UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

Solving Matrix Recurrences with Applications

Ricardo A. Baeza-Yates
Gaston H. Gonnet

Data Structuring Group
Research Report
CS-89-16

May, 1989

Solving Matrix Recurrences with Applications

Ricardo A. Baeza-Yates
Gaston H. Gonnet

Data Structuring Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1 *

Abstract

We propose a general method to solve matrix recurrences. The method also makes easier to solve some scalar recurrences. We apply this technique to the average case analysis of balanced search trees and tries. In particular we give the exact solution for a fringe analysis problem, that was unknown before.

1 Introduction

Solving recurrences is at the heart of analysis of algorithms. Classical introductions to the topic can be found in [Knu69, Lue80, GK82, GKP89]. However, matrix recurrences, as a topic, are only treated in differential equations books, and most of the time for very simple cases involving constant matrices (for example, see Hildebrand [Hil68]).

There are several algorithms that naturally lead to matrix recurrences in search trees and digital trees. Also, scalar recurrences of high order, can be expressed as a matrix recurrence of lower order [Hil68].

We propose a general method to solve matrix recurrences where all terms can be expressed as scalar functions of the independent variable and constant matrices. We also obtain the asymptotic behaviour of the solution.

We begin by giving the main ideas behind our method, followed by two applications: the solution of a fringe analysis type equation (from the expected behaviour of balanced search trees); and the expected complexity of some algorithms over tries (regular expression searching and partial match queries).

*The work of the first author was supported by an Ontario Graduate scholarship, and the second author by a Natural Sciences and Engineering Research Council of Canada Grant No. A-3353.

2 Solving Matrix Recurrences

By using matrix algebra it is possible to simplify the analysis of a set of recurrence equations. For example, consider the following problem:

$$\begin{aligned} f_1(n) &= f_1(n/2) + f_2(n/2) + c_1 \ , \quad f_1(1) = 0 \ , \\ f_2(n) &= \alpha f_2(n/2) + f_1(n/2) + c_2 \ , \quad f_2(1) = 0 \ , \end{aligned}$$

for n a power of 2 and $\alpha \geq 0$. Suppose that $\alpha = 0$, then we can reduce the problem to

$$f_1(n) = f_1(n/2) + f_1(n/4) + c_1 + c_2 \ , \quad f_1(2) = c_1, \quad f_1(1) = 0 \ .$$

Even this particular case appears to be difficult to solve.

However, writing the problem as a matrix recurrence, we obtain

$$\vec{f}(n) = \mathbf{H}\vec{f}(n/2) + \vec{c} \ ,$$

and this recurrence may be solved by simple iteration

$$\vec{f}(n) = \sum_{k=0}^{\log_2 n - 1} \mathbf{H}^k \vec{c} \ .$$

In many cases, this solution is not enough, and we need asymptotic results to obtain the complexity of the algorithm. Decomposing $\mathbf{H}$ in its Jordan normal form [Gan59], namely

$$\mathbf{H} = \mathbf{P}\ \mathbf{J}\ \mathbf{P}^{-1}$$

where $\mathbf{J}$ is a block diagonal matrix of the form

$$\mathbf{J} = \begin{bmatrix} J_1 & 0 & .. & .. \\ 0 & J_2 & 0 & .. \\ .. & .. & .. & .. \\ .. & .. & 0 & J_t \end{bmatrix} \ ,$$

