Experiments with program locality*

by JEFFREY R. SPIRN** and PETER J. DENNING***

Princeton University
Princeton, New Jersey

INTRODUCTION

For many years, there has been interest in "program locality" as a phenomenon to be considered in storage allocation. This notion arises from the empirical observation that it is possible to run a program efficiently with only some fraction of its total instruction and data code in main storage at any given time. That virtual memory systems can be made to run at all demonstrates that program locality can be used to advantage; and though it is certainly possible to write a program which violates the principles of locality, it seems one must go out of one's way to do so.

If a program is favoring a subset of its information at some particular time, we should very much like to know the identity of that subset. The set of favored pages† of information at a given time will be called the *locality* at that time. Using this information, we may answer such questions as "What behavior can be expected of the program in the near future?" or "How much storage should be allocated to the program at this time?" For some classes of programs the best we can do is estimate this locality, whereas for others we may be able to measure it exactly. The utility of this measurement is demonstrated by the fact that, for several models of program behavior, the policy "keep the current locality in memory" can be proved to be an optimal memory management policy. These models include the independent reference model,[1] the locality model,[2,3] and the least-recently-used (LRU) stack model.[4,5,6] For other locality processes, this policy appears to be nearly optimal.[3,11]

The means of measuring the locality, and the accuracy of the measurement, depend on one's definition of "locality." The definitions that have appeared so far in the literature can be classified into two categories: the *intrinsic* locality models, and the *extrinsic* ones.

Intrinsic models for locality assume that memory references emit from a program according to some (abstract) structure internal to the program itself. The locality in effect at a given time is a function of the internal state of the program at that time. Since the state of the program may not be known, it is usually not possible uniquely to determine the locality by examining the memory reference sequence of the program. Some examples of this type of locality model are page reference distribution functions,[7,8] the independent reference model,[1] the locality model,[2,3] and the LRU stack model.[4,5] Another example can be found in Reference 6, where, for $p>0$, it is assumed that there exists a sequence of sets of pages $W_p(1)$, $W_p(2), \ldots,$ $W_p(t), \ldots,$ such that $W_p(t)$ is the smallest set of pages containing the reference at time t with probability at least p.

Extrinsic models do not rely on any assumptions of internal program state. They define locality in terms of observable properties of the memory reference sequence of the program. Three examples of extrinsic locality are: (1) Given a sequence of time intervals, the "locality sequence" $L_1L_2 \ldots L_i \ldots$ is defined so that L_i is the set of pages referenced in the ith interval; (2) Given an integer $k \geq 1$, define a sequence of time intervals so that each locality L_i in the locality sequence $L_1L_2 \ldots L_i \ldots$ contains exactly k pages—i.e., exactly k distinct pages are referenced in the ith interval; and (3) A "working set" $W(t, T)$ is defined to be the set of distinct pages referenced among the last T references, and is a measure of the locality at time t.[9,10,11]

Intrinsic models are useful primarily for analysis and simulation. They are limited by the accuracy to which

* Work reported herein was supported in part by NSF Grant GJ-30126 and NASA Grant NGR-31-001-170.
** Present address: Division of Engineering, Brown University, Providence, Rhode Island 02912.
*** Present address: Department Computer Sciences, Purdue University, Lafayette, Indiana 47906
† We assume pages are all of the same size, containing at least one word each. Most of our results extend in a straightforward manner to systems in which the block size is variable, so that the assumption of paging is mostly a matter of convenience.

they simulate real programs. Due to the practical difficulty of measuring or estimating the locality, they may have little use in storage allocation. Extrinsic models are evidently more practical, since they define a measurement procedure; yet they are obviously limited by the extent to which the measurement taken reflects what the program is really doing. Such models are less suited for use in modeling, but they can be used conveniently to allocate memory.

Although there are many models for defining the concept of locality, little experimental verification of their accuracy has been undertaken. The working set model is perhaps the only exception.[12,13,14] Unless a given model can be shown to approximate closely the behavior of real programs, any analytic results obtained using the model are only of theoretical or academic interest. Accordingly, we have chosen in this paper to emphasize experiments which *test the ability of extrinsic measurements to estimate current intrinsic localities and predict future (intrinsic) localities*, and the ability of intrinsic models to simulate real world behavior.

Let us summarize the terminology that we shall be using for the various meanings of locality. If, as discussed above, a program's memory reference string is divided into (not necessarily equal) time intervals, the (extrinsic) *observed locality* L_i is defined to be the set of pages referenced in the ith interval. Since it may be difficult to determine the internal state of a program according to an intrinsic model, we usually in practice use the observed locality in the immediate past (such as the working set $W(t, T)$) as an estimate for the current intrinsic locality; this use is termed an *estimated* locality. If we assume something about the program's internal structure, we may be able to predict, on the basis of the current (estimated) locality, the most likely references in a future interval; this is termed the (intrinsic) *predicted locality*.

For some intrinsic models, the estimated locality can quite accurately (or even perfectly) determine the current intrinsic locality. Such models are clearly of special interest, and we shall discuss two of them. A third, the independent reference model, is in general not as well measured by the working set, but is presented for comparison.

Throughout this paper, it will be assumed that demand paging is being used and that a paging algorithm is optimal if it minimizes the expected probability of a page-fault in a given size memory.

