```
Foresight_Ok(n,constrained,curr)
    Tentatively schedule n in the current instruction, curr. If there is a conflict, return FALSE.
    Update absolute times (tentatively).
    Order constrained by minimum absolute time, ties are broken by maximum absolute time.
    For each m ∈ constrained
        Place m in the earliest instruction allowable.
return(can all elements of constrained be placed?)
```

Fig. 6. Foresight_Ok algorithm.

TABLE I
DIFFERENCES IN FAILURE AND TIME

	No Foresight		Foresight		Incremental		%
	# Failures	Time	# Failures	Time	# Failures	Time	
A	13	86.7	0	186.9	0	123.9	63
B	7	73.7	0	136.5	0	97.4	62
C	6	88.0	0	137.2	0	109.2	57
D	6	88.8	0	96.4	0	95.4	13

using the foresight algorithm is 62.8 units, while the excess time of using the incremental algorithm is only 23.7 units, so the excess time saved by using the incremental is 62% $((62.8 - 23.7)/62.8)$.

The need for incremental compaction grows with the number of finite maximum arcs. Overall, the incremental algorithm saves around 48% of excess time, and neither algorithm fails in compacting the examples. However, since this is still an increase over no foresight, it is better to use the incremental foresight algorithm only when the DPS fails.

III. CONCLUSIONS

Compaction with timing constraints requires new techniques. Foresighted compaction is very effective in reducing failure inherent in a greedy compaction algorithm. Incremental foresighted compaction reduces the cost to tolerable levels, but even then should only be used when discriminating compaction fails.

REFERENCES

[1] J. A. Fisher, D. Landskov, and B. D. Shriver, "Microcode compaction: Looking backward and looking forward," in *Proc. Nat. Comput. Conf.*, vol. 50, Montvale, NJ, July 1981. AFIPS Press, pp. 95–102.

[2] D. Landskov, S. Davidson, B. D. Shriver, and P. W. Mallett, "Local microcode compaction techniques," *ACM Comput. Surveys*, vol. 12, no. 3, pp. 261–294, Sept. 1980.

[3] T. Nakatani and K. Ebcioğlu, "Using a lookahead window in compaction-based parallelzing compiler," in *Proc. 23rd Microprogramming Workshop (MICRO-23)*, Orlando, FL, Nov. 1990.

[4] U. Banerjee, S. Shen, D. J. Kuck, and R. A. Towle, "Time and parallel processor bounds for fortran-like loops," *IEEE Trans. Comput.*, vol. C-28, no. 9, pp. 660–670, Sept. 1979.

[5] S. R. Vegdahl, "Local code generation and compaction in optimizing microcode compilers," Ph.D. dissertation, Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1982.

[6] K. V. Palem and B. B. Simons, "Scheduling time-critical instructions on risc machines," in *Proc. Seventeenth Annu. ACM Symp. Principles Programming Languages*, Jan. 1990, pp. 270–280.

[7] A. Aiken and A. Nicolau, "Optimal loop parallelization," in *Proc. SIGPLAN '88 Conf. Programming Language Design and Implementation*, Atlanta, GA, June 1988, pp. 308–317.

[8] A. Nicolau and R. Postasman, "An environment for the development of microcode for pipelined architectures," in *Proc. 23rd Symp. and Workshop Microprogramming and Microarchitecture*, Orlando, FL, Nov. 1990, pp. 69–79.

[9] B. Su, S. Ding, J. Wang, and J. Xia, "Microcode compaction with timing constraints," in *Proc. 20th Microprogramming Workshop (MICRO-20)*, Colorado Springs, CO, Dec. 1987, pp. 59–68.

[10] V. H. Allan and R. A. Mueller, "Compaction with general synchronous timing," *IEEE Trans. Software Eng.*, vol. 14, no. 5, pp. 595–599, May 1988.

[11] H. F. Smith, *Data Structures Form and Function*. San Diego, CA: Harcourt Brace Jovanovich, 1987.

