LOCALIZED SUFFIX ARRAY AND ITS APPLICATION TO GENOME MAPPING PROBLEMS FOR PAIRED-END SHORT READS

KOUICHI KIMURA
kouichi.kimura.hh@hitachi.com

ASAKO KOIKE
asako.koike.ea@hitachi.com

Central Research Laboratory, Hitachi Ltd, 1-280 Higashi-Koigakubo, Kokubunji, Tokyo 185-8601, Japan

We introduce a new data structure, *a localized suffix array*, based on which occurrence information is dynamically represented as the combination of global positional information and local lexicographic order information in text search applications. For the search of a pair of words within a given distance, many candidate positions that share a coarse-grained global position can be compactly represented in term of local lexicographic orders as in the conventional suffix array, and they can be simultaneously examined for violation of the distance constraint at the coarse-grained resolution. Trade-off between the positional and lexicographical information is progressively shifted towards finer positional resolution, and the distance constraint is reexamined accordingly. Thus the paired search can be efficiently performed even if there are a large number of occurrences for each word. The localized suffix array itself is in fact a reordering of bits inside the conventional suffix array, and their memory requirements are essentially the same. We demonstrate an application to genome mapping problems for paired-end short reads generated by new-generation DNA sequencers. When paired reads are highly repetitive, it is time-consuming to naïvely calculate, sort, and compare all of the coordinates. For a human genome re-sequencing data of 36 base pairs, more than 10 times speedups over the naïve method were observed in almost half of the cases where the sums of redundancies (number of individual occurrences) of paired reads were greater than 2,000.

Keywords: suffix array; rank function; genome mapping; short reads; paired-end reads; new-generation DNA sequencing.

1. Introduction

With the advent of new-generation DNA sequencers, the ongoing rapid improvements in sequencing technologies continue to expand the frontiers of genomic sciences such as personal genomics, metagenomics, and others [2, 9]. Their overwhelmingly high sequencing throughputs and relatively short read lengths have posed big challenging problems in bioinformatics [10]. In particular, genome mapping problems have been reconsidered in the light of the innovated context, and a number of powerful methods have been developed so far. Among them, methods based on the spaced seed and extension approach such as ELAND and MAQ have been developed earlier and widely used [7]. More recently, methods based on the Burrow-Wheeler transform (BWT) are attracting attention by their superior performance [3, 4, 6, 8].

One of the latest major concerns in genome mapping for short reads is involved in paired-end (PE) mapping. For the compensation of the disadvantage of short read lengths, PE sequencing has been developed: a pair of short reads are obtained by sequencing from both ends of a DNA fragment whose length is approximately controlled. Therefore, they are expected to be found on the reference genome apart from each other by a distance that corresponds to the fragment length. Given a pair of short reads, *the PE mapping problem* is to find their positions on the genome apart from each other within a given distance and in consistent orientation; in contrast, given a single short read, *the single-end (SE) mapping problem* is to find its positions on the genome. If the intra-pair distance or orientation is certainly different from expected, it can be used as evidence for genomic structural variations such as insertions, deletions, inversions and translocations.

A basic approach to PE mapping is to first find all of the possible mapping positions of each read independently (SE mappings for each read), and then choose combinations of them so that the intra-pair distance and orientation conditions are satisfied. When BWT or a suffix array (SA) is being used, the SE mapping subproblems can be solved efficiently and the intermediate results can be concisely expressed in terms of lexicographic orders (indices) no matter how many solutions are found. However, in order to choose the proper combinations among them, it is necessary to convert them into genomic coordinates (positions) and to sort and compare them. It can be time-consuming if the reads are highly repetitive (frequently occurring). In general, when BWT or SA is being used, it is difficult to restrict the search in a particular region of interest since the occurrence information is managed in terms of indices rather than positions.

In this paper, we propose a new data structure, *a localized suffix array (LSA)*, which is a modification of the conventional SA, and present a new search method that can handle repetitive occurrences efficiently as the conventional SA does, and at the same time, can be used in restricted search in particular regions of interest. We demonstrate its applications to PE mapping problems for short reads and discuss their performances using real biological data.