MODELS FOR INTRINSIC LOCALITY

Consider an n-page program whose pages constitute the set $N = \{1, 2, \ldots, n\}$. A *reference string* $r_1 r_2 \ldots r_t \ldots$ is the sequence of members of N generated by the program for given input data, where reference r_t is the number of the page containing the address referenced at time t (time being measured in terms of the number of memory references made by the program). Suppose a reference string has been divided up into intervals, and L_i is the observed locality in the ith interval. With respect to the given sequence of intervals, the reference string is considered to satisfy the *properties of locality* if:[10]

1. For almost all i, L_i is a proper subset of N;
2. For almost all i, L_i and L_{i+1} tend to have many pages in common; and
3. The observed localities L_i and L_{i+j} tend to become uncorrelated as j becomes large.

A program reference string is considered to have a high degree of locality if L_i is a small subset of N (statement 1), L_i and L_{i+1} differ by at most one page (statement 2), and the value of j for which L_i and L_{i+j} become uncorrelated is small compared to the length of the reference string.

A very general model for locality, displaying properties 1-3 intrinsically has been defined in Reference 3. It defines a sequence

$$(L_1, t_1)(L_2, t_2) \ldots (L_i, t_i) \ldots \qquad (1)$$

in which L_i is the ith intrinsic locality and t_i the *holding time* in L_i; the L_i are members of a specified set $\mathfrak{L}$ of localities associated with the program, and are subsets of N. During its stay in L_i, the program generates some sequence of references $r_{i1} r_{i2} \ldots r_{it_i}$ over the pages of L_i only. The mechanism for generating the references from L_i is unspecified and may be arbitrary. The *current locality* L_t at time t is that L_i for which $t_1 + \cdots + t_{i-1} < t \leq t_1 + \cdots + t_i$. A probability structure can be imposed by specifying a transition matrix $[p(L, L')]$ among localities L and L' of $\mathfrak{L}$, and a set of holding time distributions $h_L(t)$ for each L of $\mathfrak{L}$

In the following sections we shall discuss some special cases of this general model. These cases are of practical interest to the extent that our experiments indicate agreement between localities predicted by these cases and the localities actually observed by using the working set model.

The very simple locality model (VSLM)

This model assumes a *fixed size* locality—i.e., the localities L_i in (1) are all of the same size l, where $1 \leq l < n$. At any given time t, the probability of referencing an *interior page* (a member of L_t) is $1-\lambda$, and

the probability of referencing an *exterior page* (one not in L_t) and making a transition is λ. All l interior pages are referenced independently and with equal probability $(1/l)$. All $n-l$ exterior pages are referenced independently and with equal probability $(1/(n-l))$. When an interior page is referenced at time $t+1$, no change in locality occurs—i.e., $L_{t+1}=L_t$. When an exterior page is referenced, a change in locality occurs, but to a *demand-paging neighbor* only—i.e., $L_{t+1}=L_t+r_{t+1}-y$ where y is chosen at random from L_t. The unconditional probability of referencing an interior page is at least as large as that of referencing an exterior page, i.e.,

$$\frac{(1-\lambda)}{l} \geq \frac{\lambda}{n-l}, \qquad (2)$$

which is equivalent to the condition $\lambda \leq (n-l)/n$. This model has two important parameters—the locality size l and the transition probability λ—and will sometimes be called the *two-parameter model.* Note that the mean holding time in a locality is $1/\lambda$. For this model, the storage allocation rule "keep the current locality in memory" has been proved optimal.[3]

It can easily be shown that for programs which fit this model, it is impossible to determine absolutely the current intrinsic locality from observations on the generated reference string. We shall consider next the accuracy with which we can estimate the locality by an extrinsic model, namely, the working set.

As mentioned, the working set $W(t, T)$ is the set of distinct pages referenced among the references $r_{t-T+1} \ldots r_t$. If we desire to use the working set to estimate the locality, we must specify T, the window size. The choice of T must satisfy two criteria: (1) it must be large enough so that all pages within the locality are referenced with high probability, and (2) it must be small enough so that the likelihood of more than one locality transition within the window is low (for several transitions would introduce error). Although it is not obvious that a suitable T can be found, it is the case that for reasonable parameters of the VSLM, not only does a T exist, but its value is not especially critical. For the VSLM, condition 2 will hold whenever $T \leq 1/\lambda$, and our experiments verify that such values of T typically exist.

We shall consider the working set to be a good estimate of a VSLM locality when two criteria are satisfied: (a) the average working set size is approximately equal to l, the locality size, and (b) the average missing-page probability when the working set is kept in memory is approximately λ, the probability of referencing outside the locality. Plots of working set sizes and missing-page probabilities for various values of n, l, and λ show that,[3] for small λ (.01 or less), a value of T on the order of 5 or 10 times the locality size will do an excellent job of achieving criteria (a) and (b) above, irrespective of n and l. Furthermore, for small values of λ, the values of the working set size and missing-page probability level off and are nearly constant in a large neighborhood of T, indicating that the choice of T is not too critical for these values of λ.

For large values of λ (in excess of 0.05), the working set apparently does not provide as good an estimate of the locality. In this case, the working set size and missing page probabilities do not tend to level off at the values of l and λ, respectively. Furthermore, the value of window size needed to get the missing-page probability equal to λ gives a working set size as much as 20 percent too large.

The simple LRU stack model (SLRUM)

This model is based on the memory contention stack generated by the LRU (least-recently-used) page replacement algorithm.[5] This stack is simply a priority ordering on all pages of a program according to the time of their most recent usage. Thus, the first position (top) of the stack is the current reference, the second position is the next most recently used page, and so on. When the page in stack position i is referenced, it is moved to the top, and all the pages which were in positions $1 \ldots i-1$ are pushed down one position. Specifically, if $\mathbf{s}(t) = (x_1, \ldots, x_n)$ is the stack at time t and the page at position i is referenced, the stack at time $t+1$ is $\mathbf{s}(t+1) = (x_i, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.

