Two Methods for the Efficient Analysis of Memory Address Trace Data

ALAN JAY SMITH, MEMBER, IEEE

Abstract—**The high cost of analyzing long memory address traces has limited most researchers to short traces and analysis algorithms that are linear in the length of the trace. We suggest two methods that permit a trace to be shortened in length by one to two orders of magnitude (or more) for later further analysis. The Stack Deletion Method eliminates all references in the trace to the top k levels of the LRU stack. The Snapshot Method records the reference bits of the pages in the original tape at discrete intervals and uses these bits to generate a new trace. Extensive measurements are presented, from which we conclude that there is little or no loss in accuracy using reduced traces for many purposes for a wide range of memory sizes and degrees of reduction.**

Index Terms—**Data compression, memory management, paging, trace driven simulation.**

I. Introduction

MEMORY ADDRESS trace data, that is, a record of all memory addresses referenced during the execution of a computer program(s), has been widely used in the study of paging algorithms and scheduling for virtual memory computer systems. Belady [1] was one of the first to present extensive simulations of various paging algorithms using trace data; many other papers have appeared since that also use such data. Because of the cost of generating and analyzing traces, most researchers have been limited to a few hundred thousand or, at most, a few million memory references. Analysis of paging algorithms requiring time more than linear in the length of the trace, such as algorithms that minimize the number of "pushes and pulls" rather than just the number of faults [2] is prohibitively expensive for traces longer than about 10 000 references [3]. The simulation of computer scheduling algorithms using memory trace data is likewise unreasonably expensive.

The high cost of analyzing memory trace data has led some researchers to search for efficient algorithms [4]-[6]. In this paper, we take another approach to this problem: we choose to condense the information on a trace tape in a manner that avoids significant loss in the accuracy of our measurements and conclusions. We will employ, to achieve this condensation, the well-known observation that there is a large degree of immediate rereferencing to pages, and that over short periods of time only a very small number of pages are used. Lewis and Shedler [7], among others, have observed this fact. Denning [8] characterized this phenomenon as the "principle of locality" which states that programs execute in a series of localities; that is, they use, over short periods of time, only a subset of their pages. Madison and Batson [9] have recently made some progress in identifying and studying localities. We discuss the relation of their work to ours later in this paper.

The explication of our algorithms will require some definitions. Let $R = r_1, r_2, \cdots, r_i, \cdots$ be the memory address reference string, where, without loss of generality we substitute for the memory address the name of the page in which the address is located. We define a *time-dependent paging algorithm* as one which explicitly considers time (defined as the index i in the reference string) in making replacement or removal decisions. In this class are Working Set [10] and Page Fault Frequency [11]. A *time-independent paging algorithm* will be one which does not consider time explicitly. In this class are LRU (least recently used) [12], MIN [1], and CLOCK [12], [13], the algorithm used in Multics and in CP-67.[1] We note in particular that for the latter class of algorithms, immediate rereferencing of a page has no effect on the behavior of the algorithm. Thus LRU, MIN, and CLOCK will behave exactly the same whether processing R_1 or R_2, where $R_1 = a, b, c, c, c, c, c, a, b$ and $R_2 = a, b, c, a, b$, and where a, b, and c are page names. This rereferencing does affect the behavior of Working Set (WS) and Page Fault Frequency (PFF) as may be readily demonstrated.

Examination of stack distance functions (see [16] for examples) will show that the fraction of hits to the top levels of the stack is very high. For example, in the WATFIV trace discussed in Section IV, we find that the top five levels of the LRU stack received, respectively, 42.3 percent, 36.5 percent, 7.7 percent, 4.02 percent, and 2.45 percent of all references. Because of the high hit rate to these top levels in the stack, it should be clear that for purposes of memory management, little if any useful information is contained in this repeated rereferencing to recently used pages. Any memory management scheme, to work, must succeed in keeping these frequently referenced pages in memory.

As noted above, immediate rereferences can be eliminated from program address traces without any change in the fault count for time-independent paging algorithms. From our

Manuscript received October 20, 1975; revised Feburary 9, 1976. This work was supported in part by the Joint Services Electronics Program under Contract F44260-71-C-0087, and in part by the National Science Foundation under Grant MCS75-06768. Computer time was provided by the Energy Resources Development Administration under Contract E(043)515.

The author is with the Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

[1]The CLOCK page replacement algorithm operates as follows: A reference bit is associated with every page. When the page is referenced, this bit is set to one. Imagine the reference bits arranged in a circular list, with a pointer rotating around this list. When a page frame is needed, the reference bit pointed to by the pointer is examined. If it is zero, the page is chosen for replacement, and the pointer advanced. Otherwise, the bit is turned off (to zero), and the pointer is advanced. This page is now considered in turn.

argument above, it would appear that many other references could also be removed without a significant effect on fault counts and other program behavior statistics. One way to do this would be to remove all rereferencing within "localities" as defined by Madison and Batson [9]. We shall, instead, choose two less sophisticated methods of deleting page references from the program trace. Both methods rely heavily on the locality that characterizes almost all memory traces, but do not explicitly identify localities. The first method, the Stack Deletion Method, records only pages entering the first K stack positions; the second method, the Snapshot Method, periodically records the set of recently referenced pages. We will present comparisons of analyses in Section IV using both the condensed and original tapes; these comparisons will indicate that very little loss in accuracy occurs from the use of reduced traces.

