Substring Range Reporting[*]

Philip Bille
phbi@imm.dtu.dk

Inge Li Gørtz
ilg@imm.dtu.dk

August 19, 2011

Abstract

We revisit various string indexing problems with range reporting features, namely, position-restricted substring searching, indexing substrings with gaps, and indexing substrings with intervals. We obtain the following main results.

- We give efficient reductions for each of the above problems to a new problem, which we call *substring range reporting*. Hence, we unify the previous work by showing that we may restrict our attention to a single problem rather than studying each of the above problems individually.
- We show how to solve substring range reporting with optimal query time and little space. Combined with our reductions this leads to significantly improved time-space trade-offs for the above problems. In particular, for each problem we obtain the first solutions with optimal time query and $O(n \log^{O(1)} n)$ space, where n is the length of the indexed string.
- We show that our techniques for substring range reporting generalize to *substring range counting* and *substring range emptiness* variants. We also obtain non-trivial time-space trade-offs for these problems.

Our bounds for substring range reporting are based on a novel combination of suffix trees and range reporting data structures. The reductions are simple and general and may apply to other combinations of string indexing with range reporting.

1 Introduction

Given a string S of length n the *string indexing problem* is to preprocess S into a compact representation that efficiently supports *substring queries*, that is, given another string P of length m report all occurrences of substrings in S that match P. Combining the classic suffix tree data structure [14] with perfect hashing [13] leads to an optimal time-space trade-off for string indexing, i.e., an $O(n)$ space representation that supports queries in $O(m + \mathrm{occ})$ time, where occ is the number of occurrences of P in S.

In recent years, several extensions of string indexing problems that add *range reporting* features have been proposed. For instance, Mäkinen and Navarro proposed the *position-restricted substring searching problem* [21, 22]. Here, queries take an additional range $[a, b]$ of positions in S and the goal is to report the occurrences of P within $S[a, b]$. For such extensions of string indexing no optimal time-space trade-off is known. For instance, for position-restricted substring searching one

[*]An extended abstract of this paper appeared at the 22nd Conference on Combinatorial Pattern Matching.

can either get $O(n \log^{\varepsilon} n)$ space (for any constant $\varepsilon > 0$) and $O(m + \log\log n + \mathrm{occ})$ query time or $O(n^{1+\varepsilon})$ space with $O(m + \mathrm{occ})$ query time [8, 21, 22]. Hence, removing the $\log\log n$ term in the query comes at the cost of significantly increasing the space.

In this paper, we revisit a number string indexing problems with range reporting features, namely *position-restricted substring searching*, *indexing substrings with gaps*, and *indexing substrings with intervals*. We achieve the following results.

- We give efficient reductions for each of the above problems to a new problem, which we call *substring range reporting*. Hence, we unify the previous work by showing that we may restrict our attention to a single problem rather than studying each of the above problems individually.

- We show how to solve substring range reporting with optimal query time and little space. Combined with our reductions this leads to significantly improved time-space trade-offs for all of the above problems. For instance, we show how to solve position-restricted substring searching in $O(n \log^{\varepsilon} n)$ space and $O(m + \mathrm{occ})$ query time.

- We show that our techniques for substring range reporting generalize to *substring range counting* and *substring range emptiness* variants. We also obtain non-trivial time-space trade-offs for these problems.

Our bounds for substring range reporting are based on a novel combination of suffix trees and range reporting data structures. The reductions are simple and general and may apply to other combinations of string indexing with range reporting.

1.1 Substring Range Reporting

Let S be a string where each position is associated with a integer value in the range $[0, u]$. The integer associated with position i in S is the *label* of position i, denoted label(i), and we call S a *labeled string*. Given a labeled string S, the *substring range reporting problem* is to compactly represent S while supporting *substring range reporting queries*, that is, given a string P and a pair of integers a and b, $0 \leq a \leq b \leq u$, report all starting positions in S that match P and whose labels are in the range $[a, b]$.

We assume a standard unit-cost RAM model with word size w and a standard instruction set including arithmetic operations, bitwise boolean operations, and shifts. We assume that a label can be stored in a constant number of words and therefore $w = \Theta(\log u)$. The space complexity is the number of words used by the algorithm. All bounds mentioned in this paper are valid in this model of computation.

To solve substring range reporting a basic approach is to combine a suffix tree with a 2D range reporting data structure. A query for a pattern P and range $[a, b]$ consists of a search in the suffix tree and then a 2D range reporting query with $[a, b]$ and the lexicographic range of suffixes defined P. This is essentially the overall approach used in the known solutions for position-restricted substring searching [4, 8, 9, 21, 22, 31], which is a special case of substring range reporting (see the next section).

