation of the Crochemore-Perrin algorithm for a pattern x, where $u \equiv x[0..a-1]$, $v \equiv x[a..b-1]$, $v' \equiv x[b..c-1]$, and $w \equiv x[b..m-1]$

$(uv, w) \equiv (x[0..b-1], x[b, m-1])$, whose periods $\pi(uvv') \equiv \pi(x[0..c-1])$ and $\pi(x) \equiv \pi(x[0..m-1])$ are precomputed during the pattern preprocessing (see Section 4). We denote these periods by `pi_pref` and `pi_pat` respectively. (Note that the two factorizations are displaced each other by $|v| = b - a$ positions.)

We denote the text by $T[0..n-1]$ and the location of the current text symbol by t, where $s = t - m + c$ is the end of the text as perceived by the independent prefix instance matching uvv'. The algorithm has a very simple control structure using only three aligned text position $s - a < s < t$ that advance in tandem.

Consider the prefix instance $(u, vv') \equiv (x[0..a-1], x[a, c-1])$: the text position for the current candidate occurrence of vv' is denoted by p (and so the candidate position for u is $p - a$). We apply the forward scan to $vv' \equiv x[a, c-1]$ and the back fill to $u \equiv x[0..a-1]$ in the first half of the `for` loop, so we try to match one symbol from vv' and one from u.

Consider the pattern instance $(uv, w) \equiv (x[0..b-1], x[b, m-1])$: the text position for the current candidate occurrence of $w \equiv x[b, m-1]$ is denoted by q. We just apply the forward scan to $w = x[b, m-1]$, since $uv \equiv x[0..b-1]$ is covered by the prefix instance. We just need to check for the occurrences of w, and we try to match one symbol of it in the second half of the `for` loop.

As it can be verified in Figure 3, we report an occurrence of pattern x ending at position t after $O(1)$ time and we use just $O(1)$ additional memory words.

4 Pattern Preprocessing

We now describe the pattern preprocessing, where the task is that of computing at most two critical factorizations and the corresponding periods. Namely, given a pattern x, the pattern preprocessing finds at most two factorizations (uv, w) and (u, vv') (if needed) as stated in Lemma 3, where $x = uvw$, and the corresponding period lengths $\pi(x)$ and $\pi(uvv')$.

We build upon the Crochemore-Perrin preprocessing algorithms [7] for the above task. Consider the computation of the periods. Crochemore and Perrin gave a pattern preprocessing algorithm that computes the length of the period $\pi(x)$, when the pattern x is periodic ($\pi(x) \leq |x|/2$), and gave a variation of their algorithm that works when the period is longer than half the pattern length. Other authors have shown how to compute the period length exactly when the period is longer than half of the pattern length [2,8,10]. Our real-time variation can use either approach and either compute the period lengths precisely, or use half the pattern length as an approximation for the period length (i.e. if the period is longer than $|x|/2$, then shift the pattern x by $|x|/2$ positions instead of shifting pattern by the unknown long period length).

Consider now the computation of critical factorizations. Crochemore and Perrin gave a novel constructive proof of the Critical Factorization Theorem (Theorem 1) using properties of lexicographically maximal suffixes and Lyndon words, and a variation of an algorithm by Duval [13] to find the lexicographically maximal suffix of a string. Their proof shows that given any arbitrary order $\leq$ on the input alphabet and its reverse order $\leq^R$, the shorter between the lexicographically maximal suffix of the pattern x by $\leq$ and the lexicographically maximal suffix of x by $\leq^R$, provides a critical factorization in the first $\pi(x) - 1$ positions. The readers are referred to the original paper [7] for the elegant proof.

We will only use the fact that critical factorizations of growing prefixes of the pattern x, within their first period, can be efficiently computed on-line. Before going on, we need a few more definitions and properties of critical factorizations. A factorization (u, v) of a string $x = uv$ is *left external* if $|u| \leq \mu(u, v)$ and similarly, *right external* if $|v| \leq \mu(u, v)$. A factorization is called *external* if it is both left and right external, and *internal* if it is neither left or right external. Any external factorization has a local period length that is equal to the global period length, i.e. $\mu(u, v) = \pi(uv)$, and is therefore critical. Any internal factorization has a *square* centered at the factorization.

We define the set of all the left external non-trivial factorizations $\mathcal{L}(x) = \{(u, v) \| \, x = uv \text{ and } 1 \leq |u| \leq \mu(u, v) \text{ and } 1 \leq |v|\}$. This set is not empty since it always contains the first factorization (a, v), where a is an alphabet symbol, and it also contains at least one critical factorization by Theorem 1. We now give two properties to characterize $\mathcal{L}(x)$, the first of which can also be partially found in [25, Ch.8].