II. Stack Deletion (Method 1 and 1*)

As indicated in the previous section, the behavior of *some* paging algorithms is completely unaffected by the deletion of immediate rereferencing from the page reference string. The deletion of immediate rereferences is equivalent to deleting all items in the reference string that are "hits to level 1" of the LRU stack [4], [14]. We extend this idea to what we call *Stack Deletion* (with two subvariants, Method 1 and Method 1*).[2]

Stack Deletion (Method 1): Delete all references in the reference string that are hits to levels $1, \cdots, k-1$. Define the *deletion parameter* D to be equal to k in this case. The reduced trace will then contain only those references that remain.

Example: Let the original reference string $R = a, b, c, c, c, c, b, b, b, d, d, a, b, c, d, c, b, c, a$. Let R_k denote the reduced string of parameter $D = k$. Then $R_1 = R, R_2 = a, b, c, b, d, a, b, c, d, c, b, c, a; R_3 = a, b, c, d, a, b, c, d, b, a$, and $R_4 = a, b, c, d, a, c, d, a$. Fault and program behavior statistics may then be gathered using the reduced string R_k ($k = 2, 3, 4$) by analyzing the reduced string (almost) exactly as if it were the original trace. One may see that the number of page faults observed in processing R with LRU replacement for memory capacities of 1, 2, 3, and 4 is (13, 10, 8, 4), and similarly, for R_2, R_3, R_4 we obtain (13, 10, 8, 4), (10, 10, 9, 4), and (8, 8, 5, 4) faults. To convert these number of misses to miss ratios requires some adjustment to the trace length.

Define N to be the length of the original string R and let N_i be the length of the reduced string R_i of parameter i. $F_i(C, A)$ is the number of faults observed when processing reduced string R_i in a memory of capacity C with page replacement algorithm A. We define the fault rate (miss ratio) $f_i(C, A)$ as $f_i(C, A) = F_i(C, A)/N$. We note that the fault rate is defined in such a way that the number of faults is divided by the length of the original trace, not the reduced trace. A consequence of this definition is that the fault rate as measured on the reduced trace ($f_i(C, A)$) is bounded by N_i/N and thus is almost certain to be inaccurate for small memory capacities. We will see later that the page fault rate is very accurately preserved for larger memory sizes.

It may be noted by the alert reader that Method 1 has no provision for time-dependent paging algorithms. One approach to estimating the fault rate for such algorithms is to calculate a "stretching factor" $S_i = N/N_i$. Then, instead of incrementing the time counter by 1 at every memory reference in the reduced trace, it is incremented by S_i. This approach fails, however, when the frequency of deletion is not uniform throughout the length of the trace. We therefore define a variant of Stack Deletion which we will call Method 1*.

Stack Deletion (Method 1):* Delete all references in the reference string R that are hits to levels $1, \cdots, k-1$. Define the deletion parameter D to be equal to k as before. The reduced string will consist of a two-tuple for every reference that remains after deletion. The two-tuple will be composed of the name of the remaining reference and the increment for the time counter.

Example: Let R be as before. Let R_k denote the reduced trace of parameter k as before. Then $R_2 = (a, 1), (b, 1), (c, 1), (b, 4), (d, 3), (a, 2), (b, 1), (c, 1), (d, 1), (c, 1), (b, 1), (c, 1), (a, 1)$. $R_3 = (a, 1), (b, 1), (c, 1), (d, 7), (a, 2), (b, 1), (c, 1), (d, 1), (b, 2), (a, 2)$. R_4 may be easily calculated as well.

It may be seen that Stack Deletion (both Methods 1 and 1*) follows directly from the principle of locality. We have chosen to define our "locality" as that set of pages in the top $k-1$ positions in the LRU stack. Our reduced trace then contains a record of every entry into this locality. It is possible, given a record also of departures from the locality, to calculate the fault rate exactly for LRU replacement for all capacities greater than $k-1$ [14]. More generally this is true for any stack replacement algorithm when we define our "locality" appropriately. Because the sequences of "pushes" from the LRU locality aids only in calculating the fault rate for the LRU algorithm (in this case) we have omitted this information in the reduced trace.

It is possible to bound the error introduced into the LRU fault rate when dealing with the reduced trace. For brevity, we shall provide only a heuristic argument; it is possible to prove this rigorously. When stack deletion processing occurs, only hits to levels k and lower in the stack are kept in the reduced trace. Thus all reordering among the top $k-1$ pages in the LRU stack (for the original trace) produces no output, and thus no changes in the stack for the reduced trace. Consider the top $k-1$ positions in the stack for the reduced trace. The page in position 1 could actually belong in position $k-1$ (by our argument above), and similarly, the page in position $k-1$ in the reduced stack could actually belong in position 1 of the true LRU stack. It should also be clear that all "misplacements" in the stack must occur by permutations of the top $k-1$ positions. Therefore, the page that is pushed from the $k-1$st position in the reduced stack might actually belong (as a result) anywhere from positions 2 to $2k-2$. This malordering bound clearly carries as the page migrates down the stack since a page in stack position i at time t will be in either position 1, i, or $i+1$ at time $t+1$. Thus any given page in the reduced trace stack actually belongs anywhere from $k-2$ elements lower in the stack to $k-2$ elements higher. Since no

[2] Reportedly (according to a referee), a similar method was used by Chu and Opderbeck [11].

stack hit in the reduced trace analysis can miss by more than $k-2$, the fault rate is similarly bounded. Therefore, $F(C-k+2, \text{LRU}) \leq F_k(C, \text{LRU}) \leq F(C+k-2, \text{LRU})$. In the author's experience (see data analysis sections), this bound is very loose and in fact far less stack disordering takes place than this bound permits or than one might expect.

