Chapter 5

Locality in User Interactions

In the context of human-computer interaction, *recurrence* is the phenomenon in which prior user interactions (i.e., actions and objects of actions) are referenced repeatedly in the course of a user session. One perspective of recurrence is known as *recency*. Recency is the phenomenon in which recent, individual, user interactions are referenced repeatedly during a user session; Greenberg and Witten (1988b) observed that recurring UNIX command lines were 1 to 3 command lines apart.

Another perspective on recurrence in user interactions was identified in our earlier exploratory study. During certain intervals of a user session, a user references repeatedly a small and related set of user interactions (see Table 2 in Chapter 4). This clustering of user interactions is nominally referred to as *locality* because it is akin to a behaviour by the same name observed in program memory references. Unlike recency, which characterizes recurrences in terms of the distance (i.e., references to individual user interactions that are close to each other in the history), locality characterizes recurrences in terms of periods in time where references are made solely to a small group of user interactions[1].

This chapter describes the application of computer-memory-research techniques and findings to examine locality in user interactions. First, does locality exist in user interactions and to what extent? Second, is locality a randomly occurring behaviour or is it an artifact of user interactions? Finally, does locality explain Greenberg (1988b)'s observations of history usage and the performance of history prediction.

Program Memory Reference Research

As a program executes, it makes references to portions of a computer's memory in units known as *segments*; a sequence of memory references is a *memory reference string*. It is neither feasible to allocate all of a computer's memory to a program nor efficient to allocate too much to a program since it penalizes other

[1] There can be recency without locality. However, if there is locality, then there is also recency.

simultaneously executing programs. Denning (1980) proposed a nonlookahead strategy which assigns memory to those portions of a program's memory space that will be needed immediately, based on sampling past references. This memory policy evolved from two concepts: *working sets* and *locality*.

Working Sets

A *working set*, specifiable as a program executes, is the set of segments that were referenced by an executing program within the last *T* references [Denning, 1980]. It is the basis of the nonlookahead *memory policy* proposed by Denning (1980) in which a program's working set is always kept in resident memory. Figure 1 illustrates a working set consisting of 7 segments { A, B, C, D, E, F, G } for a working-set window size of $T = 10$. The working-set definition suggests a process for dynamically estimating segments currently needed by a program; the process involves sampling past references using a moving window of size *T*.

The working-set concept has led to a class of tools and procedures for measuring and calculating intrinsic memory demands of programs. These tools and procedures have helped researchers to understand and to model program behaviour [Denning, 1980]. One such tool is the *inter-reference distribution* which is used to calculate the rate of segment faults. Given a reference string and a segment *s*, the *inter-reference interval d* is the distance between two successive references to *s*. An *inter-reference distribution* is the probability that two successive references to a segment are *d* units apart (see Figure 2). A segment fault occurs when successive references to a segment are made outside a sampling window of size *T* (i.e., $d > T$). The nature of this distribution may suggest interesting behavioural characteristics. Specifically, when the distribution is skewed towards low inter-reference intervals, it illustrates a strong *recency* effect and hints at a possible *locality* effect. For the memory reference string used in Figure 1, Figure 2 contains its inter-reference distribution which is skewed to smaller distances *d*.

Locality

Locality is the phenomenon in which a program's memory references, made in the course of the program's execution, are limited to a small subset of its virtual address space for an extended time interval. This behavioural phenomenon was observed in studies of program behaviour and is akin to behaviour observed in user interactions. That is, user interactions exhibit a temporal and spatial bias; there are periods in a user's interactions where a user repeatedly references a small subset of the user actions appearing in the user's session.

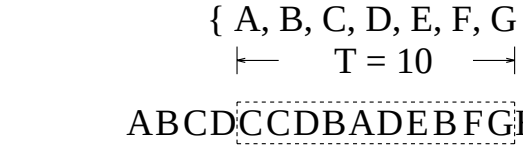


Figure 1: Working set of size 7 with a window size, T, of 10.

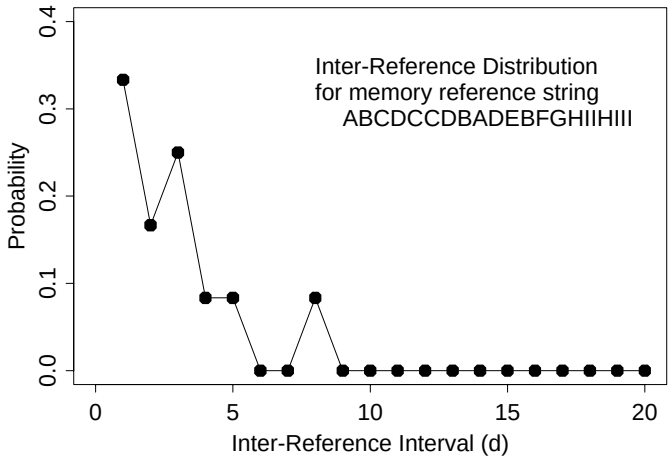


Figure 2: An inter-reference distribution for the memory reference string used in Figure 1. The distribution plots the probability that two successive references to a segment are *d* units apart. In this example, the distribution is skewed to small inter-reference intervals *d* illustrating a *recency* effect.

The locality phenomenon was formalized into a program memory reference model known as *phases* and *transitions* [Denning, 1980]. Phases are intervals in a program's memory references where locality is exhibited. A *locality set* is the set of segments referenced within a *phase*. Transitions are intervals in a program's memory references where a locality set is changing from one set to another.

While the phases and transitions model formally characterizes locality, it does not precisely define what constitutes an interval of distinctive referencing behaviour (i.e., a phase). Denning (1980) enumerates a number of precise definitions of locality each of which are based on a different criteria for identifying a phase. Madison and Batson (1976) provide one such definition and an algorithm for classifying a phase.

This locality-detection algorithm, sketched in Appendix B, makes use of a least recently used (LRU) stack. As segments are referenced, they are pushed onto the top of the LRU stack. A set of segments of size *l* is formed when the *l* segments occupy the top *l* elements of the LRU stack. A set of segments is a *locality set* if and only if all its members are each referenced at least once after the set is formed and no segments are referenced which are not in the set. Its *phase* is the maximal interval within a program's memory references in which all references made after the formation of the locality set are only to segments in the locality set. The phase begins with the reference to the newest member of the locality set and ends with the reference prior to a reference to a non-member of the locality set (i.e., minimal interval length *p* is $p = l$ where *l* is the locality set size). The interval preceding a phase, known as *phase formation* (see Figure 3), is the interval where the first reference to the oldest and newest member of the locality set is made (i.e., interval in which the locality set is being formed).