[12] V. H. Allan, "A critical analysis of the global optimization problem for horizontal microcode," Ph.D. dissertation, Comput. Sci. Dep., Colorado State Univ., Fort Collins, CO 80523, 1986.

[13] P. Wijaya, "Incremental foresighted microcode compaction," Master's thesis, Utah State Univ., Logan, UT, 1990.

[14] P. Wijaya and V. H. Allan, "Incremental foresighted local compaction," in *Proc. 22th Microprogramming Workshop (MICRO-22)*, Dublin, Ireland, Aug. 1989.

On the Complexity of Search Algorithms

Kuo-Liang Chung, Wen-Chin Chen, and Ferng-Ching Lin

***Abstract*—Consider the average complexity for searching a record in a sorted file of records that are stored on a tape. We analyze in this note the average complexity of four search algorithms, namely, sequential search, binary search, Fibonacci search, and a modified version of Fibonacci search. Our theoretical results are consistent with the recent simulation results by Nishihara and Nishino. These results show that sequential search, Fibonacci search, and modified Fibonacci search are all better than binary search on a tape.**

***Index Terms*—Complexity, Fibonacci search, generating function, modified Fibonacci search, sequential search.**

I. INTRODUCTION

Consider a search key and a sorted file of records stored on a tape. The goal of a search algorithm is to find the record saved on the tape that matches the given key.

In this note we investigate the average complexity of four known search algorithms, namely, sequential search (SS), binary search (BS), Fibonacci search (FS), and a modified version of Fibonacci search (mFS). For the purpose of analyzing the complexity of these four search algorithms, we shall only concern the average total distance that the reading head is required to move for searching a record. This

Manuscript received September 21, 1990; revised October 6, 1991. This work was supported in part by the National Science Council of R.O.C. under Grant NSC79-0408-E002-07.

K.-L. Chung is with the Department of Information Management, National Taiwan Institute of Technology, Taipei, Taiwan 10772, R.O.C.

W.-C. Chen and F.-C. Lin are with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 10764, R.O.C.

IEEE Log Number 9103107.

is due to the fact that moving the reading head of the tape takes much longer time than reading and comparing the current record with the search key. Hence, head movement time, which is proportional to the traveling distance, is the dominant factor in the search time.

To make the complexity analysis easier, we shall assume that the sorted file containing $F_n - 1$ records, where F_n is the nth Fibonacci number. Our main results derived in Section III show that the complexities of SS, BS, FS, and mFS are asymptotically equal to $0.5F_n$, F_n, $0.882F_n$, and $0.809F_n$, respectively. Our results indicate that algorithm SS is optimal and gives 50% better efficiency than BS; mFS gives 19.1% better efficiency than BS; and FS is 11.8% better than BS. Some of these results are consistent with the recent simulation results by Nishihara and Nishino [4].

II. The Performance Equations

This section first describes the basic concepts of the BS, FS, and mFS algorithms and then derives their corresponding recurrence equations for the average complexity formulas (complexity, for short). The recurrence equations for the average distance of the head movement required by the algorithms for external search on a tape were derived recently in [4]. However, they did not solve the recurrence equations to get the closed forms of the formulas. Instead, they used these recurrence equations iteratively to obtain the approximate performance values for some specific file sizes. In this note we solve the recurrence equations to get the closed forms of the formulas. These formulas are then used to compare the search complexities of the four search algorithms.

With the BS algorithm described in [3], we begin a search process by comparing the search key with the record in the middle of the sorted file. The orbit of BS on the sorted file consists of a root node containing a record and the links to the left and right subtrees which are defined in the same way. The left subtree contains those keys which are smaller than the key at the root, whereas the right subtree contains the larger keys in comparison with the root. Given, say, 12 records $K_1, K_2, \cdots, K_{12}$ and the search key K, the first record being examined is K_6 which is labeled by the index 6. If K_6 is less than K, then K_9 is examined; otherwise K_3 is examined. Continuing this process, the binary tree with root at level 0 describing this process is shown in Fig. 1(a) and the search sequences with only the first three probes on the storage are shown in Fig. 1(b).

