Implementing the Aho-Corasick Automata for Phonetic Search

Ondřej Sýkora

Charles University in Prague, Faculty of Mathematics and Physics,
Malostranské náměstí 25, 110 00 Prague 1, Czech Republic
ondrasej@gmail.com

Abstract. In phonetic search, the goal is to find in a text all words with the same pronunciation as the search phrase. The user writes the word down using a different alphabet and transcription rules. Mrázová et al. proposed a new method for phonetic search based on searching for all possible transcriptions with Aho-Corasick automata [8]. Their algorithm offers better precision than the previous existing algorithms for phonetic search, but the size of the search automata grows exponentially with the length of the search phrase. In this paper, we examine the utilisation of the states of the automata created by this approach, and we discuss a memory-efficient implementation of the automata exploiting the fact that many states of the automaton are never visited during the search phase. We demonstrate that this implementation is comparable in search speed to the original automata, but it leads to significant memory savings.

Introduction

In the current world, the amount of information grows at an ever increasing rate and searching for information is one of the most common tasks. Specifying the right query is a difficult task by itself and it is even more complicated when the data is in a foreign language with different transcription rules or alphabet. Phonetic search is an approach to string matching that finds not just the exact search phrase, but it also finds all phrases that are pronounced in a similar way in the language of the text.

Mrázová et al. [8] proposed a new approach to phonetic search that uses transcription rules to create an Aho-Corasick automaton (AC-A) [1] that finds all possible transcriptions of the search phrase. They showed that for Arabic and German it offers better speed and precision than the earlier methods. However, the number of possible transcriptions grows exponentially with the length of the search phrase. In this paper we examine the number of states of the automata that are visited during the search phase and show an efficient implementation of the AC-A that avoids the exponential growth by constructing only those parts of the automaton that are required by the searched text. We show that the efficiency of the search phase is comparable to the original AC-A, but it exhibits remarkably lower memory usage.

Related Work

The first phonetic search algorithm *Soundex* was published by Russel [12] in 1918 to ease searching in indexes of surnames. It assigns a code to each name and there is a match if two names have the same code. The codes are composed of the initial letter of the name and of three numbers derived from the following consonants. There are other algorithms similar to Soundex that use more complex coding schemes, both for English [4,10,14] and other languages, such as German [6,7,11] or Arabic [3].

The Soundex-based algorithms typically have very high recall, but low precision. Mrázová et al. [8] approach the problem by building an AC-A that searches for all words that might be pronounced the same way as the search phrase. They have tested their algorithm on Arabic and German and have shown that it provides better precision than the Soundex algorithms and that for this type of search AC-A are more efficient than other string matching techniques.

Aho-Corasick automata provide strong guarantees on the worst-case time complexity, but the practical efficiency depends significantly on effective utilization of CPU caches and memory transfers [5,13]. The automata can also be implemented effectively using special hardware architectures such as FPGAs [15]. Mrázová and Sýkora have compared the efficiency of a FPGA implementation of the phonetic search algorithm [8] with an implementation running on an Intel CPU. They found that the FPGA implementation can be faster, but the time necessary to build and deploy the automaton outweighs the processing speed, except for very large texts [9].

Complexity of Phonetic Search

The main idea of the approach of Mrázová et al. [8] is to use context-free phonetic transcription rules to create all possible transcriptions of the search phrase entered by the user. An AC-A is then used to search for all these transcriptions in the text. The rules map short strings in the user language to sets of possibly empty strings in the text language. It is common that there are multiple transcriptions for a single sequence of characters or that the rules are applied to overlapping substrings of the search phrase.

Because the transcription rules are context-free, any combination of them can be applied to the search phrase, provided they cover the whole search phrase and do not overlap. As a result, the number of possible possible transcriptions grows exponentially with the length of the search phrase. Table 1 shows the number of possible transcriptions and states of the Aho-Corasick automaton matching all these transcriptions for selected Arabic and German words.

While the number of possible transcriptions grows rapidly with the length of the search phrase, most of these transcriptions do not exist as words in the language of the text. To demonstrate this, we have constructed the automata for selected Arabic and German words and measured how many times each state was visited when processing a 250 MB sample of text extracted from the Arabic and German Wikipedia. The results are summarized in Table 2.