III. The Snapshot Method (Methods 2 and 2*)

It is possible to derive another method of trace condensation from the locality principle. Rather than recording changes in the locality, we periodically record the contents of the locality by taking a "snapshot" of the reference bits for the pages. The information contained is used to generate a new, reduced trace as follows.

Snapshot Method (Method 2): At intervals of T memory references (where T is the *interval parameter*), the list of pages in memory (linearly ordered by page name or number) is scanned and the name of every page with a reference bit set is inserted into the reduced trace. Then all of the reference bits are turned off and a marker is placed in the reduced trace. This marker indicates that the snapshot interval has expired in order to permit updating of the time counter.

Example: Let $T=2$. Let $R=a,b,c,c,c,c,b,b,b,d,d,a,b,c,d,c,b,c,a$ as before. Let R_t denote the reduced trace with interval parameter $T=t$. (This notation ordinarily produces no confusion with the Stack Deletion Method, since T is usually much greater than any feasible value of D). We use "0" as a marker. Then $R_2=a,b,0,c,0,c,0,b,0,b,d,0,a,d,0,b,c,0,c,d,0,b,c,0,a,c,0$. $R_3=a,b,c,0,c,0,b,0,a,d,0,b,c,d,0,b,c,0,a,0$, and so on. The reduced trace(s) is then analyzed in exactly the same manner as the source trace, except that the marker entries are used to update the time counter as necessary.

We define the fault rate $f_t(C,A)$ in the same manner as in the previous section.

A minor deficiency is evident in Method 2 upon close inspection. Since the entries in the reduced trace are generated in the same order each time (the list is scanned linearly), processing the reduced trace will result in "reaching down into the stack" more often than necessary. For example, the reference string $R=a,b,c,c,c,b,a,a$ will yield $R_4=a,b,c,0,a,b,c,0$ with LRU misses for R for capacities of 1, 2, and 3 of (5, 4, 3), respectively, and for R_4 of (6, 6, 3). In general, we may see that for some range of memory capacities the fault rate on the reduced trace will be too high. We therefore define a variant of the Snapshot Method called Method 2*:

Snapshot Method (Method 2):* Generate a reduced trace exactly as in Method 2 except that the list of pages is scanned in random or pseudo-random order rather than repeatedly in the same order.

In our implementation, for computational simplicity, we employed the following pseudorandom scan order: proceed through the list of pages examining the reference bits by considering (linearly and circularly) each third page on one snapshot and each fifth page on the next. (This requires, respectively, three and five passes through the list to record all set bits). Because three and five are relatively prime, this provides a fair degree of randomness between the two orders. More sophisticated randomizations are easily possible but are more difficult and slower to implement.

A version of the Snapshot Method was used by Prieve [15] in his dissertation. He obtained trace data from a machine by recording the contents of the page reference bits every 10 ms. He does not describe the exact manner in which he processed this information.

In generating his reduced trace, Prieve [15] took advantage of the fact that the original program could be executed at normal speed on a machine, and that the reference bits could be examined by periodically stopping the machine with an interrupt. Thus it was possible to generate a reduced trace representing hundreds of seconds of real machine time using two or three times that much machine time, rather than the fifty or more times more machine time required for interpretive execution and trace generation.

IV. Data Analysis

A. Introduction

A number of memory address trace tapes that record the operation of programs on the IBM System 360 are available to the author; extensive analysis of these tapes has been presented elsewhere [16]. Measurements from three of these tapes will be given in this section. These three traces include "APL," a trace of a plotting subroutine written in APL; "WATFIV," the execution of the Watfiv Fortran compiler; and "FFT," the execution of an implementation of the Fast Fourier Transform algorithm [17] written in Fortran. These three tapes were chosen from among those available in order to span a wide range of program behavior. The APL trace is stationary in its deletion frequency and accesses a relatively large number of pages, the WATFIV trace is nonstationary and accesses a somewhat smaller number of pages, and the FFT trace is for a program that has a number of small tight loops which comprise the majority of its execution time.

A large number of parameters describe the behavior of a program; among them are the fault rates for different memory capacities, page sizes, and paging algorithms, interreference intervals (time between consecutive references to the same page) for pages, intervals between page faults, and the working set size as a function of time. We have chosen to consider the first two items: the fault rates for the LRU, MIN, CLOCK, and Working Set algorithms and the interreference intervals to pages. We note that the interreference interval information is equivalent to the Working Set fault statistics. It is of course possible to study other parameters and to employ sophisticated statistical techniques, but our choice is sufficient for many purposes and to indicate the utility of our reduced traces.

B. Length and Analysis Time Reduction

The reduction in the length and analysis time of a trace when reduced using either Stack Deletion Method 1* or Snapshot Method 2* (we consider 1* and 2* unless otherwise noted as they represent refinements of 1 and 2) can be expected to be substantial for programs that display strong locality of reference. We define the decrease in length as the ratio

TABLE I
LENGTH AND ANALYSIS TIME REDUCTION RATIO OF ORIGINAL/REDUCED LENGTH OR TIME

Method	Parameter	WATFIV		FFT		APL	
		Length	Time	Length	Time	Length	Time
1	3	4.735	4.45	4.017	3.94	5.79	5.24
1	4	7.484	6.56	7.698	7.42	--	--
1	5	10.704	9.07	26.6	23.4	--	--
1	6	14.514	11.64	36.2	30.7	22.6	15.34
1	8	--	--	--	--	46.2	28.96
1	12	--	--	--	--	109.2	33.9
2	25	5.51	4.86	6.25	5.9	6.25	5.72
2	100	12.65	9.23	17.4	13.6	15.8	13.2
2	400	28.57	16.36	40.6	25.8	40.3	22.3
2	1600	63.7	25.9	136	47.8	103.2	35.6