J_i is a $m_i \times m_i$ square matrix of the form

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & .. \\ 0 & \lambda_i & 1 & 0 \\ .. & .. & .. & 1 \\ .. & .. & 0 & \lambda_i \end{bmatrix} \ ,$$

where each λ_i is an eigenvalue of $\mathbf{H}$ with multiplicity m_i, and $\mathbf{P}$ has as columns the respective eigenvectors. Hence, we have

$$\mathbf{H}^k = \mathbf{P}\mathbf{J}^k\mathbf{P}^{-1}$$

where $\mathbf{J}^k$ is the block diagonal matrix $[J_i^k]$, and each J_i^k is of the form [Gan59]

$$J_i^k = \begin{bmatrix} \lambda_i^k & k\lambda_i^{k-1} & \frac{k(k-1)}{2}\lambda_i^{k-2} & .. & \binom{k}{m_i-1}\lambda_i^{k+1-m_i} \\ 0 & \lambda_i^k & k\lambda_i^{k-1} & .. & \binom{k}{m_i-2}\lambda_i^{k+2-m_i} \\ 0 & 0 & \lambda_i^k & .. & \binom{k}{m_i-3}\lambda_i^{k+3-m_i} \\ .. & .. & .. & .. & .. \\ .. & .. & .. & 0 & \lambda_i^k \end{bmatrix} .$$

Hence, we obtain the exact solution for $\vec{f}(n)$ as a function of the eigenvalues of $\mathbf{H}$, that is

$$\vec{f}(n) = \mathbf{P}\left(\sum_{k=0}^{\log_2 n-1} \mathbf{J}^k\right)\mathbf{P}^{-1}\vec{c} .$$

Hence, we have sums of the form

$$M_j = \sum_{k=0}^{\log_2 n-1} \binom{k}{j}\lambda^{k-j} .$$

In fact, we have hidden all the dependencies of the recurrences inside the matrices until this last step, where known summations that will depend on the eigenvalues of $\mathbf{H}$ will have to be solved. In our case, we have that ($\lambda > 0$)

$$M_j = \begin{cases} O((\log_2 n)^j n^{\log_2 \lambda}) & \lambda \neq 1 \\ O((\log_2 n)^{j+1}) & \lambda = 1 \end{cases} .$$

Therefore the higher order term of $\vec{f}(n)$ is given by the eigenvalue of maximum value and its multiplicity. In our example, the eigenvalues are $1+\sqrt{\alpha}$ and $1-\sqrt{\alpha}$. Then,

$$\vec{f}(n) = \begin{cases} O(n^{\log_2(1+\sqrt{\alpha})}) & \alpha > 0 \\ O((\log_2 n)^2) & \alpha = 0 \end{cases} .$$

If the exact value of the eigenvalues is unknown, we can bound them using characteristics of the matrix $\mathbf{H}$.

In summary, the main steps are

1. Express the set of recurrences as a matrix recurrence where all functions of the independent variable (n in the example) are scalars, and the matrices have constant entries.

2. Solve the recurrence using standard methods for scalar recurrences, but using matrix algebra (non-commutative multiplication, inverse instead of division, etc.).

3. Decompose all the matrices into their normal Jordan forms, expressing the solution as a function of the eigenvalues of the matrices.

4. Compute or bound the eigenvalues to obtain the asymptotic complexity of the solution.

3 Fringe Analysis

Fringe analysis was formally introduced by Yao in 1974 [Yao74, Yao78] and was also discovered independently by Nakamura and Mizoguchi [NM78]. However, the first fringe type analysis was done by Knuth [Knu73, Solution to problem 6.2.4.10, pages 679-680]. Fringe analysis is a method used to analyze search trees that considers only the bottom part or *fringe* of the tree. From the behaviour of the subtrees on the fringe, it is possible to obtain bounds on most complexity measures of the complete tree and some exact results.

Classical fringe analysis considers only insertions, and the model used is that the $n!$ possible permutations of the n keys used as input are equally likely. A search tree build under this model is called a *random* tree. This is equivalent to say that the n-th insertion has the same probability to fall in any of the $n+1$ leaves of the tree.

The theory of fringe analysis was formalized in Eisenbarth *et al.* [EZG+82]. The fringe of the tree is defined in terms of a finite collection C of trees. A collection C is *closed* if the effect of an insertion only depends in the subtree of the fringe in where is performed, and produces one or more members of the same collection.