To create the simple LRU Stack Model, we assign to each position of the stack a fixed, independent probability. We will denote these probabilities $a_1, \ldots, a_n$, where n is the number of pages in the program (and thus the number of stack positions) and $a_1+\cdots+a_n=1$. The a_i are termed *stack distance probabilities* with i being the distance (from the top of the stack). At any given time stack position i will be chosen with probability a_i; if it is chosen, the page in that position becomes the current reference and is brought to the top of the stack, as above.

If we make suitable restrictions on the a_i, we can cause this model to exhibit locality. In Reference 3, the requirement is made that the a_i be monotonically non-increasing as one goes down the stack $(a_1 \geq \cdots \geq a_n)$*. If, under this restriction the stack is divided at any point, the pages in the stack positions above the division are all more probable than those below the division. Specifically, if the stack at time t is $\mathbf{s}(t) = (x_1, \ldots, x_n)$, we can define a locality of size l

*This requirement can be weakended slightly to $\min\{a_1 \ldots a_m\} \geq \max\{a_{m+1}, \ldots, a_n\}$ for LRU paging in a memory of size m [3].

(for any l, $1 \leq l < n$) to be the pages $\{x_1, \ldots, x_l\}$. By dividing the stack at successive distances, a hierarchy of localities may be defined. This hierarchy represents a full ordering of localities, in that any given locality contains all of those smaller than it.

Note that the SLRUM is a slight generalization of the VSLM. At any given time, the probability of a locality transition is

$$\lambda = a_{l+1} + \cdots + a_n$$

since a transition occurs if and only if the distance exceeds l. Moreover, when a new locality is entered, it is the demand paging neighbor of the former locality.

It can be shown that, if m_t is the amount of memory allocated at time t, the optimal storage allocation rule for reference strings generated by this model is: "keep the top m_t elements of the stack $\mathbf{s}(t)$ in memory, for all t."[3] It follows in particular that, if m_t is the size of the working-set $W(t, T)$, the working-set policy is optimal for this model (note that $W(t, T)$ then contains precisely the top m_t elements of $\mathbf{s}(t)$). If m_t is fixed, it follows that the LRU paging algorithm is optimal for this model.

It is worth re-emphasizing that the working set $W(t, T)$ and the observed LRU stack (i.e., the one maintained by the LRU paging algorithm) both measure *exactly* the locality according to the SLRUM. For this intrinsic model, therefore, extrinsic measures provide an exact measure of locality.

The independent reference model (IRM)

In this model, the page references $r_1 r_2 \ldots r_t \ldots$ are assumed to be independent trials under some fixed probability distribution $\{c_1, \ldots, c_n\}$. In other words, the probability of referencing page i at time t is given by the stationary probability

$$\Pr[r_t = i] = c_i \tag{3}$$

Note that consecutive page references are taken according to these probabilities without regard to the previous references made by the model.

We may form a priority list for this model by ranking the pages according to decreasing probability—i.e., there is a *fixed* priority list $(1, 2, \ldots, n)$ where $c_1 \geq c_2 \geq \cdots \geq c_n$. Given a value of l, define a locality of this model to consist of pages $L(l) = \{1, 2, \ldots, l-1\}$ and that page x which was most recently fetched into memory (note that $l \leq x \leq n$), so that a locality is of the form $L(l) + x$. The rule "keep the pages of $L(m)$ in memory at all times," for memory size m pages, is known to be optimal for the IRM.[1] As in the VSLM and the SLRUM, transitions occur between demand-paging neighbors only. Unlike these other two models, however, the transition probability varies in time, being $c_l + \cdots + c_n - c_x$ whenever the locality is $L(l) + x$. The important difference between the IRM and the two previous models is, the localities of the IRM are essentially static in content whereas those of the VSLM and the SLRUM are changing in content. We shall see that, because of this difference, the IRM produces poor fits to actual programs.

CRITERIA FOR EXPERIMENTATION

It is obvious that if one were to try to correlate the reference strings produced by a model with the observed reference string of some given program, one would have very low success: Direct correlation is much too stringent a requirement to place on a model. A more reasonable, though indirect, way is to correlate interreference densities; that is, the time between consecutive references to the same page. However, even this method is likely to be inconclusive (at least over relatively short reference strings), since experimentation shows that the interreference densities of real programs tend to be quite irregular in shape, many zeros being interspersed between non-zero probabilities.

We were interested primarily in testing whether or not the reference strings generated by a given model induce the same paging behavior as those generated by a real program. Thus, we did not care about fitting strings of references within a locality, since these will be transparent to the paging system, assuming the locality is retained in memory. We were concerned, however, with locality transition behavior. Moreover, since we wanted to use the working set to estimate the locality, we desired the model to have similar working set behavior, at least on the average.

Taking these factors into account, we decided to try to fit two types of curves. The first is the *average working set size* $w(T)$, which gives the average working set size in an interval as a function of the window size T. The other is the *average missing-page probability* $q(T)$ as a function of the window size T, when exactly the working set is kept in memory at all times. It has been proved that the latter curve is (essentially) the derivative of the former,[10] and thus we are in fact fitting the model to the working set size curve and its derivative.

Now, consider the probability of a given page's not being in the working set under window size T: this can be shown equivalent to the probability of the interreference interval for the given page being greater than T.[10] Thus, the missing page probability $q(T)$ corresponds to the complementary cumulative overall

interreference probability distribution. In this way, a close fit by a model to the observed missing page probability curve guarantees a close fit to the observed overall interreference *distribution*, even though the fit to the observed *density* will, as commented earlier, tend to be quite poor.

