ENTROPY AND COMPLEXITY OF NAIVE PATTERNMATCHING

VOLKER NANNEN

AUGUST 30^{th} 2001

CONTENTS

1. NAIVE PATTERNMATCHING

When using naive patternmatching we run through a string of length n and try to match a pattern of length $m < n$ at each location l of the string. Starting with the first letter we compare all letters of the pattern with the letters of the substring beginning at l until there is a mismatch or all letters match. After that we move on one position in the string and begin afresh. When the pattern matches we compared m times two letters and the cost of the operation was $k = m$ computations.

If string and pattern consist of nothing but the same letter over and over again we have to compare all n letters of the string with all m letters of the pattern. Without further knowledge of the string the computational complexity of naive patternmatching is therefor considered to be $0(m \times n)$ or $O(n^2)$. But if there is variation in the string then with a certain regularity there will be a mismatch at letter $l \leq m$ and we only need to compare l letters. We now want to calculate $E[m]$, the expected number of computations for a pattern of lenth m at all n positions in the string.

In this paper I want to show that in most strings (natural language, DNA) with an entropy $H > 0$ the naive method has an expected linear complexity. Because I make use of entropy and geometric distributions those will be explained first.

2. ENTROPY

Let x^k be a string of length k, produced by some source X with finite memory. The predictability of this string can be measured by its entropy rate H. Claude Shannon[1], the founder of information theory, has defined this H as

$$H = \lim_{k \to \infty} \frac{1}{k} \log \frac{1}{P(x^k)} \tag{1}$$

[1] There is an introduction to his work at http://it.ucsd.edu/IT/Newsletter/archives/wyner/shannonrev.ps. The original text A MATHICAL THEORY OF COMMUNICATION, you can find at http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html

where $P(x^k)$ is the chance that a substring of length k matches with any other substring of length k that is produced by X. The most important conclusion of Shannon was that H always exists if X has finite memory and that $\frac{1}{k}\log\frac{1}{P(x^k)}$ grows monotonously as k grows.

An example of entropy is the amount of structure in an English text. There are 26 letters in the alfabet and if those letters where all used with the same probability and without relation then the entropy was $H = \log(26) = 4,7$. But some letters are used more often then others. If we therefor compare two arbitrary letters from and English text the chance p that it is the same letter is much higher then $\frac{1}{26}$. According to Shannon it is $\frac{1}{16}$.

If we take two arbitrary substrings of 4 letters length from an English text then the chance that both form the letter 'help' and match is higher than that one of them contains the string 'help' and the other contains a string 'pleh', 'lehp', 'hlep' or something like that and that they will not match, even though they contain the same letters. The chance to match substrings of 4 letters length is higher than p^4, the chance which could be calculated from the single letters. Another important fact is that the 4 letters can never show greater chaos than the single letters allow, no matter how weird the language.

Combinations of x letters can show a higher but never a lower chance for a match than the single letters. On the other hand, global H can be used to calculate the highest chance for a match that can be found in combinations of all length because it is calculated for strings of infinit length. The higher the chance for a match the more letters of a pattern will be compared until we find a mismatch and the more costly the computation is. To use global H to calculate the chance for single letters or only a few letters to match usually results in a more costly computation estimate than if we would use the actual chances. Therefor it is a worst case assumption to use H to calculate the maximum chance p to match a letter of the pattern with a letter of the string at any given moment.

$$p = \frac{1}{2^H}, \quad H = \log\frac{1}{p}. \tag{2}$$

This is important to reason independently from the position of the letter in the pattern or in the string and greatly simplifies all calculations.

3. Geometric Distributions

A Bernoulli trial has a chance p of success and a chance $q = (p-1)$ of failure. An example of a Bernoulli trial is to bet on the 6 when casting a dice. Geometric distributions describe sequences of Bernoulli trials. $f(x)$ is the chance to have $x-1$ successes in sequence and to find a failure at trial number x. An expample of such a sequence is to bet that the dice will show a 6 in three successive trials and then another number.

$$f(x) = p^{x-1}q \tag{3}$$

To calculate a geometric distribution we use the following equation:

$$\sum_{k=0}^{m} x^k = \frac{1-x^{m+1}}{1-x} \quad , \quad 0 < x < 1 \tag{4}$$

Because $p < 1$ this gives us an easy to use limit for $m \to \infty$. This and its derivatives are used for the geometric distribution:

$$g(m) = \lim_{m\to\infty} \sum_{k=0}^{m} p^k = \frac{1}{1-p} \tag{5}$$

$$g'(m) = \lim_{m\to\infty} \sum_{k=1}^{m} kp^{k-1} = \frac{1}{(1-p)^2} \tag{6}$$

$$g''(m) = \lim_{m\to\infty} \sum_{k=2}^{m} k(k-1)p^{k-2} = \frac{2}{(1-p)^3} \tag{7}$$

4. Expected Complexity

If we compare a pattern of length m with a substring of the text then we compare the single letters until we find a mismatch or until all letters match. Each single comparision of two letters costs one computation.

