SIAM J. COMPUT.
Vol. 21, No. 3, pp. 407-437, June 1992

ON THE EXACT COMPLEXITY OF STRING MATCHING: UPPER BOUNDS*

ZVI GALIL† AND RAFFAELE GIANCARLO‡

Abstract. It is shown that, for any pattern of length m and for any text of length n, it is possible to find all occurrences of the pattern in the text in overall linear time and at most $\frac{4}{3}n-\frac{1}{3}m$ character comparisons. In fact, the bound on the number of character comparisons is usually tighter than this, for the bound is expressed in terms of the structure of the pattern. The algorithm here need not have any knowledge of the alphabet. This improves the best previous bound of $1.5n-.5(m-1)$ obtained by Colussi [*Inform. and Comput.*, to appear] and Apostolico and Crochemore [Tech. Report TR89-75, LITP, Université de Paris, Paris, France, 1989]. In a companion paper [*SIAM J. Comput.*, 20 (1991), pp. 1008–1020], the authors show a lower bound for on-line algorithms that is equal to $\frac{4}{3}n-\frac{1}{3}m$ for $m=3$. For $m=1, 2$, n character comparisons is optimal. This algorithm is based on a new analysis of the string matching algorithm by Colussi. Moreover, this new analysis of Colussi's algorithm confirms the experimental results showing that his algorithm performs very well in practice [*Inform. and Comput.*, to appear].

Key words. string matching, string searching, text editing, computational complexity, worst case behavior

AMS(MOS) subject classifications. 68Q20, 68Q25, 68U15

1. Introduction.

Given a computational problem, the ultimate goal is to determine its computational complexity *exactly*. However, this task is usually impossible for several reasons. One reason is that the exact complexity or even the constant factors depend on the machine model. Consequently, an exact determination of the time required by the problem is possible only if we restrict the model of computation, for example, by using a comparison tree model or considering straight line programs over a given set of operations.

For several problems involving order statistics the exact number of comparisons is known. The simplest of these are computing the maximum or minimum in $n-1$ comparisons (folklore), computing the maximum *and* the minimum in $\lceil \frac{3}{2}n-2 \rceil$ comparisons [20] and computing the largest two elements in $n-1+\lceil \log n \rceil$ comparisons [16], [22]. For many other problems, such knowledge is partial or not available. For instance, exact bounds for selecting the third largest element are known for all but a finite set of cases [14], [15], [23]. For the classic problem of sorting, the exact number of comparisons is known only up to $n=12$ [1], [9] and for some special cases [17]. A fascinating open problem is to find the exact number of comparisons for determining the median: the best upper bound is $3n$ [21] and the best lower bound is $2n$ [4].

In this paper, we use a RAM with uniform cost criterion, or alternatively a binary decision tree model [1]. We evaluate the time complexity of algorithms counting *all* operations and the number of comparisons. For example, the straightforward algorithm for computing the minimum of n numbers takes $O(n)$ time and performs $n-1$ comparisons in our model.

* Received by the editors October 15, 1990; accepted for publication (in revised form) May 21, 1991.

† Department of Computer Science, Columbia University, New York 10027, and Tel-Aviv University, Tel-Aviv, Israel. The work of this author was supported in part by National Science Foundation grant CCR 88-14977.

‡ AT&T Bell Labs, Murray Hill, New Jersey 07974. This author is on leave from the University of Palermo, Italy. Part of this work was performed while this author was at Columbia University, while he was supported in part by National Science Foundation grant CCR 88-14977 and by the Italian Ministry of University and Scientific Research, Project "Algoritmi e Sistemi di Calcolo."

We investigate the exact complexity of string matching over a general alphabet. Let $w = w[1, s]$ be a string. We denote by $w[i, j]$ the substring of w that starts at position i and ends at position j of $w[1, s]$. We denote the positions $i, i+1, \cdots, j$ as $[i, j]$. *String Matching* is the problem of finding all occurrences of a given pattern $p[1, m]$ in a given text $t[1, n]$. We say that the pattern occurs at text position i if $t[i+1, i+m] = p[1, m]$. By general alphabet we refer to the case of an infinite alphabet or of a finite alphabet unknown to the algorithm (thus it must work if the alphabet is $\{a, b, c, d\}$ or $\{\heartsuit, \diamond, \clubsuit, \spadesuit\}$). Most known linear-time, i.e., $O(n+m)$, string matching algorithms do not depend on the knowledge of the alphabet, since they only compare symbols. Another common feature of these algorithms is that they *preprocess* the pattern, i.e., they gather knowledge about the structure of the pattern and then start looking for it in the text. We count only the character comparisons that these algorithms perform while looking for occurrences of the pattern in the text. However, we do account for the time taken by the preprocessing. We establish new upper bounds on the number of character comparisons sufficient for an algorithm to correctly perform string matching. In a companion paper we provide lower bounds [11].

For quite some time, the best upper bound known has been $2n - m$ and corresponds to the number of character comparisons made by the Knuth, Morris, and Pratt algorithm [18] (KMP for short). Even the Boyer-Moore algorithm [5] (BM for short) could not beat the $2n - m$ bound. Indeed, Knuth [18] showed that BM performs at most $7n$ character comparisons, assuming that the pattern does not occur in the text. Under the same assumptions, Guibas and Odlyzko [13] reduced this bound to $4n$ and, very recently, Cole [6] showed that $3n$ is a tight bound (up to small order magnitude terms). Apostolico and Giancarlo [3] designed a variation of BM that achieves the $2n - m$ bound, irrespective of how many times the pattern occurs in the text. All of the analyses of BM mentioned here are quite intricate. Crochemore and Perrin [8] devised a new time-space-optimal string matching algorithm that performs at most $2n - m$ character comparisons. Recently, Colussi [7] devised an ingenious algorithm, which we refer to as SM, and showed that it performs at most $1.5n - .5(m-1)$ character comparisons (however, as we shall see, the true bound is much more involved). Independently, Apostolico and Crochemore [2] obtained a bound of $1.5n - m + 1$ by means of a simple modification of KMP. Their algorithm is a special case of Colussi's algorithm. Note that we can use the failure function used by KMP to derive deterministic finite automata [1] which perform string matching in exactly n comparisons. But this class of algorithms must know the alphabet to work correctly and have a running time that does depend (by a multiplicative factor) on the alphabet size. The algorithms we are interested in have a running time independent of the alphabet size.

We denote by $c(n, m)$ the maximal number of comparisons needed by a linear-time string matching algorithm, excluding any preprocessing step. It follows from the discussion above that $c(n, m) \leq 1.5n - .5(m-1)$. Let $c_{on\text{-}line}(n, m)$ be as $c(n, m)$ for on-line algorithms, that is, algorithms whose access to the text is limited to a sliding window of size m. Moreover, the window can be aligned with $t[i, i+m-1]$ if and only if the algorithm has already decided whether an occurrence of the pattern can start at position k of the text, for all k, $1 \leq k < i$. The best bounds currently known for $c_{on\text{-}line}(n, m)$ have been the same as the ones for $c(n, m)$.

We need a few definitions. A string $x[1, m]$ is *c periodic* if and only if $x[1, m-c] = x[c+1, m]$. We refer to c as a period of x. *The period* of a string x is the minimal integer l such that l is a period of x. We say that $x[1, m]$ is *strongly periodic* if and only if $m = kl + l'$, $k > 1$, $l' < l$, and l is the period of x. Given a pattern $p[1, m]$ whose period is z, $m = kz + z'$, $0 \leq z' < z$, the *periodic decomposition* of p is a sequence of integer

triples $(z_1, z_1', k_1), \cdots, (z_s, z_s', k_s)$ such that $(z_1, z_1', k_1) = (z, z', k)$; $p[1, z_{i-1} + z_{i-1}']$ is strongly periodic of period z_i (and thus $z_{i-1} + z_{i-1}' = k_i z_i + z_i'$), for $1 < i \leqq s$; $p[1, z_s + z_s']$ is not strongly periodic. Such a decomposition can be computed in $O(m)$ time using the preprocessing algorithm in [18]. The results of this paper can be summarized as follows:

(a) A new analysis of the algorithm SM [7] showing that it performs at most $n + \lfloor (n-m)(z_s'/z_s + z_s') \rfloor \leqq n + \lfloor n - m/2 \rfloor$ character comparisons. Our sharper bound confirms the experiments by Colussi showing that his algorithm performs very well in practice because for most patterns $s = 1$ and z_1' is smaller than z_1.

(b) Based on our analysis of SM, we devise a new algorithm and show that it performs at most $n + \lfloor (n-m)(\min(\frac{1}{3}, (z_s'+2)/2(z_s + z_s'))) \rfloor \leqq \frac{4}{3}n - \frac{1}{3}m$ character comparisons. Therefore, $c(n, m) \leqq c_{on\text{-}line}(n, m) \leqq \frac{4}{3}n - \frac{1}{3}m$. In the companion paper [11], we show a lower bound for $c_{on\text{-}line}(n, m)$ that is equal to $\frac{4}{3}n - \frac{1}{3}m$ for $m = 3$.

We remark that it is possible to prove that SM performs at most $n + \lfloor (n-m) \times (\min(z, m-z)/m) \rfloor \leqq n + \lfloor n - m/2 \rfloor$ character comparisons and that our new algorithm performs at most $n + \lfloor (n-m)(\min(\frac{1}{3}, \min(z, m-z) + 2/2m)) \rfloor$ character comparisons. The interested reader is referred to [12] for proofs. Here we give proofs for the weaker bounds in (a) and (b) in order to simplify the presentation.

The $O(n+m)$ time bound of both algorithms is independent of the alphabet size. Both algorithms make at most n character comparisons for the important class of nonperiodic patterns, i.e., $z = m$. We remark that all the known linear-time string matching algorithms require more that n comparisons for this class of patterns.

The analysis of the algorithms presented here is complicated by the fact that they are oblivious: They sometimes forget the result of some comparisons when the pattern is moved over the text. These comparisons may have to be repeated in the future. Obliviousness in a string matching algorithm is not new. Indeed, the difficulty of the analysis of BM given in [3], [6], [13], and [18] is mostly due to the obliviousness of BM. However, none of the techniques in [3], [6], [13], and [18] could be used in our case since the obliviousness of BM is different than that of our algorithms, as explained in § 3.

This paper is organized as follows. In § 2 we show that, in order to bound $c(n, m)$ for any n and m, we can restrict our attention to patterns that are not strongly periodic, i.e., patterns having period of size z and length $m = z + z'$, $0 \leqq z' < z$. Such a reduction simplifies the presentation of our analysis of SM as well as of its improved version. We review SM in § 3. Section 4 contains all the combinatorial results needed for the analysis which is presented in § 5. An improved algorithm is presented and analyzed in § 6.

2. A reduction. We show that, in order to bound $c(n, m)$, we need consider only patterns that are not strongly periodic. We recall the following well-known *periodicity lemma* [18], [19].

LEMMA 1. *Let $y[1, n]$ be a string. If k_1 and k_2 are periods of y and $k_1 + k_2 \leqq n + \gcd(k_1, k_2)$, then $\gcd(k_1, k_2)$ is also a period of y.*

Let $l_i = z_i + z_i'$ denote the length of $p[1, z_i + z_i']$ in the periodic decomposition of $p[1, m]$, $1 \leqq i \leqq s$.

FACT 1. *Assume that $p[1, m]$ is strongly periodic and consider its periodic decomposition. For any fixed i, $1 < i \leqq s$, $p[1, k_i z_i]p[1, l_i] = (p[1, z_i])^{k_i+1} p[1, z_i']$ cannot be prefix of $p[1, 2z_{i-1} + z_{i-1}'] = (p[1, z_{i-1}])^2 p[1, z_{i-1}']$.*

Proof. Notice that in the periodic decomposition of $p[1, m]$, z_{j-1} cannot be a multiple of z_j, for any j, $2 \leqq j \leqq s$. Otherwise, by the periodicity lemma, z_{j-1} cannot be

the period of a strongly periodic string. Thus, $z'_{j-1} \neq z'_j$. Moreover, $z_j > z'_{j-1}$. Otherwise, $z_{j-1} + z_j \leqq l_{j-1}$ and, since z_{j-1} and z_j are both periods of $p[1, l_{j-1}]$, gcd (z_{j-1}, z_j) is also a period of that string by the periodicity lemma. But then, z_{j-1} cannot be the period of a strongly periodic string.

Assume that for some fixed i, $1 < i \leqq s$, $p[1, k_i z_i]p[1, l_i]$ is a prefix of $p[1, 2z_{i-1} + z'_{i-1}]$. The occurrence of $p[1, l_i]$ at $(k_i - 1)z_i + 1$, $z_{i-1} + 1$ and $k_i z_i + 1$ imply that $p[1, l_i]$ has periods $|l_i - z'_{i-1}|$ and $|z'_i - z'_{i-1}|(z_{i-1} + z'_{i-1} = k_i z_i + z'_i)$. Recall that it also has periods z_{i+1} and z_i and that $z'_i \neq z'_{i-1}$. Notice that $|l_i - z'_{i-1}| = l_i - z'_{i-1}$ since $z_i > z'_{i-1}$.

Assume that $z'_i - z'_{i-1} < 0$ ($z'_i - z'_{i-1} > 0$, respectively). Since $(l_i - z'_{i-1}) + (z'_{i-1} - z'_i) \leqq l_i$ ($z_i + (z'_i - z'_{i-1}) \leqq l_i$, respectively), $u_i \equiv \gcd(l_i - z'_{i-1}, z'_{i-1} - z'_i)$ ($q_i \equiv \gcd(z_i, z'_i - z'_{i-1})$, respectively) is a period of $p[1, l_i]$ be the periodicity lemma. In the first case $u_i = \gcd(z_i + z'_i - z'_{i-1}, z'_{i-1} - z'_i) \leqq z'_{i-1} - z'_i < z_i$ and u_i divides z_i. Also in the second case $q_i < z_i$ and q_i divides z_i. But then, z_i cannot be the period of a strongly periodic string. □

We now sketch an algorithm that, given all the occurrences of $p[1, l_s]$ in a text $t[1, n]$, will find all occurrences of $p[1, m]$ in $t[1, n]$. The algorithm has s stages and each stage has a list of candidates. During stage i, $i = s, s-1, \cdots, 2$, the ordered list of candidates satisfies the following invariant: all occurrences of $p[1, l_i]$ are in the list and each of those is a potential occurrence of the pattern in the text. This invariant is trivially satisfied for $i = s$ since $p[1, l_s]$ is a prefix of $p[1, m]$ and the algorithm knows all of its occurrences in $t[1, n]$. The last stage will produce all occurrences of p in t. We now describe stage i, $2 \leqq i \leqq s$.

Stage i. Consider the ordered list of candidates. By the periodicity lemma, the distance between two candidates is at least z_i. Divide the list into subsequences of maximal length such that the distance between any two candidates in the same subsequence is z_i. Let $q_1 < q_2 < \cdots < q_r$ be one of those subsequences. Consider q_b, an element of the chosen subsequence. Assume that $b > r - k_i + 1$. Notice that $p[1, l_{i-1}] = (p[1, z_i])^{k_i} p[1, z'_i]$ cannot occur at q_b; otherwise the maximality of the subsequence would be contradicted. Since $p[1, l_{i-1}]$ is a prefix of the pattern no occurrence of the pattern in the text can start at q_b for $r - k_i + 1 < b \leqq r$. Assume that $b < r - k_i + 1$. Notice that an occurrence of $(p[1, z_i])^{k_i+1} p[1, z'_i]$ starts at q_b. Thus, by Fact 1, $p[1, 2z_{i-1} + z'_{i-1}]$ cannot occur at q_b. Since $p[1, 2z_{i-1} + z'_{i-1}]$ is a prefix of the pattern and of $p[1, l_{i-1}]$, none of those two latter strings can occur at q_b, $1 \leqq b < r - k_i + 1$. If q_{r-ki+1} exists, it is the only candidate of the subsequence that is an occurrence of $p[1, l_{i-1}]$ and a potential occurrence of the pattern. Such position can be found in $O(r)$ time and only it survives for the next stage. Processing in a similar fashion the remaining subsequences of the list of candidates, we get a new list of candidates for the next stage that satisfies the invariant. Notice that the new list is smaller by at least a factor of $k_i \geqq 2$ and can be obtained in linear time.

Stage 1. We have the list $j_1, \cdots, j_r$ of potential occurrences of the pattern in the text. Moreover, it is also a list of all occurrences of $p[1, l_1]$. A position j in the list is an occurrence of the pattern in the text if and only if $j + vz_1$ is also in the list, for all v, $1 \leqq v < k$. This stage can be implemented in $O(r)$ time.

Since there can be at most $O(n)$ occurrences of $p[1, l_s]$ in the text and, at each stage, the list is processed in time linear in its size and the size is reduced by at least $\frac{1}{2}$, the total time of the algorithm is $O(n)$ and without any character comparisons. Thus we have the following lemma.