TABLE II
PAGE FAULTS—WATFIV

Algorithm	Memory Size	Faults | Per Cent Error															
		Method 1* Parameter								Method 2* Parameter							
		1	3		4		5		6		25		100		400		1600	
LRU	30	4491	4463	- .6	4496	.1	4529	.8	4603	2.5	4486	- .1	4612	2.7	5069	12.9	6163	37
	35	2802	2761	-1.5	2786	- .6	2806	.1	2859	2	2801	0	2852	1.8	3079	9.9	3541	26
	40	1033	1058	2.4	1070	3.6	1070	3.6	1106	7.1	1037	.4	1045	1.2	1103	6.8	1269	23
	45	347	348	.3	349	.6	348	.3	350	.8	345	- .6	355	2.3	347	0	354	2.0
	50	234	236	.85	236	.85	236	.85	237	1.3	234	0	235	.4	233	- .4	234	0
	55	182	184	1.1	184	1.1	184	1.1	184	1.1	184	1.1	183	.5	181	- .5	184	1.1
	60	127	129	1.6	130	2.4	130	2.4	130	2.4	128	.8	128	.8	128	.8	132	3.9
MIN	35	1203	1194	- .8	1190	-1.1	1190	-1.1	1179	- 2	1220	1.4	1210	.6	1183	- 1.7	1096	- 8.9
	40	482	481	- .2	478	- .8	478	- .8	475	- 1.5	491	1.9	488	1.2	482	0	460	- 4.6
	45	213	213	0	212	- .5	212	- .5	211	- .7	214	.5	215	.9	211	- .9	209	- 1.9
	50	144	144	0	144	0	144	0	143	- .7	145	.7	145	.7	142	- 1.4	141	- 2.1
	55	114	114	0	114	0	114	0	113	- .9	114	0	115	.9	112	- 1.8	112	- 1.8
	60	95	95	0	95	0	95	0	95	0	95	0	95	0	95	0	94	- 1.1
CLOCK	30	5572	5518	.1	5384	-2.4	5463	- .9	5217	- 5.4	5589	1.4	5456	- 1	5232	- 5.1	5591	1.4
	35	3169	3165	- .15	3097	-2.3	3099	-2.2	3071	- 2.9	3259	2.8	3223	1.7	3019	- 4.8	3072	- 3.1
	40	1380	1358	-1.6	1328	-3.7	1263	-8.5	1289	- 6.5	1384	2.9	1315	- 4.7	1202	-13	1194	-13
	45	382	390	2.1	409	7.0	402	5.2	405	6.0	376	-1.6	400	4.7	386	1	378	- 1
	50	236	236	0	235	- .5	233	-1.3	230	- 2.5	233	-1.3	241	2.1	237	.4	234	- .9
	55	181	181	0	180	- .6	188	3.9	179	- 1.2	182	.6	184	1.7	188	3.9	182	.6
	60	141	141	0	143	1.4	144	2.1	143	1.4	143	1.4	139	- 1.4	137	- 2.8	154	9.2
Working Set	100	31943	34135	6.9					43484	36.1	33102	3.7	54726	71	34881	9.2	15729	.51
	1000	8779	8825	.5					9107	3.7	8794	.1	8839	.7	8852	.8	15729	79
	10000	626	629	.5					634	1.3	627	.2	631	.8	639	2.1	592	- 5.5
	50000	122	122	0					122	0	122	.8	122	.8	119	1.7	117	- 3.3
	100000	69	69	0					69	0	69	0	69	0	66	- 5.4	64	- 7.3

of the number of page name entries (excluding markers and counters) in the original trace to the reduced trace. We define the analysis time reduction as the similar ratio taken from the observed CPU execution time.

The three traces described earlier have been reduced using Methods 1* and 2* for a number of different values of the deletion and interval parameters; we present the time and length reduction data in Table I. Our analysis program calculated the fault rates for LRU, MIN, and CLOCK page replacement algorithms and the interreference times for Working Set simultaneously, thus our figures represent an average (or sum) of the times for each of these algorithms over a range of memory capacities. The original trace was run for one million references in each case.

We observe that for the parameter values chosen: $D = 3, 4, 5, 6, 8$, and 12, $T = 25, 100, 400$ and 1600, reductions in length ranging from approximately 4 to 135 were obtained, with significant variations occurring between programs for the same parameter values. $D = 2$ was not chosen as a sample parameter, since, as it may be seen from Section I, there is no loss of accuracy at all in eliminating immediate rereferences and simultaneously keeping track of the number of deletions. We note that the analysis time has not decreased in direct proportion to the decrease in length, but somewhat more slowly. This occurs because references in the reduced trace are likely to be deeper in the stack and are likely to require more processing per reference.

C. Stack Deletion (Method 1*) Data

Some of the data for simulations of the LRU, MIN, CLOCK, and Working Set algorithms on the WATFIV and APL reduced traces (for source tape lengths of one million references) are presented in Tables II and III. The number of faults and the percent error from the correct (unreduced trace) value for a number of different parameter values and memory capacities is given. We note the following two features for the interval deletion data.

1) With the exception of a few entries involving substantial reductions in the trace length and small memory capacity, almost every entry shows an error of less than 4 or 5 percent.