Table 1. Selected Arabic and German words, their Czech transcriptions, number of transcriptions created using the rules, and the number of states of the Aho-Corasick automata

Arabic	Phonetic	Trans.	States	German	Phonetic	Trans.	States
قال	kála	33	88	Tier	týr	55	97
تاكس	taxun	726	1580	Vater	fátr	167	282
كسر	kasrun	2152	4437	schmecken	šmekn	601	641
كيستون	kysatun	8547	17332	Speise	špajze	1656	1040
ماترام	matramun	45237	94714	Zeppelin	cepelin	6867	6587
سمكة	samakatun	270781	553810	Fraulein	frojlajn	15541	10802
تزكرة	tazkyratun	1082157	2214458	Fahrenheit	fárenhajt	107884	45514

Table 2. The average numbers of states of the Aho-Corasick automata created for the selected Arabic and German words that were visited at least *Visits* times when searching in a 250MB text sample

Visits	Arabic	German
1	337.4 (±48.22)	30.11 (±2.45)
10	190.47 (±24.21)	23.01 (±1.86)
100	112.31 (±12.42)	16.14 (±1.26)
1,000	61.94 (±5.92)	11.15 (±0.84)
10,000	31.17 (±2.64)	7.62 (±0.51)
100,000	13.88 (±1.01)	5.28 (±0.4)
1,000,000	5.65 (±0.5)	3.21 (±0.22)
10,000,000	2.2 (±0.07)	1.17 (±0.07)
100,000,000	1.0 (±0.0)	1.0 (±0.0)

The experiment confirmed that most of the states are never visited. In particular, the automaton spends 99 % of time in ca 10 states. These numbers are even more striking if you compare them with the numbers of states shown in Table 1. The automaton construction would be significantly more effective if it could avoid building the states and transitions that are never used. This can be achieved multiple ways:

1. A dictionary can be used to restrict the generated transcriptions to existing words. This is an effective approach, but it reduces the recall of the algorithm and it can be used only for common words. Many words such as names of places and people do not appear in any dictionary. In Arabic text, foreign names can be written down using multiple different transcriptions.
2. The dictionary can be created from the searched text. Creating such a dictionary requires a considerable amount of computing resources and it is not possible if the text is not known in advance, e. g. in network monitoring.

3. Alternatively, one may use the transcription rules to create a non-deterministic finite automaton (NFA) from the search phrase. The automaton is first created as a chain of states accepting exactly the search phrase. Then, edges and sequences of states are added according to the transcription rules. An example of such NFA is shown on Fig. 1. The NFA is very compact, but the time complexity of the search is $O(nl)$, where l is the number of states of the NFA and n is the length of the text. Compared to the AC-A, the NFAs are remarkably slower even for short search phrases.

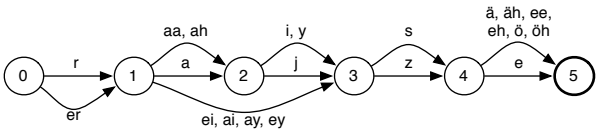

Fig. 1. An example of the NFA created from the transcription rules for the German word *Reise* using the phonetic transcription *rajze*. The final state is denoted by the thick border. Straight edges are for the characters of the phonetic transcription, rounded edges are for transitions created from the rules. For simplicity, multiple transitions between two states are represented by a single arrow with multiple labels.

Lazy Construction of Aho-Corasick Automata

We have shown that in phonetic search, the number of states visited at least once during the search is very low compared to the total number of states. Instead of determining which words appear in the text, our implementation adds states to the automaton lazily when they are required by the text, similar to the lazy evaluation of transitions described by Aho et al. [2, Ch. 3.7]. This is achieved by merging the construction and search phases of the original Aho-Corasick algorithm into a single loop driven by the searched text. In conformity with [1], we will use the notation g for the goto function, f for the failure function, and *out* for the list of words accepted by a state.

During the initialization, a minimal automaton containing only the initial state and its direct descendants is created. The automaton reads one character from the text at a time, and changes the current state using the goto or failure functions g and f in the same way as the original Aho-Corasick algorithm. When a state s is visited for the first time, it is *expanded*, i. e. its direct descendant states are added to the automaton, and the functions f, g, and *out* are updated accordingly. If the state $f(s)$ was not expanded yet, it is expanded before $out(s)$ is updated. Instead of presenting the pseudo-code of the algorithm, we illustrate its operation on Fig. 2. The figure shows how states are created and expanded when searching for the words *he*, *she* and *hers* in the text *usher*.

