[illegible]

J. Inform. Process. Cybernet. **EIK 29** (1993) 5, 283-292
(formerly: Elektron. Inform.verarb. Kybernet.)

Search Trees and Search Graphs for Markov Sources

By Günter Hotz

Abstract: We consider the search problem in ordered sets in the case that query sources have some memory. We will show that making use of this property one can find algorithms which are much more efficient than common search tree approaches that take only probability distributions into account but not their behavior over the time. Astonishingly, such query sources seem not to have been considered before.

1. Introduction

Let $(X, <, =)$ be an ordered set. We want to have a data structure that allows an efficient search for elements from X if the tests "<" and "=" are available. We will assume that for each $x \in X$ there is a distribution

$$p_x : X \to [0, 1],$$

which gives the probability that a query for x is followed by a query for y. Usually, these probabilities are described by a matrix

$$P = (p_x(y))_{x,y \in X},$$

and P is called a first order Markov process. Such Markov processes can be represented by graphs whose nodes are given by X. Two nodes x, y are connected by an edge s from x to y iff $p_x(y) \neq 0$. In this case, s is labelled by $p_x(y)$. The Markov process shown in Figure 1 is degenerated in some sense: If the process reaches x_4, it will never leave this state. In the sequel, we will exclude such processes by requiring that the graph is strongly connected, i.e. for each pair $(x, y) \in X^2$ there is a path of equally oriented edges from x to y. Obviously, this property is equivalent to the fact that the probability of a transition from x to y in the course of the process is higher than 0. The subgraph formed by x_1, x_2, x_3 in Figure 1 is strongly connected.

Let us now consider sequences of events $X_i \subset X$ at time $i = 1, 2, \ldots$. The probability $p_{xy}(n)$ of a transition from x to y in n steps is recursively given by

$$p_{xy}(n) = \sum_{z \in X} p_{xz}(r) \cdot p_{zy}(n - r),$$

where $0 < r < n$. With

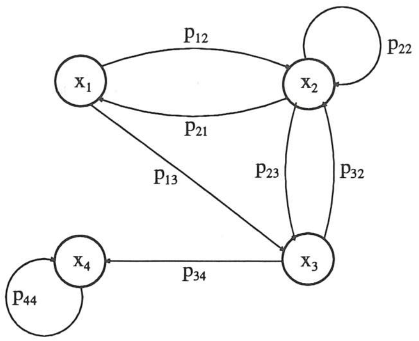


Figure 1:

$$P_n = (p_{xy}(n))_{x,y \in X}$$

this recursion can be expressed by the matrix multiplication

$$P_n = P_r \cdot P_{n-r}.$$

In particular, it is $P_n = P_1^n$.

A Markov process is said to be *ergodic* if the limes

$$p_{xy} = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} p_{xy}(i)$$

exists and is independent of x. In this case, P defines a probability distribution on X in a natural way, and we write $p(y)$ instead of $p_{x,y}$. A sufficient criterion for the ergodicity of P is that for all x, y, n it holds $p_{xy}(n) \neq 0$. Note that strong connectivity of the graph is not sufficient for that. In the following part of this paper, we will assume ergodicity of P without stating it again. The expression

$$H(P) = \sum_{x \in X} p(x) H_x$$

with

$$H_x = -\sum_{y \in X} p_x(y) \cdot \log p_x(y)$$

is called the entropy of P. In the special case $p_x(y) = p_{x'}(y)$ for all $x, x' \in X$, it obviously is $H_x = H_{x'}$, and it follows that

$$H(P) = H(p) = -\sum p(x) \log p(x).$$

Generally, $H(P)$ and $H(p)$ may be very different. This observation in connection with the coding theorems of information theory and with the relation between coding and the construction of search trees is the starting point of this note.

2. A good coding for Markov sources

If A is a set, A^* denotes the set of all finite sequences over A including the empty sequence ε. For $u, v \in A^*$, $u \cdot v$ is the concatenation of u and v. The length of u is denoted by $|u|$. A^+ means the set of all finite sequences $\neq \varepsilon$ over A. Obviously, it is $A^+ = A \cdot A^*$ if we extend the product "$\cdot$" to subsets of A^*. A subset $C \subset A^*$ is said to be *prefixfree* iff $C \cap C \cdot A^+ = \emptyset$. Thus, in this case, there is no element $u \cdot v \in C$ for some $u \in C$ such that $v \in A^+$.