In order to make the analysis easier, we may assume that the number of records in the sorted file is $F_n - 1 = 2^m - 1$ for some $m \geq 1$. We also assume that each record is searched with equal probability. Under these assumptions, the complexity of BS is given by

$$T_{BS}(n) = \left(\sum_{0 \leq i \leq m-1} 2^{m-1-i} 2^i (2^{m-i} - 1) \right) / (F_n - 1). \quad (1)$$

The reading mechanism moves 2^{m-1-i} units of length when traversing a node at level $(i-1)$ to a node at level i. Therefore, the time needed in a search step is 2^{m-1-i}. The number of subtrees whose roots are at level i is 2^i and the size of those subtrees is $2^{m-i} - 1$. Furthermore, each node in the corresponding subtree accumulates 2^{m-1-i} time units when one search step is passed.

An alternative method proposed by Ferguson [1] is FS which splits the file according to the Fibonacci sequence. The Fibonacci sequence is defined as

$$F_0 = 0, F_1 = 1,$$
$$F_i = F_{i-1} + F_{i-2} \quad \text{for} \quad i \geq 2.$$

In FS, we first examine the F_{n-2}nd record instead of the middle one. The F_{n-2}nd record corresponds to the root of the Fibonacci

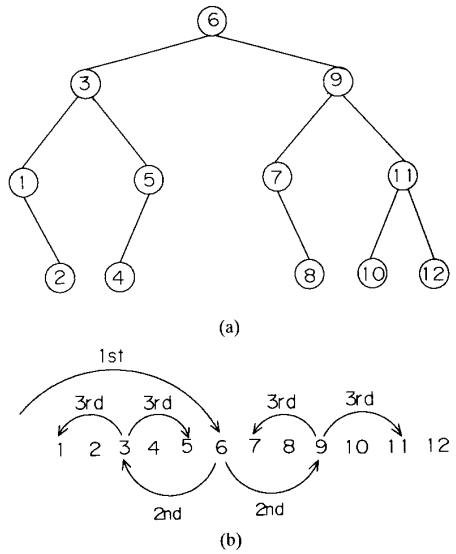

Fig. 1. (a) A binary search tree. (b) Search sequences of BS.

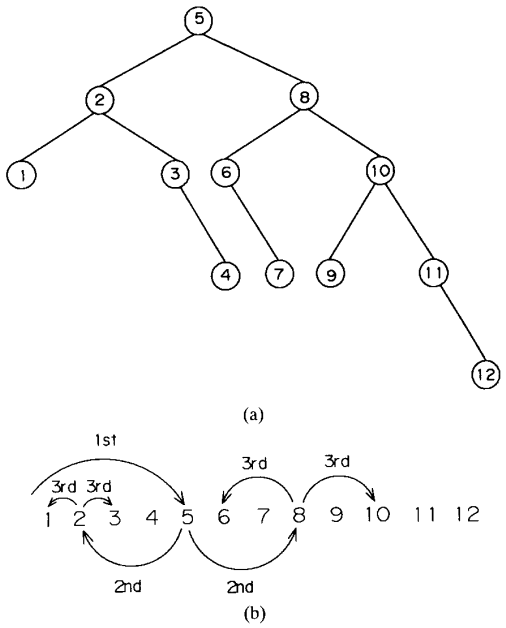

Fig. 2. (a) A Fibonacci search tree. (b) Search sequences of FS.

tree containing $F_n - 1$ nodes. Based on the comparison result, we then (recursively) search either the left subtree or the right subtree which contains $F_{n-2} - 1$ or $F_{n-1} - 1$ records. For example, given 12 records ($n = 7$), the Fibonacci tree describing the splitting process is shown in Fig. 2(a), and the search sequences of FS with the first three probes are shown in Fig. 2(b).

