THE SWAP-WITH-PARENT SCHEME: A SELF-ORGANIZING SEQUENTIAL SEARCH ALGORITHM WHICH USES NON-LEXICOGRAPHIC HEAPS

John Oommen and Juan Dong

TR-97-07 MAY 1997

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

The Swap-with-Parent Scheme: A Self-Organizing Sequential Search Algorithm which Uses Non-lexicographic Heaps *

John Oommen
Sch. of Comp. Sci.
Carleton University
Ottawa, Canada
oommen@scs.carleton.ca

Juan Dong
Sch. of Comp. Sci.
Carleton University
Ottawa, Canada
dong@turing.scs.carleton.ca

Abstract— *A common drawback to all the reported self-organizing sequential search algorithms is that they all fail to consider the size of the list when reorganizing it. The two extremes are the well-known and most commonly analyzed algorithms, the move-to-front rule and the transposition rule. This paper presents two new memory-free self-organizing sequential search algorithms, both of which overcome this drawback. The first algorithm is called* swap-with-parent *(SWP) and the second is called* move-to-parent *(MTP). Under the swap-with-parent heuristic, the accessed record is exchanged with its "parent" (considering the list as a heap structure with no ordering constraints between parents and their children) and all the other records are untouched. whereas under the move-to-parent heuristic, the accessed record is moved to its parent's positions and all the records in between get shifted back one position. It is shown that under the SWP heuristic, the Markov chain representing the scheme is time reversible. This property allows us to derive its asymptotic equilibrium probabilities a rather complicated expression for its average search cost. We conjecture that it costs no more than the move-to-front rule and its convergence is intermediate to the move-to-front rule and the transposition rule. For the performance of the move-to-parent rule, empirical comparison shows that it lies between the move-to-front heuristic and the transposition heuristic, and that it is better than the swap-with-parent rule.*

Keywords— *Self organizing lists, the move-to-front rule, the transposition rule, the swap-with-parent rule, the move-to-parent rule, Markov chains, time reversible Markov chains.*

1 Introduction

Suppose we are given a set of records $R_1, R_2, \ldots, R_n$ which are in an arbitrary order π, 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. This search costs $\pi(i)$ units of time to perform.

Assuming that each record R_i is accessed with an unknown probability s_i and the accesses are made independently. Then the expected search cost (average search length) for an ordering

*Partially supported by the National Sciences and Engineering Research Council (NSERC) of Canada

of the list π is

$$cost(\pi) = \sum_{1 \le i \le n} s_i \pi(i) \tag{1}$$

To minimize the access cost, it is desirable that the records are ordered in the descending order of their access probabilities. We shall refer to this kind of list ordering as a perfect ordering or optimal ordering and refer to a file ordered in this way as a completely organized file. However the access probabilities are seldom known *a priori* in practice, and so the list can not be arranged in the *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 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 where the most frequently accessed record is at the front of the list and the rest of the list is recursively ordered in the same manner. The most useful and common type of alteration is to move the accessed record forward one or more positions in the list. In this way more frequently accessed records move towards the front of the list so that fewer comparisons are needed on subsequent accesses.

This problem of having a file organize itself has been studied extensively. Various reviews on self organizing strategies have been published over the years. We refer the reader to the comprehensive surveys of the papers and results in the area [7, 9, 11] and the surveys on the applications [2, 11, 19]. Rather than describe the various strategies in any detail, we shall briefly highlight some of their main features.

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, as Knuth remarks ([14] p398), this counter scheme is undesirable as it requires extra memory space which could perhaps be better used by employing nonsequential search techniques.

The first memory-free self-organizing scheme proposed by McCabe in 1965 [15] is the *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 in terms of the average number of probes required to find a record in the given list (after it has reached a *steady state* where many further reorderings on the list are not expected to increase or decrease the search cost significantly) is

$$1 + 2 \sum_{1 \le i < j \le n} \sum \frac{s_i \, s_j}{s_i + s_j}. \tag{2}$$

Many other researchers [7, 9, 10, 11] have also extensively studied the MTF rule and various properties of its limiting convergence characteristics are available in the literature.

McCabe [15] introduced another memory-free scheme called the *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 [19] to have lower expected search cost per access than the move-to-front rule, and he conjectured that the transposition rule is optimal. This conjecture was further strengthened by the result by Bitner [4] that for some special distributions the transposition rule is optimal over all rules. However Anderson *et al.* [1] found a counterexample to this conjecture by deriving a rule that is better than the transposition rule for a specific distribution. Bitner [3, 4] later showed that while the transposition rule is asymptotically more efficient, the move-to-front rule converges more quickly and proposed a hybrid of these two rules that attempts to incorporate the best features

of both. This simple hybrid rule initially uses the move-to-front algorithm until its steady state is approached, and then switches to the transposition rule. However the difficulty of deciding when to switch is still a major problem.

Rivest [19] proposed a compromise between the relative extremes of the move-to-front rule and the transposition rule, namely the *move-ahead-k* heuristic where the accessed record is moved k-positions forward towards the front of the list unless it is in the first k positions, in which case it is moved to the front. This is a generalization of the transposition rule and the move-to-front rule as transposition is *move-ahead-1* and move-to-front is *move-ahead-n*. However no absolute analyses is available for this scheme.

Tenenbaum and Nemes [21] suggested a generalization of the move-to-front rule and the transposition rule, the *POS(k)* rule. The POS(k) rule moves the accessed record to position k of the list if it is in positions $k+1$ to N, or it transposes it with its preceding record if it is in positions 2 to k. If it is the first in the list, it is left unchanged. Note that *POS(1)* is the move-to-front, whereas *POS($n-1$)* is the transposition strategy.

Notice that all the algorithms described above alter the list on a single access basis. We shall call such an algorithm a *reordering algorithm* or *permutation algorithm.*

McCabe [15] considered reorganizing the list only once every k accesses to reduce the time spent reordering the list. It is to be used in conjunction with permutation algorithms. In this scheme, a counter is needed for each record to store the value of k. A record is moved forward (according to the permutation algorithm chosen) only if it has been accessed k times, not necessarily in a row, and then the counter for *that record* is reset. Bitner [4] also studied this rule and analyzed its performance when used with the move-to-front rule. Bitner also suggested a modification to it, the *wait c, move and clear* rule. After a record has been accessed c times, not necessarily in a row, it is moved forward and the counter for *every record* is reset.

Kan and Ross [12] and Gonnet *et al.* [9] proposed the *k-in-a-row* heuristics, where a record is moved forward only after it is accessed k times in a row. Gonnet *et al.* [9] proved that *($k+1$)-in-a-row* is superior to *k-in-a-row* and suggested a minor modification called the *k-in-a-batch* heuristic, where accesses are grouped into batches of size k, and a record will be moved if it is accessed k times in a batch. They proved that the batched-k rule is better than the k-in-a-row rule when in combination with either the move-to-front rule or the transposition rule. Note that these rules all require extra space for storing the values of k.

Since the Markov chains representing the schemes described above are ergodic. It means that the list can be in any of its $n!$ configurations. As opposed to ergodic representations, Markovian behavior can also be absorbing [13, 20]. In an absorbing Markov chain, the chain converges to one of a (finite) set of absorbing barriers. Oommen and Hansen [17] introduced two absorbing list organizing schemes, the *bounded memory stochastic move-to-front* scheme and the *stochastic move-to-rear* scheme. In both algorithms, the move operation is performed stochastically in such a way that ultimately no more move operations are performed. However it is shown that the first scheme is never better than the deterministic move-to-front algorithm. For the second scheme, they showed that the probability of convergence to the optimal ordering could be made as close to unity as desired. Oommen *et al.* [18] later proposed two deterministic absorbing schemes, both of which perform move-to-rear operation. One is the *deterministic linear-space move-to-rear* scheme, the other is the *deterministic constant-space move-to-rear* scheme. The former moves the accessed record to the *rear* of the list if it has been accessed k times. The scheme is asymptotically optimal, the probability of being absorbed into the optimal ordering can be made as close to unity as desired. The second scheme moves the accessed record to the *rear* of the list if it has been accessed k *consecutive* times. It is proven to be expedient. Again all the above probabilistic schemes require extra memory.

2 Drawback of Reported Schemes and Present Contributions

Notice that there is a common drawback to all the previous algorithms, 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. The move-to-front rule moves the accessed record to the front of the list and the transposition rule moves the accessed record one position ahead regardless of the size of the list. Move-ahead-k heuristic attempts to incorporate the best features of the move-to-front rule and the transposition rule by moving the accessed record forward a fixed number of positions (constant distances). Although Hester and Hirschberg [11] point out that the move-ahead-k rule can be generalized to move a percentage of the distance, no known solutions or analysis have been published to our knowledge. Other hybrid algorithms and all batched algorithms are used in conjunction with permutation algorithms, and therefore cannot avoid the drawbacks which permutation algorithms possess. Probabilistic algorithms move the accessed record, upon certain conditions, either to the front of the list or to the rear of the list, again ignoring the size of the list.

In this paper, we shall present two new memory-free self-organizing sequential search algorithms, both of which take into account the size of the list when reorganizing it. The first algorithm is called *swap-with-parent* (SWP) and the second is called *move-to-parent* (MTP). Under the *swap-with-parent* heuristic, the accessed record gets exchanged with its "parent" (considering the list as a heap structure with no ordering constraints between parents and their children), and all the other records stay unchanged. Instead of swapping the accessed record with its "parent", the *move-to-parent* heuristic moves the accessed record to its parent's position and shifts the parent and all the records between the accessed record and its parent back one position.

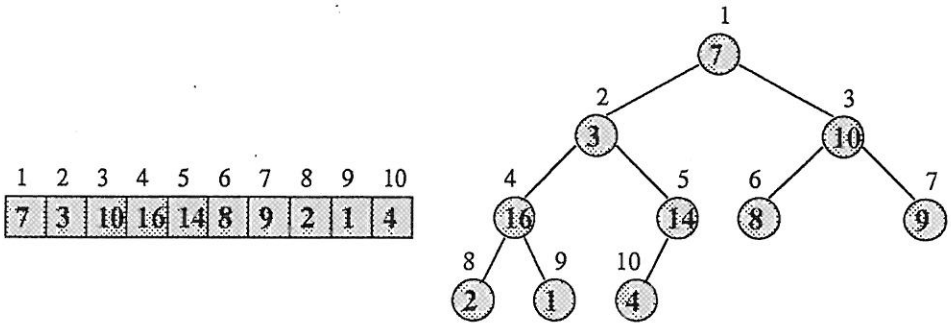
We shall show that under the swap-with-parent heuristic, the Markov chain representing the scheme is *time reversible*. (Note that this is also the property that the transposition rule has and which makes the analysis of the transposition rule greatly simplified [16, 8].) Using this property, we shall further derive various expression for the asymptotic probabilities and for the average search cost of the scheme. Although the scheme is no better than the transposition rule in terms of the average search cost per access, we conjecture that it costs no more than the move-to-front rule and its convergence is intermediate to the move-to-front rule and the transposition rule. For the performance of the move-to-parent rule, empirical comparison shows that it lies between the move-to-front heuristic and the transposition heuristic, and that it is better than the swap-with-parent rule.

As in the literature concerning the theory of self organizing data structures, we shall use the terms *algorithm, rule, scheme* and *heuristic* interchangeably.