Depending on the choice of the 2D range reporting data structure this approach leads to different trade-offs. In particular, if we plug in the 2D range reporting data structure of Alstrup et al. [2], we get a solution with $O(n \log^{\varepsilon} n)$ space and $O(m + \log\log u + \mathrm{occ})$ query time (see Mäkinen and Navarro [21, 22]). The $\log\log u$ term in the query time is from the range reporting query. Alternatively, if we use a fast data structure for the range successor problem [8, 31] to do the range reporting, we get optimal $O(m + \text{occ})$ query time but increase the space to at least $\Omega(n^{1+\varepsilon})$. Indeed, since any 2D range reporting data structure with $O(n \log^{O(1)} n)$ space must use $\Omega(\log\log u)$ query time [26], we cannot hope to avoid this blowup in space with this approach.

Our first main contribution is a new and simple technique that overcomes the inherent problem of the previous approach. We show the following result.

Theorem 1 *Let S be a labeled string of length n with labels in the range $[0, u]$. For any constants $\varepsilon, \delta > 0$, we can solve substring range reporting using $O(n(\log^{\varepsilon} n + \log\log u))$ space, $O(n(\log n + \log^{\delta} u))$ expected preprocessing time, and $O(m + \text{occ})$ query time, for a pattern string of length m.*

Compared to the previous results we achieve optimal query time with an additional $O(n \log\log u)$ term in the space. For the applications considered here, we have that $u = O(n)$ and therefore the space bound simplifies to $O(n(\log^{\varepsilon} n + \log\log u)) = O(n \log^{\varepsilon} n)$. Hence, in this case there is no asymptotic space overhead.

The key idea to obtain Theorem 1 is a new and simple combination of suffix trees with multiple range reporting data structures for both 1 and 2 dimensions. Our solution handles queries differently depending on the length of the input pattern such that the overall query is optimized accordingly.

Interestingly, the idea of using different query algorithms depending on the length of the pattern is closely related to the concept of *filtering search* introduced for the standard range reporting problem by Chazelle as early as 1986 [6]. Our new results show that this idea is also useful in combinatorial pattern matching.

Finally, we also consider *substring range counting* and *substring range emptiness* variants. Here, the goal is to count the number of occurrences in the range and to determine whether or not the range is empty, respectively. Similar to substring range reporting, these problems can also be solved in a straightforward way by combining a suffix with a 2D range counting or emptiness data structure. We show how to extend our techniques to obtain improved time-space trade-offs for both of these problems.

1.2 Applications

Our second main contribution is to show that substring range reporting actually captures several other string indexing problems. In particular, we show how to reduce the following problems to substring range reporting.

- *Position-restricted substring searching:* Given a string S of length n, construct a data structure supporting the following query: Given a string P and query interval $[a, b]$, with $1 \leq a \leq b \leq n$, return the positions of substrings in S matching P whose positions are in the interval $[a, b]$.

- *Indexing substrings with intervals:* Given a string S of length n, and a set of intervals $\pi = \{[s_1, f_1], [s_2, f_2], \ldots, [s_{|\pi|}, f_{|\pi|}]\}$ such that $s_i, f_i \in [1, n]$ and $s_i \leq f_i$, for all $1 \leq i \leq |\pi|$, construct a data structure supporting the following query: Given a string P and query interval $[a, b]$, with $1 \leq a \leq b \leq n$, return the positions of substrings in S matching P whose positions are in $[a, b]$ *and* in one of the intervals in π.

- *Indexing substrings with gaps:* Given a string S of length n and an integer d, the problem is to construct a data structure supporting the following query: Given two strings P_1 and P_2 return all positions of substrings in S matching $P_1 \circ \star^d \circ P_2$. Here $\circ$ denotes concatenation and $\star$ is a wildcard matching all characters in the alphabet.

Previous results Let m be the length of P. Mäkinen and Navarro [21, 22] introduced the position-restricted substring searching problem. Their fastest solution uses $O(n \log^{\varepsilon} n)$ space, $O(n \log n)$ expected preprocessing time, and $O(m + \log\log n + \mathrm{occ})$ query time. Crochemore et al. [8] proposed another solution using $O(n^{1+\varepsilon})$ space, $O(n^{1+\varepsilon})$ preprocessing time, and $O(m + \mathrm{occ})$ query time (see also Section 1.1). Using techniques from range non-overlapping indexing [7] it is possible to improve these bounds for small alphabet sizes [27]. Several succinct versions of the problem have also been proposed [4, 21, 22, 31]. All of these have significantly worse query time since they require superconstant time per reported occurrence. Finally, Crochemore et al. [10] studied a restricted version of the problem with $a = 1$ or $b = n$.