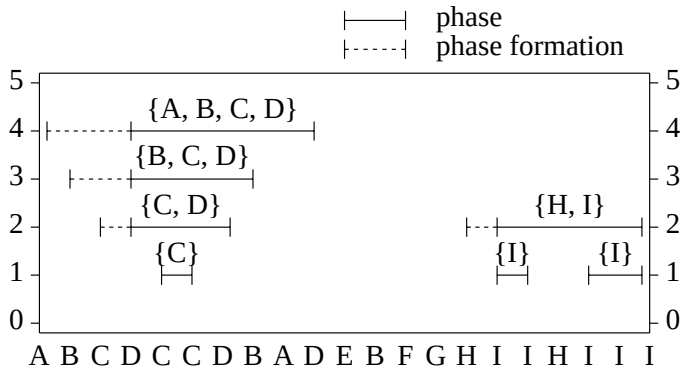


Figure 3: Locality sets of various sizes for the reference string used in Figure 1.

Unlike working sets, phases and locality sets are not determined simply; a locality set is not detected until each member of the set is re-referenced after the set is formed (i.e., at some point within a phase). Figure 3 illustrates an example of the compositions and sizes of all the locality sets appearing in the reference string used in Figure 1.

There are two reasons why a *working set* is not a *locality set*. First, a working set captures the set of segments within the last T memory references. It may include segments appearing in phases as well as transitions. Compare the working set for window size $T = 10$ with the locality sets appearing within the same working-set window in Figure 4. Second, the process for determining working sets does not differentiate phases from transitions and does not identify boundaries for phases and transitions (compare the phase diagram with the working set in Figure 4) [Henderson & Card, 1986a].

Ideally, in the absence of any characterization of the extent of locality in program memory references and the phase lengths, a memory management policy should select the locality set as the resident memory set. During transitions, where references are typically erratic, no nonlookahead memory management policy, including working set, can properly anticipate the relevant set of segments.

Relevance to the Study of Recurrences

Studies of locality of program references lead to two insights about optimal memory management which are applicable to the study of recurrence in user interactions. These two insights led computer memory researchers to conclude that the working-set policy, with an appropriately chosen window size T, was superior to all other memory policies.

First, researchers found that local memory management policies were more effective than global memory management policies[2] [Denning, 1980]. The reason is that a local memory management policy, like a working-set policy, is more

[2] Local memory policies take into account the behaviours of individual programs while global memory policies focus on the aggregate behaviour of all active programs.

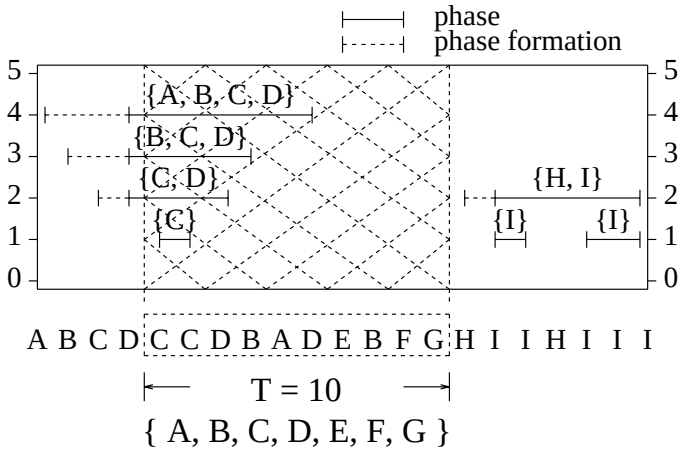


Figure 4: This figure combines Figures 1 and 3. The dashed region delimits portions of the phase diagram and the memory reference string covered by the sampling window of size $T = 10$. For the given sampling window, the working set is { A, B, C, D, E, F, G } and the largest locality set appearing in the sampling window is { A, B, C, D }. While the working set captures four segments belonging to four of the locality sets contained in the sampling window, it also includes three segments { E, F, G } which are not part of any locality set. Furthermore, the last four memory references EBFG are to segments appearing in a transition.

sensitive to changes in memory references of individual programs and can offer a much finer level of adjustment. In light of this result, it would be useful to examine command recurrences and methods for predicting recurrent commands which pay attention to a user's interactions.

Second, empirical data on program locality revealed that programs exhibited good locality. This means that a large percentage of virtual time (over 90%) was covered by long phases (consisting of 10^5 references or more to segments). [Denning, 1980]. Since the time spent in a phase was long and the mean time between two references to the same segment was short, a program's working set provides a good approximation of a program's locality set; it is effectively a program's locality set. That is, when programs exhibit good locality and phases are long, a working set captures primarily segments belonging to a phase and rarely segments belonging to a transition. It is because of these two phenomena – good locality and long phases – and because working sets are easy to determine compared to locality sets, that a program's working set provides a good approximate measure of a program's memory demands as well as a good memory policy for selecting segments to keep resident in memory. In light of this observation, it would be useful to find out the extent to which user interactions exhibit locality and the appropriateness of the working set as a policy for predicting recurrence candidates in user interactions.

Previous HCI Attempts to Apply these Concepts

Analogous to the computer memory management situation, there are HCI situations where it is necessary to identify a small subset of relevant items. As such, there have been two previous HCI efforts to employ these concepts.

Inter-reference distribution, working set, and locality were used to demonstrate that user references to multiple graphical windows exhibit locality and recency [Card, Pavel, & Farrell, 1984; Henderson & Card, 1986a]. While these results are encouraging, they are by no means definitive. One reason is that the study was not an in-depth formal analysis of window reference behaviours. Another reason is that locality in window references does not necessarily suggest locality in other user interaction behaviours (e.g., command line references). Finally, unlike command line references, window references involve a coarser grain of behaviour and a conceptually higher level of user interactions. Further study of locality for a wide range of user interactions is needed.

Inter-reference distribution was used by Greenberg and Witten (1988b) to analyze user interaction behaviours. They measured the frequency of command line recurrences as a function of the inter-reference distance *d* and found that the resulting inter-reference distribution was skewed to smaller distances. This finding, coupled with the fact that the average recurrence rate was 75%, provided evidence of *recency*. On the basis of this evidence, Greenberg and Witten (1988b) adopted the working-set strategy as the basis for predicting reuse candidates and examined its performance when different arrangements of the reuse candidates in a user history are employed.

Importance of Considering Locality

While both studies shed light on the applicability of the two concepts (i.e., working sets and locality) to HCI, there are two unanswered questions pertaining to locality in user interactions.