3 The Swap-with-Parent Heuristic

Unlike all the other schemes in the literature, the *swap-with-parent* heuristic actually takes into account the size of the list when reorganizing it. Like the *move-to-front* rule and the *transposition* rule, it is memoryless. The scheme is actually a compromise between the tradeoffs of the move-to-front rule and the transposition rule.

The idea of the scheme comes from self organizing binary search tree operations [5] where the accessed record, under certain conditions, is rotated upwards to its parent level. Since we don't utilize lexicographic ordering, we will not perform rotation operations in the swap-with-parent scheme, but simply exchange the accessed record with its parent. For any given list of records



(a) A list of 10 elements (b) The corresponding heap

Figure 1: A list viewed as a "heap".

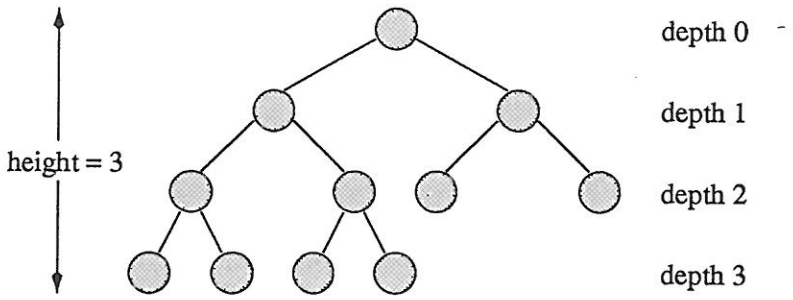


Figure 2: A s_heap of height 3.

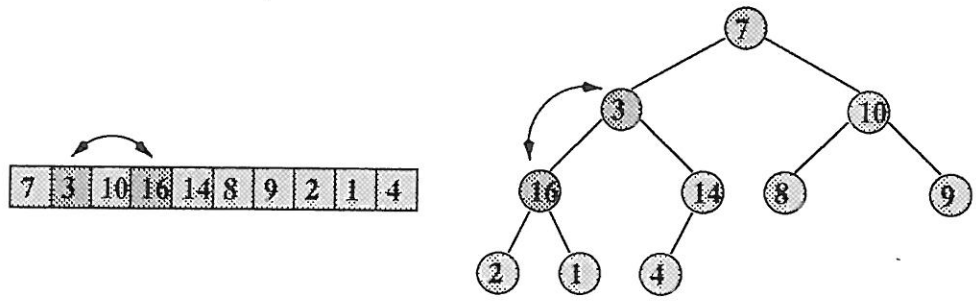
$\{R_1, R_2, \ldots, R_n\}$, we can conceptually construct a *heap structure* with *no* ordering constraints between parents and their children. For example, given a list of elements $\{7, 3, 10, 16, 14, 8, 9, 2, 1, 4\}$, the corresponding "heap" structure with *no ordering constraints* is shown in Figure 1. Note that in the context of data structures, the term "heap" is defined as an array object that can be viewed as a complete binary tree with the exception that the leaf level may not be completely filled (only filled from the left up to a point). Heaps also satisfy the heap property that the value of every parent is greater than or equal to the values of its children [6]. However because we ignore lexicographic ordering, our "heap" data structure *does not* have the heap property, and so we shall refer to this structure as a *sequential_heap* or *s_heap*. Throughout this paper, whenever we refer to heaps, we shall mean the structure defined here.

The root of the s_heap has index 1, and given the index i of a node, the index of its parent, Parent(i), can be computed simply:

$$\text{Parent } (i) = \lfloor i/2 \rfloor . \tag{3}$$

We define the *depth* of node i, depth (i), in an s_heap to be the length of the path from the root to the node i as shown in Figure 2, and we define the *height* of the s_heap to be the depth of its leaf nodes. Since an s_heap of n elements can be viewed as a complete binary tree, its height is $\lfloor \lg n \rfloor$, and the depth of node i, depth (i), is,

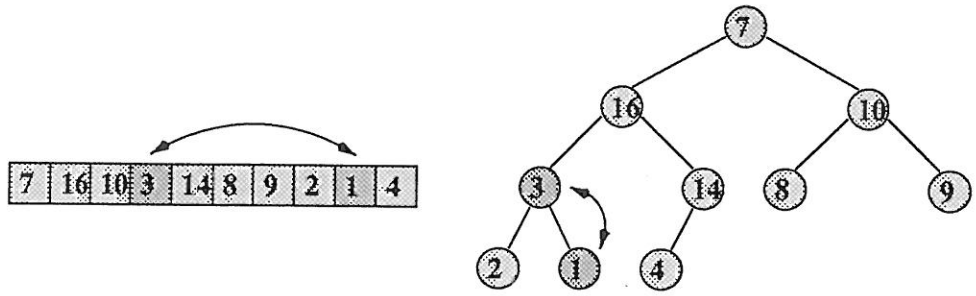
$$\text{depth } (i) = \lfloor \lg i \rfloor . \tag{4}$$



(a) list before 16 is accessed

(b) equivalent s_heap before 16 is accessed

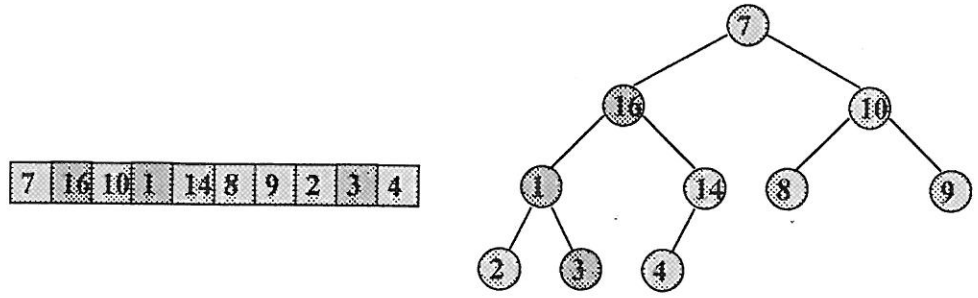
swapping 16 and 3



(c) list before 1 is accessed

(d) equivalent s_heap before 1 is accessed

swapping 1 and 3



(e) list after 16 and 1 are accessed

(f) equivalent s_heap after 16 and 1 are accessed

Figure 3: Operations of swapping with parent.

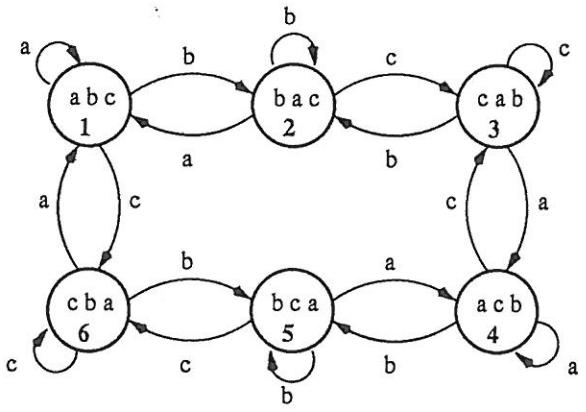


Figure 4: State diagram with three elements per state using the SWP rule.

Suppose we have a set of n records $\{R_1, R_2, \ldots, R_n\}$ which are in an arbitrary order π, so that R_i is in position $\pi(i)$ for $1 \leq i \leq n$. Using the *swap-with-parent* heuristic, whenever a record R_i is found in position $\pi(i)$, the list is rearranged by exchanging the positions of R_i and its parent which is in position $\lfloor \pi(i) / 2 \rfloor$ and leaving all the other records untouched; if R_i heads the list nothing is done. Observe, first of all, a powerful property that unlike the transposition rule, it takes as few as $\lfloor \lg n \rfloor$ steps for a frequently accessed record to be moved from the back of the list to the front (root) of the list as opposed to n accesses needed for the transposition rule, and as few as $\lfloor \lg n \rfloor$ steps for a less frequently accessed record to be moved from the front of the list to the back of the list as opposed to n accesses needed for the transposition rule.

By way of example, consider a list of 10 elements $\{7, 3, 10, 16, 14, 8, 9, 2, 1, 4\}$, using the swap-with-parent rule, Figure 3 shows the changes on the orders of the list after elements 16 and 1 are accessed consecutively.

The scheme avoids the slow convergence problem which the transposition rule has. However it may still tend to make bigger mistakes (like the move-to-front rule) by moving a record to the front about half way of its current position on the basis of a single access comparing to the transposition rule. But the "mistakes" too are far more conservative. We shall look at its performance in the next section.

4 The Performance of the SWP Heuristic

Let us consider a list of three records $\{a, b, c\}$ with respective access probabilities of $\{s_a, s_b, s_c\}$. Figure 4 shows the state diagram for the underlying Markov chain resulted from using the swap-with-parent rule.

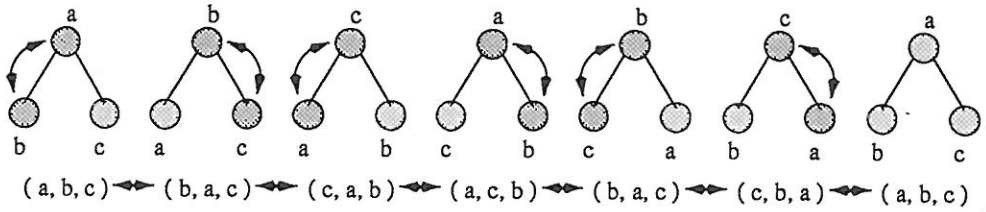
Observing the paths from state 1 to itself, one path is found if we are looking forward (clockwise), which is $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 1$, and the reversed path can be seen if we are looking backward (counter-clockwise), which is $1 \to 6 \to 5 \to 4 \to 3 \to 2 \to 1$. This gives us an indication that the Markov chain shown in the diagram is time reversible. This is actually true for all Markov chains derived from the swap-with-parent heuristic as shown in the following theorem.

Theorem 4.1 *The Markov chain which results from using the swap-with-parent heuristic is time reversible.*

Proof: To prove that the Markov chain resulting from the swap-with-parent rule is time reversible, it is sufficient to prove that starting in any state π_s, any path back to π_s has the same probability as the reversed path (see [20] p143). We will carry the proof by induction on the number of records, n, in the list.

1. Base case $n = 3$.

For any state π_s, say, $\pi_s = \{a, b, c\}$, considering the path from state π_s to itself as shown below (see also the state diagram shown in Figure 4).



The product of the transition probabilities in the forward direction is $s_b s_c s_a s_b s_c s_a = s_a^2\, s_b^2\, s_c^2$, whereas in the reverse direction, it is $s_c s_b s_a s_c s_b s_a = s_a^2\, s_b^2\, s_c^2$. Therefore, the Markov chain is time reversible.

2. Case $n = 4$.

Before going to the case for $n = k$, we shall see how we can construct the state diagram for the case of $n = 4$ from that of the case for $n = 3$. The construction is best illustrated through the diagram shown in Figure 5.