LEMMA 2. *Let $p[1, m]$, $m = kz + z'$, $k \geqq 1$, $z' < z$, be a pattern having period of size z and let $t[1, n]$ be a text. Let (z_s, z'_s, k_s) be the last term in the periodic decomposition of $p[1, m]$. For any algorithm $\mathcal{A}$ that finds all occurrences of $p[1, z_s + z'_s]$ in $t[1, n]$ in $f(n, z_s + z'_s)$ character comparisons there is an algorithm $\mathcal{A}'$ that finds all occurrences of*

$p[1, m]$ in $t[1, n]$ in the same number of character comparisons spending $O(n)$ additional time.

From now on, we consider only patterns of period size z and length $z+z'$, $0 \leqq z' < z$.

3. Colussi's algorithm. The algorithm that we present is a blend of the two best known and most efficient string matching algorithms devised so far: Knuth-Morris-Pratt [18] and Boyer-Moore [5]. It has been obtained by Colussi [7] using the correctness proof of programs as a tool to improve algorithms.

Given an alignment of the pattern $p[1, m]$ with text characters $t[i+1, i+m]$, KMP checks whether these two string are equal proceeding from left to right. BM performs the same check proceeding from right to left. Both algorithms then shift the pattern over the text by some precomputed amount and the check is repeated on the resulting new alignment. The KMP and BM approaches to string matching seem to be orthogonal and no efficient and mutually advantageous way of combining them had been known. The main difficulty in combining the two approaches is to find a partition of the set of m pattern positions such that there is an efficient strategy to shift the pattern over the text when a mismatch is found. Colussi [7] designed such partition and the corresponding strategy. The starting point of our presentation is KMP.

We need a few definitions. Given a string $w[1, m]$ and an index j, $1 \leqq j < m$, we say that $k\min(j)$ is defined and equal to d if d is the minimal integer such that $w[1, j]$ is d periodic and $w[j-d+1] \neq w[j+1]$. Thus, $k\min(j)$ is defined if a periodicity ends at j. If no such d exists we say that $k\min(j)$ is undefined. We refer to each position j of the pattern, $1 < j \leqq m$, having $k\min(j-1)$ defined as *nohole.* We refer to the remaining pattern positions as *holes.* Note that 1 is always a nohole since we have not defined $k\min(0)$.

Let the pattern be aligned with $t[i+1, i+m]$. Assume that KMP has found out that $p[1, j] = t[i+1, i+j]$ and that $p[j+1] \neq t[i+j+1]$ for some j, $0 \leqq j < m$. The pattern must be shifted over the text by some amount. There are two possibilities: shift past text position $i+j+1$ (where the mismatch occurred) and not shift past text position $i+j+1$ (thus having some overlap with the part that was matched). KMP chooses the first possibility when there is no integer g such that $p[1, j]$ is g periodic and $p[j+1] \neq p[j-g+1]$, i.e., $j+1$ is a hole, since it can be easily shown that there cannot be any occurrence of the pattern in the interval $[i+1, i+j+1]$. KMP chooses the second possibility when there is an integer g such that $p[1, j]$ is g periodic and $p[j+1] \neq p[j-g+1]$, i.e., $j+1$ is a nohole. Then, KMP shifts the pattern over the text by $k\min(j)$ positions since no occurrence of the pattern can be in the interval $[i+1, i+k\min(j)]$. Moreover, there may still be an occurrence of the pattern at $i+k\min(j)+1$, since $p[1, j-k\min(j)] = p[k\min(j)+1, j] = t[i+1+k\min(j), i+j]$ and a pattern character different from $p[j+1]$ will be aligned with $t[i+j+1]$ (recall the definition of $k\min$).

Colussi observed that if $j+1$ is a nohole and $p[j+1] \neq t[i+j+1]$, we need not know that $p[1, j] = t[i+1, i+j]$ to deduce that the pattern must be shifted by $k\min(j)$ positions over the text: Let the pattern be aligned with $t[i+1, i+m]$ and assume that all the noholes in $p[1, j]$ match the corresponding text positions in $t[i+1, i+j]$ and that $p[j+1] \neq t[i+j+1]$, $j+1$ a nohole. Then, it can be shown that there can be no occurrence of the pattern in the text in the interval $[i+1, i+k\min(j)]$. Moreover, we can still use some of the knowledge acquired about $t[i+1, i+j]$ after the shift by $k\min(j)$ takes place. Indeed, all the noholes in $p[1, j-k\min(j)]$ will match the corresponding text positions in $t[i+k\min(j)+1, i+j]$. Therefore, the string matching can be restarted by comparing the first nohole beyond $j-k\min(j)$ with the corresponding text position beyond $i+j$. Let $h_1 < h_2 < \cdots < h_{nd}$, $h_{nd} \leqq m-1$, be the pattern positions such that $k\min$ is defined and let $\text{first}(x)$ denote the least integer y such that

$x \leqq h_y$. The following lemma is a precise statement of Colussi's observation [7] (see Fig. 1).

LEMMA 3. *Let the pattern be aligned with* $t[i+1, i+m]$. *Assume that* $p[h_s+1] = t[i+h_s+1]$ *for* $1 \leqq s < r \leqq nd$, *and* $p[h_r+1] \neq t[i+h_r+1]$. *Let* $i_{\text{new}} = i + k\min(h_r)$. *Then, there is no occurrence of the pattern in the interval* $[i+1, i_{\text{new}}]$ *and the pattern can be shifted over the text by* $k\min(h_r)$ *positions. Moreover,* $p[h_s+1] = t[i_{\text{new}}+h_s+1]$ *for* $1 \leqq s < \text{first}(h_r - k\min(h_r))$, *and the algorithm can be restarted by testing whether* $p[h_{\text{first}(h_r - k\min(h_r))}+1] = t[i_{\text{new}} + h_{\text{first}(h_r - k\min(h_r))}+1]$.

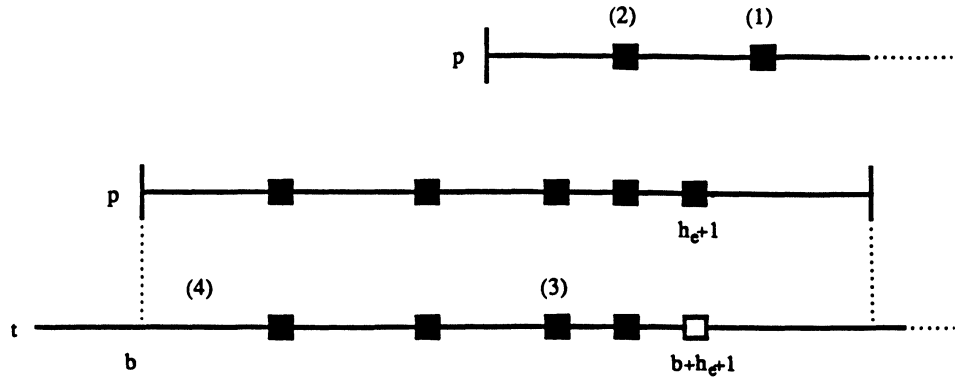

FIG. 1. *A mismatch with a selected position (a nohole) in iteration b. The shift is by the period* $k\min(h_e)$ *of the interrupted periodicity. The algorithm resumes with the first nohole aligned with a text position not before the position of the mismatch* (1), *since smaller noholes (e.g.,* (2)) *must match. The comparison of* (3) *is forgotten, because after the shift it is aligned with a hole. On the other hand, if* (4) *was not compared before iteration b, it will not be compared at all.*

An operative conclusion that can be drawn from Lemma 3 is that when checking (from left to right) if $p[1, m] = t[i+1, i+m]$, we can compare those pattern positions $j+1$ having $k\min(j)$ defined and shift by $k\min(j)$ in case of a mismatch. Now, assume we know that all the noholes of the pattern match the corresponding text characters, i.e., $p[h_s+1] = t[i+h_s+1]$, $1 \leqq s \leqq nd$. We must decide in which order to compare the holes of the pattern with the corresponding text positions and what to do in the case that a mismatch is found. If we find out that $p[\hat{\jmath}+1] \neq t[i+\hat{\jmath}+1]$, $\hat{\jmath}+1$ a hole, we can shift past text position $i+\hat{\jmath}+1$. (Recall that a shift with overlap must have $k\min(\hat{\jmath})$ defined.) In order to guarantee the longest shifts while allowing the algorithm to retain more of the knowledge it acquired about $t[i+1, i+m]$, all the holes of the pattern are processed in decreasing order, that is, from right to left. Another advantage for this order is that by filling up the holes from right to left we completely match a suffix of the pattern.

Lemma 4 describes more precisely such processing (see Fig. 2). Let $h_{nd+1} > h_{nd+2} > \cdots > h_{m-1} > h_m = 0$ be all pattern positions such that $k\min$ is undefined, $h_{nd+1} \leqq m-1$. Given a position j of the pattern, let $r\min(j)$ be the minimal period of $p[1, m]$ greater than j.

LEMMA 4. *Let the pattern be aligned with* $t[i+1, i+m]$. *Assume that* $p[h_s+1] = t[i+h_s+1]$, $1 \leqq s < r$, *and that* $p[h_r+1] \neq t[i+h_r+1]$, $r > nd$. *Let* $i_{\text{new}} = i + r\min(h_r)$. *Then, there is no occurrence of the pattern in the interval* $[i+1, i_{\text{new}}]$. *Moreover* $p[1, m - r\min(h_r)] = t[i_{\text{new}}+1, i+m]$ *and the algorithm can be restarted by testing whether* $p[h_{\text{first}(m - r\min(h_r))}+1] = t[i_{\text{new}} + h_{\text{first}(m - r\min(h_r))}+1]$.

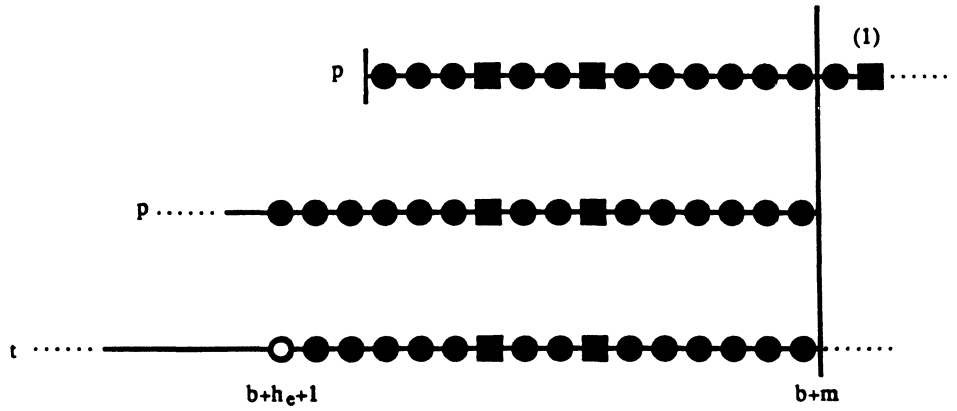

FIG. 2. *A mismatch with a hole. A complete suffix of the pattern is matched. The shift is by the smallest period of the pattern past the mismatch. (In the first alignment a suffix of the pattern is shown and after the shift a prefix is shown.) The algorithm resumes as in Fig. 1. After the shift the algorithm will only compare text positions greater than* $t_{\text{last}} = b + m$.

The operative conclusion that we can draw from Lemma 4 is that when we have completed a left to right pass and start a right to left pass, we are guaranteed that there is no need to consider text positions $t[i+r\min(h_r)+1, i+m]$ every again, since a prefix of the pattern matches $t[i+r\min(h_r)+1, i+m]$ and each future shift affects only the length of the match, since we always shift by a period of a prefix at least as large.

The subsequences $h_1 < h_2 < \cdots < h_{nd}$ and $h_{nd+1} > h_{nd+2} > \cdots > h_m$ provide a partition of the integers $\{0, \cdots, m-1\}$ which induces a partition on the pattern positions $\{1, \cdots, m\}$. The way to put these two sequences together to perform string matching is given by Lemmas 3 and 4. Indeed, we check whether $p[1, m] = t[i+1, i+m]$, using the sequence $h_1+1, h_2+1, \cdots, h_{nd}+1, h_{nd+1}+1, \cdots, h_m+1$ in the order given. This sequence can be computed in $O(m)$ time and $2m-1$ character comparisons using the preprocessing algorithm of KMP [18].

We define two tables disp and start having $m+1$ entires each as follows: $\text{disp}[i] = k\min(h_i)$ and $\text{start}[i] = \text{first}(h_i - k\min(h_i))$, $1 \leqq i \leqq nd$; $\text{disp}[i] = r\min(h_i)$ and $\text{start}[i] = \text{first}(m - r\min(h_i))$, $nd < i \leqq m$; $\text{disp}[m+1] = z$ and $\text{start}[m+1] = \text{first}(m-z)$, where z is the period of the pattern. The last entry in these tables is used to restart the algorithm correctly when an occurrence of the pattern in the text is found. These tables are computed in the preprocessing stage.

In the pseudocode given below, variable *p*start is equal to the index from which the algorithm starts examining the sequence $h_1, \cdots, h_m$ at the beginning of each iteration. Variable *t*last denotes an index such that text characters in $t[1, t\text{last}]$ are not examined anymore by the algorithm. Variable b denotes an index such that the algorithm has already found all occurrences of the pattern in $t[1, b]$ while it has still to find the ones in $t[b+1, n]$. Variable e points to the element in the sequence $h_1, \cdots, h_m$ that is currently being used for comparison.

```
    Algorithm SM
begin
  b ← 0; pstart ← 1; tlast ← 0
  repeat
    e ← pstart;
```

```
    while e ≦ m and tlast < b + h_e + 1 and p[h_e + 1] = text[b + h_e + 1] do
  e ← e + 1;
      if e = m + 1 or tlast ≧ b + h_e + 1 then occurrence found
      if e > nd then tlast ← b + m;
      b ← b + disp[e];
      pstart ← start[e];
  until b > n − m;
end
```

We say that an integer i is an iteration if Algorithm SM sets variable $b = i$ during its execution. An iteration i corresponds to an alignment of the pattern with $t[i+1, i+m]$.

We can observe a potentially serious limitation to a good performance of SM: After the pattern is shifted by some amount, there are situations in which the algorithm forgets that some of the text positions match the corresponding pattern positions and may compare them again. Consider a nohole h_s+1 of the pattern, for some fixed $s \leqq nd$. It can happen that while $k\min(h_s)$ is defined, there may exist a nohole $h_{s'}+1$, $s < s' \leqq nd$, such that $k\min(h_s - k\min(h_{s'}))$ is undefined. In other words, noholes are unstable under shifts corresponding to interrupted periodicities. The effect of this instability is that a text position matching a nohole in a given iteration may be aligned with a hole in the next iteration and be considered again if the algorithm starts matching pattern positions from right to left. Figure 1 gives an example of matched text positions that may be compared again.

Obliviousness in a string matching algorithm is not new: BM forgets part of what it knows about the text after a shift. This phenomenon has been extensively studied. However, none of the techniques used by KMP [18], Guibas and Odlyzko [13], Galil [10], Apostolico and Giancarlo [3], and Cole [6] to account for the obliviousness of BM could be used in our case. Intuitively, the obliviousness of BM is macroscopic whereas that of SM is microscopic. Indeed, BM forgets that contiguous chunks of text match some pattern substrings, whereas SM forgets that some text characters, not necessarily contiguous and irregularly distributed over the text, match some pattern characters.

Another property of SM is that it skips over some text positions while matching noholes. Then the pattern is shifted over the text. The text positions skipped over and to the left of the current alignment may be left untested, i.e., they are never compared with any pattern position. Figures 1 and 2 give a qualitative description of this phenomenon. Again, we remark that the text positions left untested are irregularly distributed over the text.

In order to obtain a tight analysis of SM, we would like to use all the untested text positions to pay for the mismatches and some of the repeated comparisons resulting from the obliviousness of SM. Achieving this goal requires tight upper bounds on the number of comparisons forgotten after a shift and a flexible and precise charging scheme that allows to amortize character comparisons against untested text positions. Such tasks are complicated by the arbitrariness and irregularity of the distributions (over the text) of forgotten comparisons and untested positions.

4. Some properties of strings. In order to analyze Algorithm SM and its improvement, we need to derive some properties of $k\min$ and to study some new combinatorial properties of strings. Throughout this section, let l be the maximal integer such that a string x has $x[1] = \cdots = x[l]$ and $x[l] \neq x[l+1]$.

FACT 2. *Let $x[1, h]$ be a k periodic prefix of x and assume that $h > k+l$. There can be no other period k' of $x[1, h]$ such that $|k-k'| \leqq l$.*

Proof. We give a proof only for the interval $[k+1, k+l]$. The proof for the other interval is similar. Assume that $x[1, h]$ is k' periodic, $k+1 \leqq k' \leqq k+l < h$. This implies that $x[k+l+1] = x[k+l+1-k'] = x[1]$. But, since $x[1, h]$ is also k periodic, $x[k+l+1] = x[l+1]$. Therefore, $x[1] = x[l+1]$, a contradiction to the definition of l. □

FACT 3. *If $j \geqq l+1$ and $k\min(j)$ is defined, then $k\min(j) > l$.*

Proof. Assume that, for $j \geqq l+1$, $k\min(j) \leqq l$. Since $k\min(j) \leqq l$ is a period of $x[1, j]$ we must have that $x[1, j] \in x[1]^*$, which is impossible since $x[l+1] \neq x[1]$ and $j \geqq l+1$. □

In what follows we derive relationships between the holes of a string and the periodicities of its substrings.