First, does locality exist in command line references and if so, is it a randomly occurring behaviour or is it an artifact arising from user interactions? There is evidence to suggest that locality is exhibited in graphical window and command line references (see Card, Pavel, and Farrell (1984) and Henderson and Card (1986a), and our exploratory study) but there has been no formal study of locality in user interactions. Recurrences in user interactions arise because of 1) the inherently repetitive nature of certain user tasks, 2) the nature of and difficulties encounted in user interactions (e.g., engaging in multiple activities, improvisations, and error corrections), 3) the inflexibilities in user interfaces which constrain the user's choice of actions, and 4) the bias toward the path of least cognitive resistance. We claim that:

> *Claim 1*: Recurrences, and in particular locality, in command line references is not a randomly occurring phenomenon but a behavioural artifact arising from user interactions.

Second, is locality a more useful characterization of recurrence than recency? We raise this question because of two puzzling results from Greenberg and Witten (1988b)'s study of the recency phenomenon in command line recurrences. The two puzzling results are: 1) the high recurrence rate (i.e., 75%) but low history usage

rate (i.e., 4%) and 2) the suboptimal performance of the working set strategy for predicting recurrent commands. These two findings were based on a study of the recency phenomenon in command line recurrences. In the following sections, we explain how both results can be clarified if locality is considered.

While difficulties in using history tools offer one possible explanation for poor history usage, another explanation is that the recurrence rate and the history usage rate are not estimates of the same behavioural phenomenon and it is incorrect to correlate them. The recurrence rate R is the proportion of activities in a session which occur two or more times [Greenberg, 1988b]:

$$R = \frac{number\ of\ activities - number\ of\ unique\ activities}{number\ of\ activities} \text{ x } 100\%.$$

where *number of activities* is the total number of activities in a session (for the reference string shown in Figure 1, R = (21-9)x100%}/21 = 57%). An activity is either a command line or a command name associated with each user interactions. The UNIX *csh* history tool is primarily a reuse tool and the history usage rate represents those reuse opportunities which are detectable as a result of the use of the history tool. When we associate recurrence rate to history usage rate, we are incorrectly associating a number of different user intentions (e.g., error recovery and others presented in Chapter 2). This could lead to an erroneous association of the recurrence of a command line as one reason for recurrence (i.e., reuse). Note, this is not to suggest that the recurrence rate is uninteresting in and of itself. However, we need a refined measure of Greenberg (1988b)'s recurrence rate R which is a better explanation of the reuse phenomenon and is consistent with the observed history-for-reuse usage rate. $R_{locality}$ is such a recurrence rate and represents the extent of locality in user sessions:

$$R_{locality} = \frac{number\ of\ locality\ activities}{number\ of\ activities} \text{ x } 100\%$$

where *number of locality activities* is the number of activities in a session appearing in phases (for the reference string shown in Figure 1, $R_{locality}$ = 13x100%/21 = 62%). Thus, we claim that:

> *Claim 2*: User activities, that are reused, are attributed to locality. $R_{locality}$ is a better estimate of reuse opportunities than the recurrence rate. In particular, many of the observed UNIX history-tool uses occur in phases and are attributed to history-for-reuse situations.

An important concern in the design of history tools is the development of strategies to predict effectively a small set of user interactions which will be needed in the near future (i.e., history candidates). The working-set policy offers one such history prediction strategy. Greenberg and Witten (1988b) used this strategy to predict reuse candidates and found that a history based on a working set (with T=10 submissions) would, on average, contain the desired command about 60% of the time and miss out about 40% of the time. The question is whether the observed performance of the working-set history prediction strategy is near optimal? Our conjecture is that the performance is suboptimal and this can confirmed by examining the extent of locality in user command references. Recall that computer memory research demonstrated that good locality is an important prerequisite for the optimal performance of the working-set policy. Specifically, we claim that:

Claim 3: Because command line references exhibit poor locality, a working-set history prediction strategy would perform suboptimally.

In order to prove this claim, we compare the relative merits of this strategy when locality is good (i.e., optimal performance of the working-set policy) and when locality is poor (i.e., suboptimal performance of the working-set policy).

The studies, described in the balance of this chapter, examine the phenomenon of locality in command line references in UNIX. Study 1 uses Madison and Batson (1976)'s locality-detection method to determine if locality is indeed present in command line references and if so, to characterize the extent of locality in command line references. We conducted Studies 2 and 3 to corroborate Claims 1-3. Study 2 examines Claim 1 – the question of whether locality is a random behaviour or an artifact of user interactions. Study 3 provides evidence to support Claims 2 and 3 (i.e., locality accounts for history usage and good locality in command line references can enhance the working-set strategy for predicting reuse candidates). Prior to describing these studies, user traces used in the studies are described.

Users' Session Traces

Session traces used in subsequent studies are those collected by Greenberg (1988a). A total of 168 users from four groups participated in his study:

a) *experienced programmers* — 36 senior computer science undergraduates with a fair knowledge of programming languages and the UNIX environment.

b) *computer scientists* — 52 faculty, graduates and researchers from the Computer Science Department at the University of Calgary having varying experience with UNIX, but all being experts with computers in general.

c) *non-programmers* — 25 office staff and members of the Faculty of Environment Design at the University of Calgary who had minimal knowledge of UNIX and only used UNIX applications (e.g., word-processing).

d) *novice programmers* — 55 students from an introductory Pascal course with little or no previous exposure to programming, operating systems or UNIX-like command-based interfaces.

Each user's trace data was strung together as one long session in which session boundary commands like "logout" were retained. Command lines in which an error occurred were excluded from the session. Each command line had a note indicating whether *csh* history was used to help form the command line but nothing indicating which history feature was used.

Study 1 : Does Locality Exist in User Interactions?

The initial question is: "Does locality, by Madison and Batson (1976)'s definition, exist in user interactions?" This question is explored using two different criteria to qualify repetition of user directives. The first criterion is based on the repetition of full command lines (known as the command-lines case) while the second criterion is based on the repetition of command names only (known as the command-names case). Descriptive statistics were collected to characterize the

extent of the locality (i.e., $R_{locality}$) and the extent of the recurrence (i.e., R) in each user session trace.

Results and Discussion

Locality was observed in both the command-lines case and the command-names case. Table 1 summarizes the locality set sizes that were observed in each case and the number of subjects, in each group and the sample as whole, that produced these locality set sizes. In each of the two subsequent sections, we examine the nature of locality in each case.