In the diagram, each small box represents a subset of the chain of the states. Thus, for example, $\boxed{\text{a}\sim\text{b}\sim\text{c }\mathbf{d}}$, contains 6 states. Inside each box (sub-chain), the last element remains in the same position in all 6 states. For example, in sub-chain $\boxed{\text{a}\sim\text{b}\sim\text{c }\mathbf{d}}$, element **d** is the last element in every state inside the sub-chain; only the other three elements a, b, c permute among one another. Thus, from any states inside the box, accesses to a, b, c always transform the state to another state inside *this* sub-chain; to reach any state in another sub-chain, element **d** must be accessed. Note that since we are modifying the list using the swap-with-parent rule, we can only return to the sub-chain $\boxed{\text{a}\sim\text{b}\sim\text{c }\mathbf{d}}$ from the states where **d** is the second element in the list. Consequently, the sub-chain of the Markov chain which is constrained to be entirely within sub-chain $\boxed{\text{a}\sim\text{b}\sim\text{c }\mathbf{d}}$ is actually equivalent to the Markov chain of three elements $\{a, b, c\}$ per state as shown in Figure 4, and thus it is time reversible. Note also that all four constituent boxes which represent the sub-chains are symmetric, and totally constitute the entire chain.

Next we will show that the Markov chain consisting of four elements, $\{a, b, c, d\}$, per state is time reversible. Thus, we shall prove that starting in any state π_s, any path back to π_s has the same probability as the reversed path. Since all four small sub-chains are symmetric, it will be sufficient to prove that if we start in any one of the sub-chains, say, $\boxed{\text{a}\sim\text{b}\sim\text{c }\mathbf{d}}$, starting in any state π_s from that sub-chain ($\boxed{\text{a}\sim\text{b}\sim\text{c }\mathbf{d}}$), any path back to π_s has the same probability as the reversed path.

Without loss of generality, let $\pi_s = \{a, b, c, d\}$, then any path back to π_s from inside the sub-chain will have the same probability as the reversed path since we have shown that the Markov chain consisting of states entirely within this sub-chain is time reversible. So it remains to show that any path back to π_s which goes *outside* the box has the same probability as the reversed path. We know that to reach any states outside the sub-chain, element **d** must be accessed. After

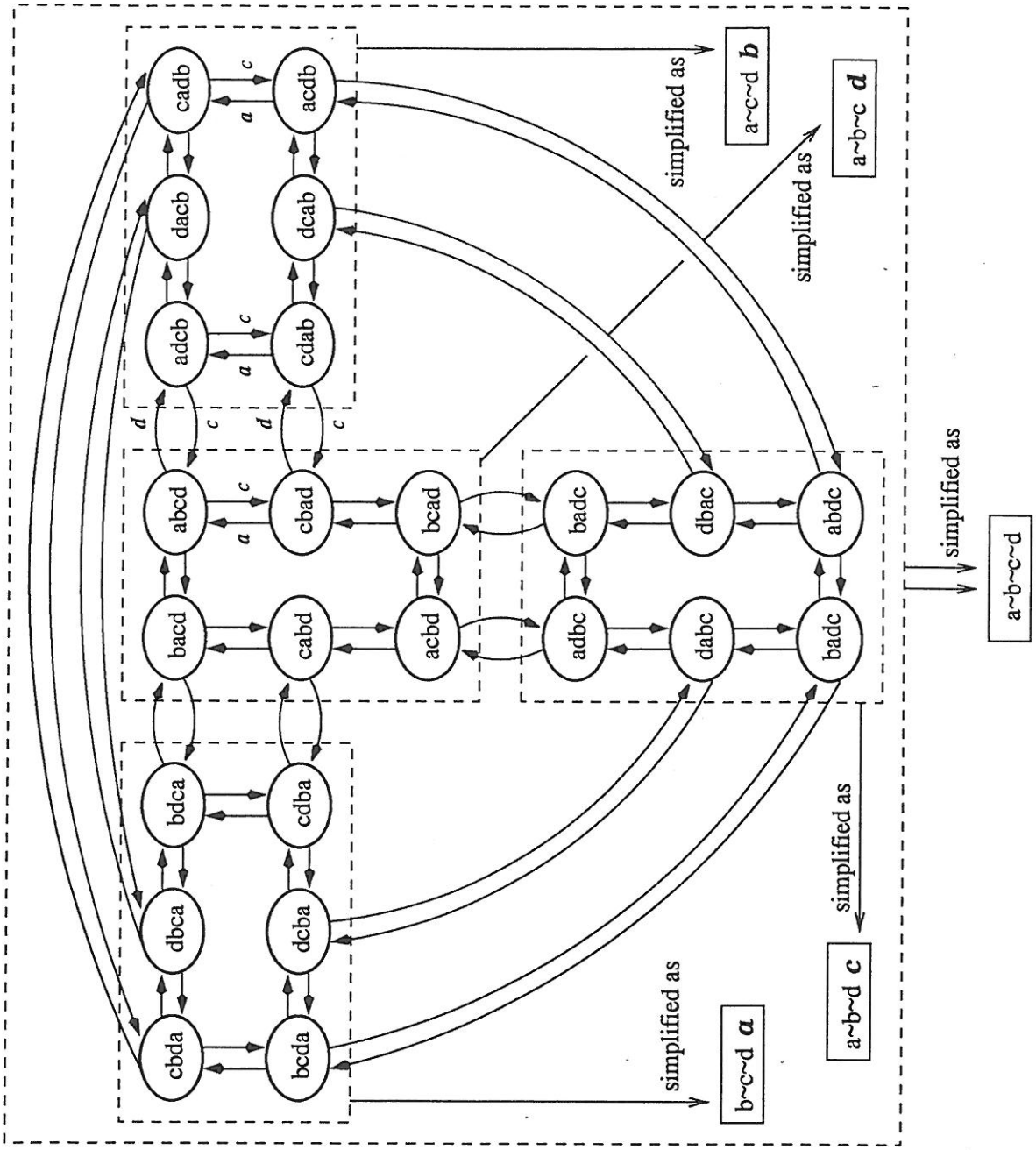


Figure 5: State diagram with four elements per state using the SWP rule when partitioned into four sub-chains. Note that the transition from a state to itself is not shown in the diagram for simplification purposes.

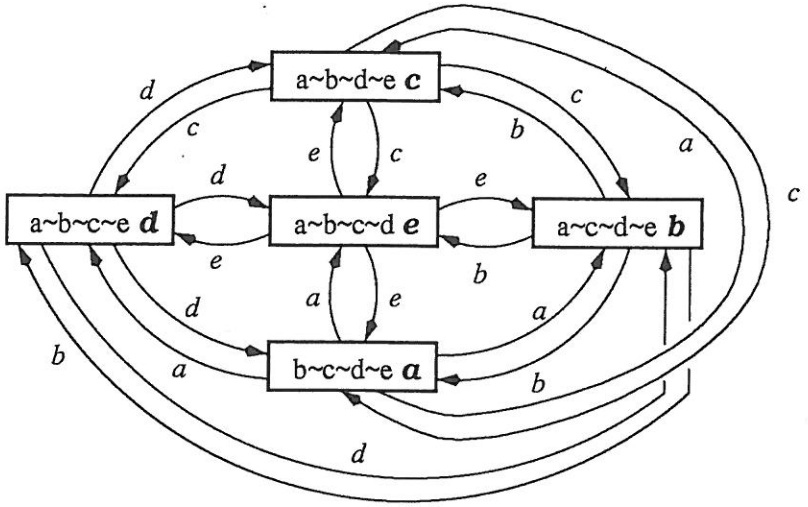


Figure 6: Simplified state diagram with five elements per state using the SWP rule.

d is accessed, we are in state $\{a, d, c, b\}$ in sub-chain $\boxed{\text{a}\sim\text{c}\sim\text{d b}}$, and so to get back to state π_s, we have to get back to sub-chain $\boxed{\text{a}\sim\text{b}\sim\text{c d}}$. Since all sub-chains are symmetrical, we only need to consider the path back to $\boxed{\text{a}\sim\text{b}\sim\text{c d}}$ directly from $\boxed{\text{a}\sim\text{c}\sim\text{d b}}$, which, in turn, means that the last state we will visit inside $\boxed{\text{a}\sim\text{c}\sim\text{d b}}$ right before we get back to $\boxed{\text{a}\sim\text{b}\sim\text{c d}}$ must be a state where d is the second element because of the nature of the swap-with-parent rule. Since $\{a, d, c, b\}$ has already been visited when we first entered the sub-chain $\boxed{\text{a}\sim\text{c}\sim\text{d b}}$, we will not allowed to use it again. There is thus only one choice - we must get back from state $\{c, d, a, b\}$. Obviously, the shortest path is thus

$$\{a, b, c, d\} \rightleftharpoons \{a, d, c, b\} \rightleftharpoons \{c, d, a, b\} \rightleftharpoons \{c, b, a, d\} \rightleftharpoons \{a, b, c, d\}\,.$$

The product of the transition probabilities in the forward direction is $s_d s_c s_b s_a$, and in the reverse direction is $s_c s_d s_a s_b$. . Therefore the forward path has the same probability as the backward path.

The argument holds for a larger path which goes around visiting more states inside $\boxed{\text{a}\sim\text{c}\sim\text{d b}}$. This is because the large path inside $\boxed{\text{a}\sim\text{c}\sim\text{d b}}$ also starts from state $\{a, d, c, b\}$ and ends at state $\{c, d, a, b\}$, and it is already proven that sub-chain $\boxed{\text{a}\sim\text{c}\sim\text{d b}}$ is time reversible. A similar argument can be given for every state in $\boxed{\text{a}\sim\text{b}\sim\text{c d}}$. Therefore, the Markov chain for $n = 4$ is time reversible.

Similarly, we can construct the state diagram for the Markov chain with five elements. It can be seen from Figure 6 that it consists of five symmetric sub-chains which are equivalent to the Markov chains with four elements.

3. Induction hypothesis.

Assuming that the Markov chain for k records $\{R_1, \ldots, R_k\}$ is time reversible, we shall prove that the Markov chain for $(k+1)$ records $\{R_1, \ldots, R_k, R_{k+1}\}$ is time reversible. The state diagram with $n = k + 1$ is shown in Figure 7.

As before, considering any state π_s, say $\pi_s = \{R_1, \ldots, R_k, R_{k+1}\}$, from one of the sub-chains $\boxed{R_1 \sim \ldots \sim R_k\ R_{k+1}}$. We need to prove that starting in π_s, any path back to π_s has

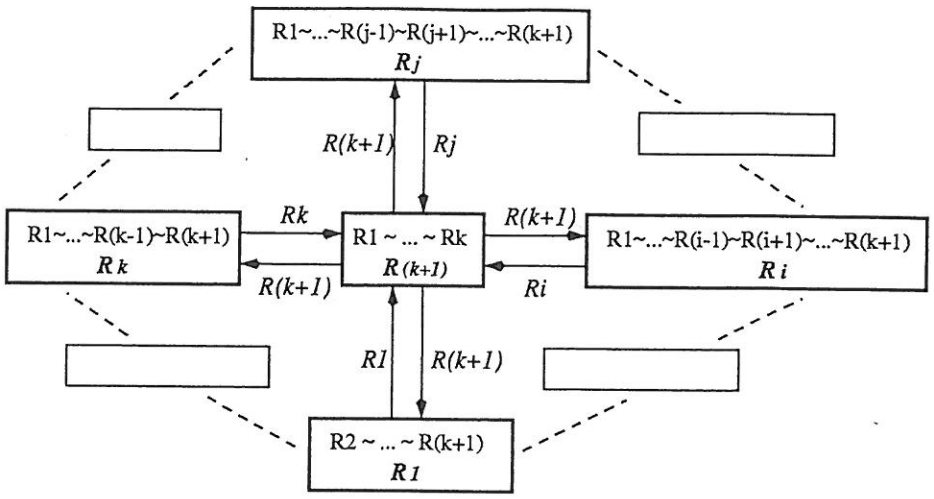


Figure 7: Simplified state diagram with $(k+1)$ elements per state using the SWP rule.