2. Localized Suffix Array (LSA)

We discuss the subjects mainly in terms of PE genome mapping problems in order to present our idea concretely. However, most of the discussion in this section can be generally applicable to other text search problems in different fields. We begin with presenting a basic idea of localizing lexicographic order information, and consider refining it into a feasible method. For ease of understanding, we first explain how the localization should work using a tiny text string example. Then a formal description of the algorithm and data structure follows.

2.1. *Basic idea*

One of the drawbacks in text searching methods based on suffix array techniques is that it is not easy to consider constraints on the occurrence positions during the searching process. For example, consider the PE mapping problem: locating a pair of short reads such that their positions are apart from each other within a given distance. For simplicity, only exact matching is considered here. One can efficiently locate all of the occurrences of each read independently using suffix array techniques. In particular, the occurrences of a read can be concisely represented as a single interval of suffix indices no matter how repetitive it is. However, in order to find a combination of the occurrences of the paired reads within a given distance, one should calculate, sort, and compare all of the occurrence positions. It would be time-consuming if the reads are highly repetitive.

One of the remedies would be to employ local suffix arrays that are restricted to subregions. One might recursively divide up the whole genome region and build a suffix array for each subregion as shown in Figure 1. Positional constraints would be considered by restricting the searches to some subregions and locally repetitive elements would be treated efficiently on the corresponding suffix arrays. However, this approach does not seem to work efficiently at all since there is extremely high redundancy among the local suffix arrays for subregions at different levels.

Since only short prefixes of suffixes are relevant in search applications, a subarray of the suffix array (referred to as *suffix subarray*) that corresponds to a subregion can be used in place of a local suffix array restricted to the subregion (Figure 2). However, note that the orders of indices in them are not necessarily consistent with each other because the corresponding suffixes have different tail ends. Since the values in the suffix array represent positions, binary representations of values in the

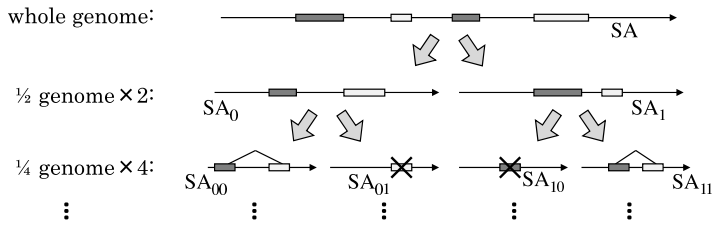

Fig. 1. Basic idea of localizing lexicographic order information.

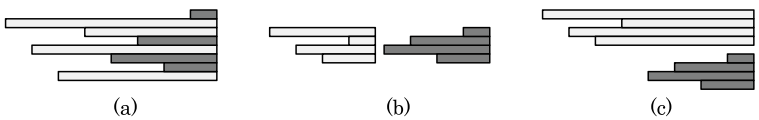

Fig. 2. (a) list of suffixes corresponding to the suffix array for a whole region, (b) lists of suffixes corresponding to the local suffix arrays for half subregions, (c) lists of suffixes corresponding to the suffix subarrays for half subregions. Suffixes in the second half subregions are shaded dark gray.

suffix subarray share common higher bits that reflect the position and width of the subregion, namely, their global positional information. Meanwhile, they have distinct lower bits, which convey lexicographic order information within the subregion (Figure 3(a)). Making the subregion narrower increases the width of the common higher bit fields representing the positional information, and decreases the width of distinct lower bit fields representing the order information.

The process of narrowing the subregion is referred to as *localization*. PE mapping based on a recursive localization (RL) method is schematically illustrated in Figure 3(b). Initially, occurrences of each read are independently given in terms of lexicographic order in the suffix array, and there are no positional information at all. By the recursive localization of the occurrences, one can get progressively detailed positional information. Along the way, positional constraint (intra-pair distance) can be checked up to the positional resolution at the time. As a special case, when one of the reads is uniquely located by itself, after immediately calculating its position, the occurrences of the other read in the vicinity can be located in progressive resolution by the recursive localization. Finally, the positional information of the mapping results satisfying the constraint are precisely determined.

