**Response to Review Comments**

**Response to Reviewer #1 and #3:**

We made the following modifications to enhance the content of our paper.

1. We show the framework of the algorithms designed by the proposed checking methods with each step explained in detail in Sections 2 and 3.
2. We evaluate the performance of our programs in comparison with that of the state-of-the-art sequential and parallel construction solutions in Section 4. For a further study, we also discuss the ways of improving the current algorithmic design and implementation.
3. We provide an illustrating example of the IS method in Appendix A.

As observed from the experimental results in Section 4, our programs are faster than the sequential construction solution but 2~3 times slower than the 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. As reported in [20, 23], eSAIS outperforms pSAscan when the size of the input string is considerably greater than the memory capacity. Therefore, it is more reasonable to compare the results of our programs with that of the sequential construction solution for a big n. We mention that the current versions of our programs are for experimental study only, and there is still 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.

In our opinions, a checker should be not only fast but also general. From this aspect, Method A is capable of verifying any (sparse) suffix/LCP array of any order.

**Response to Reviewer #2:**

From our point of view, testing code is commonly used to locate programming errors. However, the algorithms proposed recently are becoming more complicated than before. This makes it difficult for programmers to find all the implementation bugs in their programs. Against this background, some widespread software packages provide users a checker to verify constructed suffix and LCP arrays.

In addition to help avoid implementation bugs, a checker is also demanded for an array constructed by a probabilistic algorithm. In this case, the array is 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 verifying an array built by an infinite-order SA builder and vice versa. This motivates us to design Method A for checking any sparse or full suffix/LCP array of any order.

We hope that the reviewers will be satisfied with our response to their comments. Thank you so much.