Type 1 Type 2

Figure 1: 2-3 tree fringe collection of height one

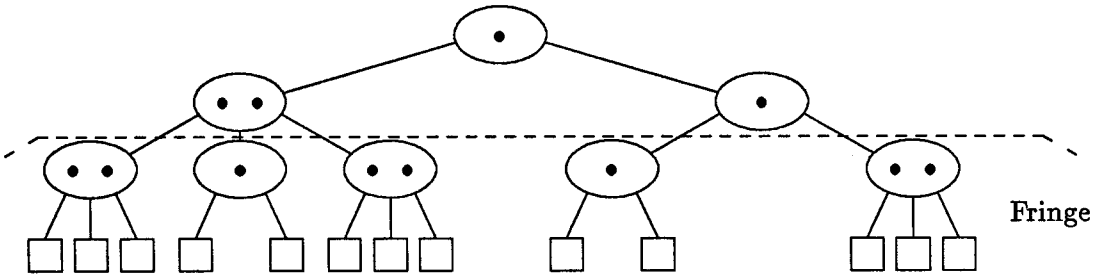

Figure 2: A 2-3 tree and its fringe of height one

Example 1. Figure 1 shows a tree collection that defines a fringe of height one in a 2-3 tree, while figure 2 shows a 2-3 tree and the fringe corresponding to this collection. The composition of the fringe can be described by the number of subtrees of each type [Yao78], or by the probability that a randomly chosen leaf belongs to each member in C [EZG+82]. An insertion in a type I subtree produces a type II subtree, while an insertion in a type II subtree produces two type I subtrees. This

process defines a Markov chain [EZG+82]. Let $A_i(n)$ be the expected number of subtrees of type i in the fringe, L_i the number of leaves in type i, and

$$p_i(n) = \frac{A_i(n)L_i}{n+1}$$

be the probability of a leaf belonging to a subtree of type i in a random 2-3 tree with n keys. Then,

$$\begin{aligned} A_1(n) &= A_1(n-1) - \frac{2A_1(n)}{n} + 2\frac{2A_2(n)}{n} \quad \text{and} \\ A_2(n) &= A_2(n-1) - \frac{3A_2(n)}{n} + \frac{2A_1(n)}{n} \quad . \end{aligned}$$

Using $p_i(n)$ and matrix notation we have

$$\vec{P}(n) = (\mathbf{I} + \frac{1}{n+1}\mathbf{H})\vec{P}(n-1) \ ,$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{H}$ is called the *transition* matrix, given by

$$\mathbf{H} = \begin{bmatrix} -3 & 4 \\ 3 & -4 \end{bmatrix} .$$

By using the condition $p_1(n) + p_2(n) = 1$ we obtain $\vec{P}(n) = [4/7, 3/7]^T$ for $n \geq 6$. ■

In general, $\mathbf{H} = \mathbf{H_2} - \mathbf{H_1}$ where $\mathbf{H_2}$ represents the transitions from one type to another type, that is, h_{2ij} is the the probability that an insertion in type j produces one or more subtrees of type i times the number of leaves of type i created; and $\mathbf{H_1} = \text{diag}(L_i) + \mathbf{I}$ represents the leaves lost by each type after an insertion plus one. It is not difficult to see that $det(\mathbf{H}) = 0$.

A fringe analysis is *connected* [EZG+82], if there exist $\mathbf{H}_i$ such that $det(\mathbf{H}_i) \neq 0$, where $\mathbf{H}_i$ is the matrix $\mathbf{H}$ with the i-th row and i-th column deleted.

The solution to a connected fringe analysis is given by the following theorem:

THEOREM **3.1**

$$\vec{P}(n) = \vec{x} + O(n^{Re(\lambda_2)}) \ ,$$

where $\vec{x}$ is the solution to $\mathbf{H}\vec{x} = 0$, normalized such that $\sum p_i = 1$, and λ_2 is the eigenvalue with second largest real part, and $Re(\lambda_2) < 0$ [EZG+82]. ■

In many cases, it is possible to use the structure of $\mathbf{H}$ to simplify the solution of the system of equations [BYP85]. In the following paragraphs we give the solution of the recurrence for all n, and we show that the second order term also depends on the multiplicity of λ_2.