For the indexing substrings with intervals problem, Crochemore et al. [8, 9] gave a solution with $O(n \log^2 n)$ space, $O(|\pi| + n \log^3 n)$ expected preprocessing time, and $O(m + \log\log n + \mathrm{occ})$ query time. They also showed how to achieve $O(n^{1+\varepsilon})$ space, $O(n^{1+\varepsilon} + |\pi|)$ preprocessing time, and $O(m + \mathrm{occ})$ query time. Several papers [3, 17, 20] have studied the property matching problem, which is similar to the indexing substrings with intervals problem, but where both start and end point of the match must be in the same interval.

Iliopoulos and Rahman [18] studied the problem of indexing substrings with gaps. They gave a solution using $O(n \log^{\varepsilon} n)$ space, $O(n \log n)$ expected preprocessing time, and $O(m + \mathrm{loglog}\, n + \mathrm{occ})$ query time, where m is the length of the two input strings. Crochemore and Tischler recently proposed a variant of the problem [11].

Our results We reduce position-restricted substring searching, indexing substrings with intervals, and indexing substrings with gaps to substring range reporting. Applying Theorem 1 with our new reductions, we get the following result.

Theorem 2 *Let S be a string of length n and let m be the length of the query. For any constant $\varepsilon > 0$, we can solve*

(i) Position-restricted substring searching using $O(n \log^{\varepsilon} n)$ space, $O(n \log n)$ expected preprocessing time, and $O(m + \mathrm{occ})$ query time.

(ii) Indexing substrings with intervals using $O(n \log^{\varepsilon} n)$ space, $O(|\pi| + n \log n)$ expected preprocessing time, and $O(m + \mathrm{occ})$ query time.

(iii) Indexing substrings with gaps using $O(n \log^{\varepsilon} n)$ space, $O(n \log n)$ expected preprocessing time, and $O(m + \mathrm{occ})$ query time (m is the size of the two input strings).

This improves the best known time-space trade-offs for all three problems, that all suffer from the trade-off inherent in 2D range reporting.

The reductions are simple and general and may apply to other combinations of string indexing with range reporting.

2 Basic Concepts

2.1 Strings and Suffix Trees

Throughout the section we will let S be a labeled string of length $|S| = n$ with labels in $[0, u]$. We denote the character at position i by $S[i]$ and the substrings from position i to j by $S[i, j]$. The substrings $S[1, j]$ and $S[i, n]$ are the *prefixes* and *suffixes* of S, respectively. The *reverse* of S is S^R. We denote the label of position i by $\mathrm{label}_S(i)$. The *order* of suffix $S[i, n]$, denoted $\mathrm{order}_S(i)$, is the lexicographic order of $S[i, n]$ among the suffixes of S.

The *suffix tree* for S, denoted T_S, is the compacted trie storing all suffixes of S [14]. The *depth* of a node v in T_S is the number of edges on the path from v to the root. Each of the edges in T_S is associated with some substring of S. The children of each node are sorted from left to right in increasing alphabetic order of the first character of the substring associated with the edge leading to them. The concatenation of substrings from the root to v is denoted $\mathrm{str}_S(v)$. The *string depth* of v, denoted $\mathrm{strdepth}_S(v)$, is the length of $\mathrm{str}_S(v)$. The *locus* of a string P, denoted $\mathrm{locus}_S(P)$, is the minimum depth node v such that P is a prefix of $\mathrm{str}_S(v)$. If P is not a prefix of a substring in S we define $\mathrm{locus}_S(P)$ to be $\perp$.

Each leaf ℓ in T_S uniquely corresponds to a suffix in S, namely, the suffix $\mathrm{str}_S(\ell)$. Hence, we will use $\mathrm{label}_S(\ell)$ and $\mathrm{order}_S(\ell)$ to refer to the label and order of the corresponding suffix. For an internal node v we extend the notation such that

$$\mathrm{label}_S(v) = \{\mathrm{label}_S(\ell) \mid \ell \text{ is a descendant leaf of v}\}$$
$$\mathrm{order}_S(v) = \{\mathrm{order}_S(\ell) \mid \ell \text{ is a descendant leaf of v}\}.$$

Since children of a node are sorted, the left to right order of the leaves in T_S corresponds to the lexicographic order of the suffixes of S. Hence, for any node v, $\mathrm{order}_S(v)$ is an interval. We denote the left and right endpoints of this interval by l_v and r_v. When the underlying string S is clear from the context we will often drop the subscript $_S$ for brevity.

The suffix tree for S uses $O(n)$ space and can be constructed in $O(\mathrm{sort}(n))$ time, where $\mathrm{sort}(n)$ is the time for sorting n values in the model of computation [12]. We only need a standard comparison-based $O(n \log n)$ suffix tree construction in our results. Let P be a string of length m. If $\mathrm{locus}_S(P) = \perp$ then P does not occur as a substring in S. Otherwise, the substrings in S that match P are the suffixes in $\mathrm{order}_S(\mathrm{locus}_S(P))$. Hence, we can compute all occurrences of P in S by traversing the suffix tree from the root to $\mathrm{locus}_S(P)$ and then report all suffixes stored in the subtree. Using perfect hashing [13] to represent the outgoing edges of each node in T_S we achieve an $O(n)$ solution to string indexing that supports queries in $O(m + \mathrm{occ})$ time (here occ is the total number of occurrences of P in S).