TABLE III
PAGE FAULTS—APL

Algorithm	Memory Size	Method 1* Parameter: Faults (1)	Method 1* Parameter: Per Cent Error (1)	Faults (3)	Per Cent Error (3)	Faults (6)	Per Cent Error (6)	Faults (8)	Per Cent Error (8)	Faults (12)	Per Cent Error (12)	Method 2* Parameter: Faults (25)	Per Cent Error (25)	Faults (100)	Per Cent Error (100)	Faults (400)	Per Cent Error (400)	Faults (1600)	Per Cent Error (1600)
LRU	15	5452		5549	1.8	5620	3.1	6106	12	5153	- 5.5	5584	2.4	5787	6.1	7442	36.5	5920	8.6
	20	2852		2848	- 1.5	2880	1.0	3045	6.8	2752	- 3.5	2825	- 1	2863	.4	3139	1	3766	3.2
	25	1456		1452	- .3	1440	- 1.1	1474	1.2	1430	- 1.8	1436	- 1.4	1428	- 2	1421	- 2.5	1709	17
	30	799		800	.1	801	.25	806	.75	848	6	799	0	791	- 1	802	.4	835	4.5
	35	528		528	0	531	.6	533	.94	536	1.5	526	- .4	529	.2	528	0	537	1.7
	40	395		395	0	393	- .5	394	- .3	408	3.3	395	0	392	- .8	398	.3	405	2.5
	45	310		311	.32	312	.64	313	.97	318	2.6	311	.4	313	1.0	313	1	316	1.9
MIN	20	1547		1527	- 1.3	1477	- 4.6	1437	- 7.1	1165	-24	1511	- 2.4	1539	- .5	1504	- 2.8	1406	- 9.2
	25	795		793	- .3	769	- 3.3	756	- 4.9	685	-14	793	- .3	794	- .1	785	- 1.3	757	- 4.8
	30	472		471	- .2	460	- 2.5	456	- 3.5	440	- 7.8	470	- .4	471	- .2	468	.8	457	- 3.2
	35	331		331	0	326	- 1.6	322	- 2.8	317	- 4.3	330	- .3	333	.6	328	- .9	328	- .9
	40	252		252	0	248	- 1.6	244	- 3.2	243	- 3.6	251	- .4	253	.4	249	- 1.2	249	- 1.2
	45	205		205	0	202	- 1.5	199	- 3	198	- 3.5	204	- .5	205	0	203	- 1.0	201	- .2
CLOCK	15	5938		5938	0	6081	2.4	6138	3.4	4820	-19	5934	0	6125	3.1	7119	20	6176	4.0
	20	3010		3007	- .1	3020	.3	3040	1.0	2567	-15	3010	0	2995	- .5	3128	3.9	3903	29.7
	25	1458		1434	- 1.7	1452	- .5	1425	- 2.3	1354	- 7.2	1468	.7	1493	2.4	1460	.14	1617	10.9
	30	854		860	.7	820	- 4	811	- 5	822	- 3.8	843	- 1.3	870	1.9	839	- 1.8	866	1.4
	35	556		566	1.8	569	2.4	561	.9	558	.35	549	- 1.3	559	.5	577	3.8	561	- .9
	40	422		427	1.2	414	- 1.9	407	- 3.6	419	- .7	430	1.9	435	3.1	418	- 1.0	416	- 1.5
	45	341		328	- 3.9	327	- 4.2	345	1.2	353	3.5	343	.6	337	- 1.2	329	- 3.5	334	- 2.1
	50	279		278	- .4	279	0	305	9.3	301	7.9								
Working Set	100	19897		2205	10.6	25348	28	18756	- 6	9155	.54	24834	25	42527	114	21105	6.2	9696	- 51
	1000	4172		4254	2.0	4728	13.3	5063	21	5746	38	4286	2.9	4270	25	4187	.55	9696	133
	10000	645		647	.3	660	2.3	673	4.3	717	11	658	2.2	649	.77	647	.46	640	- .6
	50000	249		250	.4	250	.4	253	1.6	238	- 4.5	250	0	250	0	250	0	249	- .4
	100000	153		154	.65	155	1.3	139	- 9	138	-10	154	0	154	0	154	0	154	0

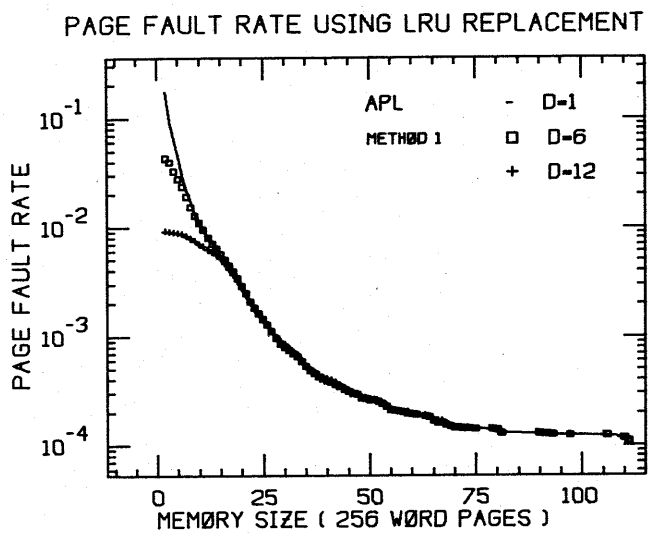

Fig. 1. Page fault rate using LRU replacement; APL trace, Stack Deletion Method (1) used for reduction, deletion parameter values = 1, 6, 12.