Consequently, the total search time is equal to $F_{n-2}(F_n - 1)$ plus the total amount of time needed in the left and right subtrees. As shown in Fig. 3, the distances from the root of the original tree to the roots of the left and right subtrees are both F_{n-3}. In fact, for any node, the distance from this node to its two subtrees are equal. However, since the splitting point is decided by the Fibonacci

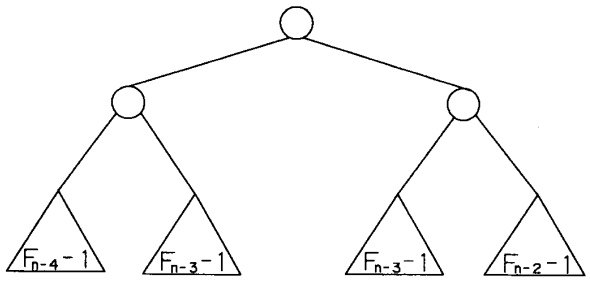

Fig. 3. The recursive tree of FS.

sequence, the sizes of these two subtrees are different, namely, $F_{n-2}-1$ for the left subtree and $F_{n-1}-1$ for the right subtree.

Thus, the complexity of FS is $T_{FS}(n) = p_{n-2}/(F_n - 1)$, where p_n is subject to

$$p_n = F_n(F_{n+2} - 1) + p_{n-1} + q_{n-1},$$
$$q_n = F_n(F_{n+1} - 1) + p_{n-2} + q_{n-2} \quad \text{for} \quad n \geq 2 \qquad (2)$$

with the initial conditions $p_0 = 0$, $p_1 = 1$, $q_0 = 0$, and $q_1 = 0$ [4]. Note that the initial condition can be obtained by checking the case $n = 4$.

A modified version of FS (mFS) is proposed to keep the amount of head movement as small as possible [4]. In FS, the sizes of the splitted pair of subfiles (subtrees) always form the two numbers $F_i - 1$ and $F_{i+1}-1$. In mFS, while the moving manner of the reading mechanism is similar to that of FS, the splitting position is decided such that the splitted part with smaller length F_i always abuts the current position of the reading head. The probe sequence defined in this way can keep the head movement small since the reading mechanism moves only F_i units of length in each search step.

The modified Fibonacci tree describing the splitting process is shown in Fig. 4(a) and the search sequences of mFS with the first three probes are shown in Fig. 4(b). The complexity of mFS, which can be derived by virtue of Fig. 5, is equal to $T_{mFS}(n) = s_{n-2}/(F_n - 1)$, where s_n is subject to recurrence

$$s_n = F_n(F_{n+2} - 1) + s_{n-1} + s_{n-2} \quad \text{for} \quad n \geq 2 \qquad (3)$$

with initial values $s_0 = 0$ and $s_1 = 1$ [4].

III. The Analysis of Complexities

In this section, we analyze the complexities of SS, BS, mFS, and FS. First, the complexities of SS and BS can be analyzed rather easily.

Theorem 1: The complexity of SS is $T_{SS}(n) \approx 0.5F_n$.

Proof: In SS, the complexity $T_{SS}(n)$ is given by

$$T_{SS}(n) = \Big(\sum_{1 \leq i \leq F_n - 2} i\Big)/(F_n - 1) \approx 0.5F_n.$$

This completes the proof.

Theorem 2: The complexity of BS is $T_{BS}(n) \approx F_n$.

Proof: Replacing the term 2^m in (1) by F_n, we have

$$T_{BS}(n) = \Big(\sum_{0 \leq i \leq m-1} 2^{m-1-i}2^i(2^{m-i} - 1)\Big)/(F_n - 1)$$
$$\approx \frac{1}{F_n - 1}(2^m(2^m - 1) - m2^{m-1})$$
$$\approx F_n.$$

■

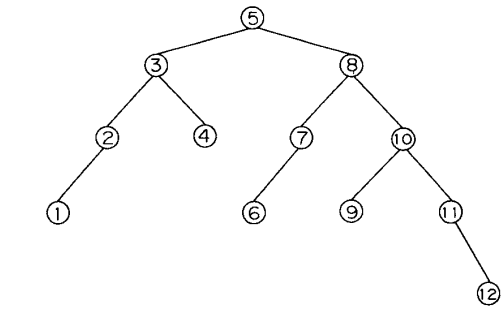

Fig. 4. (a). A modified Fibonacci search tree. (b) Search sequences of mFS.