the same probability as the reversed path. Again, by our hypothesis, we don't need to consider paths that are inside the sub-chain, but only paths that go outside the sub-chain. To reach any state outside the sub-chain, element R_{k+1} has to be accessed, let R_p be the parent of R_{k+1}, this makes the transition from π_s to $\{R_1, \ldots, R_{p-1}, R_{k+1}, R_{p+1}, \ldots, R_k, R_p\}$ in sub-chain $\boxed{R_1 \sim \ldots \sim R_{p-1} \sim R_{p+1} \sim \ldots \sim R_{k+1}\ R_p}$. Since all sub-chains are symmetrical, to get back to π_s, we only need to consider the path that is directly from sub-chain $\boxed{R_1 \sim \ldots \sim R_{p-1} \sim R_{p+1} \sim \ldots \sim R_{k+1}\ R_p}$. A shortest path which visits only two states inside $\boxed{R_1 \sim \ldots \sim R_{p-1} \sim R_{p+1} \sim \ldots \sim R_{k+1}\ R_p}$ is

$$\begin{aligned}
\{R_1, R_2, \ldots, R_k, R_{k+1}\} &\rightleftharpoons \{R_1, R_2, \ldots, R_{p-1}, R_{k+1}, R_{p+1}, \ldots, R_k, R_p\} \\
&\rightleftharpoons \{R_2, R_1, \ldots, R_{p-1}, R_{k+1}, R_{p+1}, \ldots, R_k, R_p\} \\
&\rightleftharpoons \{R_2, R_1, \ldots, R_{p-1}, R_p, R_{p+1}, \ldots, R_k, R_{k+1}\} \\
&\rightleftharpoons \{R_1, R_2, \ldots, R_k, R_{k+1}\}\,.
\end{aligned}$$

The product of the transition probabilities in the forward direction is $s_{k+1}s_2s_ps_1$, whereas in the reverse direction, it is $s_2s_{k+1}s_1s_p$. The argument holds for any large path for the same reason stated for the case where $n = 4$. Therefore, the Markov chain is time reversible, and the inductive step is complete. □

Theorem 4.2 *For a given set of records* $\{R_1, R_2, \ldots, R_n\}$ *with respective access probabilities* $\{s_1, s_2, \ldots, s_n\}$, *under the swap-with-parent heuristic the stationary probabilities obey:*

$$\frac{P\{R_{i_1} \ldots \mathbf{R_{i_j}} \ldots \mathbf{R_{i_{2j}}} \ldots R_{i_n}\}}{P\{R_{i_1} \ldots \mathbf{R_{i_{2j}}} \ldots \mathbf{R_{i_j}} \ldots R_{i_n}\}} = \frac{s_{\mathbf{i_j}}}{s_{\mathbf{i_{2j}}}} \tag{5}$$

and

$$\frac{P\{R_{i_1} \ldots \mathbf{R_{i_j}} \ldots \mathbf{R_{i_{2j+1}}} \ldots R_{i_n}\}}{P\{R_{i_1} \ldots \mathbf{R_{i_{2j+1}}} \ldots \mathbf{R_{i_j}} \ldots R_{i_n}\}} = \frac{s_{\mathbf{i_j}}}{s_{\mathbf{i_{2j+1}}}} \tag{6}$$

for $1 \le j < n$ *if* $s_k \neq 0$ *for* $1 \le k \le n$. □

Proof: The results follows immediately from Theorem 4.1 as a consequence of Equation (7.1) of [20] p138. □

Theorem 4.2 shows a relationship between the stationary probabilities of two states where the list orderings differ by two records which have a parent/child relationship. Corollary 4.1 below shows the relationship between the stationary probabilities of two states where the two swapped records do not have parent/child relationship. In a more general case, for a given set of records $\{R_1, \ldots, R_n\}$ with respective access probabilities $\{s_1, \ldots, s_n\}$, we are interested in the relationship between the stationary probabilities of two *arbitrary* states. This is given by Theorem 4.3.

Corollary 4.1 *Under the swap-with-parent rule the stationary probabilities obey:*

$$\frac{P\{\ldots, \mathbf{R_i}, R_{i_1}, \ldots, R_{i_k}, \mathbf{R_j}, \ldots\}}{P\{\ldots, \mathbf{R_j}, R_{i_1}, \ldots, R_{i_k}, \mathbf{R_i}, \ldots\}} = \left(\frac{s_\mathrm{i}}{s_\mathrm{j}}\right)^{k_{ij}} \tag{7}$$

where $k_{ij} = |\,\lfloor \lg(\pi(i)) \rfloor - \lfloor \lg(\pi(j)) \rfloor\,|$, *for* $1 \le i, j \le n$ *if* $s_j \neq 0$.

Proof: This follows easily by successive swapping using Equation (5) and (6). □

It is advantageous to see the results shown above through an example. Considering a list of 10 elements $\{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{10}\}$. Figure 8 illustrates the sequence of transformations from state π_s to state π_t, where $\pi_s = \{R_1, R_2, \mathbf{R_3}, R_4, \ldots, R_8, \mathbf{R_9}, R_{10}\}$ and $\pi_t = \{R_1, R_2, \mathbf{R_9}, R_4, \ldots, R_8, \mathbf{R_3}, R_{10}\}$.

To go from state π_s to state π_t, swapping element R_9 with its parent until it reaches the root and then swapping R_3 with it, now R_9 is in the position of R_3. Then swapping R_1 with R_3, then R_2 with R_3, then R_4 with R_3. We have reached state π_t. Let π_i denote all the states during the transformation, then

$$\begin{aligned}
\pi_s &= \{R_1, R_2, \mathbf{R_3}, R_4, R_5, R_6, R_7, R_8, \mathbf{R_9}, R_{10}\}, \\
\pi_1 &= \{R_1, R_2, R_3, \mathbf{R_9}, R_5, R_6, R_7, R_8, \mathbf{R_4}, R_{10}\}, \\
\pi_2 &= \{R_1, \mathbf{R_9}, R_3, \mathbf{R_2}, R_5, R_6, R_7, R_8, R_4, R_{10}\}, \\
\pi_3 &= \{\mathbf{R_9}, \mathbf{R_1}, R_3, R_2, R_5, R_6, R_7, R_8, R_4, R_{10}\}, \\
\pi_4 &= \{\mathbf{R_3}, R_1, \mathbf{R_9}, R_2, R_5, R_6, R_7, R_8, R_4, R_{10}\}, \\
\pi_5 &= \{\mathbf{R_1}, \mathbf{R_3}, R_9, R_2, R_5, R_6, R_7, R_8, R_4, R_{10}\}, \\
\pi_6 &= \{R_1, \mathbf{R_2}, R_9, \mathbf{R_3}, R_5, R_6, R_7, R_8, R_4, R_{10}\}, \\
\pi_t &= \{R_1, R_2, \mathbf{R_9}, R_4, R_5, R_6, R_7, R_8, \mathbf{R_3}, R_{10}\}.
\end{aligned}$$

From Equation (5), we have

$$\begin{aligned}
P\{\pi_s\} &= (s_4 / s_9)\, P\{\pi_1\}, \\
P\{\pi_1\} &= (s_2 / s_9)\, P\{\pi_2\}, \\
P\{\pi_2\} &= (s_1 / s_9)\, P\{\pi_3\}, \\
P\{\pi_3\} &= (s_9 / s_3)\, P\{\pi_4\}, \\
P\{\pi_4\} &= (s_3 / s_1)\, P\{\pi_5\}, \\
P\{\pi_5\} &= (s_3 / s_2)\, P\{\pi_6\}, \\
P\{\pi_6\} &= (s_3 / s_4)\, P\{\pi_t\}.
\end{aligned}$$

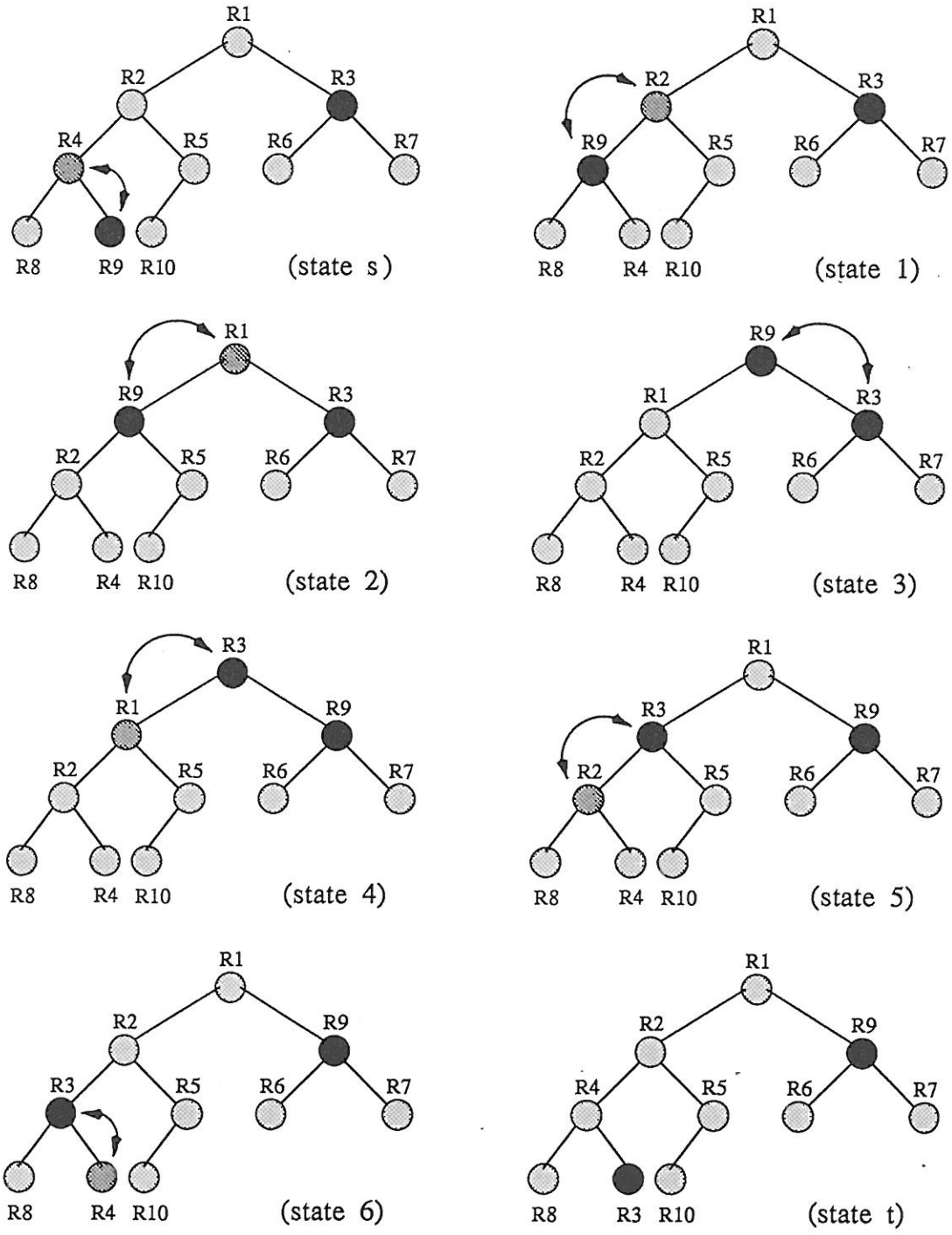


Figure 8: Transformations from state π_s to π_t through $\pi_1, \ldots, \pi_6$.

