Response to Review Comments

Comments:

(reviewer #1) The paper is well written up to Section 2. In contrast, Section 3 requires readers to have prior knowledge about induced suffix sorting algorithm, and in the current format, it is nearly impossible for readers without background to understand the details, not to mention how to check the correctness.

While I believe that the results in Section 3 are correct, but with the current writing, it is hard for me to verify its correctness. A major revision is needed, most suitably by adding enough examples, and perhaps a brief introduction to induced sorting as well.

(reviewer #3) The only thing I miss throughout the manuscript is an appendix where the checking methods could be illustrated by means of an example. That is, to exemplify the different detailed steps and main structures involved into each proposal, by using a sample input string. I would really appreciate that authors could include that section as, in my opinion, it would help to enhance the paper's content even more.

Response:

We have rewritten the whole paper to make it easier to follow. In Sections 2 and 3, we illustrate the algorithmic framework for the proposed checking methods with each step explained in detail. In Section 4, we evaluate the performance of our programs in comparison with the state-of-the-art sequential and parallel construction solutions. In the same section, we also discuss the ways of improving the current implementations of our programs. For a better understanding of our checking methods, we provide an example in Section 2 to show the process of comparing the target substrings by the fingerprinting technique and another example in Appendix A to show the process of inducing the suffix and LCP arrays by the induced sorting method.

Comments:

(reviewer #1) Your proposed methods require sort(n) I/Os to perform. Theoretically speaking, the fastest suffix array construction method and LCP construction method also require sort(n) I/Os. Am I correct? If this is the case, what is the benefit of applying your method, instead of implementing the above methods (or, asking an independent programmer to implement these methods if you are using that already) and compare the results? To my understanding, a checker should take much less time or I/Os than a brute-force re-implementation. Please comment the above in your revision.

Response:

We conducted more experiments to enhance the content of our paper. As observed from the experimental results in Section 4, our programs are faster than the state-of-the-art sequential construction solution but slower than the current fastest parallel construction solution under the given conditions. It should be pointed out that the time and space complexities of pSAscan and Sparse-Φ are proportional to n^2/M. This is much higher than that of eSAIS and our programs when n increases, and thus poses a strict limitations to the scalability of the parallel construction solution. As reported in [23], eSAIS outperforms pSAscan when the size of the input string is

considerably greater than the memory capacity. **Hence, for a big n, it is more reasonable to compare the results of our programs with that of the sequential construction solution.**

Comments:
(reviewer #3) Besides that, and just as a small detail, it would also be interesting that authors could point out the specific opportunities they find useful to improve their contributions in the near future, at the end of the conclusions.
Response:
In Section 4, we make a discussion on how to improve the implementations of our programs. For example, it can be observed from our experiments that the disk-based sorter has a great influence on the performance of our programs. Currently, we employ the containers provided by the STXXL library to sort fixed-size items using external memory. It is possible to speed up the sorting process by high-performance radix-sort GPUs algorithms.
We point out that the current version of our programs are for experimental study only, and there is a big margin for better implementations. For example, a recent work [20] for engineering the IS method achieved a significant improvement over the previous results [10, 12, 13]. This indicates a great potential for speeding up ProgB and ProgB+, because the induced sorting process is the performance bottleneck for both the programs. **An optimized engineering of our methods is out of the scope of this paper and will be addressed elsewhere.**

Comments:
(reviewer #2) The paper seems to be well-written. But I have a serious problem with the motivation of this work. It is to check suffix trees and LCP arrays produced by some existing algorithms in case of implementation bugs or occasional errors. The Karp-Rabin fingerprinting function is used to do the task.
My concern is, the implementation bugs and occasional errors should be checked by testing codes, not by running another program since the program itself may have implementation bugs or occasional errors. So we have to run a third program to check the second. Finally, we will end up with examining the codes of some algorithm.
The program verification can be useful. But the paper does not fall in this category. It does not contribute anything to improving the efficient construction of suffix trees and LCP arrays. Given this, I find it very difficult to support the acceptance of the paper.
Response:
From our point of view, testing code is commonly used by the programmers to locate programming errors. However, the algorithms proposed recently are becoming more complicated than before, which makes it hard to find all the implementation bugs in the programs. Against this background, some widespread software packages provide users a checker to verify suffix and LCP arrays after construction. In addition to help avoid implementation bugs, a checker is also demanded for arrays constructed by

probabilistic methods. In this case, the arrays are correct with a probability and hence must be verified to ensure its correctness.
In practice, we usually check a constructed array from one builder by comparing it with that from another builder. However, this is not feasible in all the cases, because, for example, a finite-order SA builder is not capable of checking an infinite-order suffix array and vice versa. From this aspect, our first checking method is rather general, it can verify any sparse or full suffix/LCP array of any order.
As shown in the paper, our programs are faster than the state-of-the-art sequential construction solution and more scalable than the fastest parallel construction solution in terms of time and I/O complexities. Therefore, we believe that our programs could be efficient verification solutions distributed with the state-of-the-art SA/LCP builders for the situations where checking must be done immediately after building.

Thanks a lot!