2.2 Range Reporting

Let $X \subseteq \{0, \ldots, u\}^d$ be a set of points in a d-dimensional grid. The *range reporting problem* in d-dimensions is to compactly represent X while supporting *range reporting queries*, that is, given a rectangle $R = [a_1, b_1] \times \cdots \times [a_d, b_d]$ report all points in the set $R \cap X$. We use the following results for range reporting in 1 and 2 dimensions.

Lemma 1 (Alstrup et al. [1], Mortensen et al. [24]) *For a set of n points in $[0, u]$ and any constant $\gamma > 0$, we can solve 1D range reporting using $O(n)$ space, $O(n \log^\gamma u)$ expected preprocessing time and $O(1 + \mathrm{occ})$ query time.*

Lemma 2 (Alstrup et al. [2]) *For a set of n points in $[0,u] \times [0,u]$ and any constant $\varepsilon > 0$, we can solve 2D range reporting using $O(n \log^{\varepsilon} n)$ space, $O(n \log n)$ expected preprocessing time, and $O(\log\log u + \mathrm{occ})$ query time.*

3 Substring Range Reporting

We now show Theorem 1. Recall that S is a labeled string of length n with labels from $[0,u]$.

3.1 The Data Structure

Our substring range reporting data structure consists of the following components.

- The suffix tree T_S for S. For each node v in T_S we also store l_v and r_v. We partition T_S into a *top tree* and a number of *bottom trees*. The top tree consists of all nodes in T_S whose string depth is at most $\log\log u$ and all their children. The trees induced by the remaining nodes are the forest of bottom trees.
- A 2D range reporting data structure on the set of points $\{(\mathrm{order}_S(i), \mathrm{label}_S(i)) \mid i \in \{1,\ldots,n\}\}$.
- For each node v in the top tree, a 1D range reporting data structure on the set $\{\mathrm{label}_S(i) \mid i \in \mathrm{order}_S(v)\}$.

We analyze the space and preprocessing time for the data structure. We use the range reporting data structures from Lemmas 1 and 2. The space for the suffix tree is $O(n)$ and the space for the 2D range reporting data structure is $O(n \log^{\varepsilon} n)$, for any constant $\varepsilon > 0$. We bound the space for the (potentially $\Omega(n)$) 1D range reporting data structures stored for the top tree. Let V_d be the set of nodes in the top tree with depth d. Since the sets $\mathrm{order}_S(v)$, $v \in V_d$, partition the set of descendant leaves of nodes in V_d, the total size of these sets is as most n. Hence, the total size of the 1D range reporting data structures for the nodes in V_d is therefore $O(n)$. Since there are at most $\log\log u + 1$ levels in the top tree, the space for all 1D range reporting data structures is $O(n \log\log u)$. Hence, the total space for the data structure is $O(n(\log^{\varepsilon} n + \log\log u))$.

We can construct the suffix tree in $O(\mathrm{sort}(n))$ time and the 2D range reporting data structure in $O(n \log n)$ expected time. For any constant $\gamma > 0$, the expected preprocessing time for all 1D range reporting data structures is

$$O\left(\sum_{v \text{ in top tree}} |\mathrm{order}_S(v)| \log^{\gamma} u\right) = O(n \log\log u \log^{\gamma} u) = O(n \log^{2\gamma} u).$$

Setting $\delta = 2\gamma$ we use $O(n(\log n + \log^{\delta} u))$ expected preprocessing time in total.

3.2 Substring Range Queries

Let P be a string of length m, and let a and b be a pair of integers, $0 \le a \le b \le u$. To answer a substring range query we want to compute the set of starting positions for P whose labels are in $[a,b]$. First, we compute the node $v = \mathrm{locus}_S(P)$. If $v = \perp$ then P is not a substring of S, and we return the empty set. Otherwise, we compute the set of descendant leaves of v with labels in $[a,b]$. There are two cases to consider.

(i) If v is in the top tree we query the 1D range reporting data structure for v with the interval $[a, b]$.

(ii) If v is in a bottom tree, we query the 2D range reporting data with the rectangle $[l_v, r_v] \times [a, b]$.