We start from the matrix recurrence equation

$$\vec{P}(n) = (\mathbf{I} + \frac{1}{n+1}\mathbf{H})\vec{P}(n-1) \ .$$

Let m be the dimension of the recurrence. Because the rank of $\mathbf{H}$ is $m-1$, we include the condition that $\sum_i p_i(n) = 1$, to obtain the following equation

$$\vec{P}(n) = (\mathbf{I} + \frac{1}{n+1}\mathbf{T})\vec{P}(n-1) \ + \ \frac{1}{n+1}\vec{F} \ ,$$

where $\mathbf{T} = \mathbf{H}_{(m-1)\times(m-1)} - \mathbf{H}'$, $\vec{F}$ is the last column of $\mathbf{H}$ up to the $(m-1)$-th row, and $\mathbf{H}'$ is a $(m-1)\times(m-1)$ matrix where every column is equal to $\vec{F}$.

THEOREM **3.2** *The solution for a connected fringe analysis of a closed collection of trees is given by*

$$\vec{P}(n) = -\mathbf{T}^{-1}F \ + \ \frac{(-1)^{n+1}}{(n+1)!}(-\mathbf{T}-\mathbf{I})^{\underline{n+1}}\vec{C} \ ,$$

where $x^{\underline{n}} = (x-n+1)\cdots(x-1)x$ denotes descendent factorials, with $\vec{C}$ obtained from the initial condition

$$\vec{P}(n_0) = [1,0,0,...,0]^T \ ,$$

where n_0 is the number of elements in the smallest subtree type of the fringe collection.

Proof: Introducing the generating function

$$\vec{P}(z) = \sum_{n\geq 0} \vec{P}(n)z^n \ ,$$

in the matrix recurrence, we obtain the following first order non-linear differential equation

$$\frac{d\vec{P}(z)}{dz} = \left(\frac{2z-1}{z(1-z)}\mathbf{I} + \frac{1}{1-z}\mathbf{T}\right)\vec{P}(z) \ + \ \frac{1}{z(1-z)^2}\vec{F} \ ,$$

whose solution is

$$\vec{P}(z) = \frac{1}{z}e^{-(\mathbf{T}+\mathbf{I})\ln(1-z)}\vec{C} \ - \ \frac{1}{z(1-z)}\mathbf{T}^{-1}\vec{F} \ ,$$

where $\vec{C}$ is obtained from the initial condition.

For $n \geq n_0$, we have

$$\vec{P}(n) = [z^n]\vec{P}(z) = -\mathbf{T}^{-1}\vec{F} \ + \ \sum_{k\geq 0}(-\mathbf{T}-\mathbf{I})^k[z^{n+1}]\frac{\ln^k(1-z)}{k!}\vec{C} \ ,$$

where $[z^n]P(z)$ denotes the coefficient in z^n of $P(z)$. But

$$[z^n]\frac{\ln^k(1-z)}{k!} = \frac{(-1)^n}{n!}S_n^{(k)} ,$$

for $n \geq k$ where $S_n^{(k)}$ denotes Stirling numbers of the first kind [Knu69]. Therefore, for $n \geq n_0$

$$\vec{P}(n) = -\mathbf{T}^{-1}\vec{F} + \frac{(-1)^{n+1}}{(n+1)!}\sum_{k=0}^{n+1} S_{n+1}^{(k)}(-\mathbf{T}-\mathbf{I})^k\vec{C} ,$$

or [Knu69]

$$\vec{P}(n) = -\mathbf{T}^{-1}\vec{F} + \frac{(-1)^{n+1}}{(n+1)!}(-\mathbf{T}-\mathbf{I})^{\underline{n+1}}\vec{C} .$$