Locality in Full Command Lines

The recurrence criterion in the command-lines case is fairly restrictive and makes the algorithm sensitive to any variation in the command lines (i.e., a different ordering of the command line objects would terminate a locality set). Despite this restriction, locality set sizes 1 to 12 and 14 were observed with 135 of the 168 users producing a locality set size of 4, 49 of the 168 users producing a locality set size of 6, 20 of the 168 users producing a locality set size of 8, and a small number of users producing locality set sizes larger than 8 (see Cmd Line columns in Table 1).

Figure 5 provides, by way of locality maps from one subject in each of the four groups, a visual characterization of the nature and extent of locality exhibited by the 168 subjects. The locality maps illustrate that while the extent of locality and the locality set sizes vary, locality is exhibited at various periods in a user session. Table 2 summarizes, by way of descriptive statistics, the extent of recurrence (i.e., R), the extent of locality (i.e., $R_{locality}$), and the percentage of R that was accounted for by $R_{locality}$.

Many of the observed locality set sizes were attributed to novice programmers (i.e., 50, 18, and 8 novice programmers produced locality set sizes 4, 6, and 8, respectively). On average, novice programmers exhibited locality in 51% of their sessions. 62% of their recurrences ($R = 81\%$) were accounted for by locality ($R_{locality} = 51\%$). The explanation for both of these results is that novice users do not have a large command vocabulary and do not deviate substantially in the commands they issue. In comparison, users in the other three user groups exhibited far less locality (see Table 2) because they possess more skill and are more likely to vary the construction of their command lines.

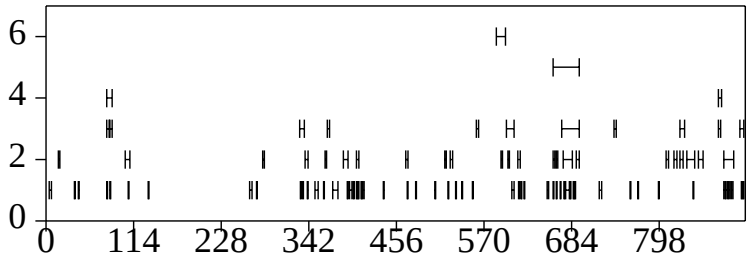
For each user and each observed locality set size, the average number of full command lines in a phase and the average number of occurrences of the same locality set were determined. Then, for each user group and the sample as a whole, the mean and standard error for each of the two measures were computed. Figure 6 plots the mean for the first measure (i.e., average number of command lines in a phase) and Table 3 lists the mean and standard error for the first six locality set sizes. Figure 7 plots the mean for the second measure (i.e., average number of occurrences of the same locality set) and Table 4 lists the mean and standard error for the first six locality set sizes. The graphs and tables illustrate differences across the four groups for each of the two measures.

Set Sizes	Computer Scientists		Experienced Programmers		Non Programmers		Novice Programmers		All Subjects	
	Cmd Line	Cmd Name	Cmd Line	Cmd Name	Cmd Line	Cmd Name	Cmd Line	Cmd Name	Cmd Line	Cmd Name
1	52	52	36	36	25	25	55	55	168	168
2	51	52	36	36	25	25	55	55	167	168
3	47	52	33	36	21	25	54	55	155	168
4	38	50	31	34	16	23	50	55	135	162
5	21	44	24	31	8	20	33	49	86	144
6	9	35	16	25	6	15	18	41	49	116
7	5	21	9	20	2	15	6	33	22	89
8	4	13	7	13	1	7	8	29	20	62
9	1	7	3	10		6		18	4	41
10	1	6	2	7		3		14	3	30
11	1	5	1	3		1		13	2	22
12		1	1			3		6	1	10
13		1		2				4		7
14			2	3				3	2	6
16								1		1
17								1		1
20								1		1
29								1		1

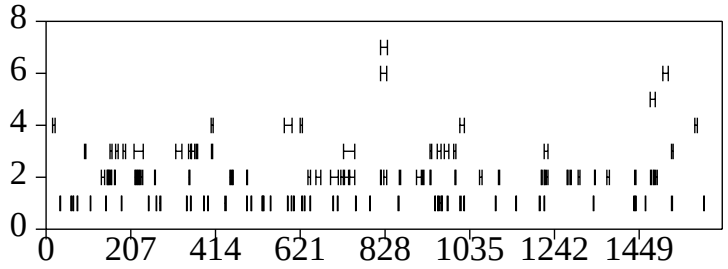
Table 1: The number of users observed producing the locality set sizes for the command-lines and command-names criteria.

User Group	No. of Users	R		$R_{locality}$		% of R Accounted for by $R_{locality}$ $R_{locality}$ x 100% / R	
		Mean	Std. Err.	Mean	Std. Err.	Mean	Std. Err.
Computer scientists	52	69.4	1.1	17.0	1.2	24.4	1.6
Experienced programmers	36	77.7	2.0	26.7	2.5	32.9	2.3
Non programmers	25	68.3	1.7	25.0	2.8	36.1	3.6
Novice programmers	55	80.5	1.0	50.7	2.0	62.3	2.0
All subjects	168	74.6	.8	31.3	1.5	40.4	1.6

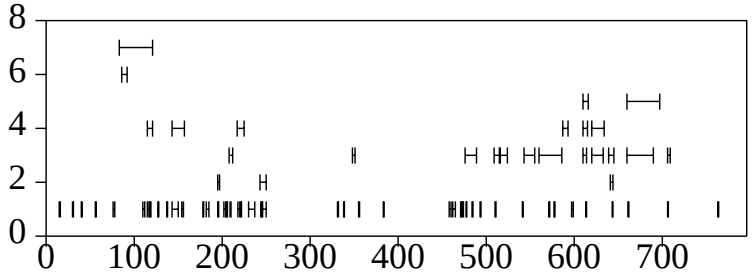
Table 2: For all users and the command-lines criterion, the mean and standard error of R, $R_{locality}$, and percentage of R that was accounted for by $R_{locality}$ are tabulated.



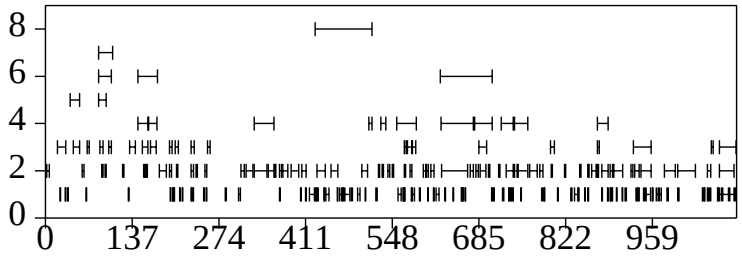
computer-scientist-3 with session length of 909 lines



experienced-programmer-34 with session length of 1651 lines



non-programmer-16 with session length of 795 lines



novice-programmer-15 with session length of 1091 lines

Figure 5: Phase diagrams for four of the users, one from each user group for the command-lines criterion. The bars delimit phases, the y-axis is the locality set sizes, and the x-axis is the command line number in a user session.