FACT 4. *Let $x[1, h]$ be a k periodic prefix of x. If $j-k$ is a nohole, $k < j \leqq h$, then j is a nohole.*

Proof. We assume that $k\min(j-1-k)$ is defined and show that $k\min(j-1)$ is defined. By definition of $k$min, there exists an integer r such that $x[1, j-k-r-1] = x[r+1, j-k-1]$ and $x[j-k-r] \neq x[j-k]$. Using the k periodicity of $x[1, h]$, we obtain $x[1, j-k-r-1] = x[r+k+1, j-1]$ and $x[j-k-r] \neq x[j]$. Thus, $k\min(j-1)$ is defined. □

Given a k periodic prefix $x[1, h]$ of x, a hole $j-k$ may not carry over to j. The next fact establishes a relationship between "missing holes" and periodicity.

FACT 5. *Let $x[1, h]$ be a k periodic prefix of x. Assume that $j-k$ is a hole while j is not, for some j, $k < j \leqq h$. Then $i = k\min(j-1)$ satisfies $l < i < k$.*

Proof. Assume that $i \geqq k$ for the given j satisfying the hypothesis. If $i = k$, we have that $x[1, j-1-k] = x[k+1, j-1]$ and that $x[j] \neq x[j-k]$. This latter inequality contradicts the k periodicity of $x[1, h]$. If $i > k$, we note that because $i \leqq j-1, j-k-1 > 0$. We show that the k periodicity of $x[1, h]$ and the fact that $k\min(j-1) = i$ must imply that $k\min(j-1-k)$ is defined. This will contradict the assumption that $j-k$ is a hole: From the definition of $k$min, $x[1, j-i-1] = x[i+1, j-1]$ and $x[j-i] \neq x[j]$. Combining these conditions with the facts that $i > k$ and $x[1, h]$ is k-periodic, we obtain that $x[i+1, j-1] = x[i+1-k, j-k-1] = x[1, j-i-1]$ and $x[j-k] \neq x[j-i]$, which imply that $k\min(j-k-1) \leqq i-k$, a contradiction. Thus, $i < k$. There is no $j \leqq l$ satisfying the hypothesis of the fact. Now, for $j \geqq l+1$, $i > l$ by Fact 3. □

Let v be a position of x such that $k\min(v-1)$ is defined. We say that v is *reachable* from position $u < v$ if there exists an increasing sequence of integers $u_1, u_2, \cdots, u_s$ such that $u_1 = u$, $u_s = v$, $k\min(u_1 - 1)$ is undefined and $k\min(u_{i+1} - 1) = u_i - 1$, for $1 \leqq i < s$. Notice that $u > 1$ and is the only hole in the sequence. We denote the sequence of integers $u = u_1, u_2, \cdots, u_s = v$ as $(u \Rightarrow v)$. We say that $(u_1 \Rightarrow v_1)$ and $(u_2 \Rightarrow v_2)$ are disjoint if they have no integers in common.

LEMMA 5. *Let $x[1, h]$ be a k periodic prefix of x. Assume that there is a position j, $k < j \leqq h$, such that $j-k$ is a hole whereas j is not. Then, there is a position q, $l+1 < q \leqq k$, from which j is reachable. Moreover, if two distinct positions j_1 and j_2 satisfy the assumptions of the lemma then $(q_1 \Rightarrow j_1)$ and $(q_2 \Rightarrow j_2)$ are disjoint.*

Proof. There is no $j \leqq l+1$ satisfying the hypothesis, since by definition of l and of $k$min, no position j in $[1, l]$ can have $k\min(j-1)$ defined and $x[1, l+1]$ is not k periodic, for any $k < l+1$. Therefore, assume that $l+1 < j$.

Since j is nohole, $i = k\min(j-1)$ is defined. Since $j-k$ is a hole by hypothesis, we can apply Fact 5 to prefix $x[1, h]$ to obtain $l+1 < i+1 \leqq k$. If $k\min(i)$ is undefined, j is reachable from $i+1$, $l+1 < i+1 \leqq k$. Otherwise, we notice that $x[1, j-1]$ is i periodic and that $i+1$ is a nohole in this string whereas position 1 is a hole. Thus, we can now

apply Fact 5 to $x[1, j-1]$ with $j' = i+1$, $k' = i$, and $i' = k\min(i)$. Iterating the above reasoning, we get a decreasing sequence of indices that must end in a hole q, $l+1 < q \leqq i+1 \leqq k$.

Given q_1 and q_2 from which j_1 and j_2 are reachable, respectively, we now show that $(q_1 \Rightarrow j_1)$ and $(q_2 \Rightarrow j_2)$ are disjoint. The proof is by contradiction. Assume that such two sequences are not disjoint. We first notice that j_1 cannot be in $(q_2 \Rightarrow j_2)$, since $k\min(j_2 - 1) < k < j_1$. Similarly, j_2 cannot be in $(q_1 \Rightarrow j_1)$. Since the two sequences are not disjoint (by assumption), there must exist two indices f and g such that $u_f \in (q_1 \Rightarrow j_1)$, $u'_g \in (q_2 \Rightarrow j_2)$, $u_f = u'_g$ and $u_{f+1} \neq u'_{g+1}$. Without loss of generality, let $u'_{g+1} < u_{f+1}$. From the definition of reachability, we know that $x[1, u_{f+1} - 1]$ is $u_f - 1$ periodic, with $x[u_{f+1}] \neq x[u_{f+1} - u_f + 1]$. Analogously, we know that $x[1, u'_{g+1} - 1]$ is $u_f - 1$ periodic, with $x[u'_{g+1}] \neq x[u'_{g+1} - u_f + 1]$. This latter inequality contradicts the $u_f - 1$ periodicity of $x[1, u_{f+1} - 1]$.

Thus, $(q_1 \Rightarrow j_1)$ and $(q_2 \Rightarrow j_2)$ are disjoint and $q_1 \neq q_2$. □

LEMMA 6. *Let $x[1, m]$ be a string, z be its period, $m = z + z'$, $0 \leqq z' < z$. Let l be the maximal integer such that x has a prefix equal to $x[1]^l$ and let $x[1, h]$ be a k periodic prefix of x, $k < h$. Choose an integer $b \leqq h$. Assume that $j_i > \max(k, b)$, $1 \leqq i \leqq s$, are all the noholes in $x[1, h]$ such that $j_i - k$ is a hole and j_i is reachable from a hole in $x[1, \min(k, b)]$.*

If $b \leqq k$,

$$s \leqq \max\left(0, \left\lfloor \frac{b-1}{2} \right\rfloor\right). \tag{1}$$

Otherwise,

$$s \leqq \max\left(0, \left\lfloor \frac{k-2}{2} \right\rfloor\right). \tag{2}$$

Proof. For each j_i satisfying the hypothesis of the lemma, let $q_i \leqq \min(k, b)$ denote a hole from which j_i is reachable. By Lemma 5, each q_i is distinct and greater than $l+1$. From the inequality $q_i > l+1$, we can immediately deduce that $s = 0$ when $\min(k, b) \leqq l+1$. In what follows, we consider the case $\min(k, b) > l+1$.

For each $(q_i \Rightarrow j_i)$, let u_i be the minimal integer in that sequence greater than b. By Lemma 5, all $(q_i \Rightarrow j_i)$ are disjoint. Thus all u_i's are distinct and the correspondence between q_i's and u_i's is one to one. Let $d_i = k\min(u_i - 1) + 1$. Note that all d_i's are distinct ($d_i \in (q_i \Rightarrow j_i)$). We obtain the bounds by bounding the number of d_i's. In a one-to-one fashion, we associate with the d_i's a set of integers r_i in the interval I containing the d_i's. The two sets are disjoint and therefore their size is bounded by half the size of their union or half the number of the elements of I that are "used."

We must have $l+1 < d_i \leqq \min(k, b)$. Indeed, $l+1 \leqq q_i - 1 \leqq k\min(u_i - 1) = d_i - 1 \leqq k\min(j_i - 1)$, and $k\min(j_i - 1) < k$ by Fact 5. Using the definition of reachability and the fact that u_i is the least integer in $(q_i \Rightarrow j_i)$ larger than b, $k\min(u_i - 1) = d_i - 1 < b$.

We divide the d_i's in two sets B and C defined as follows: $B = \{d_i : d_i \leqq \min(b, k) - l\}$; $C = \{d_i : b - l < d_i \leqq \min(b, k)\}$.

It is obvious that B and C partition the d_i's when $b \leqq k$. We prove that we have a partition also in the case that $k < b$. The proof is by contradiction. Assume that there exists a d_g such that $d_g \notin B \cup C$. Then, $k - l < d_g \leqq \min(k, b - l)$. Since $x[1, h]$ is k periodic, $x[1, u_g - 1]$ is $d_g - 1$ periodic, $k < b$ and $u_g > b$, $x[1, b]$ is both k periodic and $d_g - 1$ periodic. Since $b > d_g + l - 1$, we apply Fact 2 to $x[1, b]$ (with $k = d_g - 1$), to obtain that no period of $x[1, b]$ is in $[d_g, d_g + l - 1]$. But k is in that interval. A contradiction.

We associate each $d \in B$ with $d+l$. Obviously, the correspondence is one to one. Moreover, $d+l \neq d_i$ for $1 \leqq i \leqq s$, since by the $(d_i - 1)$ and $d-1$ periodicity of $x[1, b]$ we have $x[l+1] = x[d+l] \neq x[1] = x[d_i]$. We also have $l+1 < d+l \leqq \min(k, b)$. We can now obtain the claimed bounds by assigning a suitable number of positions in $[1, \min(k, b)]$ to elements in C that nave not been assigned to elements of B and that are different from any d_i, $1 \leqq i \leqq s$. We distinguish two cases.

If $k < b$ ($k \geqq b$, respectively), then $|C| \leqq l-1$ ($|C| \leqq l$, respectively). Noting that positions $[1, l+1]$ are different from any d_i and have not been assigned to elements in B, we can assign to elements in C a corresponding number of these positions to obtain bound (2) ((1), respectively). □

5. Analysis of Algorithm SM. We need a few definitions. An iteration r *matches* a suffix of the pattern if $t[r+h_i+1] = p[h_i+1]$, $1 \leqq i \leqq nd$, and, at least, $t[r+m] = p[m]$. Two iterations r and u, $r < u$, have a *suffix-prefix* overlap if r matches a suffix of the pattern and $u \leqq r+m-1$. For any iteration u, let $t\text{last}(u)$ be the value of $t$last in algorithm SM when u starts.

LEMMA 7. *$u < t\text{last}(u)$ if and only if u has a suffix-prefix overlap with an iteration $u' < u$.*

Proof. Assume $u < t\text{last}(u)$. Let $u' < u$ be the last iteration that advanced t last. Note that $0 = t$ last (0), so u cannot be the first iteration. We have t last $(u) = u' + m$ and since $u < t$ last (u), $u \leqq u' + m - 1$. Therefore u has an overlap with u'. Moreover, $e > nd$ at the end of iteration u', otherwise t last could not be updated during that iteration. Since $\max(h_{nd}, h_{nd+1}) = m-1$, and $e > nd$ at the end of iteration u', we know that $p[m]$ has been compared with $t[u'+m]$ during that iteration and that u' cannot use $k$min as a shift function since all the noholes match. If u' did not match any suffix of the pattern, we must have $p[m] \neq t[u'+m]$. But, in that case, there would be a shift by m of the pattern over the text (recall the definition of $r$min and the fact that u' cannot use $k$min for shifts). Therefore, the iteration succeeding u' is $\hat{u} = u' + m > u$, a contradiction. Thus $p[m] = t[u'+m]$ and u has a suffix-prefix overlap with u'.

Assume that u has a suffix-prefix overlap with u', $u' < u$. We first show that u cannot have a suffix-prefix overlap with any other iteration $\hat{u} < u$. Assume the contrary. Thus, $u - \min(\hat{u}, u') \leqq m-1$. Since both $\hat{u}$ and u' matched a suffix of the pattern, there was a shift by at least z of the pattern over the text at the end of both iterations. Thus $\min(\hat{u}, u') + z \leqq \max(\hat{u}, u')$ and $\max(\hat{u}, u') + z \leqq u$ implying that $\min(\hat{u}, u') + 2z \leqq u$. A contradiction since $u - \min(\hat{u}, u') \leqq m - 1 < 2z$ $(m = z + z', z' < z)$.

Since u has a suffix-prefix overlap with u', u' must have matched a suffix of the pattern and set t last $= u' + m$. We claim that no iteration $\hat{u}$, $u' < \hat{u} < u$ can advance t last. Assume $\hat{u}$ advances t last and hence $e > nd$ at the end of iteration $\hat{u}$. Since $\max(h_{nd}, h_{nd+1}) = m-1$, and $e > nd$ at the end of iteration u', we obtain that $p[m]$ has been compared with $t[\hat{u}+m]$. If $p[m] \neq t[\hat{u}+m]$, the pattern would be shifted by at least m implying $u - u' \geqq m$. A contradiction, since u and u' overlap. If $p[m] = t[\hat{u}+m]$, then $\hat{u}$ matches a suffix of the pattern and u has a suffix-prefix overlap with $\hat{u}$ (recall $u' < \hat{u} < u$ and $u - u' \leqq m-1$). A contradiction to the fact that u has suffix-prefix overlap with u'. Therefore, when u starts t last $(u) = t$ last $= u' + m$ and since $u - u' \leqq m-1$, $u < t$ last (u). □

The iterations of Algorithm SM are divided into sequences of consecutive iterations. There are two kinds of sequences: *light* and *heavy*. Informally, a heavy sequence is a maximal sequence of iterations such that the first matches a suffix of the pattern and every other iteration has a suffix-prefix overlap with some iteration in the sequence preceding it. A heavy sequence can be followed by a heavy or light sequence. A

nonempty sequence of iterations between two consecutive heavy sequences is a light sequence. As will be explained later, light sequences are easier to analyze and pay for themselves.

Formally, a sequence $s_1, s_2, \cdots, s_y$ of consecutive iterations is *light* if

(1) s_1 does not match a suffix of the pattern and it is either the first iteration of the string matching algorithm or it is preceded by an iteration that ends a heavy sequence.

(2) $s_i \geqq t$ last (s_i), $1 < i \leqq y$ and either s_y is the last iteration of the algorithm or u matches a suffix of the pattern, where u is the iteration succeeding s_y.

A sequence $u_1, u_2, \cdots, u_g$ of consecutive iterations is *heavy* if

(1) u_1 matches a suffix of the pattern and it is either the first iteration of the string matching algorithm or it is preceded by an iteration that ends a sequence, either light or heavy. Notice that in both cases u_1 advances t last to $u_1 + m$ since it matches a suffix of the pattern.

(2) $u_i < t$ last (u_i), $1 < i \leqq g$ and either u_g is the last iteration of the algorithm or $s \geqq t$ last (s), where s is the iteration succeeding u_g.

By Lemma 7, the preceding definition implies that s has no suffix-prefix overlap with any iteration preceding it. Moreover, it also implies that u_i, $1 < i \leqq g$, has a suffix-prefix overlap with some u_j, $1 \leqq j < i$, as we show next. Our claim follows from Lemma 7 if u_1 is the first iteration of the algorithm. Assume u_1 is not the first iteration of the algorithm. By Lemma 7, u_i has a suffix-prefix overlap with some $u' < u_i$, $u_i - u' \leqq m-1$. Assume also that $u' < u_1$. Since both u' and u_1 match a suffix of the pattern, the pattern is shifted over the text by at least z at the end of both iterations. Therefore, $u' + z \leqq u_1$ and $u_1 + z \leqq u_i$ implying $u_i - u' \geqq 2z > m = z + z'$, $z' < z$, a contradiction, since we assume that $u_i - u' \leqq m-1$. Therefore $u' = u_j$ for some j, $1 \leqq j < i$.

In order to ensure that each iteration of the algorithm has a successor, we add a dummy iteration that follows the last iteration of the algorithm. Such dummy iteration does nothing, is not part of any light or heavy sequence, and is not regarded as the last iteration of the algorithm. We set it equal to n.

We will use a charging mechanism in which each position in the text may pay for at most one comparison. Given an iteration j we say that a position $d > j$ of the text is *free* if $t[d]$ has never been considered by the string matching algorithm or it has been compared with one or more characters of the pattern but, for each comparison, a different position d', $d' \leqq j$, of the text paid for it. A position $d \leqq j$ of the text is *busy* if it paid for exactly one comparison performed by the algorithm.

Given an iteration s, let start (s) be the value of e in algorithm SM at the beginning of iteration s. Given an iteration v starting either a light or a heavy sequence, let $c(v)$ be the number of noholes in the pattern between zero and $h_{\text{start}(v)}$. Notice that $c(v) = \text{start}(v) - 1$. For notational convenience, we define $c(v) = 0$ for the dummy iteration. In our account of the number of character comparisons performed by algorithm SM we maintain the following invariant for any iteration s starting a sequence.