Of course, this method of model fitting has its disadvantages. Its primary limitation is that both the curves $w(T)$ and $q(T)$ are averages measured over an interval. If the interval is too large, any non-stationary behavior will tend to be masked on the average. For this reason, most measurements were taken over what we consider to be a suitably small interval (short compared to the lifetime of the program), in most cases 20K references (about 10,000 instructions). This necessitated taking several measurements in various parts of a given program's reference string. For comparison, measurements over a larger interval, 300K references long, were also taken.

We decided for these experiments to ignore the distinction between instruction and data references. Modern computers tend to make great use of such operations as register-to-register instructions, indirect references, and multiple data reference instructions (such as LM and STM on the 360). The more a program makes use of these operations, the less true the tendency for instruction and data references to alternate. We decided not to make detailed studies of how instruction and data references are in fact mixed in practical reference strings, as this question was secondary to our interest in locality behavior. Moreover, most modern systems do not make any serious attempts to distinguish "instruction working sets" from "data working sets" in their storage allocation procedures. Nonetheless, the effects of such a distinction may be significant, and constitute a worthwhile project for future research.

For each reference string segment tested, the observed (OBS) working set curve, independent reference probabilities $\{c_i\}$, and LRU stack distance probabilities $\{a_i\}$ were measured. A single-pass algorithm for measuring the working set curve is given in Reference 10. The independent reference and LRU stack probabilities were determined by counting references to each page and to each stack position, respectively. The value used for n, the size of the total program's page set, was the total number of distinct pages actually referenced in the reference string segment being studied. Pages which were never referenced in the interval of measurement were not counted in the value of n. This was done mostly for convenience, and should have little or no effect on the results.

Also included for comparison was an attempt to fit the working set curve to the following exponential function

$$w(T) = n(1 - e^{-BT}) \quad B > 0$$

where B is a parameter. This will be termed the *exponential model* (EXP).

Using the measured values for the independent reference and LRU stack probabilities, the working set curves for these two models were computed. (Algorithms for computing the working set curves of the various models are given in Reference 3.) For the locality model, the parameters l and λ were chosen to give the lowest mean-squared relative error for the set of window sizes 10, 20, 30, . . . , 1000, against the observed $w(T)$ curve. The same procedure was repeated for the exponential model to determine a value of the parameter B.

DESCRIPTION OF RESULTS

Programs on two machines were tested for fits with the various models. The PAL assembler on the Digital

CHART 1—Description of Programs Measured

Ref. Str. No.	Machine	Page Size (words)	Description	Refs. Skipped	Refs. Measured
0	PDP-8	128	Assembler, Pass 1	0	20K
1	PDP-8	128	Assembler, Pass 1	100K	20K
2	360	256	FORTRAN (G) COMPILER	1K	20K
3	360	256	FORTRAN (G) COMPILER	200K	20K
4	360	256	Small FORTRAN job. One main loop.	1K	20K
5	360	256	Small FORTRAN job. One main loop.	100K	20K
6	360	256	Small FORTRAN job. One main loop.	1K	300K
7	360	256	Medium FORTRAN job. Several Subroutines.	1K	20K
8	360	256	Medium FORTRAN job. Several Subroutines.	100K	20K
9	360	256	Medium FORTRAN job. Several Subroutines.	1K	300K

CHART 2—Values of Parameters

Ref. Str. No.	Total pages refd.	VSLM l	VSLM λ	EXP B
0	11	4	.0025	.0013
1	12	4	.0027	.0015
2	35	5	.014	.00080
3	38	7	.022	.0012
4	20	3	.030	.0025
5	20	4	.020	.0020
6	20	4	.021	.0021
7	22	4	.024	.0020
8	20	3	.029	.0024
9	31	4	.014	.00085

Equipment PDP-8 was run using a page size of 128 words, the standard page size for the machine. Several IBM 360 programs were run, including two FORTRAN jobs and the FORTRAN (G level) compiler itself. The 360 page size was chosen arbitrarily to be 256 words. Chart 1 gives a description of each program. The "reference string number" refers to a reference string segment from each program. In particular, we expressed the reference string in the form $r_1 r_2 \ldots r_k r_{k+1} \ldots r_{k+x} \ldots$, where k is the number of "references skipped" and x the number of "references measured." In other words, $r_{k+1} \ldots r_{k+x}$ is the reference string segment over which we attempted to fit the models.

Curve fit results

Chart 2 gives the values of various measured or best-fit parameters. It is important to note that the best-fit

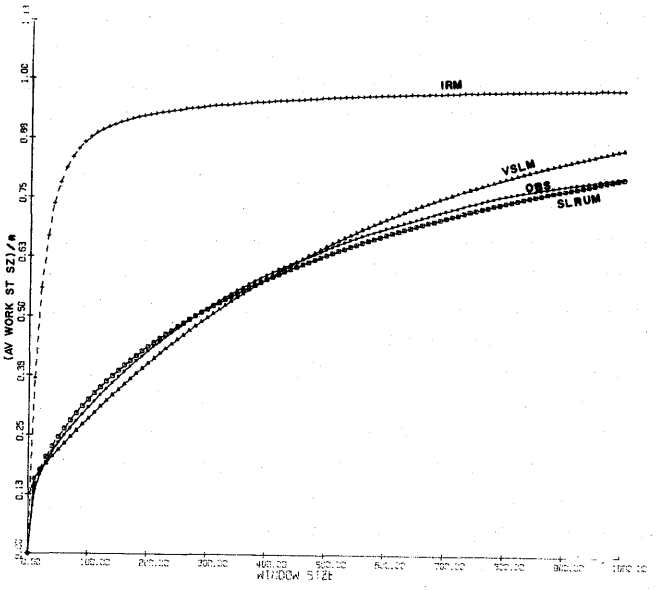

Figure 2—Working set size (Ref. St. 4)

VSLM locality size l was typically under 20 percent of the program size n, that the locality transition probability λ was typically in the range 0.01 to 0.03, and that the locality transition time $1/\lambda$ was typically in the range 30 to 100 references. Reference strings 0 and 1 were exceptions, having much lower transition probability λ than the others, this being due undoubtedly to the severely limited amount of memory on the PDP-8 (4K words).