The mapping

$$c : M \to A^*$$

is prefixfree if both (1) and (2) are true:

(1) c is injective,
(2) $c(M)$ is prefixfree in A^*.

Shannon's coding theorem for *noiseless* zero-memory sources tells us:

There exist optimal prefixfree codings $c : X^n \to A^*$, and these codings satisfy

$$\frac{1}{\log m} H(p) \leq \frac{1}{n} \cdot \sum_{x \in X^n} p(x) \cdot |c(x)| \leq \frac{1}{\log m} H(p) + \frac{1}{n},$$

where $m = \text{card}(A)$.

For $A = \{0, 1\}$ and $n = 1$, we get

$$H(p) \leq \sum_{x \in X} p(x) \cdot |c(x)| \leq H(p) + 1.$$

Thus, these codings are optimal in the sense that they minimize the expected code length.

A corresponding theorem also holds for Markov processes P. Usually, this theorem is formulated only for a noisy channel. Hence, the case $n = 1$ is not of interest there. But for $n > 1$ the application of the coding theorems to search trees leads to impractical solutions. This may be the reason why this case has not been considered in the context of search problems. However, it is not difficult to derive a theorem for P from the theorem given above.

For each $x \in X$, we choose a coding

$$c_x : X \to \{0, 1\}^*$$

with property

$$H_x \leq \sum_{y \in X} p_x(y) \cdot |c_x(y)| < H_x + 1.$$

For brevity we write

$$L_x(k) = \sum_{y \in X^k} p_x(y) \cdot |c_x(y)|.$$

Now we define for a special initial state x_0

$$c_0(x_1 x_2 \ldots x_k) = c_{x_0}(x_1) \sqcup c_{x_1}(x_2) \sqcup \ldots \sqcup c_{x_{k-1}}(x_k),$$

where $\sqcup$ is a separation symbol, which is different from 0 and 1. Then we have for $u = x_1 x_2 \dots x_k$

$$\begin{aligned} L_{x_0}(k) &= \sum_{i=1}^{k} \sum_{u \in X^k} p_{x_0}(u) \cdot |c_{x_i}(x_{i+1})| + k - 1 \\ &= L_{x_0}(1) + \left(\sum_{i=1}^{k-1} \sum_{x_i \in X} p_{x_0 x_i}(i) L_{x_i}(1) \right) + k - 1. \end{aligned}$$

We divide by k and rearrange by elements from X. Thus we have

$$\frac{1}{k-1} \cdot L_{x_0}(k) = \sum_{x \in X} \left(\frac{1}{k-1} \sum_{i=1}^{k-1} p_{x_0 x}(i) \right) \cdot L_x(1) + 1 + \frac{1}{k-1} \cdot L_{x_0}(1).$$

From this one gets:

Theorem *The coding of the Markov source (X, P) is prefixfree and on the average asymptotically transmits between $H(P) + 1$ and $H(P) + 2$ binary symbols per $x \in X$.*

Proof. Setting

$$L = \sum_{x \in X} p(x) L_x(1)$$

we get from

$$H_x \le L_x(1) \le H_x + 1$$

and the inequality proven above

$$\frac{1}{k} \cdot L_{x_o}(k) \longrightarrow_{k \to \infty} L + 1$$

$$H(P) + 1 \le \lim_{k \to \infty} \frac{1}{k} L_{x_0}(k) < H(P) + 2$$

$\square$

Remark: The coding c_0 may be non optimal. We used the alphabet $0, 1, \sqcup$ for the coding of X and, hence, get only $\frac{1}{\log 3} H(P)$ as lower bound from Shannon's theorem. However, one can easily use such a binary coding for $\sqcup$ that c_0 remains prefixfree, and the additive contribution $|c_0(\sqcup)|$ appears instead of the factor $\frac{1}{\log 3}$.

3. Prefixfree search trees

Let W be a finite set, for instance the German vocabulary. If we want to look up the English translation of the German word $w \in W$ in a German-English dictionary, we first must find the entry for w in the dictionary. For that we use the lexicographical order of words in correct spelling. If a computer has to do this task, we somehow have to store W in the computer. It is not necessary to do the coding in a symbol by symbol manner, but we can employ some easy to compute function

$$\sigma : W \to X,$$

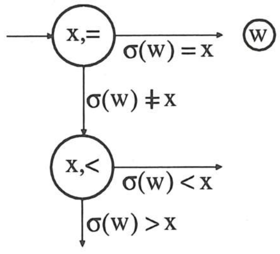
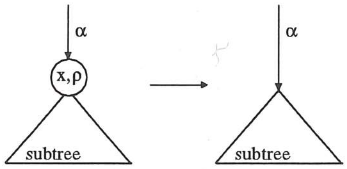


Figure 2:

such that for example $\sigma(w)$ has enough space in a memory cell of the machine. $\sigma(w)$ is called a key for w. For the sake of simplicity we assume that σ is injective. Hence, it is easy to store some information in connection with $\sigma(w)$ which tells us where the translation of w can be found.