Therefore,

$$P\{\pi_s\} = \frac{s_4}{s_9} \times \frac{s_2}{s_9} \times \frac{s_1}{s_9} \times \frac{s_9}{s_3} \times \frac{s_3}{s_1} \times \frac{s_3}{s_2} \times \frac{s_3}{s_4} \times P\{\pi_t\} = \left(\frac{s_3}{s_9}\right)^2 P\{\pi_t\}. \tag{8}$$

Since $depth(R_3) = 1$ and $depth(R_9) = 3$, the difference between the depths of R_3 and R_9 is 2. Therefore Equation (8) above verifies the result shown in Corollary 4.1.

Theorem 4.3 *Under the swap-with-parent heuristic, the stationary probabilities between any two arbitrary states π_s and π_t obey:*

$$\frac{P\{\pi_s\}}{P\{\pi_t\}} = \prod_{1 \le i \le n} s_i^{\xi_i(\pi_t, \pi_s)} \tag{9}$$

where $\xi_i(\pi_t, \pi_s) = \lfloor \lg(\pi_t(i)) \rfloor - \lfloor \lg(\pi_s(i)) \rfloor$ for $1 \le i \le n$ is the difference between the depths of R_i in ordering π_s and π_t.

Proof: Since the Markov chain representing the scheme is time reversible, so we can reach any state from any other by a sequence of one or more transformations. Assuming that the sequence of intermediate states which transforms state π_s to state π_t are $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}$ for $1 \le k \le (n! - 2)$, we can write

$$\frac{P\{\pi_s\}}{P\{\pi_t\}} = \frac{P\{\pi_s\}}{P\{\pi_{i_1}\}} \times \frac{P\{\pi_{i_1}\}}{P\{\pi_{i_2}\}} \times \ldots \times \frac{P\{\pi_{i_{k-i}}\}}{P\{\pi_{i_k}\}} \times \frac{P\{\pi_{i_k}\}}{P\{\pi_t\}}.$$

Corollary 4.1 gives the expression for each of the fractions on the right hand side of the above equation. Thus

$$\frac{P\{\pi_s\}}{P\{\pi_t\}} = \prod_{u,v} \left(\frac{s_u}{s_v}\right)^k \tag{10}$$

for some u and v where $1 \le u, v \le n$ and $0 \le k \le \lfloor \lg n \rfloor$.

Note that each transformation from state π_{i_j} to state $\pi_{i_{j+1}}$ swaps record R_u and record R_v. Note also that if the *depth* of a record in π_s is shallower than its *depth* in π_t, then its access probability will appear as a numerator in the expression shown by Equation (10) above. Similarly, if the *depth* of a record in π_s is deeper than its *depth* in π_t, its access probability will appear as a denominator in Equation (10).

The number of times that record R_i will be *swapped* during the transformations from π_s to π_t is the difference in depths from when R_i is in π_s to it being in π_t, i.e, $\xi_i(\pi_t, \pi_s) = \lfloor \lg(\pi_t(i)) \rfloor - \lfloor \lg(\pi_s(i)) \rfloor$. If the depth of record R_i in π_s is shallower than its depth in π_t, then the value of $\xi_i(\pi_t, \pi_s)$ will be positive, whereas if its depth in π_s is deeper than its depth in π_t, the value of $\xi_i(\pi_t, \pi_s)$ will be negative. If the depth of R_i in π_s is the same as its depth in π_t, then the value of $\xi_i(\pi_t, \pi_s)$ is 0. Therefore,

$$\frac{P\{\pi_s\}}{P\{\pi_t\}} = \prod_{1 \le i \le n} s_i^{\xi_i(\pi_t, \pi_s)},$$

and the result is proved. □

The procedure of the proof is now clarified by means of an example, which illustrates how we actually transform an arbitrary state to another through a sequence of swaps. Suppose that $\pi_s = \{3, 4, 9, 8, 7, 5, 2, 1, 6\}$ and $\pi_t = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. To transform from π_s to π_t, one path is $\pi_s \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \pi_3 \rightarrow \pi_4 \rightarrow \pi_5 \rightarrow \pi_6 \rightarrow \pi_7 \rightarrow \pi_t$ as shown in Figure 9.

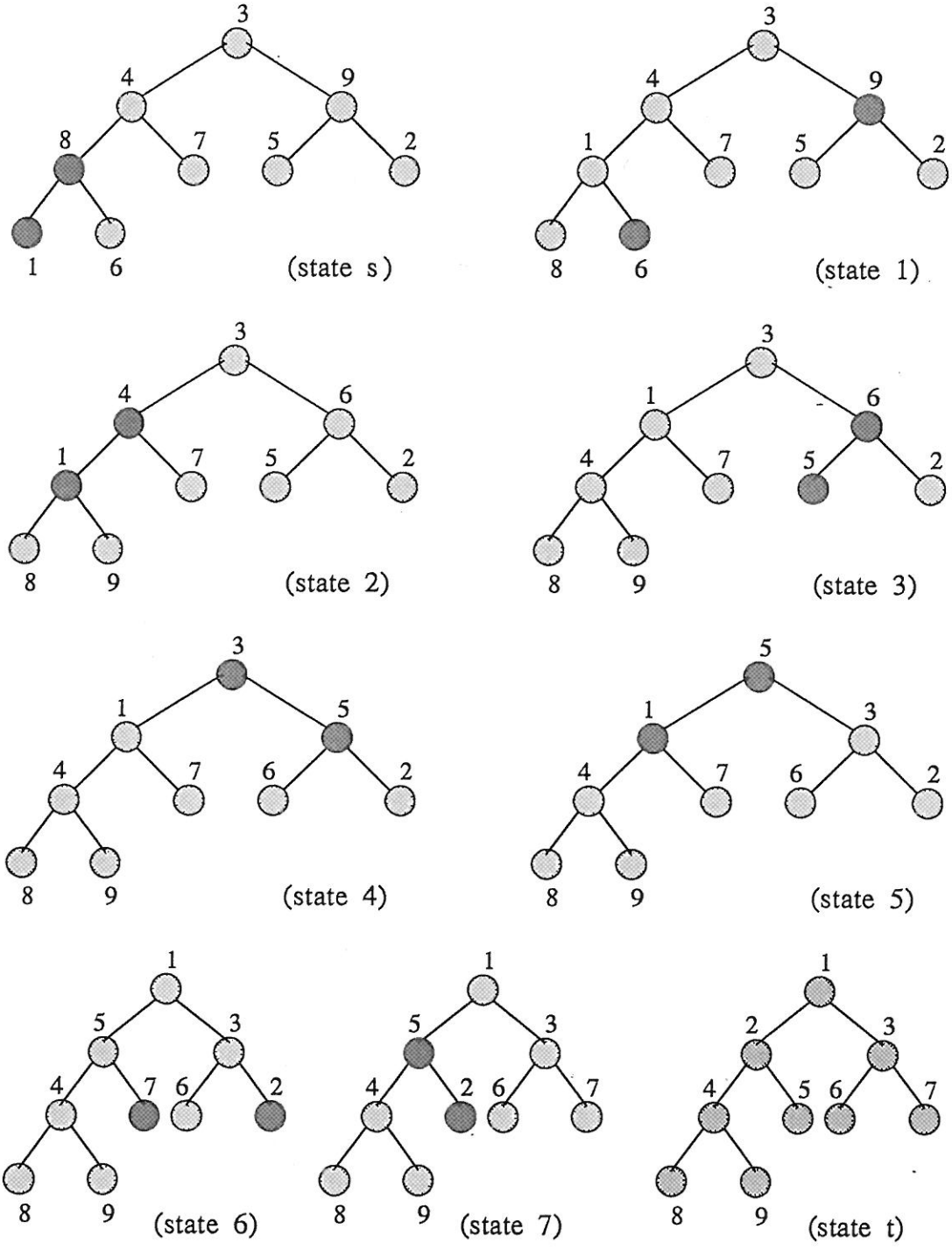


Figure 9: Transformations from state π_s to π_t through $\pi_1, \ldots, \pi_7$.

Using the stationary probabilities, we see that

$$\begin{aligned}
\frac{P\{\pi_s\}}{P\{\pi_t\}} &= \frac{P\{\pi_s\}}{P\{\pi_1\}} \times \frac{P\{\pi_1\}}{P\{\pi_2\}} \times \frac{P\{\pi_3\}}{P\{\pi_4\}} \times \frac{P\{\pi_5\}}{P\{\pi_6\}} \times \frac{P\{\pi_7\}}{P\{\pi_t\}} \\
&= \left(\frac{s_8}{s_1}\right)\left(\frac{s_9}{s_6}\right)^2\left(\frac{s_4}{s_1}\right)\left(\frac{s_6}{s_5}\right)\left(\frac{s_3}{s_5}\right)\left(\frac{s_5}{s_1}\right)\left(\frac{s_7}{s_2}\right)^0\left(\frac{s_5}{s_2}\right) \\
&= (s_3\, s_4\, s_8\, s_9^2)\,/\,(s_2\, s_6\, s_1^3) \\
&= s_1^{-3}\, s_2^{-1}\, s_3^1\, s_4^1\, s_5^0\, s_6^{-1}\, s_7^0\, s_8^1\, s_9^2 .
\end{aligned}$$

It can be easily verified that Theorem 4.3 produces the same result.

Rivest gave an expression for the asymptotic search cost of the transposition rule (see [19]). We can now give an analogous result for the average search cost of the swap-with-parent rule.

Theorem 4.4 *Let π_0 denote the identity permutation on n records $\pi_0(i) = i$ for $1 \le i \le n$, which is the optimal ordering under our assumption that $s_i \ge s_{i+1}$ for $1 \le i \le n$. Let $\xi_i(\pi_0, \pi)$ denote the quantity $(\lfloor \lg(i) \rfloor - \lfloor \lg(\pi(i)) \rfloor)$ for any permutation π and $1 \le i \le n$, which is the difference between the depths of R_i in the optimal ordering and in ordering π. Then the asymptotic search cost for the swap-with-parent heuristic is*

$$P\{\pi_0\} \sum_{all\ \pi} \left(\prod_{1 \le i \le n} s_i^{\xi_i(\pi_0,\pi)} \sum_{1 \le j \le n} s_j \pi(j) \right) \tag{11}$$

where

$$P\{\pi_0\} = \left(\sum_{all\ \pi} \prod_{1 \le i \le n} s_i^{\xi_i(\pi_0,\pi)} \right)^{-1} . \tag{12}$$

Proof: Note that the average search cost of an algorithm can be calculated by

$$\sum_{\text{all } \pi} \left(P\{\pi\} \sum_{1 \le j \le n} s_j\, \pi(j) \right) .$$

(See [7] p16, Equation 2.6.) Using Theorem 4.3, we have,

$$P\{\pi\} = P\{\pi_0\} \prod_{1 \le i \le n} s_i^{\xi_i(\pi_0,\pi)} \qquad \text{for all } \pi . \tag{13}$$

Consequently

Average search cost for swap-with-parent

$$\begin{aligned}
&= \sum_{\text{all } \pi} \left(\left(P\{\pi_0\} \prod_{1 \le i \le n} s_i^{\xi_i(\pi_0,\pi)} \right) \sum_{1 \le j \le n} s_j\, \pi(j) \right) \\
&= P\{\pi_0\} \sum_{\text{all } \pi} \left(\left(\prod_{1 \le i \le n} s_i^{\xi_i(\pi_0,\pi)} \right) \sum_{1 \le j \le n} s_j \pi(j) \right),
\end{aligned}$$

thus Equation (11) is proved.
Since

$$\sum_{\text{all } \pi} P\{\pi\} = 1,$$

applying Equation (13), we have

$$P\{\pi_0\} \sum_{\text{all } \pi} \left(\prod_{1 \le i \le n} s_i^{\xi_i(\pi_0,\pi)} \right) = 1,$$

which proves Equation (12). □

Using time reversibility, we can now prove the following results.