Chart 3 gives the results of the working set curve

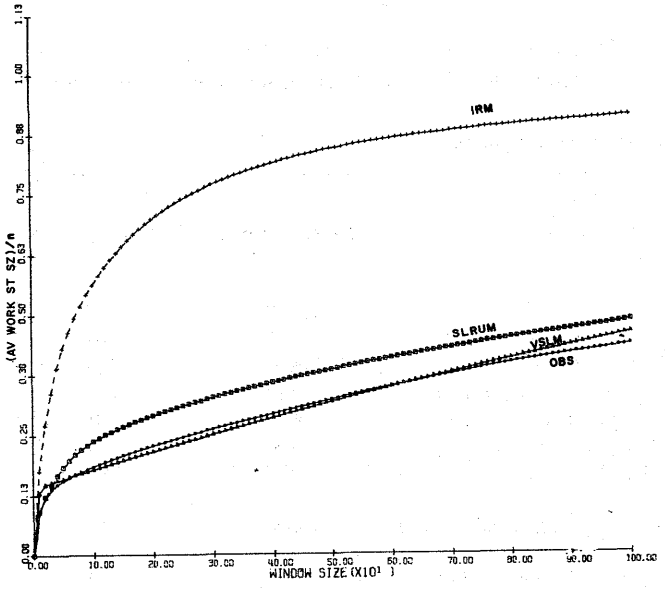

Figure 1—Working set size (Ref. St. 2)

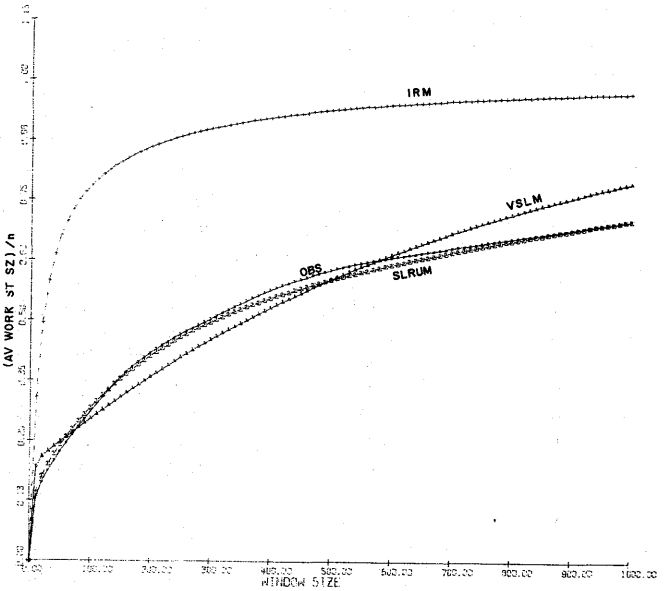

Figure 3—Working set size (Ref. St. 6)

CHART 3—Fits to mean working set size curve

Ref. Str. No.	VSLM avg. error	VSLM worst error	EXP avg. error	EXP worst error	IRM avg. error	IRM worst error	SLRUM avg. error	SLRUM worst error
0	7.5%	56%	37%	99%	84%	97%	28 %	33 %
1	6.2	49	38	99	97	106	19	24
2	6.0	49	32	97	161	208	20	29
3	11	58	32	96	109	146	6.5	8.2
4	5.3	30	20	95	95	246	2.6	7.7
5	10	53	26	96	77	200	2.3	7.9
6	9.9	54	25	96	86	210	2.6	8.1
7	8.0	51	24	96	98	225	2.8	7.9
8	6.5	29	22	95	85	207	2.9	8.9
9	10	56	31	97	162	291	8.3	9.4

fittings for the various models, and Chart 4 gives the corresponding results for the missing page probability curve. Two error measures are listed for each fit: "average relative error" over the curve, and the "worst case relative error." Except for the IRM, the worst errors occurred for very small values of T (less than 10); for the IRM, the worst errors occurred for the largest values of T (above 500). All errors are shown as per cent of the observed value. The "average relative error" is only an approximate value: it is found by taking the square root of the previously mentioned mean squared relative error (it can be shown that this represents an upper bound to the true average of the absolute values of the errors). The worst case error is the largest relative error considered over all integer window sizes in the range 1 to 1000.

It seems apparent from the data that the SLRUM performs the best over all in approximating the two curves, with the VSLM a close second. The fits of these two models are usually very good on the working set curve. The errors in fitting the missing page probability curve are larger, even unacceptably large in some cases. However, it can at least be said that even for this curve, these two models perform much better than either of the others, again with the SLRUM slightly superior. We can conclude from this that the models are better for predicting a program's memory demands than for predicting its page-fault probability; further refinements to the models are required to achieve the latter goal.

Because of its static treatment of locality, the independent reference model is the worst model of the four. It consistently overestimates the working set size, usually by a factor of 2 or 3.

Figures 1-3 show typical working set curves, and Figures 4-6 show typical missing page probability curves. All six figures show the observed (measured) curve OBS, and the results of attempting to fit each model. (EXP was omitted to aid in readability). Figures 7-9 show typical stack distance probabilities; all such curves show that the monotonically non-increasing assumption of the a_i tends to be valid for the majority of values of i. (Note the logarithmic vertical axis on these figures).