Given the points returned by the range reporting data structures, we output the corresponding starting positions of the corresponding suffixes. From the definition of the data structure it follows that these are precisely the occurrences of P within the range $[a, b]$. Next consider the time complexity. We find $\text{locus}_S(P)$ in $O(m)$ time (see Section 2). In case (i) we use $O(1 + \text{occ})$ time to compute the result by Lemma 1. Hence, the total time for a substring range query for case (i) is $O(m + \text{occ})$. In case (ii) we use $O(\log\log u + \text{occ})$ time to compute the result by Lemma 2. We have that $v = \text{locus}_S(P)$ is in a bottom tree and therefore $m \geq \text{strdepth}(\text{parent}(\text{locus}_S(v))) > \log\log u$. Hence, the total time to answer a substring range query in case (ii) is $O(m + \log\log u + \text{occ}) = O(m + \text{occ})$. Thus, in both cases we use $O(m + \text{occ})$ time.

Summing up, our solution uses $O(n(\log^{\varepsilon} n + \log\log u)$ space, $O(n(\log n + \log^{\delta} u))$ expected preprocessing time, and $O(m + \text{occ})$ query time. This completes the proof of Theorem 1.

4 Applications

In this section we show how to improve the results for the three problems position-restricted substring searching, indexing substrings with intervals, and indexing gapped substrings, using our data structure for substring range reporting. Let $\text{REPORT}_S(P, a, b)$ denote a substring range reporting query on string S with parameters P, a, and b.

4.1 Position-Restricted Substring Searching

We can reduce position-restricted substring searching to substring range reporting by setting $\text{label}(i) = i$ for all $i = 1, \ldots, n$. To answer a query we return the result of the substring range query $\text{REPORT}_S(P, a, b)$. Since each label is equal to the position, it follows that the solution to the substring range reporting instance immediately gives a solution to position-restricted substring searching. Applying Theorem 1 with $u = n$, this proves Theorem 2(i).

4.2 Indexing Substrings with Intervals

We can reduce indexing substrings with intervals to substring range reporting by setting

$$\text{label}(i) = \begin{cases} i & \text{if } i \in \varphi \text{ for some } \varphi \in \pi, \\ 0 & \text{otherwise.} \end{cases}$$

To answer a query we return the result of the substring range reporting query $\text{REPORT}_S(P, a, b)$. Let I be the solution to the indexing substrings with intervals instance and let I' be the solution to the substring range reporting instance derived by the above reduction. Then $i \in I \Leftrightarrow i \in I'$.

To prove this assume $i \in I$. Then $i \in \varphi$ for some $\varphi \in \pi$ and $i \in [a, b]$. From $i \in \varphi$ and the definition of $\text{label}(i)$ it follows that $\text{label}(i) = i$. Thus, $\text{label}(i) = i \in [a, b]$ and thus $i \in I'$. Assume $i \in I'$. Then $\text{label}(i) \in [a, b]$. Since $a > 0$ also $\text{label}(i) > 0$, and it follows that $\text{label}(i) = i$. By the reduction this means that $i \in \varphi$ for some $\varphi \in \pi$. Since $i = \text{label}(i)$, we have $i \in [a, b]$ and therefore $i \in I$.

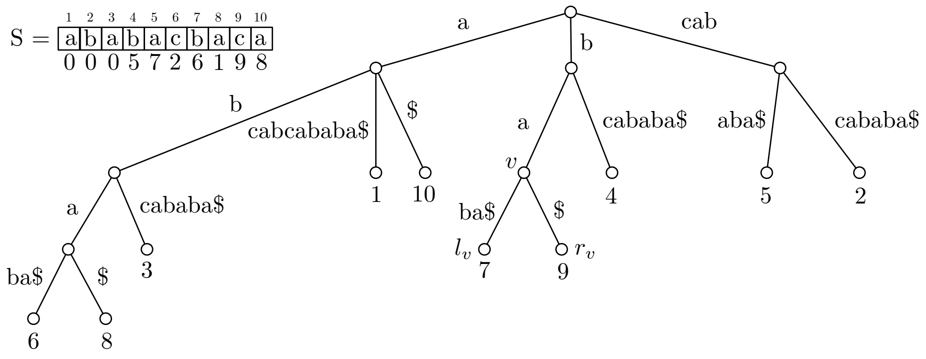

Figure 1: A string S, the labeling for $d = 2$ (below the string), and the suffix tree of T_{S^R}. Given a query $P_1 = \mathrm{ab}$ and $P_2 = \mathrm{bac}$ we find $v = \mathrm{locus}_{S^R}(\mathrm{ba})$ (marked in the suffix tree). We have $l_v = 6$ and $r_v = 7$ from the left-to-right-order in the T_{S^R}. The substring range reporting query $\textsc{report}_s(P_2, 6, 7)$ returns 7. Hence, we report the occurrence at position $7 - 2 - 2 = 3$.

We can construct the labeling in $O(n+|\pi|)$ if the intervals are sorted by startpoint or endpoint. Otherwise additional time for sorting is needed. A similar approach is used in the solution by Crochemore et al. [8].