2.2. *Procedural introduction of recursive localization (RL) and LSA*

In this subsection, we concretely explain how to realize the recursive localization (RL) using a tiny text example, and also make some observations that will lead to an efficient algorithm presented in the subsequent subsection. Remember that the binary representation of an occurrence is initially lexicographic order information alone. In each step of RL, a single bit of positional information will be gained in exchange for loss of a single bit of lexicographic order information.

Let us consider a tiny example text data: $S =$ "`abracad`". The length of S is seven and the entire region over S is $[0, 7]$. Any occurrence of a substring of S

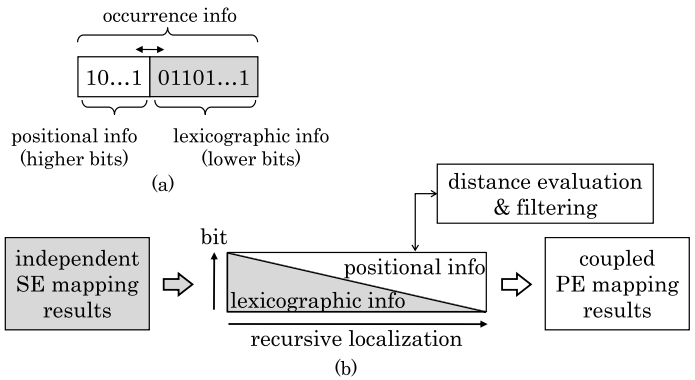

Fig. 3. Paired-end (PE) mapping by the recursive localization (RL) method.

can be specified by an index in the suffix array (Figure 4(a)). The values in the suffix array, $(7, 0, 3, 5, 1, 4, 6, 2)$, are sorted according to the lexicographic order of the corresponding suffixes, and all of the bits in the binary expression convey the lexicographic order information (Figure 4(b)). At the first step of RL, by sorting the indexed values with respect to the most significant bit, we get two suffix subarrays, $(0, 3, 1, 2)$ and $(7, 5, 4, 6)$, which correspond to two subregions $[0, 3]$ and $[4, 7]$, respectively. The binary representations of values in these suffix subarray have three bits PLL, where the most significant bit P indicates the global positional information of the subregions and the lower bits LL convey the local lexicographic order information within the subregion (Figure 4(c)). At the second step of RL, by sorting the indexed values again with respect to the secondly significant bit, we get four suffix subarrays: $(0, 1)$, $(3, 2)$, $(5, 4)$ and $(7, 6)$, which correspond to four (sub)subregions: $[0, 1]$, $[2, 3]$, $[4, 5]$ and $[6, 7]$, respectively. The binary representations of values in these suffix subarray have three bits PPL, where the higher bits PP indicate the global positional information of the (sub)subregions and the lower bit L conveys the local lexicographic order information within the (sub)subregions (Figure 4(d)). At the final step of RL, by further sorting the indexed values again according to the last bit, we get eight suffix subarrays: $(0), (1), \ldots, (7)$, which are completely sorted according to the positions since we have so far sorted them according to all of the three bits (Figure 4(e)). The final values in these suffix subarrays consist exclusively of positional bits PPP. They correspond to eight subregions of length one: $[0, 0], [1, 1], \ldots, [7, 7]$, respectively.

In the intermediate steps of RL, occurrences of locally repetitive substrings restricted to particular subregions are concisely represented by the mixed bit representations: the higher bits are used to specify the subregion and the lower bits are used to specify the local lexicographic order within the subregion. The latter have similar utility in the subregion as the conventional suffix array has in the whole region.

Note that the indexed values are sorted in each step with respect to *the ref-*

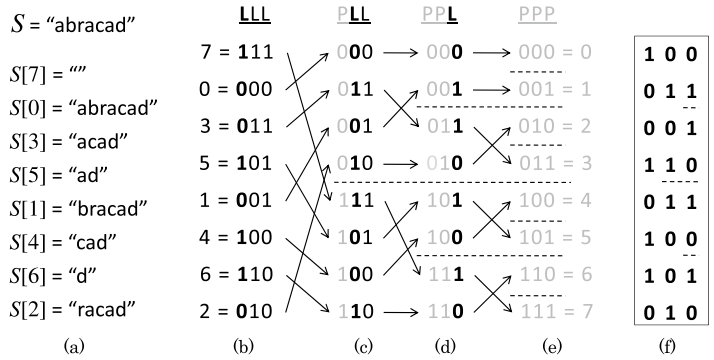

Fig. 4. An example of recursive localization (RL). The reference bits, highlighted in boldface type in (b), (c) and (d), constitute the localized suffix array (LSA) in (f).