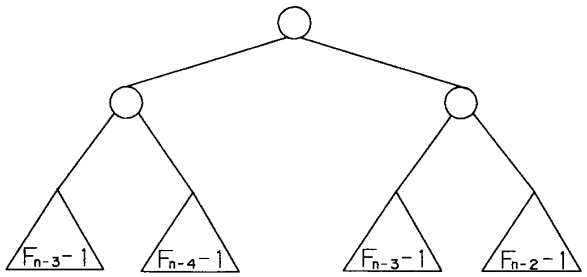

Fig. 5. The recursive tree of mFS.

Before analyzing the complexities of mFS and FS, we need some important lemmas and corollary. The following three lemmas are from [2].

Lemma 1: $F_{n+m} = F_{m-1}F_n + F_{n+1}F_m$.

Lemma 2:

$$F_n = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n);$$
$$\phi^n = F_n\phi + F_{n-1};$$
$$\hat{\phi}^n = F_n\hat{\phi} + F_{n-1}$$

where $\phi \approx 1.618$ and $\hat{\phi} \approx -0.618$.

Lemma 3: $\sum_{0 \leq k \leq n} F_kF_{n-k} = \frac{n-1}{5}F_n + \frac{2n}{5}F_{n-1}$.

From Lemma 1 and the fact that $F_{n+1} \approx 1.618F_n$ [2], we obtain the following corollary.

Corollary 1: $F_{2n} \approx 2.236F_nF_n$.

Theorem 3: The complexity of mFS is $T_{mFS}(n) \approx 0.809F_n$.

Proof: Since $T_{mFS}(n) = s_{n-2}/(F_n - 1)$, we first solve s_{n-2} in (3) as follows:

$$s_i = F_i(F_{i+2} - 1) + s_{i-1} + s_{i-2} \quad \text{for} \quad i \geq 2,$$

where $s_0 = 0$ and $s_1 = 1$.

Taking the generating function on both sides of the above equation, we have

$$S(z) = z + zS(z) + z^2S(z) + \sum_{i \geq 2} F_i(F_{i+2} - 1)z^i$$

$$= \frac{z}{1-z-z^2} + \frac{z}{1-z-z^2}\Big(\sum_{i\geq 2} F_i(F_{i+2}-1)z^{i-1}\Big)$$
$$= G(z) + G(z)\Big(\sum_{i\geq 2} F_i(F_{i+2}-1)z^{i-1}\Big)$$
$$= G(z)\sum_{i\geq 0} F_iF_{i+2}z^{i-1} - G(z)\sum_{i\geq 0} F_iz^{i-1} \quad (4)$$

where $S(z) = \sum_{i\geq 0} s_iz^i$ and $G(z) = z/(1-z-z^2) = \sum_{i\geq 0} F_iz^i$ as shown in [2].

The value of s_{n-2} is equal to the coefficient of z^{n-2} on the right-hand side of (4), thus

$$s_{n-2} = \sum_{0\leq k\leq n-1} F_kF_{n-k-1}F_{n-k+1} - \sum_{0\leq k\leq n-1} F_kF_{n-k-1}.$$

From Lemma 3, the above equation is

$$s_{n-2} = \sum_{0\leq k\leq n-1} F_kF_{n-k-1}F_{n-k+1} - \frac{n-2}{5}F_{n-1} - \frac{2n-2}{5}F_{n-2}. \quad (5)$$

Then using Lemma 2 and Corollary 1, we can calculate the summation term in the right-hand side of (5).