Cost of a match: The chance[2] for a match of the whole pattern is p^m and the cost is m computations. Therefor the expected cost of a match is smaller or equal to mp^m computations.

Cost of a mismatch: The chance for a mismatch at the first letter is $(1-p)$. The cost of a mismatch at the first letter is one computation. The expected cost of such a mismatch is $(1-p)$ computations. The chance for a mismatch at letter 2 is $p(1-p)$. Such a mismatch costs 2 computations and the expected cost is $2p(1-p)$. The chance for a mismatch at letter $k \leq m$ is $p^{k-1}(1-p)$ and the expected cost is $kp^{k-1}(1-p)$.

The expected number of computations and with that the expected cost $E[K, m]$ which we need to match the pattern with the substring of the text is the sum of these expected costs and is a geometric series:

$$E[K, m] = mp^m + (1-p)\sum_{k=1}^{m} kp^{k-1}, \tag{8}$$

If $E[K, m]$ rises monotonously as m gets bigger and if $\lim_{m\to\infty} E[K, m]$ exists than this has to be a real upper limit, the maximum expected cost that no pattern can exceed. This must not be confused with the maximum possible cost. This will always be $m \times n$.

$\lim_{m\to\infty} E[K, m]$ can be found with the help of formula (6):

$$\begin{aligned} \lim_{m\to\infty} E[K, m] &= \lim_{m\to\infty} \left(mp^m + (1-p)\sum_{k=1}^{m} kp^{k-1} \right) \\ &= \frac{1}{1-p} \end{aligned} \tag{9}$$

[2]Under the worst case assumption where global entropy is used for all situations.

The prove that $E[K,m]$ rises monotonously:

$$E[K,m+1] - E[K,m] = (m+1)p^{m+1} + (1-p)\sum_{k=1}^{m+1} kp^{k-1} - \left(mp^m + (1-p)\sum_{k=1}^{m} kp^{k-1} \right)$$

(10)

$$\begin{aligned} &= mp^{m+1} + p^{m+1} + (1-p)(m+1)p^m - mp^m \\ &= p^m \left(mp + p + m + 1 - mp - p - m \right) \\ &= p^m \\ &> 0 \end{aligned}$$

Independent of the length m of the pattern the expected number of computations per substring of the text shall not be greater than $\frac{1}{1-p}$. With the naive method a substring begins at each of the $n-m$ positions of the text and is compared with the pattern. Because of this the number of computations for the complete text shall never be greater than $n\frac{1}{1-p}$:

$$E[K,n] < \frac{n}{(1-p)} \tag{11}$$

With $p = \frac{1}{2^H}$ this can also be written as:

$$E[K,n] < \frac{n}{1-\frac{1}{2^H}} \tag{12}$$

The expected complexity is linear in n and besides that depends only on H, which should be constant for any given source. The greater the entropy the smaller p and the more the fraction approaches n. The limit lies at

$$\lim_{H\to\infty} \frac{n}{1-\frac{1}{2^H}} = n. \tag{13}$$

For small H the number of computation rises:

$$\lim_{H\to 0} \frac{n}{1-\frac{1}{2^H}} = \infty. \tag{14}$$

An entropy of 0 means that the text consists of a single letter, repeated over and over. We cannot devide by 0 but the complexity at $H = 0$ according to the worst case assumption which is normally used for the naive patternmatching is $O(n^2)$. This is a trivial case because once we made sure that the entropy is 0 it is of no use at all to search something in the text. A text that possibly but not necessarily has an entropy of 0 belongs to the set of texts that possibly but not neccessarily have an entropy of 0. Because some members of this set will have an entropy > 0 the whole set has an entropy $H > 0$ too. For any non-trivial search the naive method therefor is of the order

$$O\left(n\frac{1}{1-\frac{1}{2^H}}\right) = O\left(n\right). \tag{15}$$

5. Variance

Because we calculated an expected complexity $E[K, m]$ we are interested in the variance σ_m^2.

$$\begin{aligned}\sigma_m^2 &= E[K^2, m] - E[K, m]^2 \\ &= m^2 p^m + (1-p)\sum_{k=1}^{m} k^2 p^{k-1} - E[K, m]^2\end{aligned} \tag{16}$$

σ_m^2 rises monotonously with m. This needs some calculation but can be proven:

$$\begin{aligned}\sigma_{m+1}^2 - \sigma_m^2 &= E[K^2, m+1] - E[K, m+1]^2 - E[K^2, m] + E[K, m]^2 \\ &= (m+1)^2 p^{m+1} + (1-p)\sum_{k=1}^{m+1} k^2 p^{k-1} \\ &\quad - \left(m^2 p^m + (1-p)\sum_{k=1}^{m} k^2 p^{k-1}\right) - E[K, m+1]^2 + E[K, m]^2 \\ &= m^2 p^{m+1} + 2mp^{m+1} + p^{2m+2} + (1-p)(m+1)^2 p^m - m^2 p^m \\ &\quad - \left(E[K, m+1] + E[K, m]\right)\left(E[K, m+1] - E[K, m]\right) \\ &= p^m\left(mp + 2p + p^{m+2} + m - mp + 1 - p - m\right) \\ &\quad - \left(E[K, m+1] + E[K, m]\right)p^m \\ &= p^m\left(p + p^{m+2} + 1 + E[K, m] + E[K, m+1]\right) \\ &> 0\end{aligned} \tag{17}$$