We build up the data structure for searching as it is shown in Figure 2. We construct a tree whose node set equals the set $W \cup (X,=) \cup (X,<) \cup \{\$\}$. The nodes W are terminal nodes of the tree. From nodes $(x,=)$ two edges are starting; one of these pointing to w, the other one to $(x,<)$. From $(x,<)$ two edges are starting, one ending in some node $(x_1,=)$ with $x_1 > x$ and the other one ending in some node $(x_2,=)$ with $x_2 > x$ if there exists such a key. If there is none such key, the corresponding edge is missing. If we assume that the dictionary is complete with respect to the queries, the corresponding test will never be true. By labelling each edge of a valid test "=" and "<" by 1 and 0, resp., we obviously get a prefixfree code for W if we map to w the labelling of the unique path from the root to node w. This tree may contain nodes from which only one edge is starting. Such nodes are eliminated by the following rule:



Here $\rho \in \{<,=\}$. The tree is called *reduced* if this rule can not be applied. Obviously, trees which can be transformed into each other by reductions yield the same output. By reductions our tree may loose its regular structure which is characterized by the alternate tests "<" and "=". However, this is not essential. Moreover, this suggests the generalization to allow individual predicates for the node tests. This is interesting in the context of estimating the running time of computer programs.

Now one can see that, the other way, each reduced prefixfree coding $c : W \to \{0,1\}^*$ can be obtained by the above construction. For that one defines

$$X = \left\{ \begin{array}{ll} u \in \{0,1\}^* | & u \text{ proper prefix of } c(w), w \in W \\ & \text{or } u = c(w) \text{ for } c(w) = v0 \end{array} \right\}$$

and

$$u0 < u < u1 \text{ for each } u \in \{0,1\}^*.$$

Now let

$$\sigma(w) = \left\{ \begin{array}{ll} c(w) & \text{for } c(w) = u0 \\ u & \text{for } c(w) = u1. \end{array} \right.$$

Since c is a prefixfree coding, σ is injective. If for σ the construction described above with subsequent tree reduction is done, one gets back the prefix code C.

The time for searching w in the tree is given by the length of the path $c(w)$ which starts in the tree's root and ends in node w. Hence, the expected search time in the tree is equal to the expected code length. Note that this is not true for all search trees that have been considered in literature. In particular, it has to be noted that we did not require that the coding c preserves X's order in $\{0,1\}^*$. Of course, we therefore have the problem of calculating σ. However, compared to other search trees, losses in search times of prefixfree trees are so slight that they do not affect the benefits gained by making use of the query source's memory. Hence, for the sake of simplicity, the restriction to prefixfree search trees is justified. Nevertheless, I would like to add that one has to put up with slight poorer bounds for the optimal access time if one employs alphabetical codings, i.e. codings which preserve a given order [1,2,3,4]. In this case, one only has

$$H(p) \leq \sum_W p(w)|c(w)| < H(p) + 2.$$

4. Search graphs and reductions in search graphs

To keep the notation consistent with Section 2., we will write again X instead of W. Inner nodes of the search tree are not needed any more in the sequel.

Actually, the complete representation of the continuous search in a search tree is described by a graph since the algorithm returns to the root each time a search problem has been solved. In the case of query sources with memory, this completion of the tree by "return edges" is essential. Let (X, P) be a Markov process and $c_0 : X^* \to \{0,1,\sqcup\}^*$ the coding given earlier. We will construct a search graph for c_0 whose paths are labelled by $c_0(u)$ if $u = (x_1, x_2, \ldots, x_k)$ is the sequence of queries.

Let S_x for $x \in X$ be an optimal search tree for the distribution $p_x : X \to [0,1]$. The nodes of S_x are denoted by $\{x\} \times X$ where x is an index which guarantees that S_x and $S_{x'}$, $x \neq x'$, have no nodes in common. Now we will consider a *graph* G for (X, P) whose node set is $X \times X$, that means the union of the node sets of all trees S_x. G's edge set comprises all edges of the S_x's and, in addition to that, contains the following edges: Each node (x, y) of S_x is connected to S_y's root by an edge labelled $\sqcup$. This completely defines $G = G(P)$.