INVARIANT 1. Start $(s) \leqq nd + 1$ *and all positions of the text in* $t[s+1, s+h_{\text{start}(s)}]$ *aligned with holes of the pattern are free whereas those remaining match the corresponding positions of the pattern. All positions in* $t[s+h_{\text{start}(s)}+1, n]$ *are free and, in the worst case, all positions in* $t[1, s]$ *are busy.*

5.1. The analysis of light sequences. Let $s_1, s_2, \cdots, s_y$ be a light sequence and let u_1 be either the iteration that starts the following heavy sequence or the dummy iteration. We estimate the number of comparisons performed by the algorithm during iteration s_1 and we prove that, if s_2 is not the dummy iteration, it satisfies invariant 1

when it starts. Then, we inductively extend the same analysis to the remaining iterations in the sequence. We will charge each iteration in the sequence. This charge is related but not equal to the number of character comparisons done by the algorithm during that iteration. If iteration s makes r comparisons, we will charge it $r+c(s)-c(s')$ comparisons, where s' is the next iteration. Thus s pays for the $c(s)$ comparisons done before s, but does not pay for the $c(s')$ comparisons that s' is going to inherit. We show that s can pay for the number of comparisons being charged to it using only free text positions. The free text positions used to pay become busy at the end of iteration s. (This is what we meant by saying that light sequences pay for themselves. In fact, each item of a light sequence pays for itself.)

We assume that s_1 satisfies invariant 1 when it starts. Thus, $t[s_1+h_1+1]=p[h_1+1], \cdots, t[s_1+h_{\text{start}(s_1)-1}+1]=p[h_{\text{start}(s_1)-1}+1]$ at the beginning of s_1. This amounts to $c(s_1)$ matches that Algorithm SM has identified during previous iterations. None of these text characters is going to be compared to any pattern character during iteration s_1.

During iteration s_1, the algorithm performs some $r \geqq 1$ comparisons. By the definition of a light sequence, $r-1$ of these comparisons are matches and one is a mismatch. Thus, at the end of iteration s_1, the algorithm has the additional knowledge that $t[s_1+h_{\text{start}(s_1)}+1]=p[h_{\text{start}(s_1)}+1], \cdots, t[s_1+h_{\text{start}(s_1)+r-2}+1]=p[h_{\text{start}(s_1)+r-2}+1]$, $t[s_1+h_{\text{start}(s_1)+r-1}+1] \neq p[h_{\text{start}(s_1)+r-1}+1]$. Let $N=\{s_1+h_i+1: 1 \leqq i \leqq \text{start}(s_1)+r-1\}$. Notice that the algorithm performs $r=|N|-c(s_1)$ comparisons during iteration s_1. We bound this number by bounding $|N|$.

If s_2 is the dummy iteration, $|N| \leqq m \leqq s_2-s_1+c(s_2)$, since $s_2=n$, $s_1 \leqq n-m$ and $c(s_2)=0$. Therefore, the algorithm performs at most $s_2-s_1+c(s_2)-c(s_1)$ character comparisons during iteration s_1 paying for s_2-s_1 comparisons.

Assume s_2 is not the dummy iteration, i.e., $s_2 \leqq n-m$. We consider two mutually exclusive cases: start $(s_1)+r-1 \leqq nd$ and start $(s_1)+r-1>nd$. For brevity, let $\hat{h}$ denote $h_{\text{start}(s_1)+r-1}$.

Assume that start $(s_1)+r-1>nd$. By definition of a light sequence, this case can happen only if $h_{nd}<m-1$. Otherwise s_1 matches a suffix of the pattern. Moreover, we must have start $(s_1)+r-1=nd+1$, $h_{nd+1}=m-1$ and $t[s_1+m] \neq p[m]=p[h_{nd+1}+1]$. The algorithm shifts the pattern over the text by $r\min(m-1)=m>m-1$ positions. Thus, $s_2=s_1+m$. Obviously, $|N| \leqq m=s_2-s_1$ and $c(s_2)=0$. Therefore, the number of character comparisons performed by the algorithm during iteration s_1 is at most $s_2-s_1+c(s_2)-c(s_1)$. Moreover, since none of the text positions in $[s_1+1, s_2]$ is busy (by invariant 1), they can pay for $c(s_1)+r-c(s_2)=|N|$ comparisons and become busy at the end of iteration s_1. Obviously, s_2 satisfies invariant 1 when it starts.

Let us consider now the case start $(s_1)+r-1 \leqq nd$. Since $k\min(\hat{h})$ is defined and $t[s_1+\hat{h}+1] \neq p[\hat{h}+1]$, we have that $s_2=s_1+k\min(\hat{h})$. We divide the set N into four disjoint subsets N_{nomatch}, N_{busy}, N_{holes}, N_{noholes} defined as follows.

- $N_{\text{nomatch}}=\{s_1+\hat{h}+1\}$,
- $N_{\text{busy}}=\{i \in N: i \leqq s_2\}$.
- N_{holes} is the set of i's in $N-\{s_1+\hat{h}+1\}$ such that $i>s_2$ and each i is aligned with a hole of the pattern at the beginning of iteration s_2.
- N_{nohole} is the set of i's in $N-\{s_1+\hat{h}+1\}$ such that $i>s_2$ and each i is aligned with a nohole of the pattern at the beginning of iteration s_2.

At the beginning of iteration s_2, Algorithm SM sets variable $e=\text{start}(s_2)$ to first $(\hat{h}-k\min(\hat{h}))$. We claim that

$$|N_{\text{noholes}}|=\text{start}(s_2)-1=c(s_2). \tag{3}$$

Consider j, one of the first $c(s_2)$ noholes in the pattern not compared in iteration s_2. By Fact 4, $j+k\min(\hat{h})$ is a nohole in iteration s_1. Thus, in iteration s_1, j is aligned with an element of N_{noholes}. Conversely, each element of N_{noholes} is aligned with a nohole of the pattern in both iterations and the nohole in iteration s_2 is one of the first $c(s_2)$ of the pattern (start (s_2) = first $(\hat{h}-k\min(\hat{h}))$). Thus, (3) holds.

We claim that

$$|N_{\text{holes}}| \leqq s_2 - s_1 - 1 - |N_{\text{busy}}|. \tag{4}$$

Indeed, consider $p[1, \hat{h}]$ and let J denote the set of positions j in this string such that $k\min(\hat{h}) < \hat{h}$, j is nohole and $j-k\min(\hat{h})$ is a hole. At the end of iteration s_1, each $j \in J$ has a match with exactly one text position in N_{holes} and vice versa. Thus, $|N_{\text{holes}}| = |J|$. If $k\min(\hat{h}) \leqq l$, $\hat{h} \leqq l$ by Fact 3 and $|J| = |N_{\text{busy}}| = 0$ since there cannot be any noholes in $p[1, l]$. Thus (4) obviously holds. Assume now that $k\min(\hat{h}) > l$. From Lemma 5 (with $k = k\min(\hat{h})$), we know that each $j \in J$ is reachable from a distinct hole q in $p[2, k\min(\hat{h})]$. The number of holes in $p[1, k\min(\hat{h})]$ is $s_2 - s_1 - |N_{\text{busy}}| = k\min(\hat{h}) - |N_{\text{busy}}|$, since by definition, the elements of N_{busy} are exactly those aligned with noholes in $p[1, k\min(\hat{h})]$. Thus, $|N_{\text{holes}}| = |J| \leqq s_2 - s_1 - 1 - |N_{\text{busy}}|$ and (4) follows. We notice that the holes in $p[2, k\min(\hat{h})]$ were aligned with free positions at the beginning of iteration s_1, by invariant 1. Such text positions were ignored during iteration s_1 because s_1 tried to match noholes first and start $(s_1) + r - 1 \leqq nd$. Thus, there are enough free text positions from s_1+2 to s_2 to pay for the comparisons that involved text positions in N_{holes}. That is, at the end of iteration s_1 all text positions in N_{holes} are free, provided that the text positions aligned with holes in $p[2, k\min(\hat{h})]$ during iteration s_1 are declared busy.

We also observe that position s_1+1 is free at the beginning of iteration s_1 since it is aligned with $p[1]$ (a hole) and Invariant 1 holds. Therefore, it can pay for the mismatch. Thus, text position $s_1+\hat{h}+1$ is still free provided that s_1+1 becomes busy at the end of iteration s_1. All text positions in N_{busy} are declared busy at the end of iteration s_1.

We remark that all text positions declared busy at the end of iteration s_1 were either free at the beginning of this iteration or they were part of the $c(s_1)$ matches that s_1 inherited from previous iterations. Moreover, they are not going to be considered again by Algorithm SM after iteration s_1.

Using the identities (3), $|N_{\text{nomatch}}| = 1$, and inequality (4), it follows that $|N| \leqq s_2 - s_1 + c(s_2)$. Since $c(s_1)$ of those comparisons were performed during previous iterations, the algorithm performs at most $s_2 - s_1 + c(s_2) - c(s_1)$ comparisons during iteration s_1. Moreover, $c(s_2)$ of them are left unpaid by s_1.

Iteration s_2 satisfies Invariant 1 when it starts. Indeed, all text positions in $t[s_2+1, s_2+h_{\text{start}(s_2)}]$ aligned with holes of the pattern are free. Such text positions are a subset of N_{holes}. All text positions in $t[s_2+1, s_2+h_{\text{start}(s_2)}]$ aligned with noholes of the pattern match the corresponding characters. These positions are in N_{noholes}. All text positions in $t[s_2+h_{\text{start}(s_2)}+1, n]$ are free because either they were free at the beginning of s_1 and the algorithm did not consider them during s_1, or free text positions in $t[s_1+1, s_2]$ paid (by becoming busy) for their comparisons.

The analysis of the other iterations in the sequence is similar. Consequently, iteration s_i makes r_i comparisons and charges $r_i + c(s_i) - c(s_{i+1})$ $(s_{y+1} = u_1)$ to free positions in $[s_i+1, u_1]$. Summing up these charges we get $\sum_{i=1}^{y} r_i + c(s_1) - c(u_1) \leqq u_1 - s_1$, and hence $\sum_{i=1}^{y} r_i \leqq u_1 - s_1 + c(s_1) - c(u_1)$. Moreover, Invariant 1 holds for u_1 inductively. Therefore we have the following Lemma.

LEMMA 8. *Let $s_1, s_2, \cdots, s_y$ be a light sequence and let u_1 be the iteration that starts the succeeding heavy sequence or the dummy iteration. Assuming that s_1 satisfies Invariant* 1, *Algorithm* SM *performs at most $u_1 - s_1 + c(u_1) - c(s_1)$ character comparisons during the execution of such a light sequence. Moreover, iteration u_1 satisfies Invariant* 1.

5.2. The analysis of heavy sequences. Consider a heavy sequence $u_1, u_2, \cdots, u_g$ and let s_1 be the iteration succeeding it. We estimate the total number of comparisons that the algorithm performs during the execution of such a sequence. We also show that s_1 satisfies Invariant 1 if it is not the dummy iteration.

We partition the sequence $u_1, u_2, \cdots, u_g$ into d consecutive subsequences $U_1, U_2, \cdots, U_d$ as follows. Let $u_{i_1} < u_{i_2} < \cdots < u_{i_{d-1}}$ be the iterations among $u_1, u_2, \cdots, u_g$ that match a suffix of the pattern. Then, $U_1 = \{u_1, \cdots, u_{i_1}\}$, $U_2 = \{u_{i_1+1}, \cdots, u_{i_2}\}$, $U_d = \{u_{i_{d-1}+1}, \cdots, u_g\}$. We denote the first and last element in each sequence U_j as first (U_j) and last (U_j), respectively. For each j, $1 \leqq j \leqq d$, last (U_j) can be thought of as being m positions to the left of a boundary line on the text that we denote by $\text{line}_j = \text{last}(U_j) + m$. Moreover, we set $\text{line}_0 = u_1$. Note that after iteration last (U_j), $1 \leqq j < d$, this boundary line is never crossed to the left because the algorithm sets $t$last to line_j, i.e., $t\text{last} = \text{line}_j$ when iteration last (U_j) ends. We bound the number of character comparisons that the algorithm can perform between two consecutive boundary lines during the given heavy sequence.

We assume that u_1 satisfies Invariant 1 when it starts. Since u_1 starts a heavy sequence, it matches a suffix of the pattern. Thus, $U_1 = \{u_1\}$. By Invariant 1, there are $c(u_1)$ noholes in the pattern that match the text characters in $t[\text{line}_0 + 1, \cdots, \text{line}_1]$ aligned with them at the beginning of iteration u_1. Such positions are not going to be considered by the algorithm during u_1, since $\text{start}(u_1) = c(u_1) + 1$. Moreover, the remaining text positions in $t[\text{line}_0 + 1, n]$ are free. During iteration u_1, in the worst case, the algorithm can compare all other symbols of $p[1, m]$. In any case, the pattern is shifted $u_2 - u_1 \geqq z$ positions to the right over the text. Fact 6 follows from these observations.

FACT 6. *Assume that u_1 satisfies Invariant* 1. *The number of character comparisons performed by the algorithm during u_1, the first iteration of a heavy sequence, is bounded by $m - c(u_1)$.*

We now derive an upper bound on the number of comparisons performed by the algorithm during iterations in U_j, $1 < j \leqq d$. For each $u \in U_j$, $1 < j \leqq d$, let $C(u)$ denote the set of text positions in $[\text{line}_{j-1} + 1, \text{line}_j]$ that are aligned with noholes in the pattern at the beginning of iteration u and that have been successfully matched against pattern characters during iterations in the heavy sequence preceding u. Of course, $C(u)$ may be the empty set. (Unlike $c(u)$, $|C(u)|$ only counts text positions between two lines that are aligned with noholes at the beginning of u.)

LEMMA 9. *Assume that $|U_j| > 1$ for some j, $1 < j \leqq d$. The number of character comparisons performed by the algorithm during an iteration $u \in U_j$, $u < \text{last}(U_j)$, is bounded by* $\max(1, \lfloor (u' - u)/2 \rfloor) + |C(u')| - |C(u)|$, *where u' is the iteration succeeding u in U_j.*

Proof. We give a proof for the case in which U_2 satisfies the hypothesis of the lemma. The proof can be immediately extended to the case $j > 2$.

Consider an iteration $u \in U_2$, $u < \text{last}(U_2)$, and let u' be the iteration succeeding it in U_2. Iteration u inherits $|C(u)|$ matches in $[\text{line}_1 + 1, \text{line}_2]$ from previous iterations in U_2. (Recall that all comparisons before U_2 were to the left of line_1.) Since u fails to match any suffix of the pattern, we must have $e \leqq nd$ at the end of iteration u. Thus, the algorithm performs q comparisons between noholes in $p[h_{\text{start}(u)} + 1, m]$ and the

corresponding text positions in $[\text{line}_1+1, n]$, skipping text positions in $C(u)$. One of those comparisons is a mismatch and $q-1$ are matches. Then, the pattern is shifted over the text by $u'-u = k\min(h_{\text{start}(u)+q-1})$ positions. Among the $|C(u)|+q-1$ text positions in $[\text{line}_1+1, \text{line}_2]$ that matched the corresponding pattern positions during iteration u, the algorithm records only $|C(u')|$ during iteration u' and such character comparisons will not be repeated during u'. We bound q by bounding $S = |C(u)|+q-|C(u')|-1$, i.e., the number of matches that the algorithm forgets in going from iteration u to u'. Let $k = u'-u$, $h = h_{\text{start}(u)+q-1}$, $b = \text{line}_1 - u$. We notice that $b \leqq h$ (since after u_1 the algorithm only compares text symbols beyond line_1) and that $k = k\min(h)$ is a period of $p[1, h]$. We next show that $u' < \text{line}_1$ which implies $k < b$ for our choice of k and b. Indeed, by the way we partition the heavy sequence, $\text{last}(U_2) \geqq u' > u$ is the only iteration in U_2 that can increase the value of variable $t\text{last}$ in algorithm SM when that iteration ends. Since $t\text{last} = \text{line}_1$ when iterations in U_2 start and $u' < t\text{last}(u')$ by definition of heavy sequence, we obtain $u' < \text{line}_1$.

Observe that the S matches that the algorithm forgets at the end of iteration u correspond to positions j in $p[1, h]$ such that j is a nohole, $j > b = \max(k, b)$, and $j-k$ is a hole. By Lemma 5, each of these pattern positions is reachable from a distinct hole in $p[1, k]$, $k = \min(k, b)$. Since our choice of k, b and h satisfies the hypotheses of Lemma 6, we obtain $S \leqq \max(0, \lfloor (k-2)/2 \rfloor)$ from bound (2). Therefore $q = S+1+|C(u')|-|C(u)| \leqq \max(1, \lfloor (u'-u)/2 \rfloor)+|C(u')|-|C(u)|$. □

We now consider the last iteration in U_j, $1 < j < d$. Again, without loss of generality, we limit our discussion to the case $j = 2$. Iteration $\text{last}(U_2)$ matches a suffix of the pattern since it advances $t\text{last}$ to $\text{last}(U_2)+m$ and, by assumption, U_2 does not conclude a heavy sequence. Thus, in the worst case, the algorithm performs $\text{line}_2 - \text{line}_1 - |C(\text{last}(U_2))|$ comparisons of text positions in $[\text{line}_1+1, \text{line}_2]$, skipping text positions in $C(\text{last}(U_2))$. Then the pattern is shifted over the text by at least z positions and $t\text{last}$ is set equal to line_2. Thus, all text positions in $[\text{line}_1+1, \text{line}_2]$ are not going to be considered again by the algorithm. Fact 7 summarizes these observations.