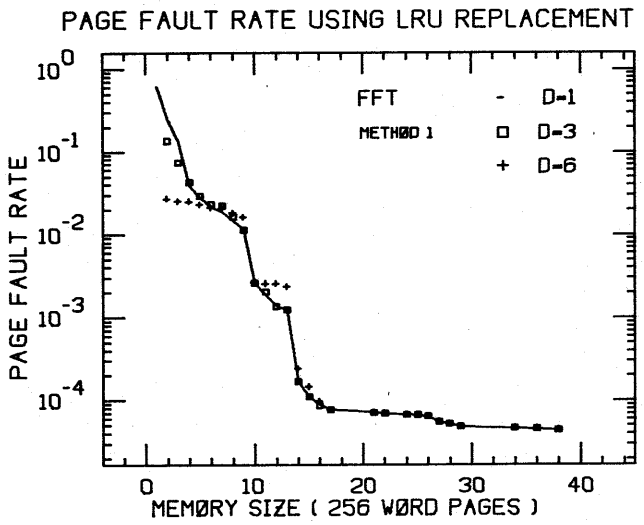

Fig. 2. Page fault rate using LRU replacement; FFT trace, Stack Deletion Method (1) used for reduction, deletion parameter values = 1, 3, 6.

2) For large memory capacities, the number of faults observed is generally small, and thus errors of one or two in the number of faults loom disproportionately large. An appropriate level of significance for the error in the number of faults $|F(C,A) - F_i(C,A)|$ is $\sqrt{F(C,A)}$, which is the standard deviation for counts of rare events (page faults) [18].

Fig. 1 shows the fault rate for the APL trace and LRU page replacement for deletion parameter values 1, 6, and 12. As indicated earlier, the maximum fault rate is bounded by N_i/N, but we see that the observed fault rate quickly converges to the true one.

The LRU fault rate for the FFT trace is given in Fig. 2 where we observe 1) the underestimate of the fault rate for small memory sizes (as noted earlier), and 2) the overestimate of the fault rate at the "corners" of the curve. The latter effect is due to the very tight loop structure of the FFT program which gets slightly distorted by eliminating loops within less than D pages. Some general rules of thumb for the degree of deletion producing acceptable error rates are presented in part E of this section. The results for the WATFIV trace are similar to that for the APL trace.

Interreference intervals were taken for Method 1* and when plotted on a scale extending from 1000 to 100 000 time units, were found to be indistinguishable from the true values.

D. The Snapshot Method

In Tables II and III we also present measurements for the Snapshot Method (2*) for parameter values of 25, 100, 400, and 1600 for both the WATFIV and APL traces. We observe that for large memory capacities or small parameter values the error introduced is negligible, being generally less than four percent. The exceptions are for small memory sizes and large parameter values where the error is sometimes substantial. We illustrate this behavior further in Figs. 3 and 4 for the APL

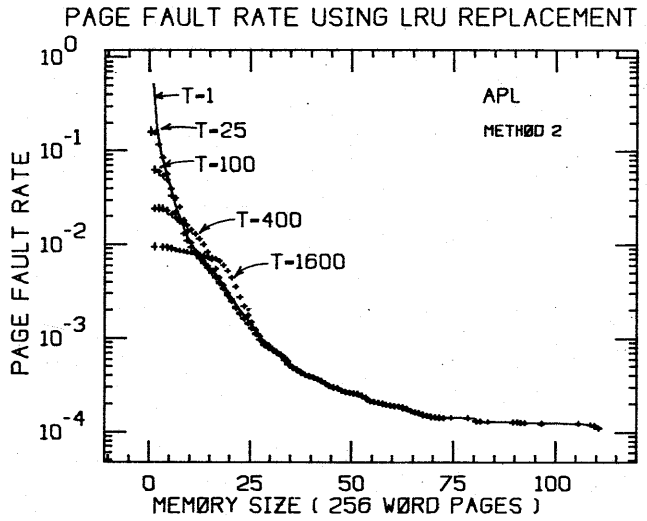

Fig. 3. Page fault rate using LRU replacement; APL trace, Snapshot Method (2) used for reduction, interval parameter values = 1, 25, 100, 400, 1600.

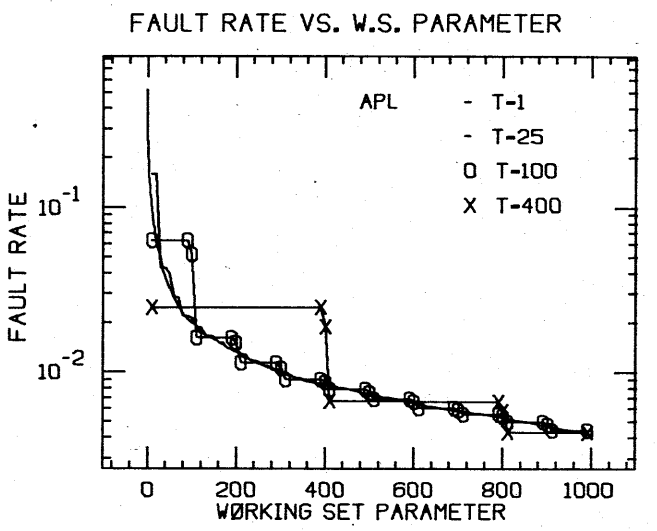

Fig. 5. Fault rate versus working set parameter; APL trace, Snapshot Method (1) used for reduction, interval parameter values = 1, 25, 100, 400.