■

Example 2. In Example 1, we have $\vec{P}(1) = [1, 0]$, $T = -7$, $F = 4$ and $C = 1/35$, obtaining

$$P(z) = \frac{4}{7z(1-z)} + \frac{(1-z)^6}{35z} ,$$

or

$$p_1(n) = \frac{4}{7} - \frac{(-1)^n}{35}\binom{6}{n+1} .$$

Note that the second term is 0 for $n > 5$. ■

The next step is to obtain the asymptotic behaviour of the solution. Let $\mathbf{R} = -\mathbf{T} - \mathbf{I}$. We want the asymptotic value of

$$\vec{\epsilon}(n) = \frac{(-1)^{n+1}}{(n+1)!}\sum_{k=0}^{n+1} S_{n+1}^{(k)}\mathbf{R}^k\vec{C}$$

THEOREM **3.3** *Asymptotically, every component of* $\vec{\epsilon}(n)$ *is*

$$O(n^{Re(\lambda_1)}\log^{m-1} n) ,$$

where λ_1 *is the eigenvalue of* $\mathbf{T}$ *with largest real component, and* m *is its multiplicity. Note that* λ_1 *is equal to the the second largest eigenvalue of* $\mathbf{H}$ *and that* $Re(\lambda_1) < 0$.

Proof: We decompose $\mathbf{R}$ in its upper normal Jordan form [Gan59]

$$\mathbf{R} = \mathbf{P}\ \mathbf{J}\ \mathbf{P}^{-1} .$$

Then, we obtain

$$\vec{\epsilon}(n) = \mathbf{P}\frac{(-1)^{n+1}}{(n+1)!}\sum_{k=0}^{n+1} S_{n+1}^{(k)}\mathbf{J}^k\mathbf{P}^{-1}\vec{C} .$$

Therefore, we have summations of the form

$$M_j = \frac{(-1)^{n+1}}{(n+1)!} \sum_{k=0}^{n+1} S_{n+1}^{(k)} \binom{k}{j} \lambda^{k-j} .$$

By using a well known combinatorial relation [Knu69] we have

$$M_0 = (-1)^{n+1} \binom{\lambda}{n+1} .$$

From Eisenbarth *et al.* [EZG+82] we know that $Re(\lambda) > 0$ for λ an eigenvalue of $-\mathbf{T}$. If λ is an integer we have $M_0 = 0$ for $n \geq \lambda$. If λ is not an integer (that is, λ is real or complex), we have [Knu69]

$$M_0 = (-1)^{n+1} \binom{\lambda}{n+1} = \binom{n-\lambda-1}{-\lambda-1} = \frac{\Gamma(n-\lambda)}{\Gamma(-\lambda)\Gamma(n+1)} .$$

Therefore, asymptotically, we have

$$M_0 = \frac{n^{-(\lambda+1)}}{\Gamma(-\lambda)} + O(n^{-\lambda-2})$$

(see [AS72] for details about the Γ function).

Analogously, we obtain

$$M_j = \Big((\Psi(n-\lambda) - \Psi(-\lambda))^j + O(1/n) \Big) M_0 = O(M_0 \log^j n) ,$$

where $\Psi(x) = \frac{d}{dx}(\ln \Gamma(x)) = \ln n + O(1)$ [AS72].

But if λ is an eigenvalue of $\mathbf{T}$, then $-(\lambda+1)$ is an eigenvalue of $-\mathbf{T} - \mathbf{I}$. Then, we can state that every component of $\vec{\epsilon}(n)$ is

$$O(n^{Re(\lambda_1)} \log^{m-1} n) ,$$

where λ_1 is the eigenvalue of $\mathbf{T}$ with largest real component, and m is its multiplicity. ■

In all the analyses that appear in the literature, the multiplicity of the second eigenvalue is 1. However, in general this may be not true.