erence bits in the corresponding column, which are shown in bold face fonts in Figure 4(b)(c)(d). Namely, these reference bits in columns constitute all the information necessary to carry out the above procedure. We define *the localized suffix array (LSA)* as these reference bits in columns lumped together (Figure 4(f)). Each column of LSA is an rearrangement of the bits in the corresponding column of the binary representation of the original suffix array. Hence the LSA and the conventional suffix array have the same number of bits. Moreover they have the equivalent information because the conventional suffix array can be restored conversely from LSA by rolling back the above RL procedures starting from the completely sorted position list. Finally, note that the internal orders in a suffix subarray are inherited from its parent subarray.

2.3. *Algorithms for LSA construction and RL of index intervals*

Motivated by the previous observations, we can now formalize the algorithm for RL and data structure of LSA. Let S be a text data, \$ be a termination symbol that does not appear in S, and B be the suffix array of $S\$$. For simplicity, we assume that the length of $S\$$ is a power of two, $\ell(S\$) = 2^w$, which is not an essential restriction. Then each value in B is uniquely represented by a w-bit binary string. Let Q be a query text string that appears at least once in the text data S. Each occurrence of Q in S is represented by an index of B, and the set of all occurrences (exact matches) of Q in S is represented by a half-open index interval $R = [s, t[$ (greater than or equal to s and less than t) of the suffix array B and the number of occurrences is given by $t - s$. Here only exact matches are being considered for simplicity. In the case of approximate matches, multiple intervals should be considered instead, and the following discussion could be modified accordingly in a straightforward way.

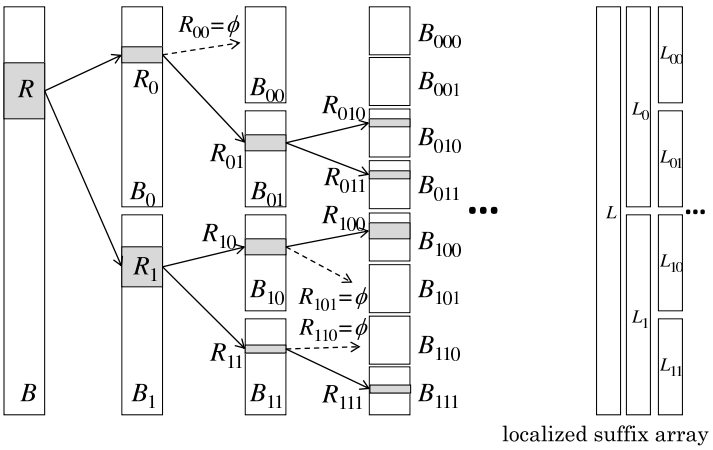

Fig. 5. Recursive localization (RL) of suffix array and index interval. B is a suffix array and R is an index interval representing the occurrences of a query string. Suffix subarrays and localized intervals are denoted by B_α and R_α, respectively, with a binary string α.

Suffix subarrays, denoted by B_α, are recursively defined as follows. Figure 5 illustrates the process in parallel with Figure 4. Generally, α denotes a binary string over 0 and 1 of length (denoted by $\ell(\alpha)$) up to w. α may represent an empty string ε. We define $B_\varepsilon = B$ and that $B_{\alpha 0}$ and $B_{\alpha 1}$ be the subarray of B_α whose $(d+1)$-th significant bit, referred to as *the reference bit*, is 0 or 1, respectively, where $d = \ell(\alpha) < w$. Consequently, B_α is the subarray of B with the highest d bits equal to α where $\ell(\alpha) = d$ for $0 \le d \le w$. Let L_α denote the concatenation of the reference bits in B_α. The *localized suffix array (LSA)* is constituted from these L_α altogether for $0 \le \ell(\alpha) < w$. It contains all of the required information as noted below. A pseudo-code of construction algorithm is given in Figure 6.