Applying Theorem 1 with $u = n$, this proves Theorem 2(ii).

4.3 Indexing Substrings with Gaps

We can reduce the indexing substrings with gaps problem to substring range reporting as follows. Construct the suffix tree of the reverse of S, i.e., the suffix tree T_{S^R} for S^R. For each node v in T_{S^R} also store l_v and r_v. Set

$$\mathrm{label}_S(i) = \begin{cases} \mathrm{order}_{S^R}(n - i + d + 2) & \text{for } i \geq d + 2, \\ 0 & \text{otherwise.} \end{cases}$$

To answer a query find the locus node v of P_1^R in T_{S^R}. Then use the substring range reporting data structure to return all positions of substrings in S matching P_2 whose labels are in the range $[l_v, r_v]$. For each position i returned by $\textsc{report}_S(P_2, l_v, r_v)$, return $i - |P_1| - d$. See Fig. 1 for an example.

Correctness of the reduction We will now show that the reduction is correct. Let I be the solution to the indexing substrings with gaps instance and let I' be the solution to the substring range reporting instance derived by the above reduction. We will show $i \in I \Leftrightarrow i \in I'$. Let $m_i = |P_i|$ for $i = 1, 2$.

If $i \in I$ then there is an occurrence of P_1 at position i in S and an occurrence of P_2 at position $i' = i + m_1 + d$ in S. It follows directly, that there is an occurrence of P_1^R at position $i'' = n - (i + m_1) + 2$ in S^R. By definition,

$$\mathrm{label}_S(i') = \mathrm{label}_S(i + m_1 + d) = \mathrm{order}_{S^R}(n - (i + m_1 + d) + d + 2) = \mathrm{order}_{S^R}(i''),$$

where the second equality follows from the fact that $i + m_1 + d \geq d + 2$. Since there is an occurrence of P_1^R at position i'' in S^R, we have $\mathrm{label}_S(i') = \mathrm{order}_{S^R}(i'') \in \mathrm{order}_{S^R}(\mathrm{locus}_{S^R}(P_1^R))$.

Thus, $\text{label}_S(i') \in [l_v, r_v]$, and since there is an occurrence of P_2 at position i' in S, we have $i' - m_1 - d = i \in I'$.

If $i \in I'$ then there is an occurrence of P_2 at position $i' = i + m_1 + d$ with $\text{label}(i')$ in the range $[l_v, r_v]$, where $v = \text{locus}_{S^R}(P_1^R)$. We need to show that this implies that there is an occurrence of P_1 at position i in S. By definition,

$$\text{label}_S(i') = \text{order}_{S^R}(n - i' + d + 2) = \text{order}_{S^R}(n - i - m_1 + 2).$$

Let $i'' = n - i - m_1 + 2$. Since $\text{order}_{S^R}(i'') = \text{label}_S(i') \in [l_v, r_v]$, there is an occurrence of P_1^R at position i'' in S^R. It follows directly, that there is an occurrence of P_1 at position $n - i'' - m_1 + 2 = n - (n - i - m_1 + 2) - m_1 + 2 = i$ in S. Therefore, $i \in I$.

Complexity Construction of the suffix tree T_{S^R} takes time $O(n \log n)$ and the labeling can be constructed in time $O(n)$. Both use space $O(n)$. It takes $O(m_1)$ time to find the locus nodes of P_1^R in T_{S^R}. The substring range reporting query takes time $O(m_2 + \text{occ})$. Thus the total query time is $O(m + \text{occ})$.

Applying Theorem 1 with $u = n$, this completes the proof of Theorem 2(iii).

5 Substring Range Counting and Emptiness

We now show how to apply our techniques to *substring range counting* and *substring range emptiness*. Analogous to substring range reporting, the goal is here to count the number of occurrences in the range and to determine whether or not the range is empty, respectively. A straightforward way to solve these problems is to combine a suffix tree with a 2D range counting data structure and a 2D range emptiness data structure, respectively. Using the techniques from Section 3 we show how to significantly improve the bounds of this approach in both cases. We note that by the reductions in Section 4 the bounds for substring range counting and substring range emptiness also immediately imply results for counting and emptiness versions of position-restricted substring searching, indexing substrings with intervals, and indexing substrings with gaps.

5.1 Preliminaries

Let $X \subseteq \{0, \ldots, u\}$ be a set of points in a d-dimensional grid. Given a query rectangle $R = [a_1, b_1] \times \cdots \times [a_d, b_d]$, a *range counting query* computes $|R \cap X|$, and a *range emptiness query* computes if $R \cap X = \emptyset$. Given X the *range counting problem* and the *range emptiness problem* is to compactly represent X, while supporting range counting queries and range emptiness queries, respectively. Note that any solution for range reporting or range counting implies a solution for range emptiness with the same complexity (ignoring the occ term for range reporting queries). We will need the following additional geometric data structures.