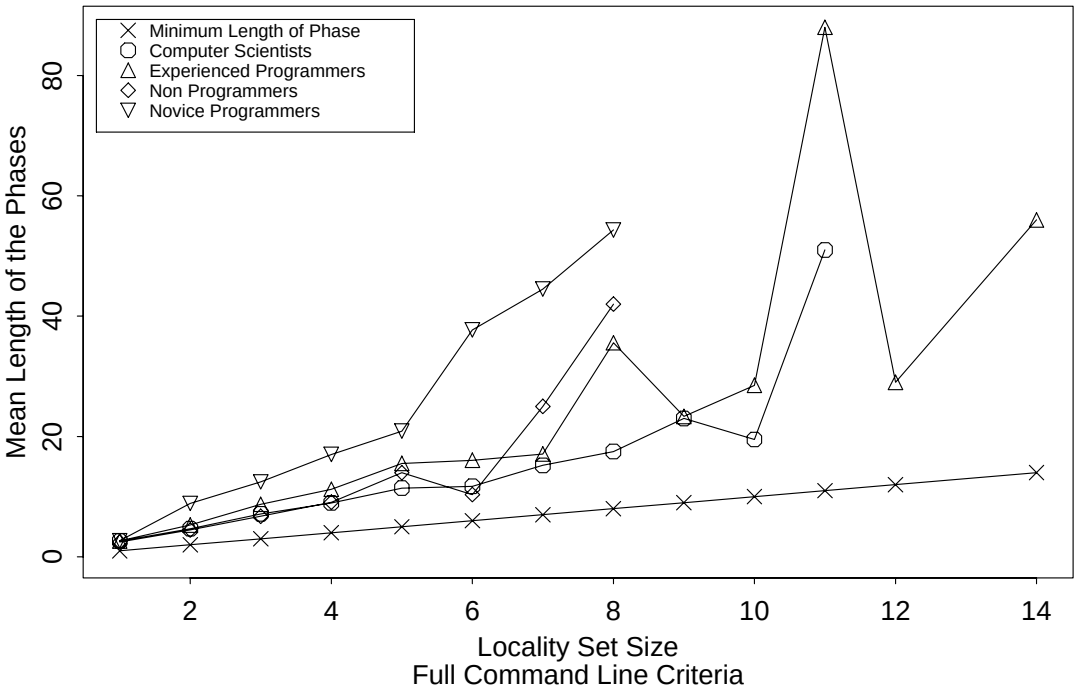


Figure 6: A plot of the mean number of full command lines in a phase for each user group based on user means as a function of locality set size.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	2.62	4.68	7.14	8.96	11.42	11.72
	Std. Err.	0.11	0.27	0.35	0.53	1.34	1.44
Experienced programmers	Mean	2.64	5.28	8.72	11.22	15.52	16.05
	Std. Err.	0.08	0.29	0.61	1.77	1.78	1.76
Non programmers	Mean	2.48	4.47	6.74	9.07	13.97	10.33
	Std. Err.	0.12	0.21	0.40	0.86	1.85	1.09
Novice programmers	Mean	2.68	8.87	12.46	17.01	20.90	37.70
	Std. Err.	0.05	0.52	0.90	1.05	2.31	6.08
All subjects	Mean	2.62	6.16	9.27	12.48	16.44	22.51
	Std. Err.	0.05	0.25	0.41	0.66	1.15	2.84

Table 3: For the command-lines criterion, the mean and standard error ($\sigma/\sqrt{n}$) of the number of command lines in a phase for locality set sizes 1 to 6. For instance, the mean number of command lines in a phase for a locality set of size 1 for subjects in the Computer Scientist group is 2.62 command lines.