Localized intervals, denoted by $R_\alpha = [s_\alpha, t_\alpha[$, are recursively defined as follows. $R_\varepsilon = R$, namely, $s_\varepsilon = s$ and $t_\varepsilon = t$. For α with $\ell(\alpha) < w$, $R_{\alpha 0} = [s_{\alpha 0}, t_{\alpha 0}[$ and $R_{\alpha 1} = [s_{\alpha 1}, t_{\alpha 1}[$ are given by

$$s_{\alpha 0} = \mathrm{rank}(L_\alpha, s_\alpha, 0), \qquad t_{\alpha 0} = \mathrm{rank}(L_\alpha, t_\alpha, 0),$$
$$s_{\alpha 1} = \mathrm{rank}(L_\alpha, s_\alpha, 1), \qquad t_{\alpha 1} = \mathrm{rank}(L_\alpha, t_\alpha, 1),$$

where rank() denotes the rank function. In general, given a binary string L over 0 and 1, $\mathrm{rank}(L, s, 0)$ is the number of occurrences of 0 in the first s bits of L; likewise, $\mathrm{rank}(L, t, 1)$ is the number of occurrences of 1 in the first t bits of L. Fast algorithms for computing rank functions with a little space overheads for precomputed data, for example with less than 3% extra overheads, are known [1, 3]. The validity of this definition is guaranteed by the above-mentioned inheritance property of the internal orders in a suffix subarray. Finally with $\ell(\alpha) = w$, the localized intervals acquire precise positions. Note that all the information that is required to compute $(R_\alpha)_{\ell(\alpha) \le w}$, *the recursive localization (RL)* of R, is given by LSA.

3. Application to Paired-End (PE) Mapping Problems

Given a pair of short reads, the paired-end (PE) mapping problem is to find their locations on the reference genome that are apart from each other within a given distance and in consistent orientation. It can be solved by first solving single-ended (SE) mapping problems for each read independently and then searching for combination of the occurrences of the paired reads that satisfy the intra-pair distance and orientation conditions. In the latter stage, the RL method developed in the preceding section can be applied: the locations of localized intervals were coarsely determined in progressive accuracy, which enables checking the intra-pair conditions and filtering out illegal candidates in bulk (Figures 1, 3). We explain details of the implementation of our experimental program and show the results.

3.1. *Single-end (SE) mapping method*

In solving SE mapping problem, a few mismatches at the nucleotide level are usually allowed in consideration of possible polymorphism or sequencing errors. We developed a fast SE mapping program using BWT that allowed at most one mismatch

(substitution, insertion or deletion of a single base) per read [3]. Descending priorities were given to exact matches, substitutions and indels (insertions or deletions) in this order. Occurrences with higher priority were searched first, and if they were found, other occurrences with lower priorities were not searched. The SE mapping results were represented as a set of index intervals of the suffix array of the reference genome, but their genomic positions were not calculated.

3.2. *Paired-end (PE) search methods*

A set of intervals of the suffix array, given by the SE mapping program, were recursively localized, and they were repeatedly tested and filtered in terms of the intra-pair conditions. The minimum and maximum intra-pair distances were specified in advance, and the pairs of positions with minimum intra-pair distance within the specified range were output for each pair of reads (as an initial attempt). When no legal positions were found, we returned to the SE mapping to collect all occurrences down to the lowest priority in order to get an extended set of intervals, and

```
 1: build_LSA(int length, int SA[length], bool LSA[height][length]) {
 2:   int val[length], buffer[2][length / 2];
 3:   for (int j = 0; j < length; ++j)
 4:     val[j] = SA[j];
 5:   for (int ht = height; ht > 0; --ht) {
 6:     int block_len = 1 << ht, num_block = ceil(length / block_len);
 7:     for (int bl = 0; bl < num_block; ++bl) {
 8:       int *pt[2] = {buffer[0], buffer[1]};
 9:       for (int i = 0; i < block_len; ++i) {
10:         int j = bl * block_len + i;
11:         if (j >= length)
12:           break;
13:         bool t = val[j] & (block_len / 2);
14:         LSA[ht - 1][j] = t;
15:         *pt[t]++ = val[j];
16:       }
17:       int j = bl * block_len;
18:       for (int t = 0; t < 2; ++t)
19:         for (int *q = buffer[t]; q < pt[t]; ++q)
20:           val[j++] = *q;
21:     }
22:   }
23: }
```