$\lim_{m\to\infty}\sigma_m^2$ therefor will not be exceeded by any pattern. With (5)-(7) we can find this limit:

$$\begin{aligned}\lim_{m\to\infty}\sigma_m^2 &= \lim_{m\to\infty}\left(m^2p^m+(1-p)\sum_{k=1}^{m}k^2p^{k-1}-E[K,m]^2\right)\\ &= \lim_{m\to\infty}\left(m^2p^m+p(1-p)\sum_{k=2}^{m}k(k-1)p^{k-2}\right.\\ &\qquad\left.+(1-p)\sum_{k=1}^{m}kp^{k-1}-E[K,m]^2\right)\\ &= \frac{2p(1-p)}{(1-p)^3}+\frac{1-p}{(1-p)^2}-\frac{1}{(1-p)^2}\\ &= \frac{p}{(1-p)^2}\end{aligned} \tag{18}$$

σ_m^2 is independent for all n positions of the string and therefor the global variance is

$$\begin{aligned}\sigma_{n,m}^2 &= n\sigma_m^2\\ &< n\frac{p}{(1-p)^2}\\ &= n\frac{\frac{1}{2^H}}{(1-\frac{1}{2^H})^2}\end{aligned} \tag{19}$$

σ_n^2 too is linear in n and besides that only dependent on H. For great H σ_n^2 becomes small and for small H σ_n^2 becomes great. The limits lie at

$$\lim_{H\to\infty}\sigma^2=0 \tag{20}$$

$$\lim_{H\to 0}\sigma^2=\infty \tag{21}$$

6. Examples

Counter examples: Arguments against the use of the naive method are tasks like finding 'aaaab' in 'aaaaaaaaab' or 'abababababc' in 'abababababababababc'. These are examples of extremely low entropies. String and pattern have very little variation. In such a case the value of the fraction gets big. If all letters were the same the entropy was 0 and in (15) you would devide by zero, which is not possible.

Example: an environment where 9 out of 10 letters are 'a' and 1 out of 10 is a 'b'. This leads to strings as 'aaaaaaaaabaaaaaaaaaaaaabaaaa' and has an expected entropy of

$$\log\left(\frac{1}{\frac{9}{10}\frac{9}{10}+\frac{1}{10}\frac{1}{10}}\right)=0,29 \tag{22}$$

per letter, calculated from the chance $p = 0,82$ to match an arbitrary letter with any other arbitrary letter. The complexity is not heigher than $O(5,6n)$ but the variance is $\sigma^2 = 25,3n$.

DNA: In DNA there are 4 different nucleotides, which amounts to an entropy of $\log(4) = 2$. Nucleotides form groups of 3 for the coding of an amino acid. There are 20 amino acids but these can be encoded in different ways so that almost all 64 states of a group have a meaning and are used in nature, but not with the same probability. The entropy of nucleotides in real DNA therefor is somewhat smaller than 2, let's say 1,6. The chance to match nucleotides is $p = \frac{1}{2^{1,6}} = 0,33$ and you get a complexity of $O(1,5n)$ with $\sigma^2 = 0,74n$.

Natural language: The entropy of an arbitrary letter in an English text is $\log(26) = 4,03$ but the entropy of a real English text is much lower, according to Shannon about $1,3$. In that case $p = 0,4$ and the complexity is $O(1,68n)$ with $\sigma^2 = 1,15n$.

7. Conclusion

The entropy-based evaluation of naive patternmatching in a normal environment leads to linear complexity, even for large patterns and big databases. This stands in sharp contrast to the claim that this method is of geometric complexity. Other algorithms could be reevaluated in the same way and might show similar surprising results.

An argument against the use of entropy for the estimation of complexity might be that the data emitted by a source can change and under adverse circumstances can still lead to a geometric complexity. But as long as it is not proven that a given source has an entropy of 0 even the entropy of unreliable sources has to be considered as $H > 0$ and the naive method stays linear. For applications which work on fixed databases, e.g. a text corpus or a DNA database, this is not relevant at all because the entropy cannot change. What is more, once the entropy is known it can be used to accurately evaluate a search algorithm.

Another argument against expected complexity is the variance. Normally the complexity of an algorithm is based on the worst case assumption. This sort of complexity can guarantee the maximum amount of computation that an algorithm shall ever need. Against this I can say what has been shown about naive patternmatching: the variance in the normal case is not of a higher order than the expected complexity. In the case of naive patternmatching, by adding a value $c\,\sigma^2$ to the expected complexity $O(n)$ you can arbitrarily and effectively minimize the chance to exceed the expected complexity. And even so $O(n) + c\,\sigma^2$ stays linear in n because σ^2 is linear in n.