Lemma 4. *Let (u_1, v_1) and (u_2, v_2) be two factorizations in $\mathcal{L}(x)$, such that $|u_1| \leq |u_2|$. Then the local period lengths satisfy $\mu(u_1, v_1) \leq \mu(u_2, v_2)$.*

Proof. Let x' be a prefix of $x = u_1v_1 = u_2v_2$ of length $|x'| = \min(|x|, |u_1| + \mu(u_1, v_1), |u_2|+\mu(u_2, v_2))$. Then, x' has periods of lengths $\mu(u_1, v_1)$ and $\mu(u_2, v_2)$. Assuming by contradiction that $\mu(u_1, v_1) > \mu(u_2, v_2)$, then the shorter prefix of length $|u_1| + \mu(u_2, v_2)$ has a period of length $\mu(u_2, v_2)$ and therefore (u_1, v_1) has a local period of length at most $\mu(u_2, v_2) < \mu(u_1, v_1)$, establishing that $\mu(u_1, v_1) \leq \mu(u_2, v_2)$. □

Lemma 5. *Let (u, v) be a factorization in $\mathcal{L}(x)$. Then, there exist a prefix x' of x and a left external critical factorization (u_0, v_0) of x', such that $x' = u_0v_0$ and $\mu(u_0, v_0) = \mu(u, v)$.*

Proof. If (u, v) is a critical factorization of $x = uv$ then simply take $(u_0, v_0) = (u, v)$. Otherwise, (u, v) cannot be right external. Let x' be the prefix of x of length $|u| + \mu(u, v) < |x|$. Then x' has period length $\pi(x') = \mu(u, v)$ and left external critical factorization (u_0, v_0), such that $\mu(u_0, v_0) = \pi(u_0v_0) = \mu(u, v)$. □

Given any on-line algorithm that computes left external critical factorizations of growing prefixes of the pattern x (e.g. the Crochemore-Perrin pattern preprocessing algorithm), consider the factorizations induced in x by the critical factorizations of the pattern prefixes. By Lemma 4, the local period lengths of such factorizations are non-decreasing for increasing lengths prefixes, and by Lemma 5, all local period lengths in $\mathcal{L}(x)$ are represented. The constuctive algorithm for finding the factorization in Lemma 3 boils down to the prefix critical factorizations that are representative of the largest two local period lengths in $\mathcal{L}(x)$. We can now describe how to obtain the pattern preprocessing.

Theorem 3. *There exist an on-line constant-space pattern preprocessing algorithm that produces the critical factorizations required in Lemma 3 and the corresponding periods.*

Proof. We already discussed how to find the periods at the beginning of this section. As for the desired critical factorizations, take any on-line constant-space algorithm that computes the left external critical factorizations of growing prefixes of the pattern x and record the last two different critical factorizations (u_0, v_0) and (u_1, v_1) of the pattern prefixes $x_0 = u_0v_0$ and $x_1 = u_1v_1$, such that $|x_0| < |x_1|$ and $|u_0| < |u_1|$ and stop when x_1 is the first critical factorization in the second half of the pattern x, such that $|u_1| \geq |x|/2$, or when the end of the pattern is reached. Since any left external critical factorization in the second half of x is also right external in x and therefore a critical factorization, the last prefix critical factorization x_1 that was found is critical for the whole pattern x. If either of these last two prefix critical factorizations is a critical factorization of the whole pattern x in the first half of x, then we found one critical factorization satisfying case (1) in Lemma 3. Otherwise, (u_0, v_0) induces a non-critical factorization (u_0, v_0v') of u_0v_0v', where u_0v_0v' is the longest prefix of x having period $\mu(u_0, v_0)$ and v_0 is a prefix of v_0v'. Notice that since above prefix critical factorization (u_0, v_0) is left external then $|u_0| \leq |v_0| \leq |v_0v'|$. The second factorization (u_1, v_1) induces a critical factorization in the whole pattern $x = u_1v_1$ since $|u_1| \geq |x|/2$. Letting $u = u_0$, $u_1 = u_0v$ and $x = uvw$, we get the desired three way factorization satisfying case (2) of Lemma 3. □

While the Crochemore-Perrin preprocessing algorithm is works in constant space and in linear time, it does not poroduce the period lengths and critical factorizations in real time. However, the $O(m)$ time preprocessing results of Theorem 3 are produced in time so that the ensuing text scanning works in real-time.

5 Concluding Remarks

We have shown that the Crochemore-Perrin constant-space algorithm [7] can be directly transformed into a real-time string matching algorithm by interleaving its back fill with its forward scan. Other on-line constant-space string matching algorithms can be transformed into real-time algorithms as well, provided that they report the locations of pattern occurrences in increasing order of their locations and no later than cm locations after the end the occurrence, for some fixed constant c and any length m pattern. The general transformation searches for occurrences of a pattern prefix and uses such occurrences as an anchor while catching up with the remaining pattern suffix, utilizing periodicity properties to count highly repetitive pattern prefixes.