Theorem 4.5 *The swap-with-parent heuristic is expedient since*

$$P\{R_j \text{ precedes } R_i\}_{SWP} > \frac{1}{2} \qquad \text{if } s_j > s_i.$$

Proof: By Equation (7) we have

$$P\{\ldots, \mathbf{R_i}, R_{i_1}, \ldots, R_{i_k}, \mathbf{R_j}, \ldots\} = \left(\frac{s_i}{s_j}\right)^{k_{ij}} P\{\ldots, \mathbf{R_j}, R_{i_1}, \ldots, R_{i_k}, \mathbf{R_i}, \ldots\}$$

where $k_{ij} = |\,\text{depth}\,(i) - \text{depth}\,(j)\,| \ge 0$.

Since $s_j > s_i$, therefore $\left(\frac{s_i}{s_j}\right)^{k_{ij}} < \left(\frac{s_i}{s_j}\right)^0 = 1$. The above equation implies that

$$P\{\ldots, R_i, R_{i_1}, \ldots, R_{i_k}, R_j, \ldots\} < P\{\ldots, R_j, R_{i_1}, \ldots, R_{i_k}, R_i, \ldots\},$$

summing over all states for which R_j precedes R_i using the above, we have

$$\sum_{\text{state } \pi} P\{\pi \mid R_i \text{ precedes } R_j \text{ in } \pi\} < \sum_{\text{state } \pi} P\{\pi \mid R_j \text{ precedes } R_i \text{ in } \pi\},$$

which implies that

$$P\{R_i \text{ precedes } R_j\} < P\{R_j \text{ precedes } R_i\}.$$

Since $P\{R_i \text{ precedes } R_j\} = 1 - P\{R_j \text{ precedes } R_i\}$, it yields

$$P\{R_j \text{ precedes } R_i\} > \frac{1}{2},$$

which completes the proof. □

Theorem 4.6 *For any two records R_i and R_j, let $k_{ij} = |\,\lfloor \lg(\pi(i)) \rfloor - \lfloor \lg(\pi(j)) \rfloor\,|$, for $1 \le i, j \le n$. Then under the swap-with-parent heuristic we have:*

$$P\{R_j \text{ precedes } R_i \mid k_{ij} = d\} = \frac{1}{1 + \left(\frac{s_i}{s_j}\right)^d} = \frac{s_j^d}{s_i^d + s_j^d}. \tag{14}$$

Proof: By Equation (7) we have

$$P\{\ldots, \mathbf{R_i}, R_{i_1}, \ldots, R_{i_k}, \mathbf{R_j}, \ldots\} = \left(\frac{s_i}{s_j}\right)^{k_{ij}} P\{\ldots, \mathbf{R_j}, R_{i_1}, \ldots, R_{i_k}, \mathbf{R_i}, \ldots\},$$

summing over all states for which R_j precedes R_i and $k_{ij} = d$ using the above, we have

$$\sum_{\text{state } \pi} P\{\pi \mid R_i \text{ precedes } R_j \text{ in } \pi \text{ and } k_{ij} = d\} =$$

$$\left(\frac{s_i}{s_j}\right)^d \sum_{\text{state } \pi} P\{\pi \mid R_j \text{ precedes } R_i \text{ in } \pi \text{ and } k_{ij} = d\},$$

which implies that

$$P\{R_i \text{ precedes } R_j \mid k_{ij} = d\} = \left(\frac{s_i}{s_j}\right)^d P\{R_j \text{ precedes } R_i \mid k_{ij} = d\}.$$

Since $P\{R_i \text{ precedes } R_j\} = 1 - P\{R_j \text{ precedes } R_i\}$, it yields

$$P\{R_j \text{ precedes } R_i \mid k_{ij} = d\} = \frac{1}{1 + \left(\frac{s_i}{s_j}\right)^d} = \frac{s_j^d}{s_i^d + s_j^d}.$$

Hence the result. □

Using Theorem 4.1, it is very easy to show that the swap-with-parent rule is not as good as the transposition rule in terms of the asymptotic cost. However we conjecture that it is better than the move-to-front rule when $n > 3$. Experimental results shown in Table 1 support our conjecture. (See section 3.5 for the details of the experiments.)

We shall now present various enumeration arguments which lead us to believe that the swap-with-parent rule is better than the move-to-front rule. To prove rule A is better than rule B, it is sufficient to prove that the asymptotic probability that record R_j precedes record R_i under rule A is greater than that under rule B given that $s_j > s_i$ for all $j \neq i$. (See [7] Theorem 2.1, p14.) Therefore, to prove that the swap-with-parent rule is superior to the move-to-front rule, we need to prove that

$$P\{R_j \text{ precedes } R_i\}_{SWP} > P\{R_j \text{ precedes } R_i\}_{MTF} = \frac{s_j}{s_i + s_j}$$

given that $s_j > s_i$ for $1 \leq i, j \leq n$ and $j \neq i$.

By Theorem 4.6, we know that

$$P\{R_j \text{ precedes } R_i \mid k_{ij} = d\}_{SWP} = \frac{s_j^d}{s_i^d + s_j^d} \qquad \text{for } 0 \leq d \leq \lfloor \lg n \rfloor.$$

Notice that if $s_j > s_i$, then

$$P\{R_j \text{ precedes } R_i \mid k_{ij} = d = 0\}_{SWP} = \frac{1}{1+1} < \frac{1}{1 + \left(\frac{s_i}{s_j}\right)} = \frac{s_j}{s_i + s_j}. \tag{15}$$

$$P\{R_j \text{ precedes } R_i \mid k_{ij} = d = 1\}_{SWP} = \frac{s_j}{s_i + s_j}. \tag{16}$$

$$P\{R_j \text{ precedes } R_i \mid k_{ij} = d > 1\}_{SWP} = \frac{1}{1 + \left(\frac{s_i}{s_j}\right)^d} > \frac{1}{1 + \left(\frac{s_i}{s_j}\right)} = \frac{s_j}{s_i + s_j}. \tag{17}$$

Invoking the law of total probability, we have that in the average case

$$P\{R_j \text{ precedes } R_i\}_{SWP} = \sum_{d=0}^{\lg n} \left(P\{R_j \text{ precedes } R_i \mid k_{ij} = d\} * P\{k_{ij} = d\}\right). \qquad (18)$$

Unfortunately we are unable to find a bound or an expression for the value of $P\{k_{ij} = d\}$, because it is related in a fairly complex way to both the access probabilities of the records R_i and R_j, and the required swapping operations. Consequently we have not, as yet, formally proven that the swap-with-parent rule is better than the move-to-front rule. However, alluding to Equation (15), (16) and (17) above, we know that to make $P\{R_j \text{ precedes } R_i\}_{SWP} > (s_j/(s_i+s_j))$ given that $s_j > s_i$, we would want to make $P\{k_{ij} = 0\}$ as small as possible and $P\{k_{ij} = d > 0\}$ as large as possible. That is to say that given two records R_i and R_j with $s_j > s_i$, under the swap-with-parent heuristic, we hope that the *asymptotic* probability of these two records being in the same level in the corresponding s_heap is much (we don't know how much) smaller than that of them not being in the same level.

Theorem 4.7 below seems to indicate that this is true, even though the results shown in the theorem ignore the access probabilities and the required reordering rule. (The proof of the theorem can be found in [7], Appendix A. A general result can be found in [8].)

Theorem 4.7 *Consider a complete binary tree with n records $\{R_1, \ldots, R_n\}$ which are in a* random *order, (note that $n = 2^q - 1$ for some $q > 1$.) For any two records R_i and R_j, let $k_{ij} = |\lfloor \lg(\pi(i)) \rfloor - \lfloor \lg(\pi(j)) \rfloor|$ for $1 \leq i, j \leq n$. Then the following equations hold:*

$$\begin{aligned} P\{k_{ij} = d = 0\} &= \frac{1}{3}, \\ P\{k_{ij} = d = 1\} &= \frac{1}{3} + \frac{1}{n}, \\ P\{k_{ij} = d > 1\} &= \frac{1}{3} - \frac{1}{n}. \end{aligned}$$

□

Theorem 4.7 tells us that when $n > 3$,

$$P\{k_{ij} = 0\} = \frac{1}{3} \quad \text{and} \quad P\{k_{ij} > 0\} = \frac{2}{3}. \qquad (19)$$

Notice that the results shown in the above equations are based on the condition that the tree is complete, i.e., $n = 2^q - 1$ for some $q > 1$. If this condition is not true, then we have the following corollary (see [7], Appendix B for the proof. A general result can be found in [8].)

Corollary 4.2 *Consider a incomplete binary tree (the leaf level is not full) with n records $\{R_1, \ldots, R_n\}$ which are in a* random *order, (i.e., $n \neq 2^q - 1$ for some $q > 1$.) Then for any two records R_i and R_j, if $k_{ij} = |\lfloor \lg(\pi(i)) \rfloor - \lfloor \lg(\pi(j)) \rfloor|$ for $1 \leq i, j \leq n$, the following inequalities hold.*

$$P\{k_{ij} = 0\} < \frac{1}{3} \quad \textit{and} \quad P\{k_{ij} > 0\} > \frac{2}{3}. \qquad (20)$$

□

Therefore, (19) and (20) together informally seem to imply that the number of cases in which the swap-with-parent rule performs better than the move-to-front rule exceeds the number of cases in which it does not. Thus we conjecture that

$$P\{R_j \text{ precedes } R_i\}_{SWP} > P\{R_j \text{ precedes } R_i\}_{MTF} \qquad \text{if } s_j > s_i.$$

The formal conjecture is given below.

Conjecture 4.1 *The asymptotic search cost of the swap-with-parent heuristic is less than or equal to that of the move-to-front heuristic when* $n > 3$. □

The worst case cost of the swap-with-parent rule is unknown. If locality is allowed in the access sequence, it may be much worse than the move-to-front rule. For example, if we encounter the same situation that happens in the transposition rule, where two records are alternately accessed many times, they will continue to exchange places without advancing toward the front of the list. This same phenomenon will also happen with the swap-with-parent rule.

The rate of convergence of the swap-with-parent rule is open, no general formula is currently known for the *overwork* measure of convergence [3]. Intuition tells us that it should lie between the move-to-front rule and the transposition rule.

5 Implementation Details

Implementation of the swap-with-parent heuristic is very simple. It can be done with no additional pointers. The parent of the accessed record can be easily computed by using Equation (3), i.e.,

$$\text{Parent}(i) = \lfloor i/2 \rfloor .$$

Thus the array implementation is straightforward. Since each record in the list can be referred to by its index directly, whenever record R_i is found in position $\pi(i)$, its parent must be in position $\lfloor \pi(i)/2 \rfloor$. Simply switching the indices of these two records yields the actual swap-with-parent operation.