Example 3. For the second order analysis of 2-3 trees [Yao78, EZG+82] (a seven type collection), the second eigenvalue of $\mathbf{H}$ is $-6.55 + 6.25i$, and then, the order of each component in $\vec{\epsilon}(n)$ is proportional to

$$99.01 \cos(6.25 \ln n) n^{-6.55} + O(n^{-7.55}) .$$

Note that the periodic term has logarithmic period. ■

4 Algorithms on Tries

The analysis of several algorithms on tries can be expressed by matrix recurrence equations. Examples are the simulation of finite automata over a binary trie [BYG89] and partial match queries in k-d tries [FP86]. Here, we extend both cases to an alphabet of size σ (that is, σ-tries). A σ-trie is a σ-ary digital tree, which each node has a descendant for every symbol of the alphabet.

One algorithm to search what strings are members of the language defined by a given regular expression, is to simulate the corresponding deterministic finite automaton over the trie built from the set of strings. The main steps of the algorithm are [BYG89]:

1. Convert the regular expression passed as a query into a minimized DFA without outgoing transitions from final states (see justification in step 3).

2. Simulate the DFA on the σ-trie from all strings. That is, associate the root of the tree with the initial state, and, for any internal node associated with state i, associate its k-th descendant with state $j = \delta(i, x_k)$ where δ is the transition function of the DFA and x_k is the k-th symbol of the alphabet.

3. For every node of the trie associated with a final state, accept the whole subtree and halt the search in that subtree. (For this reason, we do not need outgoing transitions in final states).

4. On reaching an external node, run the remainder of the automaton on the single string determined by this external node.

It is shown in [BYG89] that the expected number of internal nodes visited by the automaton starting from a non-final state i, in a σ-trie of n random infinite strings can be expressed as

$$N_i(n) = 1 + \frac{1}{\sigma^n} \sum_{j_1+\cdots+j_\sigma=n} \frac{n!}{j_1!\cdots j_\sigma!}(N_{j_1}(n_1) + \cdots + N_{j_\sigma}(n_\sigma)) \quad (n > 1)$$

where $j_k = \delta(i, x_k)$ and x_k is the k-th symbol of the alphabet. This is similar to the analysis for the expected number of nodes in σ-tries [Knu73]. For final states we have $N_f(n) = 1$ and for undefined states we have $N_{undef}(n) = 0$ for all n. The initial conditions are $N_i(0) = N_i(1) = 0$.

Introducing exponential generating functions in the above equation, that is

$$N_i(z) = \sum_{n\geq 0} N_i(n)\frac{z^n}{n!}$$

we obtain

$$N_i(z) = e^{z/\sigma}(N_{j_1}(z/\sigma) + \cdots + N_{j_\sigma}(z/\sigma)) \; + \; e^z - 1 - z$$

Writing all the equations as a matrix functional equation, we have

$$\vec{N}(z) = e^{z/\sigma}\mathbf{H}\vec{N}(z/\sigma) \; + \; f(z)\vec{F}$$

where $f(z) = e^z - 1 - z$ and $\mathbf{H}$ is the incidence matrix of the automaton (that is, an $s \times s$ matrix where s is the number of states in the DFA and h_{ij} is the number of transitions from state i to state j, with $h_{ij} = 0$ if i is a final state) and $\vec{F}$ be a constant vector such that F_i is 1 for all i. The initial state is labelled 1.

This functional equation may be solved formally by iteration [FP86] obtaining

$$\vec{N}(z) = \sum_{k\geq 0} e^{z(1-1/\sigma^k)} f(z/\sigma^k)\mathbf{H}^k\vec{F}$$

From here, it is easy to obtain $\vec{N}(n)/n!$ by computing $[z^n]\vec{N}(z)$ using the series expansion of e^x. Then, we have

$$\vec{N}(n) = \sum_{k\geq 0} \tau_{n,k}\mathbf{H}^k\vec{F}$$

where

$$\tau_{n,k} = 1 - \left(1 - \frac{1}{\sigma^k}\right)^n - \frac{n}{\sigma^k}\left(1 - \frac{1}{\sigma^k}\right)^{n-1} .$$