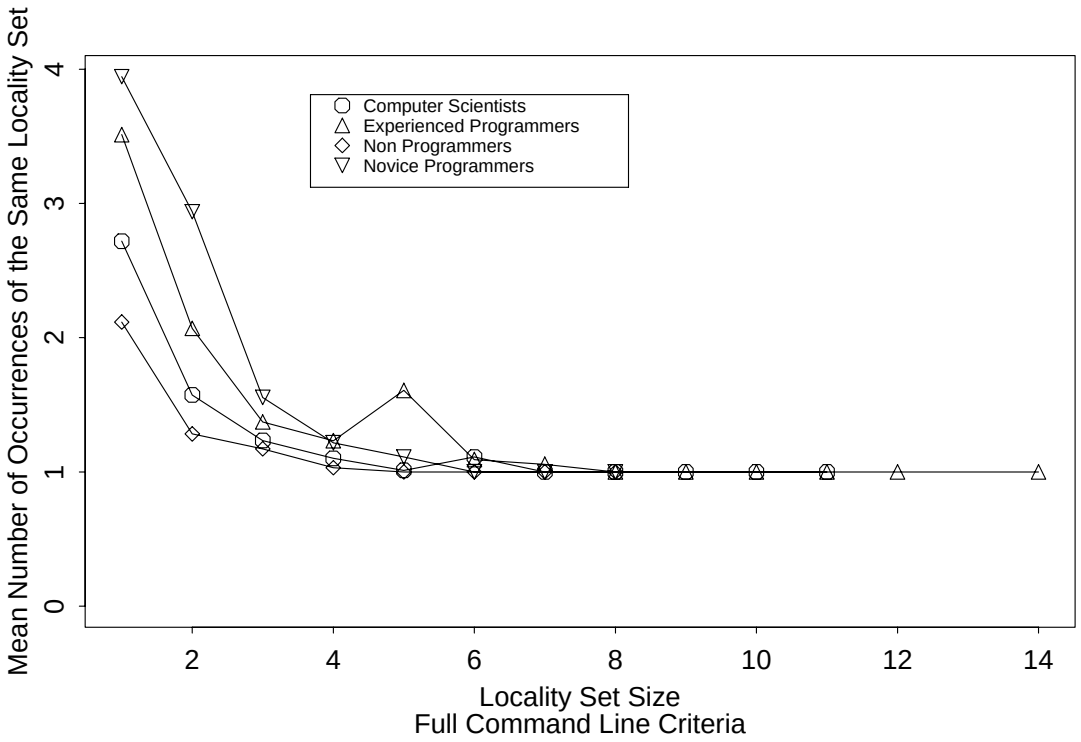


Figure 7: A plot of the mean number of occurrences of the same locality set for each user group based on user means as a function of locality set size for the command-lines criterion.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	2.72	1.57	1.23	1.10	1.01	1.11
	Std. Err.	0.16	0.07	0.06	0.04	0.01	0.11
Experienced programmers	Mean	3.51	2.07	1.37	1.23	1.61	1.09
	Std. Err.	0.38	0.20	0.09	0.10	0.27	0.06
Non programmers	Mean	2.12	1.28	1.17	1.03	1.00	1.00
	Std. Err.	0.19	0.06	0.06	0.03		
Novice programmers	Mean	3.94	2.94	1.56	1.22	1.11	1.00
	Std. Err.	0.32	0.17	0.10	0.06	0.04	
All subjects	Mean	3.20	2.09	1.37	1.17	1.22	1.05
	Std. Err.	0.15	0.09	0.05	0.03	0.08	

Table 4: For the command-lines criterion, the mean and standard error of the number of occurrences of the same locality set for sizes 1 to 6. For instance, the mean number of occurrences of the same locality set of size 1 for subjects in the Computer Scientist group is 2.72.

Several users in the experienced programmer group with extensive programming and UNIX experience generated locality set sizes of 9 and larger. This is not surprising because experienced programmers have larger command sets, perform sophisticated tasks, and have very well-learned behaviours. In contrast, all but one of the subjects in the other three user groups have locality set sizes of 8 and less because these subjects have smaller command sets and less experience with UNIX and *csh*.

An examination of the mean number of occurrences of the same locality sets reveals that users tended to repeat the same locality sets for smaller set sizes (especially locality set sizes 1 and 2) more than for larger set sizes (i.e., size 4 and up). A comparison of user groups with respect to the mean number of occurrences of the same locality set indicates that novice and experienced programmers repeat the same locality set more often (see Table 4 and Figure 7).

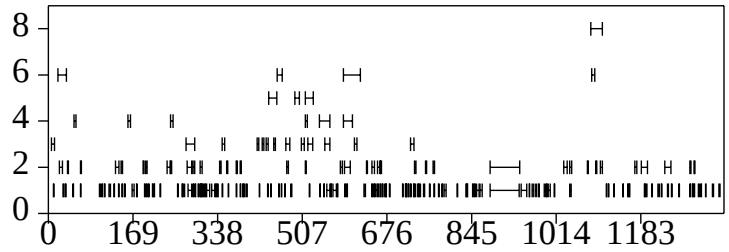
Locality in Command Names Only

Locality set sizes 1 .. 14, 16, 17, 20, and 29 were observed in the command-names case (see Table 1). Figures 8, 9, and 10 and Tables 1, 5, 6, and 7 summarize the extent of locality observed in the command-names case. Larger locality sets were observed because activities repeat if command names repeat.

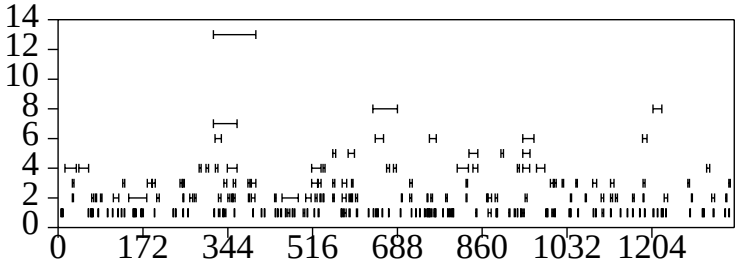
Many of the larger locality set sizes were produced by novice programmers. On average, novice programmers have longer phases and repeated the same locality set more often than the other three user groups. Unlike the command-lines case, non-programmers, on average, have the next longest phases and they repeated the same locality set about as often as novice programmers. In general, computer scientists and experienced programmers were comparable in terms of the size of their locality sets, the average number of command names in their phases, and the average number of occurrences of the same locality set.

User Group	No. of Users	R		$R_{locality}$		% of R Accounted for by $R_{locality}$ $R_{locality}$ x 100% / R	
		Mean	Std. Err.	Mean	Std. Err.	Mean	Std. Err.
Computer scientists	52	96.0	.4	52.3	1.8	55.0	1.9
Experienced programmers	36	95.2	.5	53.9	2.0	56.6	2.1
Non programmers	25	95.2	.7	75.4	3.3	79.1	3.3
Novice programmers	55	97.4	.2	82.9	1.6	85.1	1.6
All subjects	168	96.0	.2	66.1	1.5	68.8	1.5

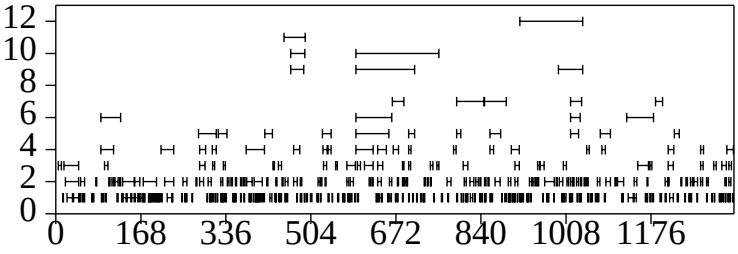
Table 5: For command-names criterion, the mean and standard error of R, $R_{locality}$, and percentage of R that was accounted for by $R_{locality}$.



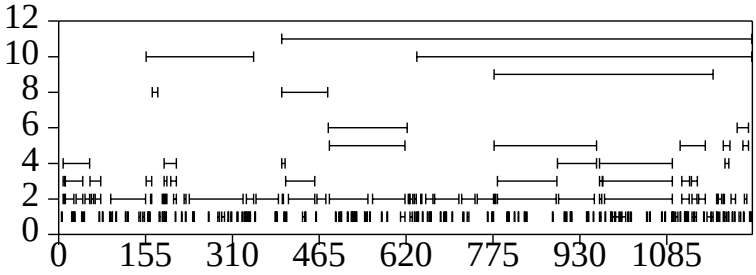
computer-scientist-15 with session length of 1348 lines



experienced-programmer-11 with session length of 1370 lines



non-programmer-18 with session length of 1339 lines



novice-programmer-44 with session length of 1237 lines

Figure 8: Phase diagrams for four of the users, one from each user group for the command-names criterion. The bars delimit phases, the y-axis is the locality set sizes, and the x-axis is the command line number for a user session.