$$\sum_{0\leq k\leq n-1} F_kF_{n-k-1}F_{n-k+1}$$
$$= \frac{1}{5\sqrt{5}} \sum_{0\leq k\leq n-1} (\phi^k - \hat{\phi}^k)(\phi^{n-k-1} - \hat{\phi}^{n-k-1}) \cdot (\phi^{n-k+1} - \hat{\phi}^{n-k+1})$$
$$= \frac{1}{5\sqrt{5}} \sum_{0\leq k\leq n-1} (\phi^{2n-k} - \phi^{n-1}\hat{\phi}^{n-k+1} - \phi^{n+1}\hat{\phi}^{n-k-1} + \phi^k\hat{\phi}^{2n-2k} - \hat{\phi}^k\phi^{2n-2k} + \hat{\phi}^{n+1}\phi^{n-k-1} + \hat{\phi}^{n-1}\phi^{n-k+1} - \hat{\phi}^{2n-k}). \quad (6)$$

Although, there are 8 summation terms in the right-hand side of (6), only the following two terms need be considered:

$$\sum_{0\leq k\leq n-1} \phi^{2n-k} = \frac{\phi^{2n+1} - \phi^{n+1}}{\phi - 1} \approx 5.854F_{2n} \approx 13.090F_nF_n,$$
$$\sum_{0\leq k\leq n-1} \hat{\phi}^k\phi^{2n-2k} = \frac{(\hat{\phi}^n - \phi^{2n})\phi^2}{\hat{\phi} - \phi^2} \approx 1.81F_{2n} \approx 4.047F_nF_n. \quad (7)$$

The other 6 summation terms can be ignored because their values are too small to affect the analytical result. For example, the fourth term is

$$\sum_{0\leq k\leq n-1} \phi^k\hat{\phi}^{2n-2k} = \frac{(\phi^n - \hat{\phi}^{2n})\hat{\phi}^2}{\phi - \hat{\phi}^2} \approx 0.691F_n$$

which is quite small, compared with the above two summation terms. Plugging the right-hand sides of (7) into (6), we have

$$\sum_{0\leq k\leq n-1} F_kF_{n-k-1}F_{n-k+1} \approx 0.809F_nF_n. \quad (8)$$

Finally, from (5) and (8) we get

$$s_{n-2} \approx 0.809F_nF_n.$$

Thus, $T_{mFS}(n) = s_{n-2}/(F_n - 1) \approx 0.809F_n$. ∎

Theorem 4: The complexity of FS is $T_{FS}(n) \approx 0.882F_n$.

Proof: Since $T_{FS}(n) = p_{n-2}/(F_n - 1)$, we first solve p_{n-2} in (2) as follows:

$$p_i = F_i(F_{i+2} - 1) + p_{i-1} + q_{i-1};$$
$$q_i = F_i(F_{i+1} - 1) + p_{i-2} + q_{i-2} \quad \text{for} \quad i \geq 2$$

where $p_0 = 0, p_1 = 1, q_0 = 0$, and $q_1 = 0$.

We take the generating function on both sides of the above equations and obtain

$$P(z) = z + zP(z) + zQ(z) + \sum_{i\geq 2} F_i(F_{i+2} - 1)z^i, \quad (9)$$
$$Q(z) = z^2P(z) + z^2Q(z) + \sum_{i\geq 2} F_i(F_{i+1} - 1)z^i = \frac{1}{1-z^2}\Big(z^2P(z) + \sum_{i\geq 2} F_i(F_{i+1} - 1)z^i\Big) \quad (10)$$

where $P(z) = \sum_{i\geq 0} p_iz^i$, $Q(z) = \sum_{i\geq 0} q_iz^i$. Replacing $Q(z)$ in (9) by the right-hand side of (10), we have