The next step, it is to obtain the asymptotic value of $\vec{N}(n)$. Decomposing $\mathbf{H}$ in its upper normal Jordan form [Gan59], and following the same steps as before, we have

$$\vec{N}(n) = \sum_{k\geq 0} \tau_{n,k}\mathbf{H}^k\vec{F} = \mathbf{P}\left(\sum_{k\geq 0} \tau_{n,k}\mathbf{J}^k\right)\mathbf{P}^{-1}\vec{F} .$$

Then, as in Section 2, we have summations of the form

$$M_j \; = \; \sum_{k\geq 0} \tau_{n,k}\binom{k}{j}\lambda^{k-j} .$$

The convergence of M_j is guaranteed by the fact that, for fixed n and large k we have

$$\tau_{n,k} \approx 1 - e^{-n/\sigma^k} - \frac{n}{\sigma^k}e^{-(n-1)/\sigma^k} \; = \; O\left(\frac{n^2}{\sigma^{2k}}\right) .$$

The asymptotic value of M_0 for $\sigma = 2$ has already been obtained in Flajolet and Puech [FP86]. In a similar manner ($\lambda > 1$) we obtain

$$M_0 = \gamma(\log_\sigma n)n^{\log_\sigma \lambda} + O(1) ,$$

where $\gamma(x)$ is a periodic function of x with period 1, mean value depending only on λ, and with small amplitude. Similarly, for $\lambda > 1$

$$M_j = \frac{\gamma(\log_\sigma n)}{j!}\left(\frac{\log_\sigma n}{\lambda}\right)^j n^{\log_\sigma \lambda} + O(M_{j-1}) .$$

If $\lambda = 1$, we have $M_j = O((\log_\sigma n)^{j+1})$. Then, for λ_i we have that the main order term is

$$\gamma_i(\log_\sigma n)\,(\log_\sigma n)^{m_i-1}\, n^{\log_\sigma \lambda_i}\ .$$

The higher order term of $N_1(n)$ is given by $\lambda_1 = \max_i(|\lambda_i|)$. In the case that there are more than one eigenvalue with the same modulus, we select λ_1 to be the one with largest multiplicity.

In [BYG89] is shown that $1 \leq \lambda_1 < 2$ for any regular expression when $\sigma = 2$, hence the number of visited nodes is sublinear in n. Analogously, it is possible to prove that $1 \leq \lambda_1 < \sigma$ for our case. A similar analysis is done to conclude that the expected number of comparisons performed by the automaton on a single string is bounded by a constant independent of n [BYG89].

Example 4. Let A be the deterministic finite automaton of the regular expression $((0+2)(1+0))^*1((1+2)(1+0))^*0$ over a ternary alphabet (see figure 3).

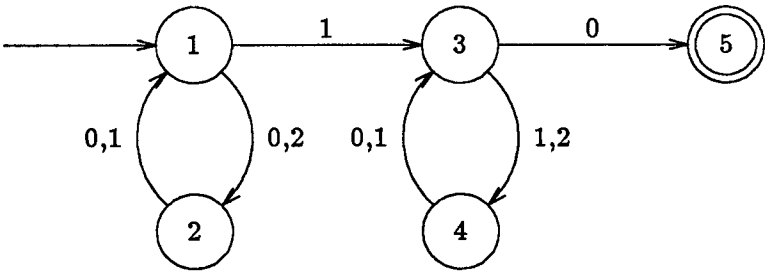

Figure 3: Minimal state deterministic finite automaton for $((0+2)(1+0))^*1((1+2)(1+0))^*0$.