Hence, we get the answer to a query sequence $x_1, x_2, \dots, x_k$ by staring in the root x_0 of some tree, calling x_1 in this tree, then jumping to S_{x_1}'s root, looking there for x_2 and so on until we reach S_{x_k}'s root. The labelling of this path in G is just $c_0(x_1, \dots, x_k)\sqcup$. Thus, it follows:

Theorem *Searching in G asymptotically takes time L with*

$$H(P) \leq L < H(P) + 2$$

in optimal prefixfree search trees and

$$H(P) \leq L < H(P) + 3$$

in optimal alphabetical prefixfree search trees.

Proof. The proof immediately follows from the construction and from the bounds for the expected code length. □

This good search time has to be paid by a large amount of memory space. Instead of space requirements $O(\text{card}(X))$ we here have $O((\text{card}(X))^2)$. Now the question arises in what situations $G(P)$ can be reduced.

We will consider the following special Markov process which can be looked upon as a *multiple language dictionary*[1].

Let $\text{card}(X) = \mathcal{N}$ and $\varepsilon > 0$. Assume that there is a decomposition

$$X = X_1 \cup X_2 \cup \ldots \cup X_K$$

where $X_i \cap X_j = \emptyset$ for $i \neq j$. For $a \in X_i$ and $b \notin X_i$ let

$$p_a(b) \leq \frac{\varepsilon}{\mathcal{N}}.$$

Hence, we have

$$p_a(X \setminus X_i) \leq \varepsilon \cdot \frac{\text{card}\ (X \setminus X_i)}{\mathcal{N}} < \varepsilon.$$

Now for each X_i, we set up a search graph $G_i = G(P_i)$, where P_i is the Markov process for $X_i \cup \{\$i\}$ which is the same as P for G_i as long as transitions from elements $x \in X_i$ to $y \in X_i$ are concerned. Instead of transitions $x \to y$, $y \notin X_i$, we introduce

$$p_x^{(i)}(\$i) = p_x(X \setminus X_i).$$

Moreover, we set for $p^{(i)}$

$$p_{\$i}^{(i)}(\$i) = p_{X \setminus X_i}(X \setminus X_i)$$

and for any fixed $a \in X_i$

$$p_{\$i}^{(i)}(a) = 1 - p_{X \setminus X_i}(X \setminus X_i).$$

Now we define $\tilde{G}(P)$ by means of $G_i = G(P_i)$ by redirecting edges in $G(P_i)$ which end in $\$i$ to their proper target in G_j. Hence, we have as expected search time in $\tilde{G}$

[1] Imagine one single dictionary which comprises English-German, French-German, Spanish-German, and so on.

$$L(\tilde{G}) \leq \sum p(X_i)L(G_i) + \varepsilon.$$

From the estimations

$$H(P_i) \leq L(G_i) < H(P_i) + 2$$

we get

$$\sum_i p(X_i)H(P_i) \leq L(\tilde{G}) < \sum p(X_i)H(P_i) + 2 + \varepsilon.$$

Now it is for $x \neq \$i$

$$\begin{aligned} -H_x(P_i) &= \sum_{X_i} p_x(y) \log p_x(y) + p_x(\$i) \log p_x(\$i) \\ &= -H_x(P) + O(\varepsilon) \end{aligned}$$

and

$$\begin{aligned} -H_{\$i}(P_i) &= \sum_{y \in X_i} p_{\$i}(y) \log p_{\$}(y) + p_{\$i} \log p_{\$i} \\ &= O(\varepsilon). \end{aligned}$$

It follows that

$$\begin{aligned} \sum_i p(X_i)H(P_i) &= \sum_i p(X_i) \sum_{y \in X_i \cup \{\$i\}} \frac{p(y)}{p(X_i)} H_y(P_i) \\ &= \sum_{y \in X} p(y)(H_y(P)) + O(\varepsilon)) = H(P) + O(\varepsilon). \end{aligned}$$

Hence, we have for alphabetical codings

$$H(P) + O(\varepsilon) \leq L(G) < H(P) + 3 + O(\varepsilon).$$

The benefit of this decomposition is that the search tree now has only

$$\sum_{i=1}^{K} N_i^2 < N^2$$

edges ~~nodes~~ instead of N^2 edges ~~nodes~~ before.

Definition If P fulfills the conditions given above, $X_1, \ldots, X_K$ are called ε-components of P.