$$P(z) = z + zP(z) + \frac{z}{1-z^2}\Big(z^2P(z) + \sum_{i\geq 2} F_i(F_{i+1} - 1)z^i\Big) + \sum_{i\geq 2} F_i(F_{i+2} - 1)z^i$$
$$= \frac{1-z^2}{1-z-z^2}\Big(z + \frac{z}{1-z^2}\sum_{i\geq 2} F_i(F_{i+1} - 1)z^i + \sum_{i\geq 2} F_i(F_{i+2} - 1)z^i\Big)$$
$$= (1-z^2)G(z) + G(z)\sum_{i\geq 2} F_i(F_{i+1} - 1)z^i + \frac{1-z^2}{z}G(z)\sum_{i\geq 2} F_i(F_{i+2} - 1)z^i$$
$$= G(z)\sum_{i\geq 0} F_i(F_{i+1} - 1)z^i + (1-z^2)G(z) \cdot \sum_{i\geq 0} F_i(F_{i+2} - 1)z^{i-1}. \quad (11)$$

The value of p_{n-2} is the coefficient of z^{n-2} in the right-hand side of (11), thus

$$p_{n-2} \approx \sum_{0\leq k\leq n-2} F_kF_{n-k-2}F_{n-k-1} + \sum_{0\leq k\leq n-1} F_kF_{n-k-1}F_{n-k+1} - \sum_{0\leq k\leq n-3} F_kF_{n-k-3}F_{n-k-1}. \quad (12)$$

Each summation term in the right-hand side of (12) is similar to the term in (11) and thus can be calculated using Lemma 2 and Corollary 1 as follows:

$$\sum_{0\leq k\leq n-2} F_kF_{n-k-2}F_{n-k-1} \approx 0.085F_{2n} \approx 0.191F_nF_n.$$
$$\sum_{0\leq k\leq n-1} F_kF_{n-k-1}F_{n-k+1} \approx 0.362F_{2n} \approx 0.809F_nF_n,$$
$$\sum_{0\leq k\leq n-3} F_kF_{n-k-3}F_{n-k-1} \approx 0.053F_{2n} \approx 0.118F_nF_n.$$

Substituting the values of the above summation terms into (12), we have

$$p_{n-2} \approx 0.882F_nF_n.$$

Therefore, $T_{FS}(n) = p_{n-2}/(F_n - 1) \approx 0.882F_n$. ∎

IV. Conclusions

Theorems 1 and 2 give $T_{SS}/T_{BS} \approx 0.5$. That is, SS is 50% better than BS. Theorems 2 and 3 give $T_{mFS}/T_{BS} \approx 0.809$, or mFS is 19.1% more efficient than BS. Theorems 2 and 4 give $T_{FS}/T_{BS} \approx 0.882$ or FS is 11.8% more efficient than BS. In summary, when searching on a tape, sequential search, Fibonacci search, and modified Fibonacci search are all better than binary search. Moreover, modified Fibonacci search indeed is better than Fibonacci search.

In [4], Nishihara and Nishino wrote computer programs to evaluate the performance values by iteratively applying the recurrence equations in Section II. When the size of the sorted file is more than 2000 records, they obtained the approximate efficiency ratios as $T_{FS}/T_{BS} \approx 0.882$ and $T_{mFS}/T_{BS} \approx 0.809$. These experimental values confirm our theoretic results derived in this note.

References

[1] D. E. Ferguson, "Fibonacci searching," *Commun. ACM*, vol. 3, no. 12, pp. 648, 1960.

[2] D. E. Knuth, *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*, 2nd ed. Reading, MA: Addison-Wesley, 1973.

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

[4] S. Nishihara and H. Nishino, "Binary search revisited: Another advantage of Fibonacci search," *IEEE Trans. Comput.*, vol. C-36, no. 9, pp. 1132–1135, 1987.