FACT 7. *The algorithm performs at most* $\text{line}_j - \text{line}_{j-1} - |C(\text{last}(U_j))|$ *character comparisons during iteration* $\text{last}(U_j)$, $1 < j < d$.

If we had enough free text positions in $[\text{line}_{j-1}+1, \text{line}_j]$, $j < d$, to pay for all character comparisons performed during iterations in U_j, the analysis of heavy sequences would be the same as for light sequences. Unfortunately, $\text{last}(U_j)$ must use most of the free text positions in $[\text{line}_{j-1}+1, \text{line}_j]$ in order to pay for its own character comparisons. The remaining iterations in U_j pay for their character comparisons using a credit line. For any sequence $S = \{x_1; x_2; \cdots; x_{|S|}\}$ let $\text{credit}(S) = \sum_{k=2}^{|S|} \max(1, \lfloor (x_k - x_{k-1})/2 \rfloor)$ (if $|S| \leqq 1$, let $\text{credit}(S) = 0$).

LEMMA 10. *For each* U_j, $1 < j < d$, *the total number of comparisons performed by the algorithm during iterations in* U_j *is bounded by*

$$\text{line}_j - \text{line}_{j-1} + \text{credit}(U_j). \tag{5}$$

Proof. By induction on j. For $j = 2$, an upper bound on the total number of comparisons performed by the algorithm during iterations in U_2 is obtained by summing the bound in Lemma 9 for each $u \in U_2$, $u < \text{last}(U_2)$, and that in Fact 7. Since $|C(u_2)| = 0$ (u_2 is the first iteration in U_2), this sum can be rewritten as (5). The proof is similar for $j > 2$. □

We now consider the number of character comparisons performed by the algorithm during iteration u_g, the last iteration in the heavy sequence. Throughout our discussion, we assume that $u_g \neq u_1$, i.e., $g > 1$ and $d > 1$.

LEMMA 11. *Let $g>1$. The number of character comparisons that the algorithm performs during iteration u_g is bounded by*

$$s_1 + c(s_1) - \text{line}_{d-1} - |C(u_g)| \tag{6}$$

when u_g is the last iteration of the algorithm, and by

$$s_1 + c(s_1) - \text{line}_{d-1} - |C(u_g)| + \max\left(1, \left\lfloor \frac{\text{line}_{d-1} - u_g + 1}{2} \right\rfloor\right) \tag{7}$$

when u_g is not the last iteration of the algorithm.

Proof. Iteration u_g inherits $|C(u_g)|$ matches in $[\text{line}_{d-1}+1, \text{line}_d]$ from previous iterations in U_d. Obviously, if $U_d = \{u_g\}$ then $C(u_g)$ is empty. The algorithm performs q comparisons between text characters in $t[\text{line}_{d-1}+1, \text{line}_d]$ and the corresponding pattern characters. Then, the pattern is shifted over the text by at least $\text{line}_{d-1} - u_g$ positions (past $\text{line}_{d-1} = t\text{last}(s_1)$). We consider two cases according to whether or not u_g is the last iteration of the algorithm. Let $y = \text{start}(u_g) + q - 1$, i.e., $p[h_y + 1]$ is the last character of the pattern compared during this iteration.

Assume that u_g is the last iteration of the algorithm. Obviously, $q \leqq \text{line}_d - \text{line}_{d-1} - |C(u_g)|$. Recalling that $s_1 = n$ when s_1 is the dummy iteration and that $\text{line}_d \leqq n$, we obtain that q is bounded by (6).

Assume u_g is not the last iteration of the algorithm. We consider two subcases: $y \geqq nd+1$ and $y \leqq nd$.

Subcase (i). $y \geqq nd+1$ and u_g not the last iteration of the algorithm: At the end of u_g, the pattern is shifted by $r\min(h_y)$, so $s_1 - u_g = r\min(h_y)$. Since $y > nd$ and the algorithm processes the sequence h_i in increasing order of i until either $i = m+1$ or a mismatch is found, $p[h_{nd}+1] = t[u_g + h_{nd} + 1]$. Recall that by definition of heavy sequence, $t\text{last}(s_1) \leqq s_1$. Thus, by Lemma 7, s_1 cannot have a suffix-prefix overlap with any iteration. It follows that $r\min(h_y) = m$ since $r\min(m-1) = m$ and $r\min(h_y) < m$ for $h_y < m-1$ would imply that s_1 has a suffix-prefix overlap with u_g. Thus, $s_1 - u_g = m$. This equality implies that $s_1 = \text{line}_d$ ($\text{line}_d = u_g + m$) and, in turn, that $c(s_1) = 0$ (no comparisons made beyond line_d). Therefore, $q \leqq \text{line}_d - \text{line}_{d-1} - |C(u_g)|$ is bounded by $s_1 + c(s_1) - \text{line}_{d-1} - |C(u_g)|$ and (7) holds.

Subcase (ii). $y \leqq nd$ and u_g is not the last iteration of the algorithm. Since $h_m = 0$, $nd < m$ ($k\min(0)$ is never defined) and given the way the algorithm processes the sequence h_i, we have that the q comparisons made during iteration u_g result in $q-1$ matches and a mismatch. At the end of iteration u_g, the total number of matches between pattern positions and text positions in $[\text{line}_{d-1}+1, \text{line}_d]$ is $|C(u_g)| + q - 1$. We bound q by bounding $|C(u_g)| + q$.

Let N be the set of text positions in $[\text{line}_{d-1}+1, \text{line}_d]$ that the algorithm has successfully matched with the corresponding pattern positions at the end of iteration u_g. $C(u_g)$ is included in N. Thus, $|N| + 1 = |C(u_g)| + q$. We partition N into three disjoint subsets as follows. N_{noholes} is the set of positions in N aligned with a nohole of the pattern at beginning of iteration s_1. N_{dead} is the set of positions in N falling in the interval $[\text{line}_{d-1}+1, s_1]$. N_{holes} is the set of positions in N aligned with holes at the beginning of iteration s_1. We claim that

$$N_{\text{holes}} \leqq s_1 - \text{line}_{d-1} - |N_{\text{dead}}| + \max\left(0, \left\lfloor \frac{\text{line}_{d-1} - u_g - 1}{2} \right\rfloor\right). \tag{8}$$

Indeed, let $h = h_y$, $k = k\min(h_y) \leqq h$ and $b = \text{line}_{d-1} - u_g$. Since $\text{line}_{d-1} = t\text{last}(s_1) \leqq s_1$ by definition of heavy sequence, $b = \text{line}_{d-1} - u_g \leqq s_1 - u_g = k\min(h_y) = k$. Thus, $b \leqq k < h$ and k is a period of $p[1, h]$. Consider $p[1, h]$ and let J be the set of

positions j in that string such that $k\min(h) < j \leqq h$, j is a nohole and $j - k\min(h)$ is a hole. At the end of iteration u_g, each $j \in J$ has a match with exactly one text position in N_{holes} and vice versa. By Lemma 5, each position in J is reachable from a distinct hole in $p[1, k\min(h)]$. We can further partition J in two subsets J' and $\hat{J}$ such that $j' \in J'$ is reachable from a distinct hole in $p[1, b]$ and $\hat{j} \in \hat{J}$ is reachable from a distinct hole in $p[b+1, k\min(h)]$. We have $|\hat{J}| + |N_{\text{dead}}| \leqq k\min(h) - b = s_1 - \text{line}_{d-1}$, since $k\min(h) = s_1 - u_g$, $b = \text{line}_{d-1} - u_g$ and at the end of iteration u_g each position in N_{dead} has a match with a distinct nohole in $p[b+1, k\min(h)]$. Therefore, $|\hat{J}| \leqq s_1 - \text{line}_{d-1} - |N_{\text{dead}}|$. We use Lemma 6 with the same k, h and b to bound J'. Each $j' \in J'$ is such that $j' > k = \max(k, b)$, $j' - k$ is a hole and j' is reachable from a hole in $p[1, b]$, $b = \min(k, b)$. Therefore, our choice of J', k, b and h satisfies the hypotheses of Lemma 6 and since $b \leqq k$, by (1) $|J'| \leqq \max(0, \lfloor (b-1)/2 \rfloor) = \max(0, \lfloor (\text{line}_{d-1} - u_g - 1)/2 \rfloor)$. Using the bounds on $|\hat{J}|$ and $|J'|$ and the fact that $|N_{\text{holes}}| = |J| = |\hat{J}| + |J'|$, we obtain (8).

Recall that N_{noholes} is the set of positions in N aligned with noholes of the pattern at the beginning of iteration s_1. We have that $|N_{\text{noholes}}| = \text{start}(s_1) - 1 = c(s_1)$ (the proof is analogous to the one that (3) holds and is omitted).

Using the identities $|N_{\text{noholes}}| = \text{start}(s_1) - 1 = c(s_1)$, $|N| + 1 = |C(u_g)| + q$, and inequality (8) we obtain that q is bounded by (7). □

We can finally bound the total number of comparisons performed by the algorithm during iterations in U_d. Again, we must use credit in order to pay for some of the comparisons performed during iterations in U_d. For any sequence $S = \{x_1; x_2; \cdots; x_{|S|}\}$ let credit* $(S) = \sum_{k=2}^{|S|-1} \max(1, \lfloor (x_k - x_{k-1})/2 \rfloor) + \max(1, \lfloor (x_{|S|} - x_{|S|-1} + 1)/2 \rfloor)$ (if $|S| \leqq 1$, credit* $(S) = 0$).

LEMMA 12. *The number of character comparisons performed by the algorithm during iterations in U_d, $d > 1$, is bounded by*

$$s_1 + c(s_1) - \text{line}_{d-1} + \text{credit*}(U_d \cup \{\text{line}_{d-1}\}), \tag{9}$$

when u_g is not the last iteration of the algorithm; by

$$s_1 + c(s_1) - \text{line}_{d-1} + \text{credit}(U_d) \tag{10}$$

when u_g is the last iteration of the algorithm.

Proof. Assume that u_g is not (is, respectively) the last iteration of the algorithm. A bound on the number of character comparisons performed during iterations in U_d is obtained by summing the bound in Lemma 9 for each $u \in U_d$, $u < u_g$, and bound (7) ((6), respectively) in Lemma 11. Since $|C(\text{first}(U_d))| = 0$, this sum can be rewritten as (9) ((10), respectively). □

We will repeatedly use the following lemma to bound the credit of each U_j, $1 < j \leqq d$. Its conditions are easily verified in each case.

LEMMA 13. *Consider the subsequence U_j, for some j, $1 < j \leqq d$. Let α and β be real numbers, $z \geqq \beta$. For each $u \in U_j \cup \{\text{line}_{j-1}\}$, $\lfloor \alpha(u - \text{first}(U_j) + \beta) \rfloor \leqq \lfloor \alpha((u - \text{last}(U_{j-1}))((z' + \beta)/m)) \rfloor$.*

Proof. Let $\text{left} = (u - \text{first}(U_{j-1}) + \beta)$ and let $\text{right} = (u - \text{last}(U_{j-1}))((z' + \beta)/m)$. We prove that left $\leqq$ right and therefore $\lfloor \alpha \text{left} \rfloor \leqq \lfloor \alpha \text{ right} \rfloor$ proving the lemma.

Since $\text{first}(U_j) - \text{last}(U_{j-1}) \geqq z$ by definition of $r\min$, $\text{left} \leqq u - \text{last}(U_{j-1}) - z + \beta$. Since $u \leqq \text{line}_{j-1}$ and $\text{line}_{j-1} - \text{last}(U_{j-1}) = m$, $u - \text{last}(U_{j-1}) \leqq m$. Moreover, $z \geqq \beta$ by assumption. Thus, $\text{left} \leqq (u - \text{last}(U_{j-1}) - (z - \beta)m/m) \leqq ((u - \text{last}(U_{j-1}))\ (1 - (z - \beta)/m)) = (u - \text{last}(U_{j-1}))((z' + \beta)/m) = \text{right}$, since $m - z = z'$. □

We are now ready to prove the following lemma.

LEMMA 14. *Let $p[1, m]$ be a pattern with period z. Let $u_1, u_2, \cdots, u_g$ be a heavy sequence and let s_1 be the succeeding iteration or the dummy iteration. Assume that u_1*

satisfies Invariant 1. *If u_g is not the last iteration of the algorithm,* SM *performs at most*

$$s_1 - u_1 + c(s_1) - c(u_1) + \left\lfloor \left(\text{line}_{d-1} - u_1 \right) \left(\frac{z'}{m} \right) \right\rfloor \tag{11}$$

character comparisons during the execution of such a heavy sequence. If u_g is the last iteration of the algorithm, the number of character comparisons is bounded by

$$s_1 - u_1 + c(s_1) - c(u_1) + \left\lfloor \left(u_g - u_1 \right) \left(\frac{z'}{m} \right) \right\rfloor. \tag{12}$$

Moreover, s_1 satisfies Invariant 1.

Proof. We consider two cases: u_1 is the only iteration in the sequence and its logical complement. We first derive a bound on the total number of comparisons performed by the algorithm during the heavy sequence for both cases. Then, we show that s_1 satisfies Invariant 1.

Assume that u_1 is the only iteration in the heavy sequence. By Fact 6 the number of comparisons during iteration u_1 is at most $m - c(u_1)$. Moreover, $u_1 + m = \text{line}_1 = t\text{last}(s_1) \leqq s_1$ and $s_1 - u_1 \geqq m$ if $s_1 \leqq n - m$ and also if $s_1 > n - m$. Therefore, $m - c(u_1)$ is bounded by (11) when u_1 is not the last iteration of the algorithm and is bounded by (12) when u_1 is the last iteration of the algorithm.

Consider now the case in which u_1 is not the only iteration in the heavy sequence, i.e., $g > 1$ and $d > 1$. We recall that the number of character comparisons performed by the algorithm during U_1 is

$$m - c(u_1) = \text{line}_1 - u_1 - c(u_1). \tag{13}$$

For each U_j, $1 < j < d$, we now bound credit (U_j). Since $\max(1, \lfloor x/2 \rfloor) \leqq \lfloor x \rfloor$, for $x \geqq 1$, and the sequence U_j is strictly increasing, we have credit $(U_j) \leqq \lfloor \text{last}(U_j) - \text{first}(U_j) \rfloor$ and by Lemma 13 (with $u = \text{last}(U_j)$, $\alpha = 1$, $\beta = 0$) credit $(U_j) \leqq \lfloor \text{last}((U_j) - \text{last}(U_{j-1}))(z'/m) \rfloor$. Using (5), the number of character comparisons performed during U_j is bounded by

$$\text{line}_j - \text{line}_{j-1} + \left\lfloor (\text{last}(U_j) - \text{last}(U_{j-1})) \left(\frac{z'}{m} \right) \right\rfloor. \tag{14}$$

Using the fact that $\max(1, \lfloor (x+1)/2 \rfloor) \leqq \lfloor x \rfloor$, for $x \geqq 1$, (9), and Lemma 13 (with $u = \text{line}_{d-1}$, $\alpha = 1$, $\beta = 0$), the number of character comparisons performed during U_d, when u_g is not the last iteration of the algorithm, is bounded by

$$s_1 + c(s_1) - \text{line}_{d-1} + \left\lfloor (\text{line}_{d-1} - \text{last}(U_{d-1})) \left(\frac{z'}{m} \right) \right\rfloor. \tag{15}$$

Using (10) and the same arguments used to bound credit (U_j), the number of character comparisons performed during U_d, when u_g is the last iteration of the algorithm, is bounded by

$$s_1 + c(s_1) - \text{line}_{d-1} + \left\lfloor (u_g - \text{last}(U_{d-1})) \left(\frac{z'}{m} \right) \right\rfloor \tag{16}$$

When u_g is not (respectively, is) the last iteration of the algorithm and $g > 1$, a bound on the number of character comparisons performed by the algorithm during a heavy sequence is obtained by adding (13), (14) for each $1 < j < d$, and (15) (respectively, (16)). Such a sum is bounded by (11) (respectively, (12)) since the sequence last (U_j) is increasing.

We next show that s_1 satisfies Invariant 1 when it starts. Indeed, since u_1 satisfies Invariant 1, text positions in $[u_1 + 1, n]$ are either free or match all the noholes in

$p[1, h_{\text{start}(u_1)}]$. Thus, text positions in $t[u_1+1, s_1]$ can become busy at the end of the heavy sequence. This amounts to s_1-u_1 text positions being declared busy. As for positions in $t[s_1+1, n]$, notice they are either free (they were free when u_1 started) or they match noholes in $p[1, h_{\text{start}(s_1)}]$. Therefore s_1 satisfies Invariant 1 when it starts.