Fig. 6. Pseudo-code for building localized suffix array (LSA). `length` and `SA` are input arguments and `LSA` is an output argument of the function. `SA` and `val` are one-dimensional arrays and `LSA` and `buffer` are two-dimensional arrays of designated sizes. `<<` and `&` denote a leftward bit shift operation and a bitwise AND operation, respectively. `height` is equal to $\lceil \log(\texttt{length}) \rceil$. The suffix subarrays, implicitly referred to as `blocks`, are temporally stored in consecutive subarrays of `val`.

reattempted the PE searches.

For comparison, we also implemented the naïve approach: all of the positions of occurrences given by SE mapping were calculated, sorted and compared, and pairs of positions satisfying the intra-pair conditions were chosen among them.

3.3. *Experimental results*

The performance of the experimental program was evaluated using real biological data from NCBI short read archive, SRA00271, human genome resequencing data of an African individual (NA18507) [11]. The read length was 36 bp and the median insert size was about 200 bp. In PE mapping, the minimum and maximum intra-pair distances were set to 0 bp and 1024 bp, respectively.

We used the human genome reference sequence data hg18 from the UCSC Genome Bioinformatics site [12]. The suffix arrays and BWTs of the human genome sequence on both strands were calculated using the algorithm of *qsufsort* [5]. It took approximately 46 minutes each on a 3.0 GHz Intel Xeon Linux PC using approximately 30 GB of memory. Then LSA on both strands were calculated in approximately 21 minutes each. These preprocessings were done only once by a formatting program and the results were stored in a hard drive as binary data. They were reloaded again into main memory and reused by the mapping program. The memory requirement for LSA was about 12GB for each strand, and the mapping program used about 29GB memory in total.

We first examined the distributions of mapping redundancies. *The SE mapping redundancy* was defined as the number of solutions of the SE mapping problems for each read. It was thought to have direct influence on the computational cost for solving the PE mapping problem especially when the naïve method was employed. *The PE mapping redundancy* was defined as the number of solutions of the PE mapping problem. It was expected to be reduced when the intra-pair constraints were effective. When a read or a read pair failed to find any mapping positions, its

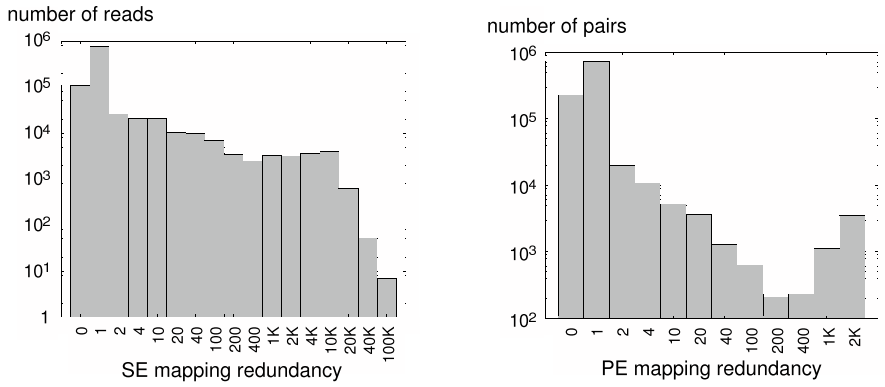

Fig. 7. Distribution of SE and PE mapping redundancy for a million pairs of reads.

redundancy was set to zero. Figure 7 shows an example of the distributions of the SE and PE mapping redundancies for a million pairs of reads. The reduction of the PE mapping redundancy compared to the SE mapping redundancy was done by the PE mapping program. The success rate of PE mapping was about 77.0%.