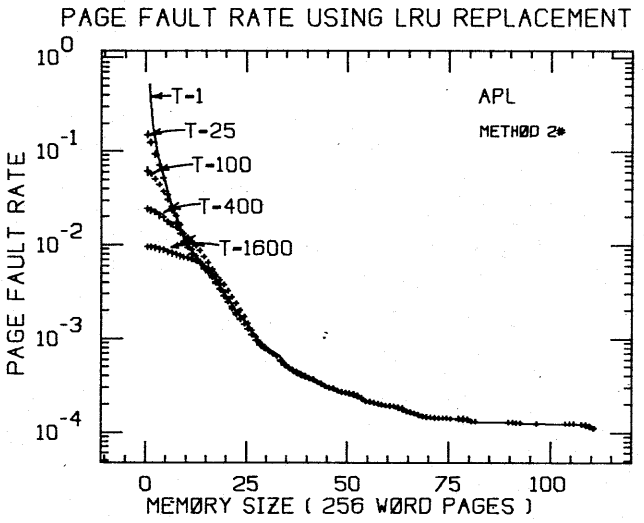

Fig. 4. Page fault rate using LRU replacement; APL trace, Snapshot Method (2*) used for reduction, interval parameter values = 1, 25, 100, 400, 1600.

TABLE IV
DIRECTION OF ERROR

Algorithm	+/- Method 1*	Method 2*
LRU	67/11	49/10
MIN	0/38	20/29
CLOCK	34/35	41/31
WS	25/5	30/9

trace. Proceeding linearly through the page list induces a substantial error in the fault rate for a range of memory sizes (Fig. 3); our pseudo-randomization of the scan order has substantially reduced this error (Fig. 4).

The fault rate for the Working Set algorithm is given in Fig. 5 for the APL trace for the Snapshot Method. We have deliberately chosen to present our results for small working set parameter values in order to demonstrate the rapid convergence of the interreference times to the actual values as the interreference times increase. The Snapshot Method produces a square wave which brackets and rapidly converges to the correct value. We note that the possible error in the time of reference to a page here is bounded by $2*T$. Thus if we denote the Working Set miss ratio as $m_T(t)$ for working set parameter t and deletion parameter T, then the miss ratio $m_T(t)$ is bounded as $m_0(t+T) \leqslant m_T(t) \leqslant m_0(t-T)$ where $m_0(t)$ is the miss ratio for the original trace.

E. Comparisons and Range of Validity

The direction of the error in the observed fault rate between the original and reduced traces, that is, whether the fault rate was over (positive error) or under (negative error) estimated, is an important parameter for many applications. In particular, any consistent pattern in the error can suggest either improvements in the algorithm or compensations to the results. In Table IV we indicate the number of positive and negative errors for each algorithm and for each method as drawn from Tables II and III and from corresponding data for FFT. Some definite and consistent patterns may be observed.

Working Set is based on the assumption that the probability of referencing a page is a monotonically declining function of the time since the last reference. LRU, similarly, is based on the assumption that the probability of referencing a page is a monotonically declining function of the number of pages that have been referenced since the last time the given page was used. If these two (similar) assumptions are well justified, then one would expect the fault rate observed from the reduced traces to increase over that from the source trace. This follows from the fact that reducing the trace allows the stack to become locally disordered (for LRU) and allows the interreference times to become inaccurate. This hypothesis is supported by the data in Table IV where we observe a preponderance of positive errors (overestimate of fault rates) for both LRU and Working Set.

One would expect the fault rate for MIN to decline when the

length of the trace decreases, since MIN is a lookahead algorithm. By removing information from the trace, the reduction algorithm allows a great deal more latitude for removal decisions, which works to the advantage of MIN. We may observe just this phenomenon in Table IV for Method 1* and to some extent for Method 2*.

There seems to be no evident pattern in the errors for the CLOCK algorithm nor does any pattern suggest itself to the author.

Comparisons of the relative degree of error for algorithms 1* and 2* for comparable reductions in the trace length are presented in Table V where we see that there is a slight but persistent bias in favor of algorithm 2*. This statistic is somewhat misleading, in that there are large and consistent errors that occur for algorithm 2* for certain parameter combinations, whereas algorithm 1* produces errors with a larger apparent variance but a less consistent bias throughout most of the parameter values examined. Also, Method 2 is guaranteed to produce absolute errors in timing no worse than $2T$, while 1* will produce LRU fault rates inaccurate at most by $k - 2$ in memory capacity.

An examination of the data presented in Tables II and III indicates that some rough rules of thumb may be given for the range over which the error rates for the reduced traces are tolerable. For algorithm 1*, it seems that for memory capacities greater than $2*D$, the error rate is extremely low and within the "noise." One can obtain a similar rule of thumb for Method 2 in a rather indirect way. A value of D corresponding to a value for T may be obtained by choosing D to yield an equivalent amount of trace reduction as T. The estimate for a safe memory capacity in this case may be taken to be $2*D$. One may also compute a result directly for timing information (e.g., working set execution); a safe parameter value is at least four to eight times as large as T. One may also observe that for the normal operating region of a program, i.e., where the fault rate is within two or three times the minimum, the reduced traces seem to give very good results for all parameter combinations tested.

F. Other Tests

In addition to the data presented here, we have conducted considerable additional experimentation and we believe that the data we have presented here are typical. There are of course many program properties that we have not tested and there are parameters that have not been varied. Page size has not been varied, although there is every reason to expect that varying it over a wide range should have no effect. Paging algorithms other than those presented have not been simulated, but those presented include three realizable algorithms, one lookahead algorithm, three stack algorithms, one non-stack algorithm (CLOCK), three time-independent algorithms, and one time-dependent algorithm. There is some doubt about the effectiveness of locality measurements (as in [9]), although Madison and Batson [9] only considered a form of reduced trace (one limited to array segments), but this requires experimentation beyond the scope of this paper. The range of possible tests and useful parameter combinations is limited only by the range of application of program traces.