Notice that the positions declared busy at the end of s_1 can be used to pay for s_1-u_1 character comparisons. This "covers" $s_1-u_1-c(u_1)$ comparisons performed by the algorithm during the heavy sequence as well as the $c(u_1)$ character comparisons (all matches) that u_1 has inherited from the previous iteration. The $c(s_1)$ matches inherited from u_g by s_1 are left unpaid. The remaining comparisons are paid by a credit line. □

5.3. Putting the sequences together.

THEOREM 1. *Let $p[1, m]$ be a pattern with period z, $m=z+z'$, and $z'<z$, and let $t[1, n]$ be a text. Algorithm* SM *finds all occurrences of p in t in $O(n+m)$ time. In the worst case, the algorithm performs at most $n+\lfloor(n-m)(z'/(z+z'))\rfloor$ character comparisons.*

Proof. The $O(n+m)$ time bound is obvious. As for the number of character comparisons, we notice that iteration zero satisfies Invariant 1 trivially and, by Lemmas 8 and 14, each iteration starting either a light or a heavy sequence satisfies the same invariant and the bounds in Lemmas 8 and 14 hold. The bound of the theorem follows by summing up these bounds: The sum of the terms $u_1-s_1+c(u_1)-c(s_1)$ or $s_1-u_1+c(s_1)-c(u_1)$ is bounded by n and the sum of the terms $\lfloor(s_1-u_1)(z'/(z+z'))\rfloor$ or $\lfloor(u_g-u_1)(z'/(z+z'))\rfloor$ is bounded by $\lfloor(n-m)(z'/(z+z'))\rfloor$. Since $c(u)=0$ for u the dummy iteration, no comparison is left unpaid. □

The bound in Theorem 1 is tight for each value of z and $m=z+z'$, $z'<z$. Indeed, let $p[1, m]=a^{z'}b^{z-z'}a^{z'}$. Choose $n=cm$, c integer, and $t[1, n]=(p[1, m])^c$. Notice that the first nohole of the pattern is $z'+1$, since $k\min(z')=1$. We show that when the algorithm has discovered all occurrences of $p[1, m]$ in $t[1, jm]$, for some j, $1\leqq j\leqq c$, it has performed $jm+(j-1)z'$ character comparisons and the pattern is aligned with $t[(j-1)m+1, jm]$. Indeed, when the algorithm has discovered all occurrences of $p[1, m]$ in $t[1, m]$, it has performed m comparisons and the pattern is aligned with $t[1, m]$. Assume that the algorithm has just found all occurrences of $p[1, m]$ in $t[1, jm]$, for some j, $1\leqq j<c$, it has performed $jm+(j-1)z'$ character comparisons and the pattern is aligned with $t[(j-1)m+1, jm]$. In order to find the next occurrence, the algorithm shifts the pattern over the text by z positions and tests whether the first nohole of the pattern matches the corresponding text character, i.e., $p[z'+1]$ is compared with $t[jm+1]$. This comparison results in a mismatch ($p[z'+1]\neq p[1]=t[jm+1]$) and the pattern is shifted by $k\min(z')=1$. This is repeated $z'-1$ additional times, since $t[jm+2]=\cdots=t[jm+z']=p[1]\neq p[z'+1]$. Then, SM finally discovers, in m comparisons, that $p[1, m]=t[jm+1, (j+1)m]$. The number of comparisons during this phase is $m+z'$ which added to the $jm+(j-1)z'$ comparisons performed to find all occurrences of $p[1, m]$ in $t[1, jm]$ gives a total of $(j+1)m+jz'$ character comparisons. Moreover, the pattern is aligned with $t[jm+1, (j+1)m]$.

Thus, SM performs $cm+(c-1)z'=n+\lfloor(n-m)(z'/(z+z'))\rfloor$ character comparisons to find all occurrences of $p[1, m]$ in $t[1, n]$, $n=cm$.

6. Saving character comparisons. The analysis of Algorithm SM carried out in the previous section and the examples given there show that the main source of inefficiency (for character comparisons) is due to shifts by one during heavy sequences of iterations: Recall that we used $\max(1, \lfloor x/2\rfloor)\leqq x$, when $x\geqq 1$, where x is the distance between two consecutive iterations, or the length of the shift. This length can be 1 exactly when

we have a mismatch with $e = 1$ and we shift by $k\min(h_1) = 1$. Whenever $x \geqq 2$, we have $\max(1, \lfloor x/2 \rfloor) \leqq x/2$. In this section we describe a variation of Algorithm SM that takes care of this shortcoming by handling differently certain shifts by 1.

We give a somewhat redundant pseudo code version of the new algorithm SM′. It behaves like Algorithm SM, except that it handles differently the shifts by one during heavy sequences. Based on the length of the suffix-prefix overlap of the current iteration, it decides to behave as Algorithm SM or look for the regular expression $(p[1])^*p[l+1]$. Recall that l is the maximal integer such that $p[1] = \cdots = p[l]$ and $p[1] \neq p[l+1]$. We assume that $p \neq (p[1])^m$, $m > 1$, because such a pattern can be easily searched for in n comparisons. We first give the pseudocode for the new algorithm and then discuss it in some detail.

```
        Algorithm SM′
begin
    b ← 0; pstart ← 1; tlast ← 0; heavy = false
  repeat
    e ← pstart;
    if heavy = true and e = 1 and tlast > b + 1 then
                begin
                —Find longest prefix of text[tlast + 1, n] that matches—
                —the regular expression p[1]*p[l + 1]—
                i ← tlast − b + 1;
                while b + i ≦ n and text[b + i] = p[1] do i ← i + 1;
                if i ≦ l or text[b + i] ≠ p[l + 1] then
                                        begin
                                          b ← b + i;
                                          pstart ← 1;
                                          tlast ← b;
                                          goto decide;
                                        end
                                    else
                                        begin
                                        pstart ← 2;
                                        tlast ← b + i;
                                        b ← tlast − (l + 1);
                                        goto decide;
                                        end
                end
    while e ≦ m and tlast < b + h_e + 1 and p[h_e + 1] = text[b + h_e + 1] do
    e ← e + 1;
    —This part is as in Algorithm SM—
        Check whether or not an occurrence has been found
        set variables tlast, b, pstart accordingly
    —Decide how to set heavy—
    decide:     if b ≧ tlast then heavy ← false;
                                    else   heavy ← true;
    until b > n − m
end
```

As before, u is an iteration if Algorithm SM′ sets variable $b = u$ during its execution. Notice that SM′ simulates SM for some iterations while it does not for others. By

simulation we mean that SM′ compares pattern positions against the corresponding text positions according to the sequence $h_1, \cdots, h_m$ defined in § 3. We point out that the sequence of iterations for algorithm SM′ is nondecreasing (for algorithm SM it is increasing) since b can be set to the same value twice. Indeed, assume that SM′ does not simulate SM during some iteration u (it executes the "new" part of code) and it tries to match a prefix of $t[t\text{last}+1, n]$ with $(p[1])^*p[l+1]$. If the match is successful, $t\text{last}$ and b are updated and the numeric value of the next iteration u' may be equal to u. (If the match is not successful, then b is increased and $u < u'$.) However, notice that an iteration u in which SM′ does not simulate SM must be followed by an iteration u' in which it does, since either heavy = false or $p\text{start} = 2$ (the next value assigned to e) and the new part of code cannot be executed during u'. So, we can still distinguish between two consecutive iterations u and u' because either $|u-u'|>0$ or SM′ does not simulate SM during only one of them. Therefore, for each iteration u of SM′, we can still define start (u) and last (u) as in the previous section, provided that, when there is ambiguity, we give the additional information of whether SM′ simulates SM during u.

THEOREM 2. SM′ *finds all occurrences of the pattern in the text.*

Proof. For any given iteration, SM′ either simulates SM or searches for the regular expression $(p[1])^*p[l+1]$. We prove the theorem by showing that when each iteration u starts the following invariant holds.

INVARIANT 2. SM′ *has correctly found all occurrences of* $p[1, m]$ *in text positions* $[1, u]$ *and* $t[u+h_1+1]=p[h_1+1], \cdots, t[u+h_{p\text{start}-1}+1]=p[h_{p\text{start}-1}+1]$. *Moreover, if* $t\text{last}-u>0$, *then* $t[u+1, t\text{last}]=p[1, t\text{last}-u]$ *and* $t\text{last}-u \leqq h_{p\text{start}}$.

Initially $(u=0)$, Invariant 2 is trivially satisfied. Consider an iteration $u>0$; assume it satisfies the invariant. We show that u', the next iteration, also satisfies the invariant. We discuss only the case in which SM′ does not simulate SM during iteration u, since the other case is implied by the way SM works and its correctness.

Assume that SM′ does not simulate SM during iteration u. Thus, when u starts, $e = p\text{start} = 1$ and $t\text{last}-u = t\text{last}-b>1$. Since h_1 is the first position of the pattern having $k\text{min}$ defined, we have that $h_1 = l$. By Invariant 2, $t[u+1, t\text{last}]=p[1, t\text{last}-u]$ and $t\text{last}-u \leqq h_1 = l$. Thus, $t[u+1, t\text{last}]$ is a prefix of $p[1, l]$.

SM′ searches for a prefix of $t[t\text{last}+1, n]$ that matches the regular expression $(p[1])^*p[l+1]$. Assume that it scans $t[t\text{last}+1, k+1]$ during this search. If $k-u<l$ or $t[k+1]\neq p[l+1]$, then there can be no occurrence of the pattern in the text in $[u+1, k+1]$ and the algorithm correctly sets the next iteration $u'=k+1$, and this iteration must start from scratch ($p\text{start}=1$ and $t\text{last}=u'$). Thus, u' satisfies the invariant. If $k-u \geqq l$ and $t[k+1]=p[l+1]$, then there can be no occurrence of the pattern in the text in $[u+1, k-l]$. However, there may be an occurrence in $k-l+1$ since $t[k-l+1, k+1]=p[1, l+1]$. The algorithm correctly sets $u'=k-l$, $t\text{last}=k+1$ and $p\text{start}=2$. Noting that $t\text{last}-u' \leqq l+1 \leqq h_2$ and that $p[h_1+1]=p[l+1]=t[k+1]=t[u'+h_1+1]$, u' satisfies the invariant. □

The analysis of Algorithm SM′ proceeds along the same lines as the analysis of Algorithm SM. We divide the iterations into light and heavy sequences. The definition of both sequences is as in the previous section. We remark that Lemma 7 does not hold anymore since the condition $u < t\text{last}\,(u)$ does not necessarily imply that u has a suffix-prefix overlap with an iteration u' preceding it.

Again, for each iteration starting a sequence we maintain Invariant 1. Thus, we can still use Lemma 8 for the analysis of a light sequences. In order to bound the number of character comparisons performed by Algorithm SM′, we need to analyze heavy sequences again and to show that the iteration succeeding a heavy sequence satisfies Invariant 1.

6.1. The analysis of heavy sequences for Algorithm SM′. Consider a heavy sequence $u_1, u_2, \cdots, u_g$ and let s_1 be the iteration succeeding it. We partition the sequence $u_1, u_2, \cdots, u_g$ into d consecutive subsequences $U_1, U_2, \cdots, U_d$ as in the preceding section. Again, for each U_j, $1 \leqq j \leqq d$, we define a boundary line, $\text{line}_j = \text{last}(U_j) + m$. Moreover, we set $\text{line}_0 = u_1$. As before, last (U_j) is the only iteration in U_j that matches a suffix of the pattern, $1 \leqq j < d$.

We need the following observations about iterations in U_j, $1 < j \leqq d$, handling a shift by one.

LEMMA 15. *Each U_j, $1 < j \leqq d$ has at most one iteration u such that $e = 1$ when it starts. It is either the next to last or the last iteration in U_j. When $u = \text{line}_{j-1} - 1$, it is the last iteration in U_j. When $u < \text{line}_{j-1} - 1$, it cannot match any suffix of the pattern.*

Proof. Assume that there is more than one iteration in U_j such that $e = 1$ holds when it starts. Let u and u', $u \leqq u'$ be the first two. We consider two cases: $u = \text{line}_{j-1} - 1$ and $u < \text{line}_{j-1} - 1$ (we cannot have $u \geqq \text{line}_{j-1}$ since $u \in U_j$). Notice that when u starts, $t$last must be equal to line_{j-1} since no iteration in U_j preceding u has matched a suffix of the pattern or it had $e = 1$, an essential condition to modify the value of $t$last in the new code. Moreover, SM′ simulates SM at least up to the iteration preceding u.

Case $u = \text{line}_{j-1} - 1$. Since $t\text{last} = \text{line}_{j-1} = u + 1$ Algorithm SM′ simulates SM during the execution of iterations in U_j at least up, and including, u. Now, if u finds a mismatch before matching any suffix of the pattern, it shifts by at least up to 1 and does not update $t$last: the next iteration $\hat{u}$ is such that $\text{line}_{j-1} = t\text{last} = t\text{last}(\hat{u}) \leqq \hat{u}$. Therefore u concludes the heavy sequence. If u matches a suffix of the pattern, u concludes U_j. In any case, u' cannot exist and u is the last iteration in U_j.

Case $u < \text{line}_{j-1} - 1$. Since $t\text{last} = \text{line}_{j-1} > u + 1$, heavy = true and $e = 1$, Algorithm SM′ looks for the longest prefix of $t[\text{line}_{j-1} + 1, n]$ matching the regular expression $(p[1])^* p[l+1]$. If no prefix matches, u ends the heavy sequence and no u'exists. If a prefix matches, then $\hat{u} = t\text{last} - (l+1) = t\text{last}(\hat{u}) - (l+1)$, the iteration succeeding u, must shift by at least $l+1$. Thus, if u' exists we have $\hat{u} < u'$. Moreover, SM′ must simulate SM during $\hat{u}$. Now, if $\hat{u}$ does not match a suffix of the pattern, it concludes the heavy sequence. Indeed, $t\text{last} = t\text{last}(\hat{u})$ cannot be updated and the next iteration is $s \geqq \hat{u} + l + 1 = t\text{last}(\hat{u}) = t\text{last}(s)$. Therefore no u' exists. If $\hat{u}$ matches a suffix of the pattern, $\hat{u}$ is the last iteration in U_j and no u' exists. Therefore, u is either the next to last or the last iteration in U_j.

Iteration u can match a suffix of the pattern only if $p[1, m] = p[1]^l p[l+1]$ (see Algorithm SM′). But, recalling the definition of heavy sequence, we can have only heavy sequences consisting of one element for $p[1]^l p[l+1]$. Indeed, each iteration that matches a suffix of the pattern must then shift the pattern by $l+1$ positions. Thus, there is no suffix-prefix overlap between consecutive iterations implying that $b \geqq t\text{last}$ always holds during the execution of SM′ for $p[1]^l p[l+1]$. So, variable heavy is always false, a contradiction since we are assuming that heavy = true when u starts. □

For each U_j, $1 < j \leqq d$, let so (U_j) denote the only iteration in U_j, if any, such that $e = 1$ and $\text{so}(U_j) < \text{line}_{j-1} - 1$ when it starts. Notice that so (U_j) is the only iteration in U_j in which SM′ does not simulate SM. We also need the following fact.

FACT 8. *Assume that* so (U_j) *exists in U_j, $1 < j \leqq d$, then the pattern must have $p[1, q] = p[m - q + 1, m] = p[1]^q$, for some q, $2 \leqq q \leqq l$.*

Proof. By the way we partition the heavy sequence, last (U_{j-1}) must match a suffix of the pattern. Moreover, so (U_j) is the first iteration in U_j that can advance $t$last. Therefore, so (U_j) has a suffix-prefix overlap with last (U_{j-1}). The length of the overlap is $\text{line}_{j-1} - \text{so}(U_j) \geqq 2$, since $\text{so}(U_j) < \text{line}_{j-1} - 1$ by hypothesis. Therefore, letting $q = \text{line}_{j-1} - \text{so}(U_j)$, $p[1, q] = p[m - q + 1, m]$, $q \geqq 2$. But $q \leqq l$ because the overlap with last (U_{j-1}) is at most l; otherwise the shift and e would be larger than 1. □

Assume that u_1 satisfies Invariant 1 when it starts. Since u_1 is the first iteration of the heavy sequence, Algorithm SM′ starts with heavy = false. This implies that SM′ simulates SM during the execution of u_1. Thus, Fact 6 holds.

We now estimate the number of comparisons that SM′ performs for iterations in U_j, $1 < j \leqq d$.

LEMMA 16. *Consider an iteration $u \in U_j$, for some j, $1 < j \leqq d$. Assume that $u <$ last (U_j) and that $u \neq$ so (U_j), if* so (U_j) *exists. The number of character comparisons performed by* SM′ *during iteration u is bounded by $\lfloor (u'-u)/2 \rfloor + |C(u')| - |C(u)|$, where u' is the iteration succeeding u in U_j.*

Proof. By the hypothesis of Lemma 16 and Lemma 15, no iteration in U_j up to, and including, u can start with a value of $e = 1$. This implies that SM′ simulates SM on these iterations. Moreover $u' - u \geqq 2$, since none of those iterations can shift by 1. Therefore, the bound in Lemma 9 holds and it reduces to $\lfloor (u'-u)/2 \rfloor + |C(u')| - |C(u)|$. □

We now consider last (U_j) and so (U_j), if it exists, for a given subsequence U_j, $1 < j < d$. Then, we consider last (U_d) and so (U_d), if it exists.

FACT 9. *Consider a subsequence U_j, for some j, $1 < j < d$. Assume that* so (U_j) *does not exist. The number of character comparisons performed by* SM′ *during iteration* last (U_j) *is bounded by* $\text{line}_j - \text{line}_{j-1} - |C(\text{last}(U_j))|$.