We also examined how the RL method reduced the computation costs when the reads are highly repetitive. As a simple measure of repetitiveness, we adopted the sum of the SE mapping redundancy of paired reads (abbreviated as SSR). Figure 8 shows a joint distribution of SSR (in abscissa) and computation time in microseconds (in ordinate) by the naïve ('×') and RL ('+') methods in a log-log plot. Each point corresponds to a pair of read, and the results of 10,000 pairs of reads are shown. The computation time is the sum of those for SE and PE mapping. When read pairs were highly repetitive, the naïve method always required long computation time as expected, and great reduction of computation time by the RL method was observed for a large majority of them. In fact, in an extended dataset

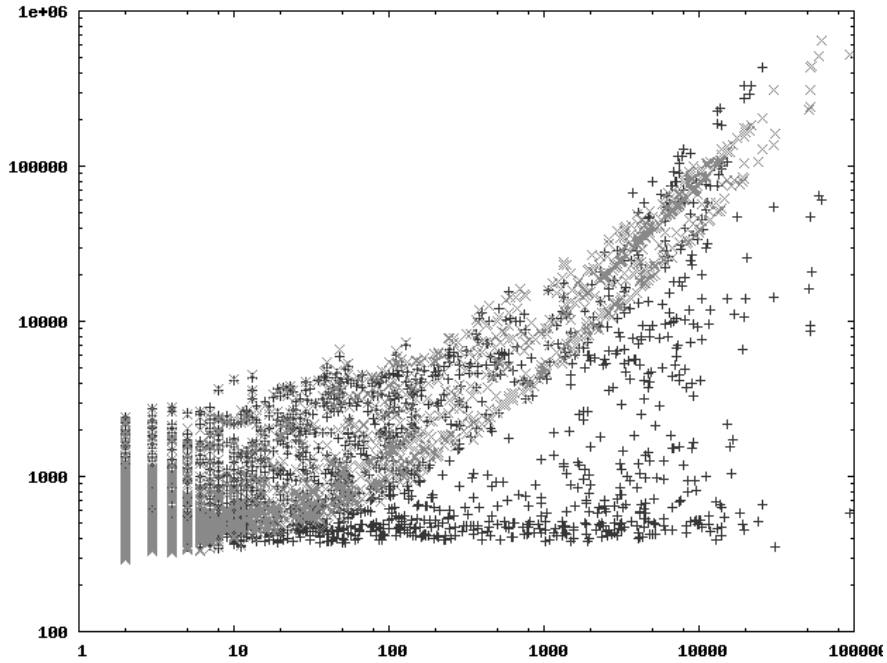

Fig. 8. Joint distribution of SSR (in abscissa) and computation time in microseconds (in ordinate) by the naïve ('×') and RL ('+') methods in a log-log plot. Each point corresponds to a pair of read, and the results of 10,000 pairs of reads are shown. SSR is a measure of repetitiveness of the reads. The computation time is the sum of those for SE and PE mapping. When read pairs were highly repetitive (SSR were large), the naïve method always required long computation time as expected, and great reduction of computation time by the RL method was observed for a large majority of them.

of 2 million pairs, more than 10 times speedup were observed in approximately 47% of the cases where SSR is greater than 2,000. In contrast, when the read pairs were hardly repetitive, the naïve method outperformed the RL method because of the light workload and its algorithmic simplicity. In addition, there are minor occasions where the read pairs were highly repetitive and yet the naïve approach outperformed the RL method. We will discuss this issue in the following section.

4. Additional Results and Discussions

In this section, we discuss some practical issues in efficiently utilizing the RL method. Clearly, when the paired reads are hardly repetitive and the mapping workload is light, the naïve method is superior to the RL method because of its simplicity. Hence, in a practical point of view, a suitable method should be selected according to the repetitiveness of paired reads. As discussed earlier, SSR can be used a simple measure of repetitiveness. We refer to switching the above two methods according to this measure as *mixed method*. An optimal threshold value of SSR was around 500 for the data of 2 million pairs of reads of length 36 bp that was used in the preceding section. Overall computation time by the naïve and RL method was 4,635 sec and 3,189 sec (1.5× speedup), respectively, and it was reduced to 2,554 sec (1.8× speedup) by the mixed method.