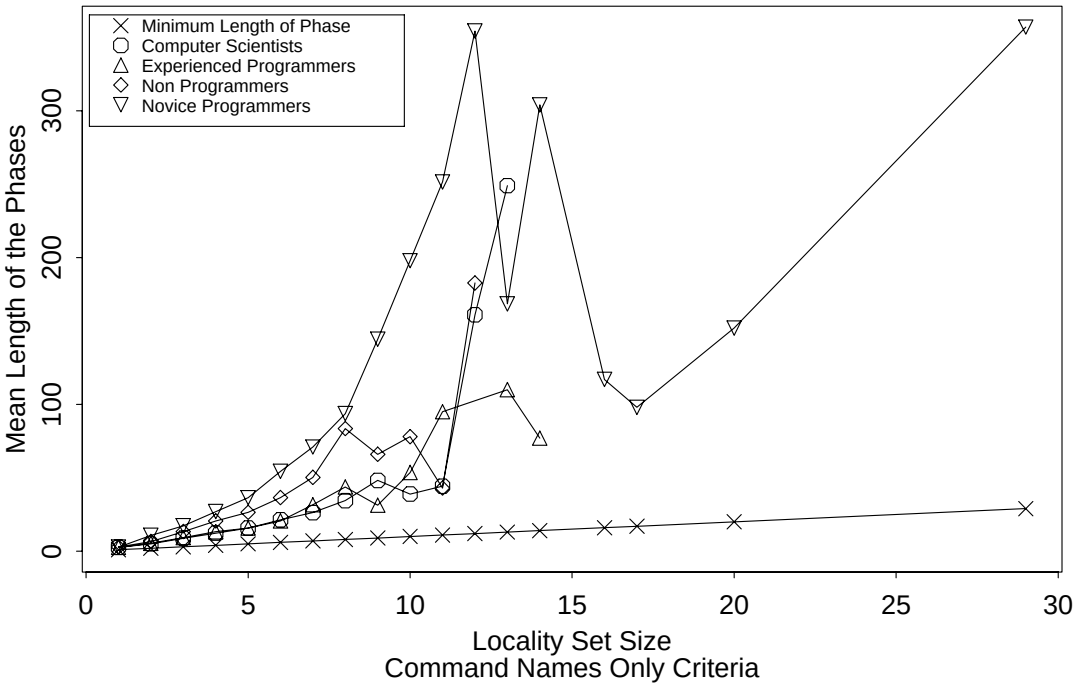


Figure 9: A plot of the mean number of command names in a phase for each user group based on user means as a function of locality set size.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	2.59	5.01	8.82	12.29	15.77	21.39
	Std. Err.	0.05	0.14	0.44	0.75	0.89	1.32
Experienced programmers	Mean	2.64	5.40	9.24	13.24	15.72	20.63
	Std. Err.	0.06	0.19	0.54	1.04	0.84	1.66
Non programmers	Mean	2.84	6.54	13.37	20.59	26.47	36.50
	Std. Err.	0.12	0.39	1.55	3.78	3.74	5.67
Novice programmers	Mean	2.78	10.77	17.31	26.98	36.45	54.37
	Std. Err.	0.04	0.74	1.13	3.40	3.70	6.44
All subjects	Mean	2.70	7.21	12.37	18.66	24.28	34.84
	Std. Err.	0.03	0.32	0.55	1.40	1.60	2.81

Table 6: For command-names criterion, the mean and standard error of the length of phases for locality set sizes 1 to 6. For instance, the mean number of command names in a phase for a locality set of size 1 for subjects in the Computer Scientist group is 2.59 command names.

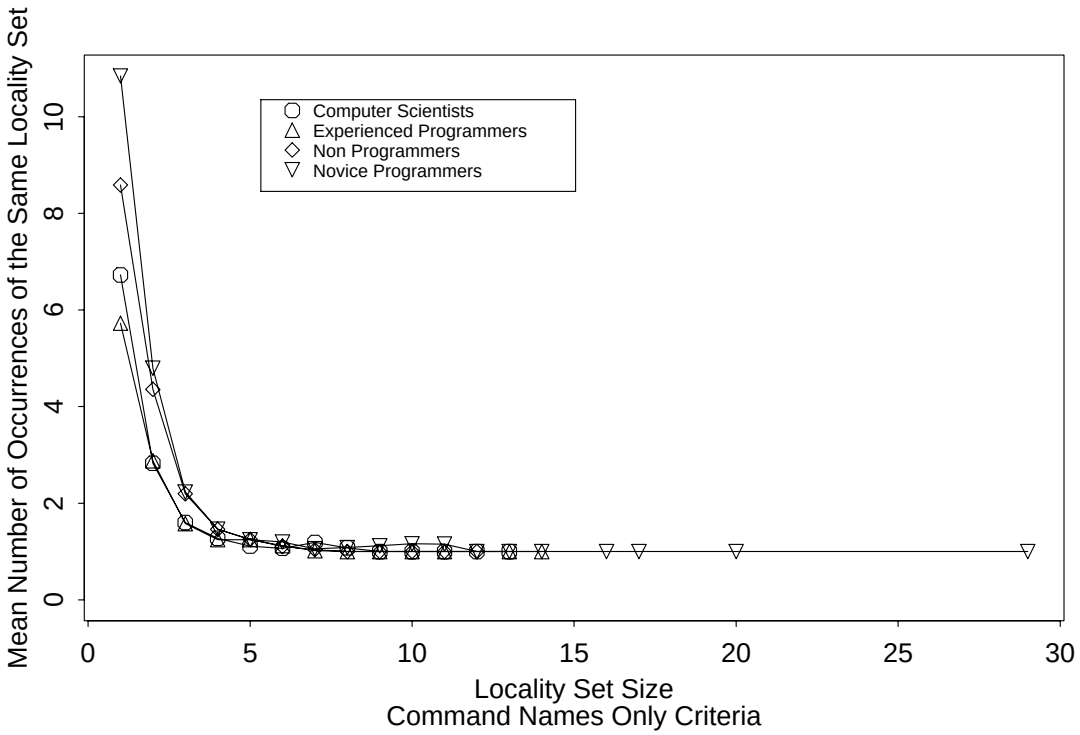


Figure 10: A plot of the mean number of occurrences of the same locality set of for each user group based on user means as a function of locality set size for the command-names criterion.