CHART 4—Fits to Missing Page Probability Curve

Ref. Str. No.	VSLM avg. error	VSLM worst error	EXP avg. error	EXP worst error	IRM avg. error	IRM worst error	SLRUM avg. error	SLRUM worst error
0	30%	228%	266%	407%	77%	376%	84 %	197%
1	36	190	301	413	167	419	85	181
2	29	132	93	127	133	426	27	118
3	82	131	157	207	70	210	5.1	26
4	32	94	40	90	103	479	16	48
5	85	157	119	224	74	417	19	43
6	72	153	100	192	81	435	18	38
7	37	158	62	110	93	450	9.5	37
8	30	92	48	91	89	420	10	37
9	57	190	117	195	102	609	14	40

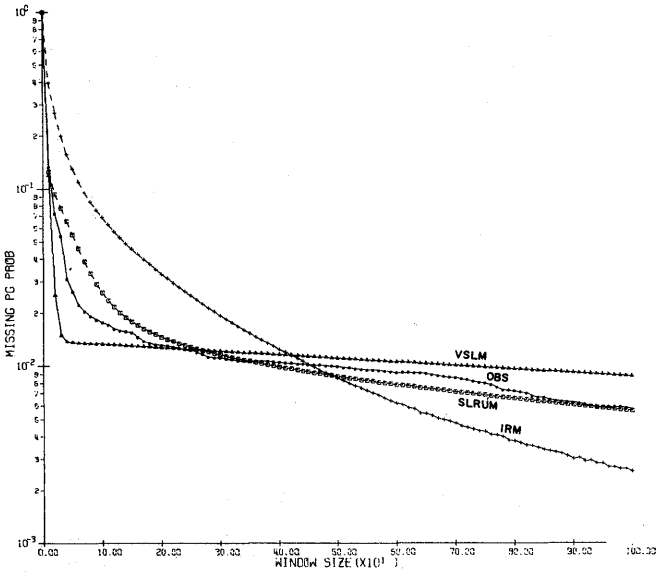

Figure 4—Missing page probability (Ref. St. 2)

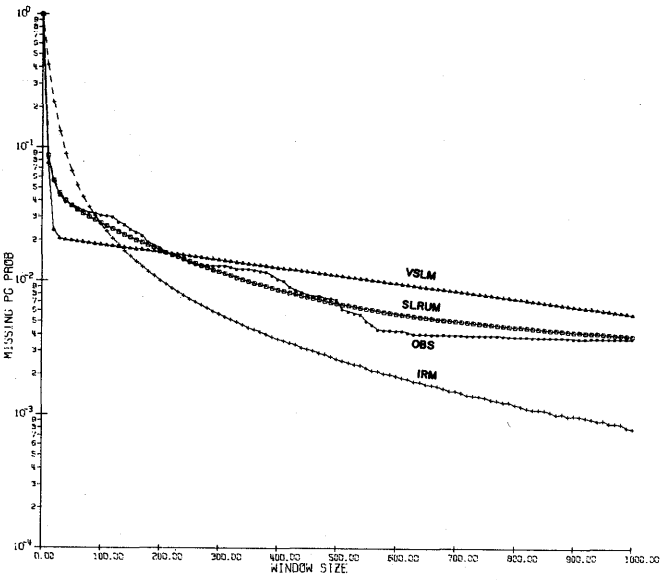

Figure 6—Missing page probability (Ref. St. 6)

Several other statistics of interest appear in Chart 5. q_I is the sum of the $n-l$ lowest measured independent reference probabilities; it gives an indication of the performance which could be expected if the program were in fact an IRM program allocated l pages of memory. q_w is the missing page probability for the working set with window size T_w, where T_w is chosen to make the average working set size equal to l. Thus, q_I and q_w apply to the same average memory size. Notice that q_I is typically an order of magnitude greater than q_w, showing much more dramatically how pronounced are the dynamic effects of locality: The assumption of static locality would have led us to predict missing page probabilities in the order of q_I whereas in fact they were in the order of q_w. This re-emphasizes the poor performance introduced by a model assuming a static locality.

It is also notable that in every case, $T_w < 1/\lambda$, where

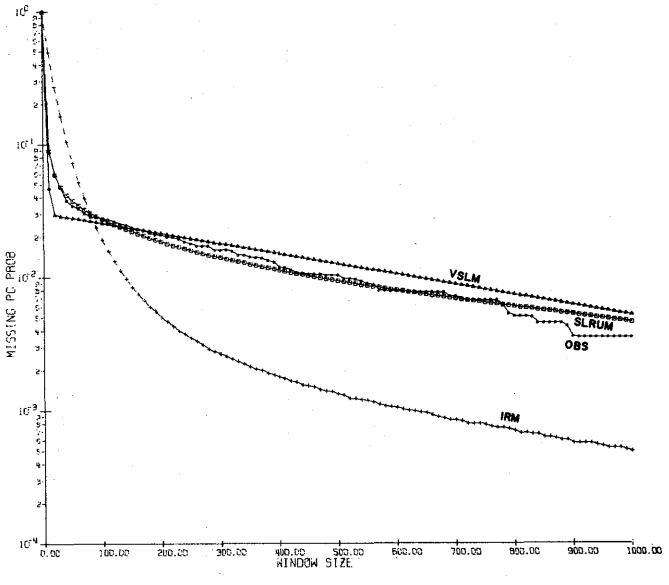

Figure 5—Missing page probability (Ref. St. 4)