The linked list implementation can be done by having a temporary variable 'parent' which is initially pointing at the front of the list, and then moved to the next position whenever the current pointer is moved twice. Whenever record R_i is found, 'swap-with-parent' simply *exchanges* the contents of the current pointer (R_i) and the parent pointer (parent of R_i).

6 Drawbacks of the SWP Heuristic

If some of the records have access probabilities zero or a negligible small value, then a potential drawback would be present when using the swap-with-parent heuristic to organize a list.

For example, if we have a list of 15 elements which are initially ordered as shown in Figure 10. Assuming element 2 has access probability 0, then eventually it will fall to the position 8, 9, 10 or 11, and it will *never* be moved again. Therefore the cost of future accesses to the elements behind element 2 in the leaf level will be increased, and the situation will never get better. The worst case happens when one of the elements with zero access probabilities falls to the leftmost position on the leaf level since about half of the elements reside on the leaf level, and so the cost of access to these elements will always be sub-optimal. Therefore the elements with zero (or negligible) access probabilities will prove to be a drag to the entire access strategy. Indeed,

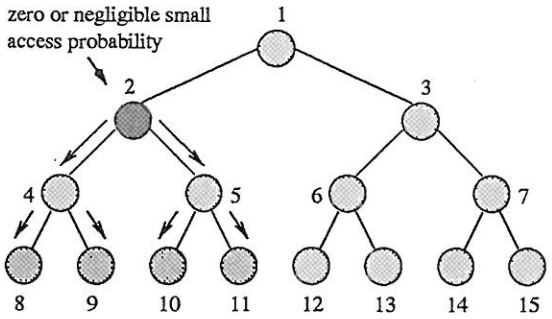


Figure 10: An s_heap of 15 elements.

in a more general setting, the elements with lower access probabilities may still cause a similar problem once they are at the leaf level.

One solution to this problem is to shift all the records between the accessed record and its parent (including the parent) back one position, instead of swapping the accessed record and its parent and leaving all the records in between unchanged. We call this scheme *move-to-parent* scheme. It is presented in the next section.

7 A Modification to the SWP Heuristic - the Move-to-Parent Heuristic

Under the *move-to-parent* (MTP) heuristic, the accessed record in the s_heap is moved to its parent's position and all the records after the parent record (including the parent record) and before the accessed record are shifted back one position. This is actually a *move-head-k* heuristic where k is not a constant but a function of the current position of the accessed record, or more explicitly its depth in the s_heap.

The move-to-parent rule avoids the drawbacks of many other algorithms. As we have seen in the previous section, it avoids the drawback caused by accessing records with zero or small access probability in the swap-with-parent heuristic. It also avoids the large asymptotic search cost of the move-to-front heuristic as every time a record with low access probability is accessed, it is moved only half way to the front of the list instead of all the way to the front of the list, which increases the costs of future accesses to only half of the records that the move-to-front rule does. It obviously avoids the slow convergence problem that the transposition rule has.

The most important of all is that when *locality* is considered, the move-to-parent heuristic avoids all the drawbacks related to locality that the move-to-front, transposition and swap-with-parent rules may have. Bitner's hybrid algorithm tends to control the locality problem, however, as we pointed out, it is difficult to know when to switch from move-to-front to transposition. Since the distance that a record is moved forward is dynamically changing under the move-to-parent heuristic, and all the records in between are moved back one position, it responds much better to the effects of locality. We shall address this problem again in Section 9.

Absolute analysis is currently not available for this scheme. We conjecture that its average search cost is in between that of the transposition rule and the move-to-front rule when $n > 3$. Note that when $n \leq 3$, the move-to-parent rule is exactly the same as the move-to-front rule. Empirical evidence shown in Table 1 supports our conjecture.

Implementation of the move-to-parent scheme is similar to that of the swap-with-parent scheme. It is achieved by keeping a temporary variable which points at the parent of the current accessed record.

8 Empirical Results

Simulating the behaviors of the algorithms is another way to compare them. In the absence of analytic results, many researchers have used simulations to test their algorithms, where in most cases, access sequences follow some known probability distributions.

A set of experiments was run to compare the asymptotic search cost of the swap-with-parent heuristic, the move-to-parent heuristic, the move-to-front heuristic and the transposition heuristic. The measure of the cost used is the number of probes required to find a record in the given list. Five different types of distributions were used to describe the access sequences:

1. *Zipf's distribution.* The individual probabilities obey:

$$s_i = \frac{1}{i\,H_n} \qquad \text{for } 1 \le i \le n$$

where $H_n = \sum_{k=1}^{n}(1/k)$ is the nth harmonic number.

2. *"80-20" distribution.* The individual probabilities obey:

$$s_i = \frac{1}{i^{1-\theta}\,H_n^{(1-\theta)}} \qquad \text{for } 1 \le i \le n \text{ and } \theta = \frac{\log 0.20}{\log 0.80} \approx 0.1386$$

where $H_n^{(1-\theta)} = \sum_{k=1}^{n}(1/k^{1-\theta})$ is the nth harmonic number of order $(1-\theta)$.

3. *Lotka's distribution.* The individual probabilities obey:

$$s_i = \frac{1}{i^2\,H_n^{(2)}} \qquad \text{for } 1 \le i \le n$$

where $H_n^{(2)} = \sum_{k=1}^{n}(1/k^2)$ is the nth harmonic number of order 2.

4. *Exponential distribution.* The individual probabilities obey:

$$s_i = \frac{1}{2^i\,K} \qquad \text{for } 1 \le i \le n$$

where $K = \sum_{k=1}^{n}(1/2^k)$.

5. *Linear distribution.* The individual probabilities obey:

$$s_i = K\,(n-i+1) \qquad \text{for } 1 \le i \le n$$

where $K = 1/\sum_{k=1}^{n} k$.

There were 12 parallel experiments conducted for lists with 100 records for each type of probability distributions. In each experiment, 300,000 accesses were performed on the list. Table 1 shows the average asymptotic search cost in terms of the number of probes required to locate a record under the move-to-front, swap-with-parent, move-to-parent and the transposition heuristics for each type of probability distributions listed above. In each case, the results

Scheme	Zipf's Distribution	80-20 Distribution	Lotka's Distribution	Exponential Distribution	Linear Distribution
MTF	26.439	31.703	4.484	27.722	41.735
SWP	24.276	29.245	4.235	27.719	41.379
MTP	22.118	27.008	3.771	23.235	38.660
SWP	19.717	24.237	3.419	20.357	34.515

Table 1: Simulation results (cost) for MTF, SWP, MTP and SWP rules.

Scheme	Zipf's Distribution (within 1%)	80-20 Distribution (within 1%)	Lotka's Distribution (within 5%)	Exponential Distribution (within 2%)	Linear Distribution (within 1%)
MTF	9000	30000	3000	5500	-1000
SWP	25000	20000	30000	20000	2500
MTP	20000	30000	20000	60000	20000
SWP	170000	160000	250000	270000	130000

Table 2: Simulation results (convergence) for MTF, SWP, MTP and SWP rules.

shown are the average result of the last 50,000 queries since the first 250,000 queries were utilized to permit the scheme to converge. Table 2 shows the *approximate* number of accesses required for the scheme to converge to its steady state (when the cost is within a certain percentage of the asymptotic cost as shown in the table) based on two experiments. [1]

From the simulation results shown in Table 1, we can see that the average search cost of the swap-with-parent scheme is no worse than the move-to-front heuristic, and the average search cost of the move-to-parent heuristic is approximately intermediate to the move-to-front heuristic and the transposition heuristic. Table 2 shows that the transposition rule is the slowest rule in terms of the number of accesses needed to converge to its steady state whereas the move-to-front rule is the fastest rule. The convergence of the swap-with-parent rule and the move-to-parent rule is between the move-to-front rule and the transposition rule.

9 The Swap-with-Parent Heuristic and Locality

In self-organizing sequential search algorithms, it is commonly assumed that the accesses to the record in the list are made independent of previous accesses and that the access probabilities do not vary with respect to time, (stationary). Consequently, the majority of the analyses on self-organizing sequential search algorithms assume that all access sequences which have the same set of access probabilities are equally likely.

[1] Twelve experiments were done for evaluating the cost values reported in Table 1. However only two experiments were done for the convergence rate since it is tedious. Unlike testing the cost where we only need the final result, testing for the convergence rate requires sampling the cost for every 1000 iterations out of total 300,000 iteration and for every distribution.

This assumption does not take into account a common attribute of access sequences called *locality* which makes the assumption unreasonable in many applications. Locality means that some "subsequences of the entire access sequence may have relative access frequencies that are drastically different from the overall relative access frequencies" [11].

Hester and Hirschberg [11] showed how the *locality* of the access sequence affects the average search cost of a scheme through the following example. Consider a list of 26 records which are the 26 English letters $\{a, b, \ldots, z\}$ in a random order. Each record is accessed exactly 10 times, that is, $s_a = s_b = \ldots = s_z = 1/26$. Let us assume that the reordering rule is the *move-to-front.*

If the access sequence is as below,

$$\underbrace{\underbrace{a, \ldots, z}, \underbrace{a, \ldots, z}, \ \ldots, \ \underbrace{a, \ldots, z}}_{10 \ times} \tag{21}$$

then under the move-to-front rule, accesses to each record (except the first access of each record) will always take 26 probes. Therefore the total number of probes required to access the records in the input sequence (except the first access to each record) is $9 \times 26 \times 26$, and the first access takes between 26 and the position of the record in the list. The best-case scenario happens when the list is initially in alphabetical order. Then the cost of the move-to-front rule will be:

$$\frac{9 \times 26 \times 26 + \sum_{i=1}^{26} i}{260} = 24.75$$

Now consider another access sequence which has the same probability distribution as the previous, but are in a different order shown below.

$$\underbrace{a, \ldots, a}_{10 \ times}, \underbrace{b, \ldots, b}_{10 \ times}, \ \ldots, \ \underbrace{z, \ldots, z}_{10 \ times} \tag{22}$$

In this case, all accesses (except the first) to each record will only take one probe. The worst-case scenario happens when the list is initially in the reversed alphabetical order, and the cost of the move-to-front rule is:

$$\frac{9 \times 26 \times 1 + \sum_{i=1}^{26} i}{260} = 3.5$$

Note that the worst-case cost of the second scenario is far less than the best-case cost of the first scenario. This indicates that the cost of a scheme can differ greatly for the same fixed probability distribution under the move-to-front rule when the order of the actual input sequences are different.

It is easy to check that the swap-with-parent rule does not have this problem in the above example. Neither does the move-to-parent rule have it. Since the move-to-parent rule is not as drastic as the move-to-front rule when moving a record forward, it can be easily verified that the cost of the first input sequence in the above example under the move-to-parent rule is less than that of the move-to-front rule. Simultaneously, the cost of the second sequence in the above example under the move-to-parent rule is more than that of the move-to-front rule. Therefore for the same fixed probability distribution, when the order of the actual input sequences are different, the cost of the move-to-parent rule does not differ as much as the move-to-front rule. This demonstrates that the move-to-parent heuristic responds in a superior fashion when it encounters this kind of 'locality'.

Another case that can happen when using the transposition rule and the swap-with-parent rule is when two records are alternately accessed many times, they will continue exchanging places without advancing toward the front of the list. However, the move-to-parent rule does not have this problem.