Proof. Since so (U_j) does not exist, SM′ simulates SM during all iterations in U_j. Thus, the bound follows as in Fact 7. □

FACT 10. *Consider a subsequence U_j, for some j, $1 < j < d$, and assume that* so (U_j) *exists. Then* so (U_j) *is not the last iteration of U_j and the number of character comparisons performed by* SM′ *during iterations* so (U_j) *and* last (U_j) *is bounded by* $\text{line}_j - \text{line}_{j-1} - |C(\text{so}(U_j))| + 1$.

Proof. We note that $C(\text{so}(U_j)) = \phi$ when so (U_j) starts, because it starts with $e = 1$. Since so $(U_j) < \text{line}_{j-1} - 1$, Algorithm SM′ tries to find the longest prefix of $t[\text{line}_{j-1}+1, n]$ that matches the regular expression $(p[1])^* p[l+1]$. Since we are assuming that U_j does not conclude a heavy sequence, the algorithm must succeed in its task (otherwise it would set $t\text{last} = b$ and then heavy = false). Therefore, it sets $b = \text{last}(U_j)$ and $t\text{last} = \text{last}(U_j) + l + 1$ and the text characters to the left of last $(U_j) + l + 2$ will not be considered again by future iterations. By inspection of the pseudocode performing the search for $(p[1]^*)p[l+1]$, the algorithm performs last $(U_j) + l + 2 - \text{line}_{j-1}$ character comparisons, one of which is a mismatch in iteration so (U_j). Now, iteration last (U_j) performs at most $m - l - 1$ character comparisons. Therefore, the total for the two iterations is $m + \text{last}(U_j) - \text{line}_{j-1} + 1 = \text{line}_j - \text{line}_{j-1} - |C(\text{so}(U_j))| + 1$. □

We can now prove the following lemma:

LEMMA 17. *The number of character comparisons performed by Algorithm* SM′ *during U_j, $1 < j < d$, is bounded by*

$$\text{line}_j - \text{line}_{j-1} + \left\lfloor (\text{last}(U_j) - \text{last}(U_{j-1})) \left(\frac{z'}{2m} \right) \right\rfloor, \tag{17}$$

if so (U_j) *does not exist. Otherwise, it is bounded by*

$$\text{line}_j - \text{line}_{j-1} + \left\lfloor (\text{so}(U_j) - \text{last}(U_{j-1})) \left(\frac{z'+2}{2m} \right) \right\rfloor \tag{18}$$

when $|U_j| > 2$; and by

$$\text{line}_j - \text{line}_{j-1} + 1 \tag{19}$$

when $|U_j| = 2$.

Proof. Assume that so (U_j) does not exist. A bound on the number of character comparisons performed during iterations in U_j is obtained by adding the bounds in Lemma 16, for each iteration $u \neq \text{last}(U_j)$, and the bound in Fact 9. This sum is bounded by $\text{line}_j - \text{line}_{j-1} + \lfloor(\text{last}(U_j) - \text{first}(U_j))/2\rfloor$, since $C(\text{first}(U_j)) = \phi$ and, in turn, by (17) since $\lfloor(\text{last}(U_j) - \text{first}(U_j))/2\rfloor \leqq \lfloor(\text{last}(U_j) - \text{last}(U_{j-1}))(z'/2m)\rfloor$ by Lemma 13 (with $u = \text{last}(U_j)$, $\alpha = \frac{1}{2}$ and $\beta = 0$).

Assume now that so (U_j) exists. If $|U_j| = 2$, bound (19) follows from Fact 10 and the fact that $C(\text{so}(U_j)) = \phi$. Consider now the subcase $|U_j| > 2$. A bound on the number of character comparisons performed by iterations in U_j is obtained by adding the bounds in Lemma 16, for each iteration $u < \text{so}(U_j)$, and the bound in Fact 10. This sum is bounded by $\text{line}_j - \text{line}_{j-1} + \lfloor(\text{so}(U_j) - \text{first}(U_j) + 2)/2\rfloor$, since $C(\text{first}(U_j)) = \phi$ and, in turn, by (18) since $z \geqq 2$ and $\lfloor(\text{so}(U_j) - \text{first}(U_j) + 2)/2\rfloor \leqq \lfloor(\text{so}(U_j) - \text{last}(U_{j-1}))((z'+2)/2m)\rfloor$ by Lemma 13 (with $u = \text{so}(U_j)$, $\alpha = \frac{1}{2}$ and $\beta = 2$). □

We now consider so (U_d) and last (U_d), when so (U_d) exists. We do not discuss the case in which so (U_d) does not exist, since SM′ simulates SM for all iterations in U_d and a bound on the number of character comparisons performed by SM′ can be obtained using Lemma 12.

When so (U_d) starts, algorithm SM′ tries to find the longest prefix of $t[\text{line}_{d-1} + 1, n]$ of length at least $l + 1 - \text{line}_{d-1} + \text{so}(U_d)$ that matches the regular expression $(p[1])^* p[l+1]$ (SM′ does not simulate SM during so (U_j)). Assume that the algorithm "scans" $t[\text{line}_{d-1}, k]$ during this search. It can be easily seen that the number of character comparisons is at most $k - \text{line}_{d-1} + 1$. If so (U_d) is the last iteration of the algorithm or in U_d (end of text or search not successful), $k = s_1$ and $c(s_1) = 0$, where s_1 is the next iteration (recall our conventions about the dummy iteration). Otherwise (search successful and not end of text), $k = \text{last}(U_d) + l + 1$. In all cases, text characters to the left of $k+1$ will not be considered again, since SM′ sets $t\text{last} = k$. So, we have Fact 11.

FACT 11. *Assume that* so (U_d) *exists. The number of character comparisons performed by Algorithm* SM′ *during* so (U_d) *is bounded by*

$$s_1 + c(s_1) - \text{line}_{d-1} + 1 \tag{20}$$

when so (U_d) *concludes either the sequence or the algorithm, and by*

$$\text{last}(U_d) + l - \text{line}_{d-1} + 2 \tag{21}$$

in all other cases.

Consider last (U_d), when so (U_d) exists. Algorithm SM′ behaves like Algorithm SM when matching pattern positions with the corresponding text positions in $[\text{last}(U_d) + l + 2, \text{last}(U_d) + m]$. We consider two cases: last (U_d) is not the last iteration of the algorithm and its logical complement. In any case, let q be the number of character comparisons that SM′ performs between $t[\text{last}(U_d) + l + 2, \text{line}_d]$ and the corresponding part of the pattern during last (U_d).

Assume that last (U_d) does not conclude the algorithm. Since it concludes the heavy sequence, the q comparisons result in $q-1$ matches and a mismatch (last (U_j) can match the pattern and conclude the heavy sequence only when $z' = 0$, but then the heavy sequence consists of one element only and so (U_j) could not exist). Then, the pattern is shifted over the text by at least $l + 1 \geqq 2$ positions (recall that last (U_d) shifts by at least $l+1$). We are interested in finding an upper bound on q. We consider two subcases: $s_1 - \text{last}(U_d) = m$ and $s_1 - \text{last}(U_d) < m$.

If $s_1 - \text{last}(U_d) = m$, we have $s_1 = \text{line}_d = \text{last}(U_d) + m$. Since no text character in $t[\text{line}_d, n]$ has been considered by the algorithm during iterations preceding s_1, we have that $c(s_1) = 0$, and q is obviously bounded by $s_1 + c(s_1) - (\text{last}(U_d) + l + 1)$.

Assume now that $s_1 - \text{last}(U_d) < m$. Let N be the set of text positions in $[\text{last}(U_d) + l + 2, \text{line}_d]$ that the algorithm has successfully matched with the corresponding pattern positions at the end of iteration $\text{last}(U_d)$. Thus, $|N| + 1 = q$. We partition N in three disjoint subsets as follows. N_{noholes} is the set of positions in N aligned with a nohole of the pattern at the beginning of iteration s_1. N_{dead} is the set of positions in N falling in the interval $[\text{last}(U_d) + l + 2, s_1]$. N_{holes} is the set of positions in N aligned with holes at the beginning of iteration s_1. We bound q by bounding $|N_{\text{holes}}| + |N_{\text{dead}}|$ and $|N_{\text{noholes}}|$.

Let $h = h_{\text{start}(\text{last}(U_d))+q-1}$. Consider $p[1, h]$ and let J be the set of positions j in that string such that $k\min(h) < j \leqq h, j$ is a nohole, and $j - k\min(h)$ is a hole. Since $s_1 - \text{last}(U_d) = k\min(h) \geqq l + 1$ ($\text{last}(U_d)$ shifts by at least $l + 1$) and $k\min(h) < j$, for each $j \in J$, each $j \in J$ has a match with exactly one text position in N_{holes} and vice versa at the end of iteration $\text{last}(U_d)$. Therefore $|N_{\text{holes}}| = |J|$. By Lemma 5 (with $k = k\min(h)$), each nohole in J is reachable from a distinct hole in $p[l + 2, k\min(h)]$. Each of the noholes in $p[l + 2, k\min(h)]$ has a match with exactly one text position in $|N_{\text{dead}}|$ at the end of iteration $\text{last}(U_d)$ (recall the definition of N_{dead}). Thus, $|N_{\text{holes}}| + |N_{\text{dead}}| \leqq k\min(h) - l - 1 = s_1 - \text{last}(U_d) - l - 1$. Obviously, $|N_{\text{noholes}}| = \text{start}(s_1) - 1 = c(s_1)$. Therefore, $q = |N| + 1 = |N_{\text{holes}}| + |N_{\text{dead}}| + |N_{\text{noholes}}| + 1 \leqq s_1 + c(s_1) - (\text{last}(U_d) + l + 1) + 1$.

If $\text{last}(U_d)$ is the last iteration of the algorithm, q is obviously bounded by $n - (\text{last}(U_d) + l + 1) \leqq s_1 + c(s_1) - (\text{last}(U_d) + l + 1)$, since $t\text{last} = \text{last}(U_d) + l + 1$ when U_d starts and no character in $[1, t\text{ last}]$ is ever considered again.

These observations can be summarized as follows.

FACT 12. *Assume that* $\text{so}(U_d)$ *exists. The number of character comparisons performed by the algorithm during iteration* $\text{last}(U_d)$ *is bounded by*

$$s_1 + c(s_1) - (\text{last}(U_d) + l + 1) \tag{22}$$

when $\text{last}(U_d)$ *concludes the algorithm or* $s_1 - \text{last}(U_d) = m$, *and by*

$$s_1 + c(s_1) - (\text{last}(U_d) + l + 1) + 1 \tag{23}$$

in all other cases.

LEMMA 18. *The number of character comparisons performed by Algorithm* SM′ *during iterations in* U_d *is bounded by*

$$s_1 + c(s_1) - \text{line}_{d-1} + \left\lfloor (\text{line}_{d-1} - \text{last}(U_{d-1})) \left(\frac{z' + 1}{2m} \right) \right\rfloor \tag{24}$$

when u_g *does not conclude the algorithm and when* $\text{so}(U_d)$ *does not exist. It is bounded by*

$$s_1 + c(s_1) - \text{line}_{d-1} + \left\lfloor (u_g - \text{last}(U_{d-1})) \left(\frac{z'}{2m} \right) \right\rfloor \tag{25}$$

when u_g *concludes the algorithm and when* $\text{so}(U_d)$ *does not exist. It is bounded by*

$$s_1 + c(s_1) - \text{line}_{d-1} + \left\lfloor (\text{line}_{d-1} - \text{last}(U_{d-1})) \left(\frac{z' + 2}{2m} \right) \right\rfloor \tag{26}$$

in all other cases.

Proof. We first consider the case in which u_g does not conclude the algorithm and so (U_d) does not exist. Algorithm SM′ simulates SM on iterations in U_d and a bound on the number of character comparisons performed by SM′ during iterations

in U_d is given by (9) in Lemma 12. We will obtain (24) from this bound by bounding credit*. Notice that the sequence $U_d \cup \{\text{line}_{d-1}\}$ is strictly increasing.

Notice that $\text{line}_{d-1} - \text{last}(U_d) + 1 \geq 2$ ($\text{line}_{d-1} - \text{last}(U_d) \geq 1$ by definition of heavy sequence and the case being considered). Moreover, for each u and u' in U_d, $\text{first}(U_d) \leq u < u' \leq \text{last}(U_d)$, $u' - u \geq 2$ since no iteration in U_d, except possibly the last one, has start $(u) = 1$ (shift by 1 in case of mismatch). Therefore, credit* $(U_d \cup \{\text{line}_{d-1}\}) \leq \lfloor(\text{line}_{d-1} - \text{first}(U_d) + 1)/2\rfloor$. By Lemma 13 (with $u = \text{line}_{d-1}$, $\alpha = \frac{1}{2}$, $\beta = 1$), credit* $(U_d \cup \{\text{line}_{d-1}\})$ is further bounded by $\lfloor(\text{line}_{d-1} - \text{last}(U_{d-1}))((z'+1)/2m)\rfloor$. Using (9), we obtain (24).

Consider the case in which u_g concludes the algorithm and so (U_d) does not exist. Algorithm SM′ simulates SM on iterations in U_d and a bound on the number of character comparisons performed by SM′ during iterations in U_d is given by (10) in Lemma 12. Employing the arguments used to bound credit*, yields credit $(U_d) \leq \lfloor(u_g - \text{last}(U_{d-1}))(z'/2m)\rfloor$. Using (10), we obtain (25).

Consider the remaining cases. That is, so (U_d) exists. Recall that so $(U_d) < \text{line}_{d-1} - 1$. Let $\hat{U} = \{x_1, \cdots, x_s\}$ be U_d up to so (U_d), i.e., $x_s < \text{so}(U_d)$. Notice that $x_i - x_{i-1} \geq 2$, $i > 1$, since no iteration in $\hat{U}$ has start $(x_i) = 1$. A bound on the number of character comparisons performed by SM′ during iterations in $\hat{U}$ is obtained by adding the bound in Lemma 16 for each $x_i \in \hat{U}$, $1 < i \leq s$. This sum is bounded by

$$\left\lfloor \frac{\text{so}(U_d) - \text{first}(U_d)}{2} \right\rfloor. \tag{27}$$

If so (U_d) is the last iteration of the algorithm or of U_d, the sum of (20) and (27) gives $s_1 + c(s_1) - \text{line}_{d-1} + 1 + \lfloor(\text{so}(U_d) - \text{first}(U_d))/2\rfloor \leq s_1 + c(s_1) - \text{line}_{d-1} + \lfloor(\text{line}_{d-1} - \text{first}((U_d)))/2\rfloor$, since so $(U_d) \leq \text{line}_{d-1} - 2$ by assumption. We derive the bound in (26) since $\lfloor(\text{line}_{d-1} - \text{first}(U_d))/2\rfloor \leq \lfloor(\text{line}_{d-1} - \text{last}(U_{d-1}))(z'/2m)\rfloor$ by Lemma 13, with $u = \text{line}_{d-1}$, $\alpha = \frac{1}{2}$ and $\beta = 0$.

If so (U_d) is neither the last iteration of the algorithm nor of U_d, the sum of (21), (23), and (27) gives $s_1 + c(s_1) - \text{line}_{d-1} + 2 + \lfloor(\text{so}(U_d) - \text{first}(U_d))/2\rfloor \leq s_1 + c(s_1) - \text{line}_{d-1} + \lfloor(\text{line}_{d-1} - \text{first}((U_d)) + 2)/2\rfloor$, since so $(U_d) \leq \text{line}_{d-1} - 2$ by assumption. We derive the bound in (26) since $\lfloor(\text{line}_{d-1} - \text{first}(U_{d-1}) + 2)/2\rfloor \leq \lfloor(\text{line}_{d-1} - \text{last}(U_{d-1}))((z'+2)/2m]\rfloor$ by Lemma 13, with $u = \text{line}_{d-1}$, $\alpha = \frac{1}{2}$ and $\beta = 2$, and $z \geq 2$. □

LEMMA 19. *Consider a heavy sequence $u_1, u_2, \cdots, u_g$ and let s_1 be the iteration succeeding it or the dummy iteration. The number of character comparisons performed by algorithm* SM′ *during the heavy sequence is bounded by*

$$s_1 + c(s_1) - u_1 - c(u_1) + \left\lfloor \frac{\text{line}_{d-1} - u_1}{l+1} \right\rfloor \tag{28}$$

when $p[1, m] = p[1]^l p[l+1]' p[1]^{l'}$, $l \geq 2$, and $p[1] \neq p[l+1]$. It is bounded by

$$s_1 + c(s_1) - u_1 - c(u_1) + \left\lfloor (\text{line}_{d-1} - u_1)\left(\frac{z'+1}{2m}\right) \right\rfloor \tag{29}$$

when the pattern does not satisfy $p[1] = p[2] = p[m-1] = p[m]$ and u_g does not conclude the algorithm. It is bounded by

$$s_1 + c(s_1) - u_1 - c(u_1) + \left\lfloor (u_g - u_1)\left(\frac{z'}{2m}\right) \right\rfloor \tag{30}$$

when the pattern does not satisfy $p[1]=p[2]=p[m-1]=p[m]$ *and* u_g *concludes the algorithm. It is bounded by*

$$s_1+c(s_1)-u_1-c(u_1)+\left\lfloor (\text{line}_{d-1}-u_1)\left(\frac{z'+2}{2m}\right)\right\rfloor \tag{31}$$

in all the other cases. Moreover, s_1 *satisfies Invariant* 1 *when it starts.*

Proof. We now consider four cases, each corresponding to one of the bounds.