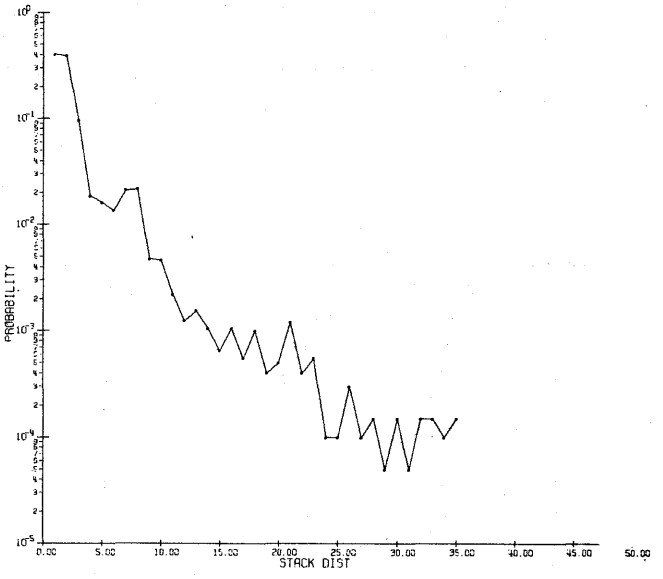

Figure 7—Distance distribution (Ref. St. 2)

$1/\lambda$ is the expected interval between locality transitions in the VSLM. Thus, it is unlikely that more than one such transition will occur in this size window, so that the working set will be a good estimator of the VSLM locality for all tested programs.

EXTENSIONS TO THE *SLRUM*

Attempts have been made to improve the SLRUM by increasing the complexity of the process by which stack distances are generated. Shedler and Tung,[5] for example, analyze a stack with a Markov process substituted for the a_i. To our knowledge, no attempts have been made to validate any extensions to the SLRUM, other than that which we shall describe below.

CHART 5—Additional Statistics

Ref. Str. No.	q_I	q_W	T_W	π_S	h_S
0	.14	.0088	75	0.	—
1	.09	.024	45	0.	—
2	.33	.035	34	.038	10.6
3	.36	.071	40	.11	12.7
4	.69	.071	15	.015	5.5
5	.48	.042	36	.015	5.7
6	.49	.041	36	.015	5.7
7	.54	.046	31	.0084	4.2
8	.63	.070	15	.0040	4.0
9	.57	.030	43	.0056	4.0

A very simple attempt was made to improve the performance of the LRU stack model. It was imagined that stack distances would be selected, as before, according to the a_i, and the a_i would be biased toward short stack distances. Occasionally, however, a new set of probabilities, the b_i, would take effect for a short time; these would be biased toward long stack distances. The distribution $\{a_i\}$ corresponds to the intuitive concept of "drifting slowly among neighboring localities," whereas the distribution $\{b_i\}$ to the notion "jumping suddenly to very different localities," or "scrambling up the entire stack." The choice between $\{a_i\}$ and $\{b_i\}$ would be determined by a 2-state Markov chain.

As has been suggested earlier, there is a distance string $d_1 d_2 \ldots d_t \ldots$ associated with the program's reference string $r_1 r_2 \ldots r_t \ldots$ being measured. Given the distance string, our problem was to determine which distances should be considered data points for the $\{a_i\}$ distribution and which for the $\{b_i\}$ distribution. Somewhat arbitrarily, we decided to count the distances toward the $\{b_i\}$ distribution whenever the majority of the last four successive distances exceeded four (four

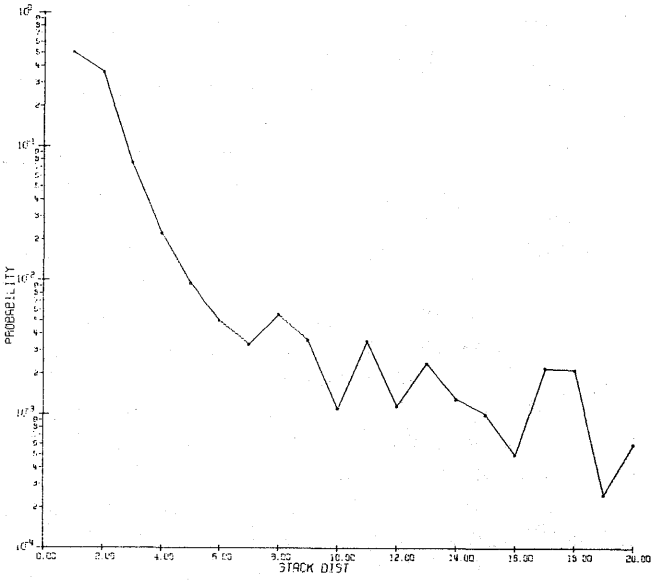

Figure 8—Distance distribution (Ref. St. 4)

was chosen since it represented a typical VSLM locality size); distances would continue to be counted toward the $\{b_i\}$-distribution until four successive distances were all at most four, in which case distances would be counted toward the $\{a_i\}$-distribution. The measured value for the steady state probability π_s of the $\{b_i\}$ (or "stack scrambling") state is shown in Chart 5; π_s is an indication of the fraction of time the program spent making large jumps between localities. Except