Lemma 3 (JáJá et al. [19]) *For a set of n points in $[0, u] \times [0, u]$ we can solve 2D range counting in $O(n)$ space, $O(n \log n)$ preprocessing time, and $O(\log n / \log \log n + \log \log u)$ query time.*

Lemma 4 (van Emde Boas et al. [29, 30], Mehlhorn and Näher [23]) *For a set of n points in $[0, u]$ we can solve 1D range counting in $O(n)$ space, $O(n \log \log n)$ preprocessing time, and $O(\log \log u)$ query time.*

To achieve the result of Lemma 4 we use a van Emde Boas data structure [29, 30] implemented in linear space [23] using perfect hashing. This data structure supports predecessor queries in $O(\log\log u)$ time. By also storing for each point it's rank in the sorted order of the points, we can compute a range counting query by two predecessor queries. To build the data structure efficiently we need to sort the points and build suitable perfect hash tables. We can sort deterministically in $O(n \log\log n)$ time [16], and we can build the needed hash tables in $O(n)$ time using deterministic hashing [15] combined with a standard two-level approach (see e.g., Thorup [28]).

Lemma 5 (Chan et al. [5]) *For a set of n points in $[0, u] \times [0, u]$ we can solve 2D range emptiness in $O(n \log\log n)$ space, $O(n \log n)$ preprocessing time, and $O(\log\log u)$ query time.*

5.2 The Data Structures

We now show how to efficiently solve substring range counting and substring range emptiness. Recall that S is a labeled string of length n with labels from $[0, u]$.

We can directly solve substring range counting by combining a suffix tree with the 2D range counting result from Lemma 3. This leads to a solution using $O(n)$ space and $O(m + \log n/\log\log n + \log\log u)$ query time. We show how to improve the query time to $O(m + \log\log u)$ at the cost of increasing the space to $O(n \log n/\log\log n)$. Hence, we remove the $\log n/\log\log n$ term from the query time at the cost of increasing the space by a $\log n/\log\log n$ factor. We cannot hope to achieve such a bound using a suffix tree combined with a 2D range counting data structure since any 2D range counting data structure using $O(n \log^{O(1)} n)$ space requires $\Omega(\log n/\log\log n)$ query time [25]. We can also directly solve substring range emptiness by combining a suffix tree with the 2D range emptiness result from Lemma 5. This solution uses $O(n \log\log n)$ space and $O(m + \log\log u)$ query time. We show how to achieve optimal $O(m)$ query time with space $O(n \log\log u)$.

Our data structure for substring range counting and existence follows the construction in Section 3. We partition the suffix tree into a top and a number of bottom trees and store a 1D data structure for each node in the top tree and a single 2D data structure. To answer a query for a pattern string P of length m, we search the suffix tree with P and use the 1D data structure if the search ends in the top tree and otherwise use the 2D data structure.

We describe the specific details for each problem. First we consider substring range counting. In this case the top tree consists of all nodes of string depth at most $\log n/\log\log n$. The 1D and 2D data structures used are the ones from Lemma 4 and 3. By the same arguments as in Section 3 the total space used for the 1D data structures for all nodes in the top tree at depth d is at most $O(n)$ and hence the total space for all 1D data structures is $O(n(\log n/\log\log n))$. Since the 2D data structure uses $O(n)$ space, the total space is $O(n \log n/\log\log n)$. The time to build all 1D data structures is $O(n(\log n/\log\log n) \cdot \log\log n)) = O(n \log n)$. Since the suffix tree and the 2D data structure can be built within the same bound, the total preprocessing time is $O(n \log n)$. Given a pattern of length m, a query uses $O(m + \log\log u)$ time if the search ends in the top tree, and $O(m + \log n/\log\log n + \log\log u)$ time if the search ends in a bottom tree. Since bottom trees consists of nodes of string depth more than $\log n/\log\log n$ the time to answer a query in both cases is $O(m + \log\log u)$. In summary, we have the following result.

Theorem 3 *Let S be a labeled string of length n with labels in the range $[0, u]$. We can solve substring range counting using $O(n \log n/\log\log n)$ space, $O(n \log n)$ preprocessing time, and $O(m + \log\log u)$ query time, for a pattern string of length m.*

Next we consider substring range emptiness. In this case the top tree consists of all nodes of string depth at most $\log \log u$. We use the 1D and 2D data structures from Lemma 1 and Lemma 5. The total space for all 1D data structures is $O(n \log \log u)$. Since the 2D data structure uses $O(n \log \log n)$ space the total space is $O(n \log \log u)$. For any constant $\gamma > 0$, the expected time to build all 1D data structures is $O(n \log \log u \log^{\gamma} u) = O(n \log^{\delta} u)$ for suitable constant $\delta > 0$. The suffix tree and the 2D data structure can be built in $O(n \log n)$ time and hence the total expected preprocessing time is $O(n(\log n + \log^{\delta} u))$. If the search for a pattern string ends in the top tree the query time is $O(m)$ and if the search ends in a bottom tree the query time is $O(m + \log \log u)$. As above, the partition in top and bottom trees ensures that the query time in both cases is $O(m)$. In summary, we have the following result.