As mentioned earlier, the RL method was occasionally outperformed by the naïve method even in the case of highly repetitive reads. These were incurred by the proliferation of localized intervals in the RL method. Although the number of localized intervals might be doubled in each localization step, the intra-pair constraint pruned isolated intervals and the proliferation was prevented in most cases. However, when paired reads were not only highly repetitive individually but also highly repetitive as a pair, the proliferation did occur. When a proliferation was observed, a practical remedy would be to give up mapping the pair at the cost of less sensitivity. When the number of localized intervals are limited to 100, the mapping time for the same data was reduced to 1,595 sec (2.9× speedup) while the number of successfully mapped pairs was not reduced at all.

When the initial attempt of PE mapping was failed, further acceleration of computation at the cost of less sensitivity was possible by giving up the second attempt of PE mapping without returning to the SE mapping. Then the mapping time was further reduced to 1,013 sec (4.6× speedup) and the success rate of mapping was reduced by 0.25%.

Read length is one of the major factors that affects the performance of PE mapping since shorter reads occur more frequently on the genome. We examined the performances when the read length is reduced to 24 bp using first 24 bp of reads from the same data. Overall computation time by the naïve and RL method was 28,011 sec and 8,997 sec (3.1× speedup), respectively, and it was reduced to 6,386 sec (4.4× speedup) by the mixed method. Thus the RL method seemed to be more advantageous when the read length was shorter.

There are some applications where PE mapping with shorter read lengths should be considered, such as spliced mapping problems for transcripts: a read should be mapped on the genome possibly after split at an unknown exon junction. This problem can be solved by first solving the PE mapping problems for possible splits and then checking the consensus of the genomic sequences around the mapped spliced positions. One of the split reads can be very short since splitting may occur at any position of the transcript read. Thus the RL method is expected to be advantageous.

5. Conclusions

We proposed a new data structure, a localized suffix array that is a modification of the conventional suffix array, and presented a new search method based on recursive localization of index intervals, which is reduced to computing rank functions. It enables local searches in regions of interest, and at the same time, efficiently treats highly repetitive occurrences as the conventional suffix array does. It was applied to genome mapping problems for paired-end short reads and its accelerating performance was demonstrated on real biological data. Finally, we pointed out its potentially advantageous applications.

References

[1] González, R., Grabowski, S., Makinen, V., et al., Practical implementation of rank and select queries, *Poster Proceedings Volume of 4th Workshop on Efficient and Experimental Algorithms*, CTI Press and Ellinika Grammata, Greece, 27–38, 2005.
[2] Holt, R.A. and Jones, J.M., The new paradigm of glow cell sequencing, *Genome Res.*, 18(3):839–846, 2008.
[3] Kimura, K., Suzuki, Y., Sugano, S. and Koike, A., Computation of rank and select functions on hierarchical binary string and its application to genome mapping problems for short-read DNA sequences, *Journal of Computational Biology*, to appear.
[4] Langmead, B., Trapnell, C., Pop, M. and Salzberg, S.L., Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, *Genome Biology*, 10(3):R25, 2009.
[5] Larsson, N.J. and Sadakane, K., Faster Suffix Sorting, *Theoretical Computer Science*, 387(3):258–272, 2007.
[6] Li, H. and Durbin, R., Fast and accurate short read alignment with Burrows-Wheeler transform, *Bioinformatics*, 25(14):1754–1760, 2009.
[7] Li, H., Ruan, J. and Durbin, R., Mapping short DNA sequencing reads and calling variants using mapping quality scores, *Genome Res.*, 18(8):1851–1858, 2008.
[8] Li, R., Yu, C., Li, Y., et al., SOAP2: an improved ultrafast tool for short read alignment, *Bioinformatics*, 25(15):1966-1967, 2009.
[9] Schuster, S.C., Next-generation sequencing transforms today's biology, *Nature Methods*, 5(1):16–18, 2008.
[10] Trapnell, C. and Salzberg, S. How to map billions of short reads onto genomes, *Nature Biotechnology*, 27(5):455–457, 2009.
[11] `http://www.ncbi.nlm.nih.gov/Traces/sra/`
[12] `http://www.genome.ucsc.edu/`