Case $p[1, m]=p[1]^l p[l+1]p[1]^l$, $l\geq 2$, and $p[1]\neq p[l+1]$. The contribution of each U_j, $1<j<d$ to the total number of character comparisons depends on how many iterations it contains. We claim that $|U_j|\leq 2$, $1<j<d$. Indeed, last (U_{j-1}) matches a suffix of the pattern and shifts the pattern by at least $z=l+1$ setting $b=\text{first}(U_j)$ and $t\text{last}=\text{line}_{j-1}=\text{last}(U_{j-1})+m$. Thus, $\text{line}_{j-1}-\text{first}(U_j)\leq l$. Since the first nohole in the pattern is $l+1$, it is aligned with a text position past line_{j-1} when first (U_j) starts. Therefore, start (first (U_j)) $=e=1$. If $t\text{last}-b=\text{line}_{j-1}-\text{first}(U_j)=1$, then SM′ simulates SM during first (U_j) and this iteration must match a suffix of the pattern; otherwise it would conclude the heavy sequence. Thus $|U_j|=1$ when $t\text{last}-b=\text{line}_{j-1}-\text{first}(U_j)=1$ and the number of character comparisons is bounded by $\text{line}_j-\text{line}_{j-1}$. If $t\text{last}-b=\text{line}_{j-1}-\text{first}(U_j)>1$, then SM′ does not simulate SM during first (U_j) as this iteration looks for the longest prefix of $t[\text{line}_{j-1}+1, n]$ that matches the regular expression $(p[1])^*p[l+1]$. It must find it, otherwise it would conclude the sequence. Now, second (U_j) must match a suffix of the pattern. Thus $|U_j|=2$ when $t\text{last}-b=\text{line}_{j-1}-\text{first}(U_j)>1$ and by Lemma 17, the number of character comparisons for $|U_j|=2$ is bounded by (19), i.e., $\text{line}_j-\text{line}_{j-1}+1$.

Since $|U_1|=1$, the sum of character comparisons performed during iterations in $U_1, \cdots, U_{d-1}$ is bounded by $\text{line}_{d-1}-u_1+(d-2)$.

Consider U_d. Notice that $|U_d|\leq 2$ and that first (U_d) must have start (first (U_d)) $=e=1$. The proof is the same as the one showing $|U_j|\leq 2$ and that first (U_j) must have start (first (U_j)) $=e=1$, $1<j<d$. We claim that the number of character comparisons for iterations in U_d is bounded by $s_1+c(s_1)-\text{line}_{d-1}+1$. This follows from Fact 11, bound (20), when $|U_d|=1$ and so (U) exists.

When $|U_d|=1$ and so (U_d) does not exist, SM′ simulates SM during first (U_d). This implies that first $(U_d)=b=t\text{last}-1=\text{line}_{d-1}-1$. Notice that, for the specific pattern we are considering, if first (U_d) matches a suffix of the pattern the next iteration must have a suffix-prefix overlap with first (U_d) and the heavy sequence would continue. Thus, first (U_d) cannot match any suffix of the pattern. For the specific pattern we are considering, $h_1=l$ and $h_2=m-1$ in the sequence $h_1, \cdots, h_m$. So, the mismatch must either be with $p[l+1]$ or with $p[2l+1]$. In either case, the shift is at least as large as the number of character comparisons performed during first (U_d) and $c(s_1)=0$. So, we obtain a bound of $s_1-\text{first}(U_d)=s_1-\text{line}_{d-1}+1=s_1+c(s_1)-\text{line}_{d-1}+1$. When $|U_d|=2$, so (U_d) must exist, otherwise Lemma 15 would be contradicted. The claimed bound is obtained as the sum of (21) and (22), since $s_1-\text{last}(U_d)=m$ (otherwise last (U_d) would completely match a suffix of the pattern).

Therefore, the number of character comparisons performed during this heavy sequence is bounded by $s_1+c(s_1)-u_1-c(u_1)+(d-1)$, where $d-1$ is the number of iterations in the heavy sequence that completely matched a suffix of the pattern (for $c(u_1)=0$). Since after such match the pattern is shifted by at least $l+1$ positions, $d-1\leq \lfloor(\text{line}_{d-1}-u_1)/l+1\rfloor$ and we obtain (28).

Case. The pattern does not satisfy $p[1]=p[2]=p[m-1]=p[m]$ and u_g does not conclude (respectively, concludes) the algorithm. In this case, so (U_j), $1<j\leq d$, cannot exist; otherwise Fact 8 would be contradicted. Thus, bound (17) holds for each

U_j, $1<j<d$, and (24) (respectively, (25)) holds for U_d. The total number of comparisons is obtained by adding these bounds to $m-c(u_1)$ (the contribution of u_1). The sum of the terms $m-c(u_1)$, $\text{line}_j-\text{line}_{j-1}$ and $s_1+c(s_1)-\text{line}_{d-1}$ is bounded by $s_1+c(s_1)-u_1-c(u_1)$. The sum of the terms $\lfloor(\text{last}(U_j)-\text{last}(U_{j-1}))(z'/2m)\rfloor$ and $\lfloor(\text{line}_{d-1}-\text{last}(U_{d-1}))((z'+1)/2m)\rfloor$ (respectively, $\lfloor(u_g-\text{last}(U_{d-1}))(z'/2m)\rfloor$) is bounded by $\lfloor(\text{last}(U_j)-u_1)((z'+1)/2m)\rfloor$ (respectively, $\lfloor(u_g-u_1)(z'/2m)\rfloor$). Therefore, (29) (respectively, (30)) holds.

Remaining cases. For each U_j, $1<j<d$, one of (17), (18), or (19) applies. Bound (19) is never larger than (18), since $\text{so}(U_j)-\text{last}(U_{j-1})\geqq z(\text{last}(U_{j-1})$ shifted the pattern by at least z since it matched a suffix of the pattern and the sequence of iterations from $\text{last}(U_{j-1})$ to $\text{so}(U_j)$ is increasing) and since $m=z+z'$, $z'<z$. Since $\text{so}(U_j)\leqq\text{last}(U_j)<\text{last}(U_{j+1})$, $1<j<d$, the sum of bounds (17) or (18) is bounded by $\text{line}_{d-1}-\text{line}_1+\lfloor(\text{last}(U_{d-1})-u_1)((z'+2)/2m)\rfloor$. Adding to this bound $m-c(u_1)$ and (26), we obtain (31) (since the bound of (26) is at least as large as the bounds of (24) and (25)).

We now show that iteration s_1 satisfies invariant 1 when it starts. Indeed, since u_1 satisfies Invariant 1, text positions in $[u_1+1, n]$ are either free or match all the noholes in $p[1, h_{\text{start}(u_1)}]$. Thus, text positions in $t[u_1+1, s_1]$ can become busy at the end of the heavy sequence. This amounts to s_1-u_1 text positions being declared busy. As for positions in $t[s_1+1, n]$, notice they are either free (they were free when u_1 started) or they match noholes in $p[1, h_{\text{start}(s_1)}]$. Therefore s_1 satisfies Invariant 1 when it starts.

Notice that the positions declared busy at the end of s_1 can be used to pay for s_1-u_1 character comparisons. This "covers" $s_1-u_1-c(u_1)$ comparisons performed by the algorithm during the heavy sequence as well as the $c(u_1)$ character comparisons (all matches) that u_1 has inherited from the previous iteration. The $c(s_1)$ matches inherited from u_g by s_1 are left unpaid. The remaining comparisons are paid by a credit line. □

LEMMA 20. *Consider a heavy sequence $u_1, u_2, \cdots, u_g$ and let s_1 be the iteration succeeding it. The number of character comparisons performed by Algorithm* SM′ *during the heavy sequence is bounded by*

$$s_1+c(s_1)-u_1-c(u_1)+\left\lfloor(u_g-u_1)\left(\min\left(\frac{1}{3},\frac{z'+2}{2m}\right)\right)\right\rfloor \tag{32}$$

when the pattern does not satisfy $p[1]=p[2]=p[m-1]=p[m]$ and u_g concludes the algorithm. It is bounded by

$$s_1+c(s_1)-u_1-c(u_1)+\left\lfloor(\text{line}_{d-1}-u_1)\left(\min\left(\frac{1}{3},\frac{z'+2}{2m}\right)\right)\right\rfloor \tag{33}$$

in all other cases.

Proof. Notice that (30) is obviously bounded by (32) since $z'/2m\leqq\frac{1}{4}$. We now prove (33). Let $\sum$ denote the alphabet. Let $\sum^{z+z'}$ denote the set of strings of length $m=z+z'$, where z is the period of the string and $z'<z$. Notice that (28), (29), and (31) in Lemma 19 partition $\sum^{z+z'}$ into three disjoint sets $\mathcal{A}=\{p[1]^l p[l+1]p[1]^{l'}$: $l\geqq 2$ and $p[1]\neq p[l+1]\}$; $\mathcal{B}=\{p[1,m]$: not $(p[1]=p[m-1]=p[1]=p[m])\}$; $\mathcal{C}=\sum^{z+z'}-(\mathcal{A}\cup\mathcal{B})$. For each of the strings in these sets we prove that each of the corresponding bounds is bounded by (33).

Consider $\mathcal{A}$. Since $l\geqq 2$, $m=2l+1$ and $z'=l$ for this set of strings, (28) is bounded by (33).

Consider $\mathcal{B}$. We notice that $(z'+1)/2m \leqq (z'+2)/2m$. Therefore, in order to obtain (33) from (29), we need to show that $(z'+1)/2m \leqq \frac{1}{3}$. Such inequality follows easily on recalling that $m = z+z'$, $z' < z$ and $z \geqq 2$ for strings in $\mathcal{B}$.

Consider $\mathcal{C}$. In order to obtain (33) from (31), we first observe that for any string in $\mathcal{C}$, $m \geqq 6$ since, for $p \in \mathcal{C}$, $p[1] = p[2] = p[m-1] = p[m]$ ($p \notin \mathcal{B}$) and $p \neq p[1]^l p[l+1] p[l]^l$ ($p \notin \mathcal{A}$). (Recall the definition of $\mathcal{C}$.) We show that $(z'+2)/2m \leqq \frac{1}{3}$ for all of the strings in $\mathcal{C}$. Indeed, recalling that $m = z+z'$, $z' < z$, this inequality holds for any string in $\mathcal{C}$ having $z \geqq 5$. So assume $z < 5$. Since $p[1] = p[2] = p[m-1] = p[m]$, we have $z' \geqq 2$ and furthermore $z' = 3$ is not possible (for $z' = 3$, $z = 4$, $p = p[1]^3 p[4] p[1]^3 \in \mathcal{A}$). Thus, $z' = 2$ and since $m \geqq 6$, $(z'+2)/2m \leqq \frac{1}{3}$. □

6.2. Putting the sequences together again. We can finally prove the following theorem.

THEOREM 3. *Let $p[1, m]$ be a pattern with period z, $m = z+z'$ and $z' < z$, and let $t[1, n]$ be the text. Algorithm* SM′ *finds all occurrences of p in t in $O(n+m)$ time. In the worst case, the algorithm performs $n + \lfloor (n-m)(\min(\frac{1}{3}, (z'+2)/2m)) \rfloor$ character comparisons.*

Proof. The $O(n+m)$ time bound is obvious. As for the number of character comparisons, we notice that iteration zero satisfies Invariant 1 trivially and, by Lemmas 8 and 19, each iteration starting either a light or a heavy sequence satisfies the same invariant and the bounds in Lemmas 8 and 20 hold. The bound of the theorem follows by adding up these bounds: The sum of the terms $u_1 - s_1 + c(u_1) - c(s_1)$ or $s_1 - u_1 + c(s_1) - c(u_1)$ is bounded by n, and the sum of the terms $\lfloor (\text{line}_{d-1} - u_1)(\min(\frac{1}{3}, (z'+2)/2m)) \rfloor$ or $\lfloor (u_g - u_1)(\min(\frac{1}{3}, (z'+2)/2m)) \rfloor$ is bounded by $\lfloor (n-m)(\min(\frac{1}{3}, (z'+2)/2m)) \rfloor$. Since $c(u) = 0$ for u the dummy iteration, no comparison is left unpaid. □

The bound in Theorem 3 is tight for $m = 3$ and any $n = 3c$, c any integer. Indeed, let $p[1, m] = aba$ and $t[1, n] = (aba)^c$. By Fact 8 and the definition of a light sequence, there is no iteration u of SM′ such that u handles a shift by one and $u < t\text{last}(u) - 1$. Therefore, SM′ simulates SM when searching for aba in any text. In the previous section we have shown that SM performs $n + \lfloor (n-m)/3 \rfloor$ character comparisons when it searches for all occurrences of aba in $t[1, n] = (aba)^c$. Since for the chosen pattern $z' = 1$ and $m = 3$, this bound is equal to the one in Theorem 3.

7. Concluding remarks. We have shown that, for any pattern of length m and any text of length n, $c(n, m) \leqq c_{\text{on-line}}(n, m) \leqq n + \lfloor (n-m)(\min(\frac{1}{3}, (z_s'+2)/2(z_s + z_s'))) \rfloor \leqq \frac{4}{3}n - \frac{1}{3}m$ character comparisons, where (z_s, z_s', k_s) is the last term of the periodic decomposition of the pattern defined in the Introduction. In a companion paper [11], we show a lower bound for on-line algorithms that is equal to $\frac{4}{3}n - \frac{1}{3}m$ for $m = 3$. Our algorithm is based on a new analysis of the string matching algorithm by Colussi [7]. Moreover, our analysis of Colussi's algorithm confirms the experimental results showing that it performs very well in practice.

Acknowledgments. We praise the endurance of the referee, who survived this tour de force in analysis of algorithms with enough energy left to give us very helpful comments.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] A. APOSTOLICO AND M. CROCHEMORE, *Optimal canonization of all substrings of a string*, Tech. Report, TR 89-75, LITP, Université de Paris 7, Paris, France, October, 1989.

[3] A. APOSTOLICO AND R. GIANCARLO, *The Boyer-Moore-Galil string searching strategies revisited.* SIAM J. Comput., 15 (1986), pp. 98-105.
[4] S. W. BENT AND J. W. JOHN, *Finding the median requires 2n comparisons*, in Proc. 17th Symposium on Theory of Computing, Association for Computing Machinery, 1985, pp. 213-216.
[5] R. S. BOYER AND J. S. MOORE, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 762-772.
[6] R. COLE, *Tight bounds on the complexity of the Boyer-Moore string matching algorithm*, in Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991.
[7] L. COLUSSI, *Correctness and efficiency of string matching algorithms*, Inform. and Comput., to appear.
[8] M. CROCHEMORE AND D. PERRIN, *Two way pattern matching*, Tech. Report, Dept. of Computer Science, Université de Paris, Paris, France, 1989.
[9] L. R. FORD AND S. M. JOHNSON, *A tournament problem*, Amer. Math. Monthly, 66 (1959), pp. 387-389.
[10] Z. GALIL, *On improving the worst case running time of the Boyer-Moore string matching algorithm*, Comm. ACM, 22 (1979), pp. 505-508.
[11] Z. GALIL AND R. GIANCARLO, *On the exact complexity of string matching: Lower bounds*, SIAM J. Comput., 20 (1991), pp. 1008-1020.
[12] R. GIANCARLO, *Efficient Algorithms on Strings*, Ph.D. thesis, Dept. of Computer Science, Columbia University, New York, 1990.
[13] L. J. GUIBAS AND A. M. ODLYZKO, *A new proof of the linearity of the Boyer-Moore string matching algorithm*, SIAM J. Comput., 9 (1980), pp. 672-682.
[14] D. G. KIRKPATRICK, *Topics in the complexity of combinatorial algorithms*, Tech. Report, Dept. of Computer Science, University of Toronto, Toronto, Canada, 1974.
[15] ———, *A unified lower bound for selection and set partitioning problems*, J. ACM, 28 (1981), pp. 150-165.
[16] S. S. KISLITSYN, *On the selection of the kth element of an ordered set by pairwise comparison*, Sibirsk. Mat. Zh., 5 (1964), pp. 557-564.
[17] D. E. KNUTH, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
[18] D. E. KNUTH, J. H. MORRIS, AND V. B. PRATT, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 189-195.
[19] R. C. LYNDON AND M. P. SCHUTZENBERGER, *The equation $a^m = b^n c^p$ in a free group*, Michigan Math. J., 9 (1962), pp. 289-298.
[20] I. POHL, *A sorting problem and its complexity*, Comm. ACM, 15 (1972), pp. 462-464.
[21] A. SCHONHAGE, M. PATERSON, AND N. PIPPENGER, *Finding the median*, J. Comput. System Sci., 13 (1976), pp. 184-199.
[22] J. SCHREIER, *On tournament elimination systems*, Mathesis Polska, 7 (1932), pp. 154-160.
[23] C. K. YAP, *New upper bounds for selection*, Comm. ACM, 19 (1979), pp. 501-508.