The incidence matrix for A (1 is the initial state, and 5 is the final state) is

$$\mathbf{H} = \begin{bmatrix} 0 & 2 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The eigenvalues of $\mathbf{H}$ are 2 and -2, each one with multiplicity 2; and 0. The expected number of visited nodes is

$$N_1(n) = \gamma(\log_2 n)\log_2 n\ n^{\log_3 2} + O(n^{\log_3 2}) = O(\log_2 n\ n^{0.63})\ .$$

■

In the case of partial match queries in k-d-tries of Flajolet and Puech [FP86], the analysis is similar to an automaton with a single cycle of length k. Namely, a partial match query over k attributes, consists in a string of length k, where some of the attributes are specified and others are unspecified. The attributes are considered to

be encoded using σ symbols, and the keys stored in a σ-trie, such that the branching is decided cyclically in one of the k attributes, like in a k-d tree [BF79].

In this case, the recurrences can be expressed as a single one, and in [FP86] a scalar recurrence is solved. We use our technique to simplify the analysis. We see the partial match query as a regular expression with a single cycle where we follow one descendant if the attribute is specified, or all the descendants if the attribute is unspecified. Let u be the number of unspecified attributes. Then, the matrix $\mathbf{H}$ consists only in one upper diagonal $(H_{1,2}, \ldots, H_{k-1,k})$, plus one entry $(H_{k,1})$, where u elements have the value σ, and $k-u$ the value 1. Hence, there is a real eigenvalue λ of maximal modulus (Perron-Frobenius theorem [Gan59]) of value $\lambda_1 = \sigma^{u/k}$ and multiplicity 1.

Using the solution given for the general automaton, with the only difference that the period of the oscillating function is divided by the number of attributes k (because we have to inspect k levels, one per attribute, to discriminate one set of keys), we have that the expected number of comparisons (visited nodes) to search the query is

$$C_n = \gamma\left(\frac{\log_\sigma n}{k}\right) n^{u/k} .$$

Therefore, the only difference between the binary case and the case $\sigma > 2$ is that the period of γ decreases when σ increases.

References

[AS72] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions.* Dover, New York, 1972.

[BF79] J.L. Bentley and J.H. Friedman. Data structures for range searching. *ACM C. Surveys*, 11(4):397–409, Dec 1979.

[BYG89] R. Baeza-Yates and G.H. Gonnet. Efficient text searching of regular expressions. In *ICALP'89*, Stresa, Italy, July 1989.

[BYP85] R.A. Baeza-Yates and P.V. Poblete. Reduction of the transition matrix of a fringe analysis and its application to the analysis of 2-3 trees. In *5th International Conference in Computer Science*, pages 56–82, Santiago, Chile, 1985.

[EZG+82] B. Eisenbarth, N. Ziviani, Gaston H. Gonnet, Kurt Mehlhorn, and Derick Wood. The theory of fringe analysis and its application to 2-3 trees and B-trees. *Information and Control*, 55(1):125–174, Oct 1982.

[FP86] P. Flajolet and C. Puech. Tree structures for partial match retrieval. *J.ACM*, 33:371–407, 1986.

[Gan59] F.R. Gantmacher. *The Theory of Matrices (2 Vols).* Chelsea Publishing Company, New York, 1959.

[GK82] D. Greene and D. Knuth. *Mathematics for the Analysis of Algorithms (2nd ed.).* Birkhauser, Boston, 1982.

[GKP89] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, Mass., 1989.

[Hil68] F Hildebrand. *Finite-Difference Equations and Simulations.* Prentice-Hall, Englewood Cliffs, New Jersey, 1968.

[Knu69] D.E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, Reading, Mass., 1969.

[Knu73] D.E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Reading, Mass., 1973.

[Lue80] G. Lueker. Some techniques for solving recurrences. *ACM Compuying Surveys*, 12(4):419–436, 1980.

[NM78] T. Nakamura and T. Mizoguchi. An analysis of storage utilization factor in block split data structuring scheme. In *VLDB*, volume 4, pages 489–495, Berlin, Sep 1978.

[Yao74] A.C-C. Yao. On random 3-2 trees. Technical Report UIUCDCS-R-74-679, Departament of Computer Science, University of Illinois at Urbana, Oct 1974.

[Yao78] A.C-C. Yao. On random 2-3 trees. *Acta Informatica*, 9(2):159–170, 1978.