10 Conclusion

In this paper, we have presented two new memory-free self-organizing sequential search algorithms - the swap-with-parent heuristic and the move-to-parent heuristic. Under the swap-with-parent heuristic, the accessed record is exchanged with its "parent" (considering the list as a heap structure with no ordering constraints between parents and their children) and all the other records are untouched, whereas under the move-to-parent heuristic, the accessed record is moved to its parent's positions and all the records in between get shifted back one position.

We have shown that under the swap-with-parent heuristic, the Markov chain representing the scheme is *time reversible*. We then derived various expressions for the stationary probabilities and for the average search cost of the swap-with-parent heuristic using this property. The result looks similar to the corresponding result of the transposition rule due to the time reversibility of the Markov chains, it is actually significantly different because of the underlying s_heap structure of the list. The scheme is no better than the transposition rule. We conjecture that it is better than the move-to-front rule and that its convergence is far superior to the transposition rule. Absolute analysis for the move-to-parent rule is not available. We conjecture that its performance is intermediate to the move-to-front rule and the transposition rule, and that it is better than the swap-with-parent rule. Empirical comparisons support our conjectures.

The difference between these two algorithms and all the other algorithms presented in this paper is that unlike all the other algorithms, these two algorithms consider the size of the list when they achieve reorganization. Although our analysis assumes no locality, it is easy to check that both these rules perform better for various scenarios when locality is considered.

Acknowledgments

We are grateful to Jason Morrison for taking the time to discuss the problem with us which provided us with very helpful and motivational comments.

References

[1] E. J. Anderson, P. Nash & R. R. Weber (1982). A counterexample to a conjecture on optimal list ordering. *J. Appl. Prob. 19, 3*, pp.730-732.

[2] J. L. Bentley & C. C. McGeoch (1985). Amortized analyses of self-organizing sequential search heuristics. *Proc. 20th Allerton Conference on Comm. Control, and Computing.*

[3] J. R. Bitner (1976). Heuristics that dynamically alter data structures to reduce their access time. *University of Illinois Report UIUCDCS-R-76-818*, July, 1976 (Ph.D. paper).

[4] J. R. Bitner (1979). Heuristics that dynamically organize data structures. *SIAM J. Computing, 8, 1*, pp. 82-110.

[5] R. P. Cheetham, B. J. Oommen & D. T. H. Ng (1993). Adaptive structuring of binary search trees using conditional rotations. *IEEE Trans. Knowledge and Data Eng. 5*, pp. 695-704.

[6] T. H. Cormen, C. E. Leiserson & R. L. Rivest (1993). *Introduction to Algorithms.* The MIT Press, Cambridge, MA.

[7] J. Dong (1997). Time reversible self-organizing sequential search algorithms. M. Sc. thesis. *School of computer science, Carleton University.*

[8] J. Dong & B. J. Oommen (1997). Some Enumeration Results on Complete k-ary Trees. *In preparation.*

[9] G. H. Gonnet, J. I. Munro & H. Suwanda (1981). Exegesis of self-organizing linear search. *SIAM J. Computing, 10, 3,* pp. 613-637.

[10] W. J. Hendricks (1973). An extension of a theorem concerning an interesting Markov chain. *J. Appl. Prob. 10,* pp886-890.

[11] J. H. Hester & D. S. Hirschberg (1985). Self-organizing linear search. *ACM Computing Surveys,* pp. 295-311.

[12] Y. C. Kan & S. M. Ross (1980). Optimal list order under partial memory constraints. *J. Appl. Prob. 17,* pp1004-1015.

[13] S. Karlin & H. M. Taylor (1975). *A First Course in Stochastic Processes.* Academic Press, New York.

[14] D. E. Knuth (1973). *The Art of Computer Programming,* Vol. 3, Addison-Wesley, Reading, Ma..

[15] J. McCabe (1965). On serial files with relocatable records. *Operations Research,* 12, pp. 609-618.

[16] B. J. Oommen & J. Dong (1997). On the time reversibility of a well known self-organizing sequential search algorithm. *In preparation.*

[17] B. J. Oommen & E. R. Hansen (1987). List organizing strategies using stochastic move-to-front and stochastic move-to-rear operations. *SIAM J. Computing, 16,* pp. 705-716.

[18] B. J. Oommen, E. R. Hansen & J. I. Munro (1990). Deterministic optimal and expedient move-to-rear list organizing strategies. *Theoretical Comp. Sci., 74,* pp. 183-197.

[19] R. L. Rivest (1976). On self-organizing sequential search heuristics. *Comm. ACM, 19,* pp. 63-67.

[20] S. M. Ross (1980). *Introduction to Probability Models,* 2nd edition. Academic Press, London.

[21] A. M. Tenenbaum & R. M. Nemes (1982). Two spectra of self-organizing sequential search algorithms. *SIAM J. Computing, 11,* pp. 557-566.

School of Computer Science, Carleton University

Recent Technical Reports

TR-96-01 **Hierarchical Load Sharing Policies for Distributed Systems**
Sivarama Dandamudi and Michael Lo, January 1996

TR-96-02 **Randomized Parallel List Ranking for Distributed Memory Multiprocessors**
Frank Dehne and Siang W. Song, January 1996

TR-96-03 **Randomized Sorting on Optically Interconnected Parallel Computer**
G. Bhattacharya, J. Chrostowski, F. Dehne, P. Palacharla, January 1996

TR-96-04 **Authenticated Multi-Party Key Agreement**
Mike Just and Serge Vaudenay, February 1996

TR-96-05 **Boolean Routing on Cayley Networks**
Evangelos Kranakis and Danny Krizanc, February 1996

TR-96-06 **Secure Non-Interactive Electronic Cash**
Patrick Morin, February 1996

TR-96-07 **An Upper Bound for a Basic Hypergeometric Series**
Lefteris M. Kirousis, Evangelos Kranakis and Danny Krizanc, March 1996

TR-96-08 **A Formal Theory for Optimal and Information Theoretic Syntactic Pattern Recognition**
B.J. Oommen and R.L. Kashyap, March 1996

TR-96-09 **A Better Upper Bound for the Unsatisfiability Threshold**
Lefteris M. Kirousis, Evangelos Kranakis and Danny Krizanc, March 1996

TR-96-10 **Minimal Sense of Direction in Regular Networks**
Paola Flocchini, March 1996

TR-96-11 **Correlation Inequality and its application to a Word Problem**
Dimitris Achlioptas, Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Michael S.O. Molloy, March 1996

TR-96-12 **On Systems with Sense of Direction**
Paola Flocchini, Alessandro Roncato, Nicola Santoro, April 1996

TR-96-13 **Computing on Anonymous Networks with Sense of Direction**
Paola Flocchini, Alessandro Roncato, Nicola Santoro, April 1996

TR-96-14 **Symmetries and Sense of Direction in Labeled Graphs**
Paola Flocchini, Alessandro Roncato, Nicola Santoro, April 1996

TR-96-15 **Distance Routing on Series Parallel Networks**
Paola Flocchini, Flaminia L. Luccio, April 1996

TR-96-16 **Maximal Length Common Non-intersecting Paths**
Evangelos Kranakis, Danny Krizanc, Jorge Urrutia, May 1996

TR-96-17 **Discrete Vector Quantization for Arbitrary Distance Function Estimation**
John Oommen, I. Kuban Altinel, Necati Aras, May 1996

TR-96-18 **Symmetry and Computability in Anonymous Networks: A Brief Survey**
Evangelos Kranakis, July 1996

TR-96-19 **Many-to-One Packet Routing via Matchings**
Danny Krizanc, Louxin Zhang, July 1996

TR-96-20 **Power Consumption in Packet Radio Networks**
Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Andrzej Pelc, August 1996

TR-96-21 **Performance of Hierarchical Processor Scheduling in Shared-Memory Multiprocessor Systems**
Sivarama P. Dandamudi, Samir Ayachi, August 1996

TR-96-22 **Performance of Hierarchical Load Sharing in Heterogeneous Distributed Systems**
Michael Lo and Sivarama P. Dandamudi, August 1996

TR-96-23 **Performance Impact of I/O on Sender-Initiated and Receiver-Initiated Load Sharing Policies in Distributed Systems**
Sivarama P. Dandamudi and Hamid Hadavi, August 1996

TR-96-24 **Characterization of Domino Tilings of Squares with Prescribed Number of Nonoverlapping 2x2 Squares**
Evangelos Kranakis, September 1996

TR-96-25 **On the Impact of Sense of Direction on Communication Complexity**
Paola Flocchini, Bernard Mans, Nicola Santoro, October 1996

TR-96-26 **The Effect of Scheduling Discipline on Dynamic Load Sharing in Heterogeneous Distributed Systems**
Sivarama P. Dandamudi, November 1996

TR-96-27 **Approximating the Unsatisfiability Threshold of Random Formulas**
Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Yannis C. Stamatiou, November 1996

TR-96-28 **Paper foldings as chaotic dynamical systems**
F. Geurts, November 1996

TR-96-29 **Compositional complexity in cellular automata: a case study**
P. Flocchini, F. Geurts, November 1996

TR-96-30 **Compositional complexity in dynamical systems**
F. Geurts, November 1996

TR-96-31 **Compositional experimental analysis of cellular automata: attraction properties and logic disjunction**
P. Flocchini, F. Geurts, N. Santoro, November 1996

TR-96-32 **Approximating Weighted Shortest Paths on Polyhedral Surfaces**
Mark Lanthier, Anil Maheshwari and Jörg-Rüdiger Sack, December 1996

TR-97-01 **Performance Comparison of Processor Scheduling Strategies in a Distributed-Memory Multicomputer System**
Yuet-Ning Chan, Sivarama P. Dandamudi and Shikharesh Majumdar, January 1997

TR-97-02 **Performance Evaluation of a Two-Level Hierarchical Parallel Database System**
Yifeng Xu and Sivarama P. Dandamudi, January 1997

TR-97-03 **Using Networks of Workstations for Database Query Operations**
Sivarama P. Dandamudi, January 1997

TR-97-04 **A Performance Study of Locking Granularity in Shared-Nothing Parallel Database Systems**
S. Dandamudi, S.L. Au, and C.Y. Chow, January 1997

TR-97-05 **Continuous Learning Automata Solutions to the Capacity Assignment Problem**
B. John Oommen and T. Dale Roberts, April 1997

TR-97-06 **Discrete Learning Automata Solutions to the Capacity Assignment Problem**
B. John Oommen and T. Dale Roberts, May 1997

TR-97-07 **The Swap-with-Parent Scheme: a Self-Organizing Sequential Search Algorithm which uses Non-Lexicographic Heaps**
John Oommen and Juan Dong, May 1997

TR-97-08 **Time reversibility: a mathematical tool for creating arbitrary generalized swap-with-parent self-organizing lists**
John Oommen and Juan Dong, May 1997

TR-97-09 **Designing Syntactic Pattern Classifiers Using Vector Quantization and Parametric String Editing**
B. J. Oommen, R.K.S. Loke, May 1997

TR-97-10 **Performance of a Parallel Application on a Network of Workstations**
A. Piotrowski and S.P. Dandamudi, May 1997

All the above-listed reports are available electronically. For further information,
(1) Access the school's worldwide web server at http://www.scs.carleton.ca.
(2) Click on to the 'Technical Reports' entry found on the main home page.
(3) Make sure that you have set the 'download to local disk' option on your mosaic or other client program. Click onto the appropriate technical reports and the postscript file will be downloaded onto your local disk.