To be able to expand states when needed, it is necessary to keep more information about the states than in the original algorithm. Each state s needs to

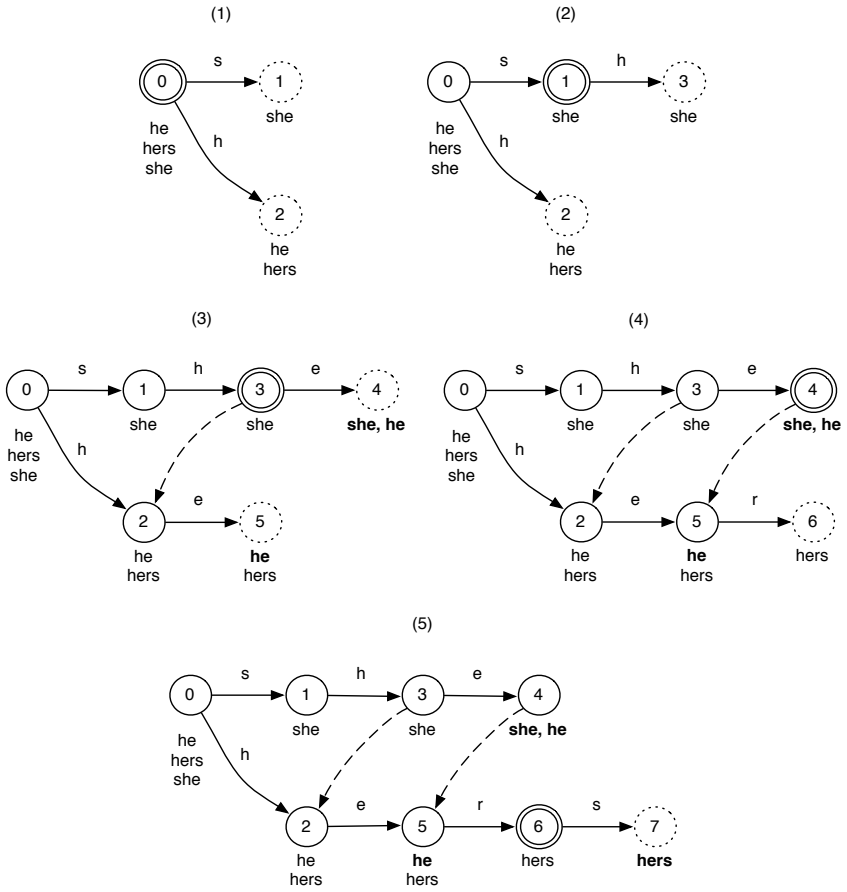

Fig. 2. Operation of the automaton while searching for words *he*, *she*, *hers* in the string *usher*. States are represented as circles. States with dotted border were not expanded yet, the state with double border represents the current state. The words below each state are the search phrases that can be accepted by a descendant of the state; words accepted by the state are printed in bold. Transitions using the goto function are represented as solid lines, failure function is represented by dashed lines. Failure edges to the initial state were ommited for clarity.

store the set of search phrases accepted by the descendants of s. This set can be represented as a set of states of the NFA for the search phrase as described in Sect. 1. In case of phonetic search, the number of states of the NFA is linear in the size of the search phrase, which makes this representation particularly effective.

The correctness of the modified algorithm can be proved the same way as for the original Aho-Corasick algorithm. Moreover, it can also be proved that the modified algorithm has the same asymptotic time complexity.

Experiments

To show the efficiency of the Lazy AC-A and compare it with searching with the original AC-A, we have implemented the algorithm as a C++ program. All tests were run on a laptop with 2.4 GHz Intel Core 2 Duo CPU and 2 GB of RAM, running a 32-bit version of Ubuntu Linux, with code compiled using GCC 4.3. The text used in the experiments was a selection of pages from static dumps of Arabic and German Wikipedia [16]. For pre-processing, all HTML tags were removed, and the documents were merged into a single 250MB file. All times presented in this section include only the time to process the data. The times to load the data and the automaton from the disk are not considered.

We have implemented the version of the algorithm that uses NFAs as a compact representation of the set of search phrases. To further improve the speed of processing of the text, whenever the algorithm uses the function f to do a state transition, it adds this transition to the function g. For the implementation of the original AC-A, we have chosen the faster deterministic version, where all transitions are stored directly in the function g.

The first set of experiments was aimed at the raw speed of string matching. We have used the algorithm with various German and Arabic words, with transcription lengths ranging from four to fourteen characters. We have measured the time necessary to process a single character of the text. In case of the original AC-A, we did not take into account the time necessary to build the automaton. It shows that both for the Lazy AC-A and the original AC-A, the times to perform a single state transition are almost the same, independent on the length of the transcription, and they are almost independent on the language used. The results are summarized in Table 3. It shows, that for both languages, the unmodified AC-A's are approximately 19 % faster than the Lazy AC-A.