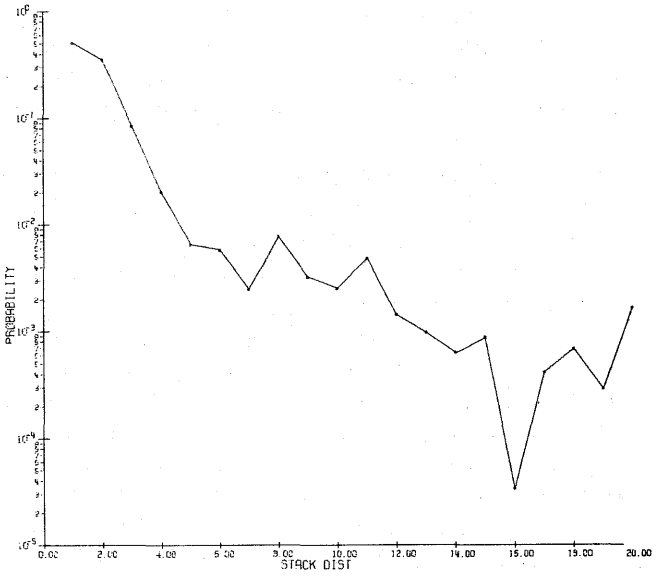

Figure 9—Distance distribution (Ref. St. 6)

for reference string 3, all the programs seemed to spend under 4 percent of their time jumping localities—i.e., they seemed to spend in excess of 96 percent of their time obeying the properties of locality.

Chart 5 also shows the mean holding time h_s in the $\{b_i\}$-state. In all cases, h_s was at least as large as the VSLM locality size l, suggesting that, when scrambling is over, the resulting locality is likely to be disjoint from the original locality.

As might be anticipated, however, the working set size and missing page probability curves generated by this extended model were in all cases indistinguishable from those produced by the SLRUM. This is because the transitions between the $\{a_i\}$ and the $\{b_i\}$-states occur independently of the process which generates stack distances. Apparently, it is necessary to make the stack-scrambling process correlated directly with the stack distance generating process, perhaps by generating distances directly from a Markov chain. Shedler and Tung's approach represents one possible solution,[5] though as yet unvalidated.

CONCLUSIONS

We have attempted here to validate experimentally several intrinsic models for the concept of program locality. We have done this with particular regard to the use of the working set as an estimator of the (intrinsic) locality. We have tried to take examples of both system software (a compiler and an assembler) and user programs, and have attempted to fit each of the models to the observed behavior of each given program.

Fitting was attempted to the measured working set size and missing page probability curves. In this way, reasonable approximations to the paging behavior of the actual programs could be obtained, without having to consider other details of the programs of less importance in paging.

Two models appear to produce good approximations to real world behavior: the two-parameter simple locality model and (especially) the LRU stack model. The independent reference model, because of its static concept of locality, does very poorly.

The working set is a good estimator of the simple two-parameter model's locality, provided the locality does not change too rapidly; we observed no case in which the locality was changing too rapidly for the working-set to be a good estimator. The working set *exactly* measures the locality in the case of the LRU stack model and is thus nearly optimal for programs whose behavior can be closely approximated by this model.

The principal conclusions to be drawn from this work are:

1. There exist non-trivial cases in which the working-set memory management policy is optimal, and evidence suggesting it will perform quite well when reference strings are generated by locality processes other than the ones studied here.
2. The concept of a "locality size" is not sharply defined, as in the case of the simple two-parameter model; it is instead a graduated concept, as in the LRU-stack model.
3. The locality at any given time receives the vast majority of references, is small compared to the program size, and is constantly changing in membership.
4. There is a tendency for transitions to occur between neighboring localities for the vast majority of the time, transitions among disjoint localities being relatively infrequent.

Stack models appear to hold great promise of being good models for program behavior, especially as we gain a better understanding of the processes by which stack distances are generated.

ACKNOWLEDGMENTS

We are grateful to J. J. Horning and K. Sevcik of the University of Toronto for many useful ideas and insights relating to intrinsic and extrinsic concepts of ocality.

REFERENCES

1 A V AHO P J DENNING J D ULLMAN
Principles of optimal page replacement
J ACM 18 1 January 1971 pp 80–93

2 P J DENNING J E SAVAGE J R SPIRN
Some thoughts about locality in program behavior
Proc Brooklyn Polytechnic Institute Symposium April 1972

3 ———
Models for locality in program behavior
Princeton University Department of Electrical Engineering Computer Science Technical Report TR-107 April 1972

4 P H ODEN G S SHEDLER
A model of memory contention in a paging machine
IBM Research Report RC-3053 September 1970

5 G S SHEDLER C TUNG
Locality in page reference strings
IBM Research Report RJ-932 October 1971

6 E G COFFMAN JR T A RYAN JR
A study of storage partitioning using a mathematical model of locality
Comm ACM 15 3 March 1972 pp 185–190

7 J E SHEMER G SHIPPEY
Statistical analysis of paged and segmented computer systems
IEEE Trans Comp EC-15 6 December 1966 pp 855–863

8 J E SHEMER S C CUPTA
On the design of Bayesian storage allocation algorithms for paging and segmentation
IEEE Trans Comp C-18 7 July 1969 pp 644–651

9 P J DENNING
The working set model for program behavior
Comm ACM 11 5 May 1968 pp 323–333

10 P J DENNING S C SCHWARTZ
Properties of the working set model
Comm ACM 15 3 March 1972 pp 191–198

11 P J DENNING
On modeling program behavior
Proc AFIPS Conf Vol 40 Spring Joint Computer Conference 1972

12 J RODRIGUEZ-ROSELL
Experimental data on how program behavior affects the choice of scheduler parameters
Proc 3rd ACM Symposium on Operating Systems Principles October 1971

13 W DOHERTY
Scheduling TSS/360 for responsiveness
Proc AFIPS Conf Vol 37 Fall Joint Computer Conference 1970 pp 97–112

14 W W CHU N OLIVER H OPDERBECK
Measurement data on the working set replacement algorithm and their applications
Proceeding of the Polytechnic Inst of Brooklyn Symposium on Computer-Communications and Teletraffic April 1972