An Efficient Composite-Alphabet Transform for String Matching under a Restricted Alphabet Size

Sung-Hwan Kim, Chang-Seok Ock, and Hwan-Gue Cho, Pusan National University, South Korea

Abstract—String matching is the problem that asks to find all occurrences of a short pattern on a relatively long reference string. While a number of methods have been presented, most published implementations assume several restrictions due to some practical issues. In particular, we focus on the restriction of the alphabet size, which is usually set to be 256 in many string matching libraries. When we have to handle strings over an alphabet with the size of more than the limit provided by the given implementation, each character should be represented as a composite alphabet, a combination of two or more characters in the restricted alphabet set. In this case, sometimes it involves potential false positives, which may cause a decline of the performance in output sensitive string matching systems such as the FM-index. In this paper, we empirically compare various configurations of composite alphabets using FM-index, and show how they affect the performance in terms of the number of false positives and the searching time.

Index Terms—Alphabet Size Limit, Composite Alphabet, FM-index, Output Sensitive Function, String Matching.

I. INTRODUCTION

String matching problem involves finding all occurrences of a given short pattern query on a relatively long reference string. This problem has a number of applications such as computer security [1], word recognition [2], data mining [3]-[5], and biological sequence processing [6].

As the result of the importance of the problem, many theoretical solutions to this problem have presented for a long time (see [7] for exact string matching algorithms, [8] for the approximate string matching algorithms). Correspondingly, a number of implementations for string matching and its applications have been published as software [9] or hardware [10].

In practice, however, there are several restrictions in the existing implementations due to the practical issues such as code optimization or some limitations of programming languages. The alphabet size is one of usual constrains existing libraries have. Some implementations only focus on a specific domain such as biological sequences [6] with a very small alphabet set (extremely, the library may support only 'A', 'C', 'G', and 'T'). In this case, the library usually uses several compression techniques so that it is very difficult to change the supported alphabet size. Another case is using the default character size that the language supports as the maximum alphabet size. According to the design, it may involve substantial changes in code to resize the alphabet size. Even without these considerations, modifying the code is so unsafe that has the potential to raise unreported errors.

To deal with this constraint of the alphabet size, we have a very straight-forward solution that can be applied without modifying the code: represents the alphabet set as the combinations of another alphabet set that is much smaller, we call this mapping as alphabet transform. As described in Fig. 1, we can simply represent a large alphabet by compositions of characters from a smaller alphabet set.

![Fig. 1. An example of an alphabet transform from \{a,c,g,t\} into \{0,1\}.](image)

This approach, however, may produce substantial false positives. In Fig. 2, every correct result should occur at the position 2i+1 for some i, since we use a composite alphabet with length 3. However, we can find the query pattern at positions not of the form 2i+1. We have to filter those false positives and extract only positions of 2i+1 to obtain the result.

![Fig. 2 False positive problem. Although the original string does not contain the query, there are some search results from its transformed string.](image)

Unfortunately, some useful string matching algorithms such as FM-index [11] involves costly operations to obtain the positional index of the search result. Moreover, they are usually sensitive to the number of occurrences of the given pattern on the indexed text. Thus false positives lead to the growth of the search time.

Although using a long composite alphabet may reduce the number of false positives by increasing the composition space, it involves longer queries and text that also affect the search performance. In this paper, we analyze this trade-off in an empirical way. By experiments we compare the search performance with respect to various composite alphabet configurations.