Theorem 4 *Let S be a labeled string of length n with labels in the range $[0, u]$. For any constant $\delta > 0$ we can solve substring range existence using $O(n \log \log u)$ space, $O(n(\log n + \log^{\delta} u))$ expected preprocessing time, and $O(m)$ query time, for a pattern string of length m.*

6 Acknowledgments

We thank Christian Worm Mortensen and Kasper Green Larsen for clarifications on the preprocessing times for the results in Lemma 3 and Lemma 5.

References

[1] S. Alstrup, G. Brodal, and T. Rauhe. Optimal static range reporting in one dimension. In *Proc. 33rd STOC*, pages 476–482, 2001.

[2] S. Alstrup, G. Stølting Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *Proc. 41st FOCS*, pages 198–207, 2000.

[3] A. Amir, E. Chencinski, C. S. Iliopoulos, T. Kopelowitz, and H. Zhang. Property matching and weighted matching. *Theoret. Comput. Sci.*, 395(2-3):298–310, 2008.

[4] P. Bose, M. He, A. Maheshwari, and P. Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *Proc. 11th WADS*, pages 98–109, 2009.

[5] T. M. Chan, K. Larsen, and M. Pătraşcu. Orthogonal range searching on the ram, revisited. In *Proc. 27th SoCG*, pages 354–363, 2011.

[6] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.

[7] H. Cohen and E. Porat. Range non-overlapping indexing. In *Proc. 20th ISAAC*, pages 1044–1053, 2009.

[8] M. Crochemore, C. S. Iliopoulos, M. Kubica, M. S. Rahman, and T. Walen. Improved algorithms for the range next value problem and applications. In *Proc. 25th STACS*, pages 205–216, 2008.

[9] M. Crochemore, C. S. Iliopoulos, M. Kubica, M. S. Rahman, and T. Walen. Finding patterns in given intervals. *Fundam. Inform.*, 101(3):173–186, 2010.

[10] M. Crochemore, C. S. Iliopoulos, and M. S. Rahman. Optimal prefix and suffix queries on texts. *Inf. Process. Lett.*, 108(5):320–325, 2008.

[11] M. Crochemore and G. Tischler. The gapped suffix array: A new index structure. In *Proc. 17th SPIRE*, pages 359–364, 2010.

[12] M. Farach-Colton, P. Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000.

[13] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31:538–544, 1984.

[14] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge, 1997.

[15] T. Hagerup, P. B. Miltersen, and R. Pagh. Deterministic dictionaries. *J. Algorithms*, 41(1):69–85, 2001.

[16] Y. Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004.

[17] C. S. Iliopoulos and M. S. Rahman. Faster index for property matching. *Inf. Process. Lett.*, 105(6):218–223, 2008.

[18] C. S. Iliopoulos and M. S. Rahman. Indexing factors with gaps. *Algorithmica*, 55(1):60–70, 2009.

[19] J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15th ISAAC*, pages 558–568, 2004.

[20] M. Juan, J. Liu, and Y. Wang. Errata for "Faster index for property matching". *Inf. Process. Lett.*, 109(18):1027–1029, 2009.

[21] V. Mäkinen and G. Navarro. Position-restricted substring searching. In *Proc. 7th LATIN 2006*, pages 703–714, 2006.

[22] V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theoret. Comput. Sci.*, 387(3):332–347, 2007.

[23] K. Mehlhorn and S. Nähler. Bounded ordered dictionaries in $O(\log\log N)$ time and $O(n)$ space. *Inform. Process. Lett.*, 35(4):183–189, 1990.

[24] C. W. Mortensen, R. Pagh, and M. Pătraçcu. On dynamic range reporting in one dimension. In *Proc. 37th STOC*, pages 104–111, 2005.

[25] M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proc. 39th STOC*, pages 40–46, 2007.

[26] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th STOC*, pages 232–240, 2006.

[27] E. Porat, 2011. Personal communication.

[28] M. Thorup. Space efficient dynamic stabbing with fast queries. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 649–658, 2003.

[29] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Process. Lett.*, 6(3):80–82, 1977.

[30] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977. Announced at FOCS 1975.

[31] C.-C. Yu, W.-K. Hon, and B.-F. Wang. Improved data structures for the orthogonal range successor problem. *Comput. Geometry*, 44(3):148 – 159, 2011.