TABLE V
RELATIVE MAGNITUDE OF ERROR

Algorithm	1* better/2* better
LRU	16/21
MIN	11/15
CLOCK	16/22
WS	8/15

V. OTHER METHODS

In addition to the two variants of both the Stack Deletion Method (1 and 1*) and the Snapshot Method (2 and 2*), experiments were run on additional deletion strategies. A third deletion strategy for the Snapshot Method, one which involved a linear scan of the page list, but in the opposite direction on alternate scans, was tested, and no significant difference was found in comparison to Method 2*. A third deletion method, which involved deleting all references to pages which had been previously referenced within T time units, was tested, and it was found to perform somewhat worse than Stack Deletion. It is possible that a more sophisticated deletion algorithm, such as one which deleted all references to the highest level locality [9] might produce better results, but the cost of running such a deletion algorithm might be substantially higher, and the possible gains to be expected are minor. It is our judgement that the two methods described throughout this paper yield consistently reliable results for a very low programming and processing cost in generating the reduced traces.

VI. APPLICATIONS

Reduced traces are useful in many aspects of memory management. If the reduced traces are generated directly, as by Prieve [15], then the reduced trace may serve as a method of studying paging algorithms over a long period of operation, such as many seconds or even minutes. When the reduced trace must be generated by interpretive simulation, the range of application is somewhat more limited due to the cost of the initial generation. One such direct application is to algorithms that are worse than linear in the length of the string. Yu and Baskett [3] have used an algorithm proposed by Horowitz *et al.* [2] to study algorithms which minimize page writes as well as page reads. They found themselves limited to traces of approximately 10 000 references. The use of reduced traces could extend that greatly.

Another and very important use for reduced traces is to do trace driven simulation of the interaction between memory management and scheduling. Chamberlin *et al.* [19] studied such interactions using a mathematical model for the fault rate as a function of the memory size; the use of reduced traces could permit substantially more realistic and believable results.

Ferrari (private communication) has indicated that reduced traces of the snapshot variety have been quite adequate for his program restructuring studies.

VII. Conclusions

The high cost of analyzing long memory address traces has limited most researchers to short traces and analysis algorithms that are linear in the length of the trace. We have proposed two methods to condense the trace information by as much as one to two orders of magnitude and we have shown by using such trace data that the errors from this condensation are minor and not such as to interfere with the utility of reduced traces for many applications. These two algorithms promise to make trace driven studies of memory management and scheduling problems easier, less expensive, and more accessible to many more researchers.

References

[1] L. A. Belady, "A study of replacement algorithms for a virtual storage computer," *IBM Syst. J.*, vol. 5, pp. 78–101, 1966.

[2] L. P. Horowitz, R. M. Karp, R. E. Miller, and S. Winograd, "Index register allocation," *J. Ass. Comput. Mach.*, vol. 13, pp. 43–61, Jan. 1966.

[3] F. Yu and F. Baskett, "The effect of page write costs on page replacement algorithms," to be published.

[4] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, pp. 78–117, 1970.

[5] L. A. Belady and F. P. Palermo, "On-line measurement of paging behavior by the multivalued MIN algorithm," *IBM J. Res. Develop.*, pp. 2–19, 1974.

[6] C. H. Lewis and R. A. Nelson, "Some one pass algorithms for the generation of OPT distance strings," IBM Rep. RC 4758, Mar. 1974.

[7] P. A. W. Lewis and G. S. Shedler, "Empirically derived micromodels for sequences of page exceptions," *IBM J. Res. Develop.*, pp. 86–100, Mar. 1973.

[8] P. J. Denning, "On modeling program behavior," in *1972 Proc. Spring Joint Comput. Conf.*, 1972, pp. 937–944.

[9] A. W. Madison and A. P. Batson, "Characteristics of program localities," *Commun. Ass. Comput. Mach.*, vol. 19, no. 5, pp. 285–294, 1976.

[10] P. J. Denning, "The working set model for program behavior," *Commun. Ass. Comput. Machinery*, vol. 11, pp. 323–333, May 1968.

[11] W. W. Chu and H. Opderbeck, "The page fault frequency replacement algorithm," in *1972 Proc. Fall Joint Comput. Conf.*, 1972, pp. 597–609.

[12] F. J. Corbato, "A Paging Experiment with the multics system," in *In Honor of P. M. Morse*, Ingard, Ed. Cambridge, MA: Mass. Inst. Technol., 1969, pp. 217–228.

[13] Y. Bard, "Characterization of program paging in a time sharing environment," *IBM J. Res. Develop.*, pp. 387–393, Sept. 1973.

[14] E. G. Coffman and B. Randell, "Performance predictions for extended paged memories," *Acta Informatica*, vol. 1, pp. 1–13, 1971.

[15] B. G. Prieve, "A page partition replacement algorithm," Ph.D. dissertation, Univ. California, Berkeley, CA, 1974.

[16] A. J. Smith, "A modified working set paging algorithm," *IEEE Trans. Comput.*, vol. C-25, pp. 907–914, Sept. 1976.

[17] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301.

[18] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1, 3rd ed. New York: Wiley, 1968.

[19] D. D. Chamberlin, S. H. Fuller, and L. Y. Liu, "An analysis of page allocation strategies for multiprogramming systems with virtual memory," *IBM J. Res. Develop.*, vol. 17, no. 5, pp. 404–412, Sept. 1973.

Alan Jay Smith (S'73–M'74) was born in New Rochelle, NY. He received the S.B. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA, the latter in 1974.

He is currently an Assistant Professor in the Computer Science Division of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, a position he has held since 1974. His research interests include the analysis and modeling of computer systems and devices, operating systems, and data compression.

Dr. Smith is a member of the Association for Computing Machinery, the Society for Industrial and Applied Mathematics, Eta Kappa Nu, Tau Beta Pi, and Sigma Xi.