Table 3. Time (in nanoseconds) necessary to process a single state transition on the selected German and Arabic words

Language	**AC-A**	**Lazy AC-A**
Arabic	2.72 (±0.07)	3.34 (±0.09)
German	2.69 (±0.06)	3.32 (±0.10)

The amount of memory necessary to represent a single state only depends on the alphabet of the text, but not on the search phrase or the searched text. To assess the memory requirements of the modified algorithm, we have measured the number of expanded and unexpanded states, and compared it to the number of states of the corresponding AC-A. The results are summarized in Table 4. While the number of states of the original AC-A grows exponentially with the length of the phonetic transcription, the size of the NFA grows linearly, and it remains reasonably small even for larger phrases, and also the growth of the number of expanded and unexpanded states of the Lazy AC-A is relatively slow

Table 4. Average numbers of states of the original AC-A, of the NFA created from the search phrase, of expanded states of the Lazy AC-A, and of unexpanded states of the Lazy AC-A for the selected Arabic and German words with different lengths of the phonetic transcription

Length	AC-A States		NFA States		Expanded		Unexpanded	
	Arabic	German	Arabic	German	Arabic	German	Arabic	German
≤ 6	1424	304	56	17	302	53	414	48
7	14925	1372	74	28	404	58	711	68
8	33305	6734	81	26	384	48	688	59
9	263948	58185	99	29	612	49	1217	56
10	613325	11272	100	37	522	49	1122	58
11-14	1447974	150058	116	43	776	48	1405	46

and for German words it even stops at ca 60 states. We assume, that a similar but higher upper bound also exists for the Arabic words.

Discussion

The modified algorithm retains the asymptotic complexity of the original AC-A. On the other hand, while the original AC-A can guarantee that every state transition will be done in a constant time, the modified version of the algorithm does not have this property. The expansion of the failure function f in state s may lead to expanding all states between the current state and the initial state. This may have important consequences in applications like network monitoring, where an attacker can send packets with keywords that would require expanding new states in each step.

While the number of states created by the modified algorithm is usually far lower than m, it is possible to create such set of search phrases and text, that all m states will be created. Except for special cases (e. g. when there is just a single search phrase), the memory requirements can be kept low by disposing states that are not required at the moment, and re-creating them when they are needed again. When the automaton is in state s, then all states $s' : depth(s') > depth(s)$ can be removed safely. Specially, when the automaton is in the initial state, all states except the initial one can be removed. Naturally, this would have a negative impact on the running time.

Conclusions

In this paper, we have shown a memory-effective implementation of the Aho-Corasick algorithm. In the worst case the modified algorithm has asymptotically the same time complexity as the original AC-A, but it only creates states that

are visited by the automaton when processing a given text. This brings a great advantage in applications like phonetic search, where the set of search phrases is extremely large, but it can be represented as a relatively small non-deterministic automaton, and most of the search phrases never appear in a real text. We have shown experimentally that in phonetic search the speed of the modified algorithm is comparable to the original AC-A even if the construction of the AC-A is not taken into account, and in practice it successfully avoids the exponential growth of the number of states.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Communications of the ACM 18(6), 333–340 (1975)
2. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: principles, techniques, and tools. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1986)
3. Al-Shamaa, K.: Arsoundex. `http://www.ar-php.org`, accessed: 2012-03-18
4. Gadd, T.N.: Phoenix: the algorithm. Program: Automated Library and Information Systems 24(4), 363–369 (1990)
5. Holub, J.: Finite automata implementations considering cpu cache. Acta Polytechnica, 47(6) (2007)
6. Michael, J.: Doppelgänger gesucht: Ein Programm für kontextsensitive phonetische Textumwandlung. c'T Magazin für Computer und Technik, vol. 25, p. 252 (1999)
7. Mokotoff, G.: Soundexing and genealogy. Avotaynu – Publisher of Works on Jewish Genealogy, `http://www.avotaynu.com/soundex.html`
8. Mrázová, I., Mráz, F., Petříček, M., Reitermanová, Z.: Phonetic search in foreign texts. In Proceedings of ANNIE'08. pp. 533–540 (2008)
9. Mrázová, I., Sýkora, O.: When is phonetic search with FPGAs efficient? In Proceedings of ICSES'10, pp. 359–362, IEEE (2010)
10. Philips, L.: Hanging on the metaphone. Computer Language Magazine 7(12), 39 (1990)
11. Postel, H.J.: Die Kölner Phonetik. Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse. IBM-Nachrichten 19, 925–931 (1969)
12. Russell, R.C.: U.S. patent No. 1, 261.16 (1918)
13. Scarpazza, D.P., Villa, O., Petrini, F.: High-speed string searching against large dictionaries on the Cell/B.E. processor. In 22nd IEEE International Symposium on Parallel and Distributed Processing, pp. 1–12 (2008)
14. Taft, R.L.: Name search techniques (1970)
15. Tan, L., Sherwood, T.: A high throughput string matching architecture for intrusion detection and prevention. In: Proceedings of ISCA'05. pp. 112–122, IEEE Computer Society, Washington, DC, USA (2005)
16. Wikimedia Foundation: Wikipedia static html dumps. `http://static.wikipedia.org/`