Exact Parametric Analysis of Stochastic Petri Nets

Man Li and Nicolas D. Georganas

***Abstract*— An algorithm for exact parametric analysis of Stochastic Petri Nets is presented. The algorithm is derived from the theory of Decomposition and Aggregation of Markov Chains. The transition rate of interest is confined into a diagonal submatrix of the associated Markov Chain by row and column permutations. Every time a new value is assigned to the transition, a smaller Markov Chain is analyzed. As a result, the computational cost is greatly reduced.**

***Index Terms*— Decomposition and Aggregation, Markov Chain, parametric analysis, stochastic Petri Nets.**

I. Introduction

Stochastic Petri Nets, introduced independently by Molloy [6] and Natkin [7], are a tool for modeling systems with concurrency, synchronization, and communication. They have been widely used in the performance analysis of communication systems, protocols, manufacturing systems, etc.

Manuscript received July 25, 1990; revised April 24, 1991. This work was supported in part by the Natural Science and Engineering Research Council of Canada under Grant A-8450 and the Ontario University Research Incentive Fund under Grant 0T9-003.

The authors are with the Department of Electrical Engineering, University of Ottawa, Ottawa, Ont., Canada K1N 6N5.

IEEE Log Number 9103109.

According to Molloy [6], the continuous time Stochastic Petri Net is defined as

$$SPN = (P, T, A, R)$$

$P = \{p_1, p_2, \cdots p_n\}$ is a set of places

$T = \{t_1, t_2, \cdots t_m\}$ is a set of transitions

$A \subset \{P \times T\} \cup \{T \times P\}$

$R = \{r_1, r_2, \cdots r_m\}$ is a set of firing rates for the exponentially distributed transition firing times.

Molloy further showed that any finite place, finite transition, marked Stochastic Petri Net is isomorphic to a Markov Chain. As a result, the usual way of analyzing a Stochastic Petri Net is to generate the Reachability Graph (RG) of the Stochastic Petri Net and at the same time construct the associated Markov Chain. The markings constitute the states of the Markov Chain (hence we will use "state" and "marking" alternatively) and the transition rates between markings constitute the infinitesimal generator matrix. Solving the Markov Chain, we obtain the steady-state probabilities. From these probabilities, we obtain the performance measures of interest.

Frequently, in the performance analysis by Stochastic Petri Nets, we are interested in the behavior of the system with respect to one parameter, or one transition. By assigning different values to this transition, we may obtain the effect of this particular transition on the whole system performance. One drawback of this analysis is that every time the transition rate is changed, we have to analyze the whole net again. This is a tedious procedure. Regarding this problem, Ammar [1] recently proposed a time scale decomposition method for analyzing a kind of Stochastic Petri Net whose transition rates differ by orders of magnitude. Approximate results were obtained. Li and Georganas [5] proposed the parametric analysis of a class of Stochastic Petri Nets whose underlying Markov Chain satisfies local balance equations. They showed that exact results can be obtained in an approach analogous to Norton's theorem. Both [1] and [5] dealt with only a special class of Stochastic Petri Nets. In this paper, we propose exact parametric analysis of Stochastic Petri Nets in general. The idea is based on the theory of Decomposition and Aggregation of Markov Chains.

The organization of this paper is as follows: Section II gives an introduction to decomposition and aggregation techniques and then derives an algorithm for parametric analysis. Section III considers the computational gain achieved by the algorithm compared with directly solving the Markov Chain by Gaussian elimination. Section IV deals with implementation of the algorithm to parametric analysis of Stochastic Petri Nets. Section V gives an example and Section VI concludes this paper.

II. Exact Parametric Analysis of Stochastic Petri Nets

We give an introduction to decomposition and aggregation techniques and then derive an algorithm for parametric analysis.

A. The Theory of Decomposition and Aggregation

The decomposition and aggregation techniques were first proposed by Simon and Ando [9] in the early 1960's. The primary feature of the decomposition and aggregation techniques is reducing the analysis of a large system into that of a set of smaller problems.

Let $\boldsymbol{Q}$ be the infinitesimal generator matrix of a continuous time Markov Chain. Assume $\boldsymbol{Q}$ is an $n \times n$ matrix. $\boldsymbol{Q}$ is partitioned into