The pattern preprocessing of the Crochemore-Perrin algorithm [2,7] computes periods and critical factorizations using symbol comparisons that test for the relative order of symbols under an arbitrary alphabet order. It is an interesting open question to find a critical factorization in linear-time using only equality comparisons without an alphabet order, even if more space is allowed. Duval et al. [12] find all local periods of a string and therefore, also all critical factorizations, in linear time over integer alphabets, but require super-linear time if only symbol comparisons are used.

Acknowledgments

We are grateful to Danny Hermelin, Gadi Landau and Oren Weimann for several discussions.

References

1. Baker, Z.K., Prasanna, V.K.: Time and area efficient pattern matching on FPGAs. In: Tessier, R., Schmit, H. (eds.) FPGA, pp. 223–232. ACM, New York (2004)
2. Breslauer, D.: Saving Comparisons in the Crochemore-Perrin String Matching Algorithm. Theoret. Comput. Sci. 158(1), 177–192 (1996)
3. Breslauer, D., Galil, Z.: Real-Time Streaming String-Matching (2011), this conference proceedings
4. Breslauer, D., Jiang, T., Jiang, Z.: Rotations of periodic strings and short superstrings. J. Algorithms 24(2), 340–353 (1997)
5. Césari, Y., Vincent, M.: Une caractérisation des mots périodiques. C.R. Acad. Sci. Paris 286(A), 1175–1177 (1978)
6. Crochemore, M.: String-matching on ordered alphabets. Theoret. Comput. Sci. 92, 33–47 (1992)

7. Crochemore, M., Perrin, D.: Two-way string-matching. J. Assoc. Comput. Mach. 38(3), 651–675 (1991)
8. Crochemore, M., Rytter, W.: Periodic Prefixes in Texts. In: Capocelli, R., Santis, A.D., Vaccaro, U. (eds.) Proc. of Sequences II: Methods in Communication, Security and Computer Science, pp. 153–165. Springer, Heidelberg (1993)
9. Crochemore, M., Rytter, W.: Text Algorithms. Oxford University Press, Oxford (1994)
10. Crochemore, M., Rytter, W.: Sqares, Cubes, and Time-Space Efficient String Searching. Algorithmica 13(5), 405–425 (1995)
11. Crochemore, M., Rytter, W.: Jewels of stringology. World Scientific, Singapore (2002)
12. Duval, J.P., Kolpakov, R., Kucherov, G., Lecroq, T., Lefebvre, A.: Linear-time computation of local periods. Theor. Comput. Sci. 326(1-3), 229–240 (2004)
13. Duval, J.: Factorizing Words over an Ordered Alphabet. J. Algorithms 4, 363–381 (1983)
14. Ergun, F., Jowhari, H., Salgan, M.: Periodicity in Streams (2010) (manuscript)
15. Galil, Z.: String Matching in Real Time. J. Assoc. Comput. Mach. 28(1), 134–149 (1981)
16. Galil, Z., Seiferas, J.: Saving space in fast string-matching. SIAM J. Comput. 2, 417–438 (1980)
17. Galil, Z., Seiferas, J.: Linear-time string-matching using only a fixed number of local storage locations. Theoret. Comput. Sci. 13, 331–336 (1981)
18. Galil, Z., Seiferas, J.: Time-space-optimal string matching. J. Comput. System Sci. 26, 280–294 (1983)
19. Gasieniec, L., Kolpakov, R.M.: Real-time string matching in sublinear space. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 117–129. Springer, Heidelberg (2004)
20. Gasieniec, L., Plandowski, W., Rytter, W.: Constant-space string matching with smaller number of comparisons: Sequential sampling. In: Galil, Z., Ukkonen, E. (eds.) CPM 1995. LNCS, vol. 937, pp. 78–89. Springer, Heidelberg (1995)
21. Gasieniec, L., Plandowski, W., Rytter, W.: The zooming method: A recursive approach to time-space efficient string-matching. Theor. Comput. Sci. 147(1&2), 19–30 (1995)
22. Jiang, T., Li, M.: K one-way heads cannot do string-matching. J. Comput. Syst. Sci. 53(3), 513–524 (1996)
23. Karp, R., Rabin, M.: Efficient randomized pattern matching algorithms. IBM J. Res. Develop. 31(2), 249–260 (1987)
24. Knuth, D., Morris, J., Pratt, V.: Fast pattern matching in strings. SIAM J. Comput. 6, 322–350 (1977)
25. Lothaire, M.: Combinatorics on Words. Addison-Wesley, Reading (1983)
26. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
27. Porat, B., Porat, E.: Exact And Approxiamate Pattern Matching In The Streaming Model. In: Proc. 50th IEEE Symp. on Foundations of Computer Science, pp. 315–323 (2009)
28. Rytter, W.: On maximal suffices and constant-space linear-time versions of kmp algorithm. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 196–208. Springer, Heidelberg (2002)
29. Seiferas, J.I., Galil, Z.: Real-time recognition of substring repetition and reversal. Mathematical Systems Theory 11, 111–146 (1977)