Generalized Swap-with-Parent Schemes for Self-Organizing Sequential Linear Lists *

John Oommen¹ and Juan Dong²
Sch. Of Comp. Sci.
Carleton University
Ottawa, Canada

¹ oommen@scs.carleton.ca
² dong@turing.scs.carleton.ca

Abstract—Self organizing linear search algorithms have been in the literature for over 30 years, and numerous schemes have been proposed during that time. Among all the previous algorithms, the move-to-front rule and the transposition rule are the most extensively analyzed schemes. Recently we proposed and thoroughly analyzed a new scheme, the swap-with-parent rule, which views the list as a heap structure with no ordering constraints between parents and their children [12]. From the analyses of the transposition rule and the swap-with-parent rule, it can be seen that the fundamental property of the corresponding Markov chain being time reversible greatly simplifies the analysis of the algorithm. In this paper, we shall show the existence of a class of time reversible Markov chains resulting from performing swaps on “implicit” trees (called ss_trees) which generalize and extend the results concerning the transposition and the swap-with-parent heuristics.

This paper introduces a generalization of the transposition rule and the swap-with-parent rule - the swap-with-parent-in-an-ss_tree heuristic and its modification - the move-to-parent-in-an-ss_tree heuristic. Detailed expressions for the asymptotic probabilities and the asymptotic search cost of the scheme have been derived.

1 Introduction

This paper presents a novel approach to designing arbitrary, user-defined self-organizing sequential search algorithms by viewing the list as an implicit tree.

Suppose we are given a set of records $R_1, R_2, \ldots, R_n$ which are in an arbitrary order $\pi$, so that $R_i$ is in position $\pi(i)$ for $1 \leq i \leq n$. At every instant of time one of these records $R_i$ is accessed. We do so by examining each record of the list starting from the first record until $R_i$ is found costing us $\pi(i)$ units of time. Thus, if we assume that each record $R_i$ is accessed with an (unknown) probability $s_i$ and that the accesses are made independently, the expected search cost (average

---

* Partially supported by the National Sciences and Engineering Research Council (NSERC) of Canada
search length) for an ordering of the list \( \pi \) is the sum of the product of every nodes' search cost and its access probability.

To minimize the access cost, it is desirable that the records are ordered optimally in the descending order of their access probabilities. However, the access probabilities are seldom known \textit{a priori} in practice, and so the list cannot be arranged into the \textit{optimal ordering} in advance. In this situation we need an algorithm which dynamically rearranges the list and gradually transforms it to a less costly ordering.

A \textit{self-organizing sequential search list} is a linear search list in which the order of the records may be altered each time a sequential search occurs so that after sufficiently many accesses, it tends to be in the optimal ordering with high probability, so that fewer comparisons are needed on subsequent accesses.

An intuitive scheme for reordering a list to a, hopefully, less costly ordering is to keep a counter of accesses for each record, and maintain the records in the descending order of their access frequencies. However, (see [9] p398), such a scheme is undesirable as it requires extra space which could perhaps be better used by employing nonsequential search techniques.

The first memory-free self-organizing scheme proposed by McCabe in 1965 [10] is the \textit{move-to-front} rule. In this scheme, each time a record is accessed, it is moved to the front of the list, and all the records before the accessed record are shifted back one position. The asymptotic search cost of the scheme is

\[
1 + 2 \sum_{1 \leq i < j \leq n} \frac{s_i s_j}{s_i + s_j}.
\]

McCabe [10] introduced another memory-free scheme called the \textit{transposition} rule. In this rule, the accessed record is moved one position closer to the front of the list by interchanging it with its preceding record unless it is at the front of the list. This was later proved by Rivest [17] to have lower expected search cost per access than the move-to-front rule.

In order to be brief we only list some of the many other strategies that have been discovered so far. These are: \textit{Move-ahead-k} (Rivest [17]), \textit{POS(k)} (Tenenbaum and Nemes [20]), \textit{Wait c}, \textit{Move-and-Clear} (Bitner [2]), \textit{k-in-a-row} (Kan and Ross [7] and Gonnet \textit{et al.} [3]), and \textit{k-in-a-batch} (Gonnet \textit{et al.} [3]). Analyses and expositions of these ergodic rules can be found in [3, 5, 16]. Besides these, the literature also reports a few absorbing rules due to Oommen and his co-authors [14, 15].

1.1 Drawback of Previous Schemes and New Contributions

Notice that there is a common drawback to all the algorithms listed above, that is, they all fail to consider the size of the list when reorganizing it. The two extremes are the well-known move-to-front rule and transposition rule. Oommen and Dong recently proposed two memory-free algorithms [12], both of which take into account the size of the list when reorganizing it. The first algorithm is called \textit{swap-with-parent} (SWP) and the second is called \textit{move-to-parent} (MTP).