With this definition we can sum up the result in the following theorem:

Theorem *If $X = X_1 \cup \ldots \cup X_K$ defines a decomposition of X in ε-components, the expected search time $L(\tilde{G})$ in the corresponding search graph $\tilde{G}$ will fulfill the following inequality*

$$H(P) + O(\varepsilon) < L(\tilde{G}) < H(P) + 3 + O(\varepsilon).$$

We will consider a further reduction of the search graph. Let for a fixed $x, x' \in X$, $x \neq x'$

$$|p_x(y) - p_{x'}(y)| < \varepsilon \text{ for all } y \in X \text{ and } p_x(y) = 0 \Leftrightarrow p_{x'}(y) = 0$$

and let $\tilde{G}$ be the search graph which results from $G(P)$ if $S_{x'}$ in $G(P)$ is replaced by S_x.

Lemma *On the conditions given above, it holds* $|L(G) - L(\tilde{G})| = O(\varepsilon)$.

Proof. It suffices to compare the expected search time in S_x for distribution $p_{x'}$ to that one for distribution p_x. If we denote the first search time by $\tilde{L}_{x'}$, we have

$$\begin{aligned} |L_x - \tilde{L}_{x'}| &= |\sum_y (p_x(y) - p_{x'}(y))|c(y)|| \\ &\leq \varepsilon \cdot \sum_y |c(y)| = O(\varepsilon). \end{aligned}$$

The second condition is technical. It guarantees $H_{x'} = H_x + O(\varepsilon)$ which simplifies the proof. □

For the case that P is decomposable in ε-components and that $p_x(y) = p_{x'}(y)$ if x, x' are lying in the same component, we have:

Theorem *On the conditions just given, there exists a search graph G consisting of $\mathcal{N} = card(X)$ nodes which has $H(P)$ as approximative expected search time.*

Proof. Since $p_x(y)$ as a function of x is constant within each component X_i, the search graph G_i can be replaced by a tree with return edges from each node to its root. This means a sequence of reductions like that one described in our lemma. Since $\varepsilon = 0$, the search time does not increase. Now we compose these graphs for X_i to the graph $\tilde{G}$ as it has been described in the previous theorem. Hence, it immediately follows the assertion of the theorem. □

Finally, we give a numerical example. Let $K, n \in \mathbf{N}$ fixed, $\mathcal{N} = K^{n+1}$, and $\text{card}(X_i) = \text{card}(X_j)$ for $i, j \doteq 1, \ldots, K^n$. Moreover, we assume that $p_x(y) = p_{x'}(y')$, for $x, x', y, y' \in X_i$ for all i and that $p_x(X_j) = \varepsilon$ for $i \neq j$. This implies $p(x) = p(x')$ for $x, x' \in X$. The entropy of p as an independent probability distribution is

$$H(p) = (n+1) \log K. \qquad \approx \lceil \log_2 \text{card} X \rceil \cdot \text{für } K = 2$$

On the other hand, we have

$$H(P) = \log K + O(\varepsilon) \qquad \simeq 1$$

as approximation for the search time in our tree.

also die Methoden divergieren mit dem Logarithmus von der Größe des Wörterbuches.

$\mathcal{N}$ („Duden") $\approx 10000 \approx 2^{13}$: Also Faktor 14

Acknowledgements

I thank Joachim Hartmann and Kurt Mehlhorn for critical reading the manuscript.

References

[1] *Ahlswede, R., I. Wegner:* Suchprobleme, Teubner Studienbücher Mathematik, 1979.
[2] *Knuth, D. E.:* The art of computer programming, Vol. 3 Sorting and Searching, Addison Wesley Publishing Co., 1973.

*) Wenn die Maschine teurer ist als die Wartezeit des Anfragenden, lohnt sich alphabetisches Sortieren der Anfrage; unabhängige Verteilung wird zu Markovprozeß. | Interessantes Problem, das sich lösen läßt.
. . . .
Wann ist Sortieren + Suchen in kleineren Baum besser als im großen Baum?

[3] *Mehlhorn, K.:* Sorting and Searching, EATCS Monographs and Theoretical Computer Science, Springer Verlag, 1984.

[4] *Ottmann, T., P. Widmayer:* Algorithmen und Datenstrukturen, BI Wissenschaftsverlag, 1990.

(Received: September 29, 1993)

Author's address:

G. Hotz
Fachbereich Informatik
Universität des Saarlandes
Saarbrücken, Germany