Memory Complexity Improvement for the Prime-Box Parallel Search Algorithm

Gregor Eesmaa, Mathias Plans, Roberts Oskars Komarovskis
University of Tartu, Institute of Computer Science, Algorithmics (MTAT.03.238)

Abstract

We focus on reducing space complexity in “Prime Box Parallel Search” by Pandey et al, describing the obtained solution and providing comparison of the methods.

Introduction

The “Prime Box Parallel Search” algorithm by Pandey et al has a serial time complexity of O(m) for dictionary search, where m is the length of the searched word. However, there were some apparent space limitations – as an example, only up to four-letter words could be used, possibly not all.

Indices of words in a lookup-array are computed as products of different prime numbers (Table 1). For example, index of “TAB” would be 71 · 103 · 251 = 1835563. As such, the memory is left underutilised by a rather sparse population of the array. We aim to improve this, while following an otherwise similar approach – retaining the time complexity and the parallelisation scheme.

(*)	A	B	C	···	Z
1	2	3	5	···	101
2	103	107	109	···	239
3	241	251	257	···	397

Table 1. a “Prime Box” by Pandey et al

Implementation

Our approach is to change the “Prime Box” indexing scheme proposed by Pandey et al. Our solution, a Tightly Spaced Indexing Scheme (TSIS) is equivalent to a multi-dimensional lookup table, where every letter supplies an additional dimension. This lets go of multiplication of prime numbers and using addition as the top-level operation, achieving a continuous value space for indices.

TSIS, similarly to the “Prime Box”, can be represented as a table (Table 2), with columns corresponding to letters in the alphabet and rows corresponding to word indices. The first value of a level is the last value of the previous level, initially 1. Every other value is an increment by the first value in the level. For example, index of “TAB” would be 20 + 26 + 2 · 26^2 = 1398.

(+)	A	B	C	···	Z
1	1	2	3	···	26
2	26	52	78	···	676
⋮	⋮	⋮	⋮	⋱	⋮
m	$1 \cdot 26^{m-1}$	$2 \cdot 26^{m-1}$	$3 \cdot 26^{m-1}$	···	26^m

Table 2. TSIS visualised as a table

TSIS can further be reduced to an equation that gives the value of a letter at a specific level (Equation 1). Doing so, no predefined table needs to exist, which is also an improvement over the “Prime Box Parallel Search”. To acquire the index of a word, the individual values for each letter have to be summed together.

$$value = letterValue \cdot 26^{level} \qquad \text{(Eq. 1)}$$

Results

The total memory consumption depends on the longest word required and the alphabet used. The memory consumption can be calculated with Equation 2, where m is the length of the longest word and x is the length of the alphabet. Figures 1 and 2 visualize the memory consumption differences between Prime Box and TSIS algorithms when different alphabet or word sizes are used.

$$\textstyle\sum_{k=1}^{m} x^k \qquad \text{(Eq. 2)}$$

Figures 1. and 2. visualize the memory consumption differences between Primebox and TSIS algorithms when different alphabet or word sizes are used. Additionally, using a 32-bit index (up to 256MiB with a bit array), TSIS can index all words of length up to 6, compared to 3 with Prime Box.

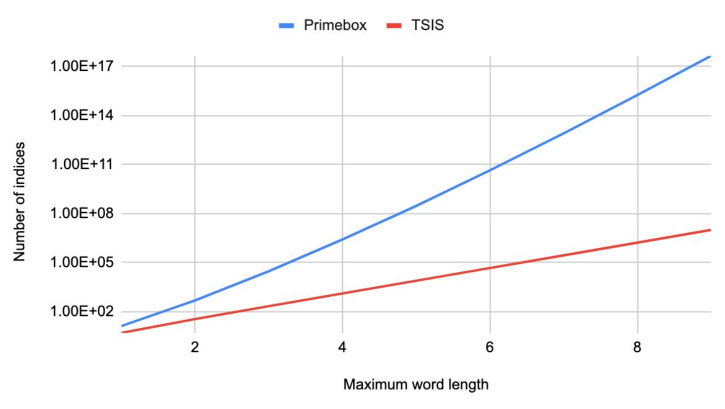

Figure 1. Array size comparison of TSIS and Prime Box algorithms based on maximum word length, with fixed alphabet of 6

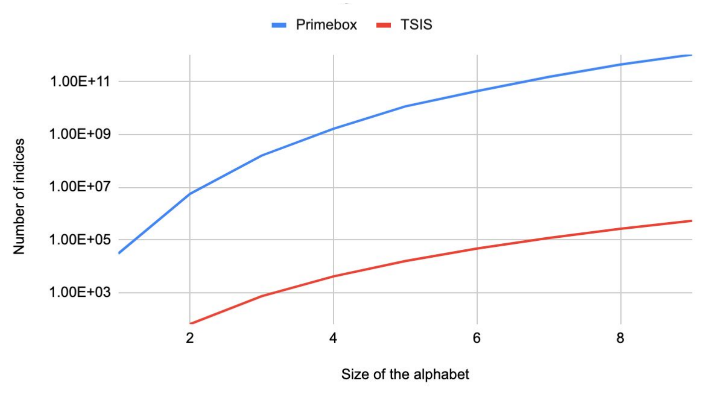

Figure 2. Array size comparison between TSIS and Prime Box algorithms based on alphabet size, with fixed max word length of 6

From a tight index space, it is not possible to improve the space complexity further without compromises to time complexity. However, simulating “hashing” using modulo could be used to trade actual space usage with performance within equivalent complexity bounds.

Conclusions

Proposed TSIS assigns a memory location for every possible letter combination, which still seems impractical, but better than “Prime Box Parallel Search”, while maintaining the same time complexity. The steps of the proposed algorithm can also be parallelised equivalently.

Further work could find better uses for a TSIS-based search. Perhaps by not limiting the alphabet to letters, but phrases, data density could further be increased. There might also be uses where every index has a differently-sized alphabet. Further, TSIS could be used to achieve better-than-ASCII encoding.