Group	Statistic	Locality Set Sizes					
		1	2	3	4	5	6
Computer scientists	Mean	6.73	2.83	1.60	1.27	1.11	1.07
	Std. Err.	0.57	0.17	0.07	0.07	0.05	0.03
Experienced programmers	Mean	5.73	2.88	1.58	1.25	1.24	1.12
	Std. Err.	0.44	0.20	0.10	0.05	0.13	0.04
Non programmers	Mean	8.59	4.36	2.19	1.46	1.25	1.11
	Std. Err.	1.31	0.53	0.20	0.13	0.08	0.06
Novice programmers	Mean	10.84	4.79	2.23	1.47	1.24	1.20
	Std. Err.	0.77	0.31	0.19	0.11	0.06	0.06
All subjects	Mean	8.14	3.71	1.89	1.36	1.21	1.13
	Std. Err.	0.41	0.16	0.08	0.05	0.04	0.03

Table 7: For the command-names criterion, the mean and standard error of the number of occurrences of the same locality set for sizes 1 to 6. For instance, the mean number of occurrences of the same locality set of size 1 for subjects in the Computer Scientist group is 6.73.

Study 2 : Does Locality Exist By Chance?

Study 2 answers the question: "Is locality a randomly occurring behaviour or is it a behaviour that arises from user-computer interactions?" Specifically, "Do *pseudo users* generate locality sets in the same proportions as *real users*?" Pseudo users generate randomly sequenced, but syntactically correct, command lines. Note, we answer this question by examining the locality behaviour for the full-command-lines case although examining locality behaviour for the command-names-only case would have been appropriate as well.

Statistical Test

Real-user sessions represent one independent sample drawn from a population. Pseudo-user sessions represent another independent sample drawn from a population. The question is: "Are the locality behaviours exhibited in real and pseudo-user samples those behaviours exhibited by individuals that come from the *same* population?" If so, then both sample means would estimate the same population mean for the locality measure. Otherwise, the sample means estimate two distinct population means. The locality measure $L_l(u)$ for a user u is the proportion of the total observed locality that is of a given locality set size:

$$L_l(u) = \frac{n_l(u)}{N(u)} \text{ x } 100\% \qquad l = 1 \text{ .. } 12, 14 \text{ and } u = p,r.$$

where $n_l(u)$ represents the number of locality sets that were observed for locality set size l for user u's session, and $N(u) = n_1(u) + \cdots + n_{12}(u) + n_{14}(u)$. r denotes a *real-user* sample and p denotes a *pseudo-user* sample. There are 13 such locality measures which correspond to the 13 locality sets observed for real users.

A sampling experiment can be conducted which draws a sample of 168 pseudo-user session traces from the pseudo-user population. For each locality set size l, the 13 sample means for the locality measure $\bar{L}_l(u=p)$ can be measured. This metric estimates the population mean μ for the locality measure $L_l(u)$. When this sampling experiment is repeated a large number of times (i.e., 1000), 13 relative frequency histograms based on the sample means $\bar{L}_l(u=p)$ can be generated, one for each locality set size l. Each relative frequency histogram approximates the probability distribution of the sample mean $\bar{L}_l(u=p)$. Specifically, if another independent sample is drawn from the pseudo-user population, the mean of the locality measures $\bar{L}_l(u=p)$ for this sample would be well described by its relative frequency histogram.

To test the claim that the mean locality measure $\bar{L}_l(u=r)$ for real-user samples comes from the same sampling distribution as $\bar{L}_l(u=p)$, we determined the probability that the mean locality measure of $\bar{L}_l(u=p)$ is equal to or more extreme than $\bar{L}_l(u=r)$. The probability condition is a statistical test of one tail of the sampling distribution. The particular probability condition depends on which tail of the sampling distribution $\bar{L}_l(u=r)$ is located in. A small probability (i.e., $P[\bar{L}_l(u=p) \geq \bar{L}_l(u=r)] < \alpha$ or $P[\bar{L}_l(u=p) \leq \bar{L}_l(u=r)] < \alpha$ where $\alpha = .01$) provides strong evidence that such an event is unlikely to occur if the claim was true. 13 such one-tailed tests were performed, one for each observed locality set l, using the corresponding sampling distribution.

Method

Each sampling distribution for each observed locality set size was obtained from a computer simulation of the sampling experiments. Each sample had 168 sessions corresponding to the 168 sessions in the real-user sample. In the sampling experiment, the sequence of command lines for each real-user session was randomly permuted using a different random number, to form the corresponding pseudo-user session. 1000 pseudo-user samples were generated.

Each session (*real* and *pseudo*) was analyzed by the locality-detection algorithm and a summary of detected locality sets was collected. Each summary included 13 $n_l(u)$ values ($l = 1 .. 12, 14$) and total number of locality sets observed (i.e., $N(u) = n_1(u) + \cdots + n_{12}(u) + n_{14}(u)$). Then, the locality measure, $L_l(u)$, for each user sample u and each set size l was computed.

Sample means for each of the 13 locality measures, $\overline{L}_l(u)$, for a *real-user* sample and 1000 *pseudo-user* samples were determined. Then, for each locality set size l, a frequency histogram for the 1000 *pseudo-user* sample means $\overline{L}_l(u=p)$ was produced. A relative frequency histogram was generated by taking the frequencies and dividing by the total number of samples (i.e., 1000).

Results and Discussion

Figure 11 displays 4 of the 13 sampling distributions of $\overline{L}_l(u=p)$ that were obtained from the sampling experiments. Table 8 lists, for each of the 13 locality set sizes, $\overline{L}_l(u=r)$ values for *real-user* sessions, probability tests, and probability test values P. P values for all locality set sizes were less than .01. The claim that the two groups come from the same population was rejected at the $\alpha = .01$ level. Therefore, locality is a behavioural artifact of user interactions and does not occur by pure chance. Figure 12 illustrates the difference between $\overline{L}_l(u=p)$ and $\overline{L}_l(u=r)$.

Study 3 : Does Locality Enhance the Recurrence Picture?

This is a two-part study which examines two claims concerning two aspects of Greenberg and Witten (1988b)'s study. In both parts of the study, we examine the locality behaviour for the full-command-lines case although it would have been appropriate to examine the locality behaviour for the command-names-only case. First, does locality account for Greenberg and Witten (1988b)'s observed history usage? Second, does limiting the history prediction strategy to locality situations enhance the quality of the prediction?

Study 3A: History-for-Reuse Occurs Within a Locality

Claim 2 states that the fraction of the recurrent activities, those appearing in a locality (i.e., $R_{locality}$), represent a more meaningful estimate of reuse opportunities than the recurrence rate estimate (i.e., R). Recall, R is the percentage of the session activities containing recurrent activities while $R_{locality}$ is the percentage of the session activities containing phase activities.

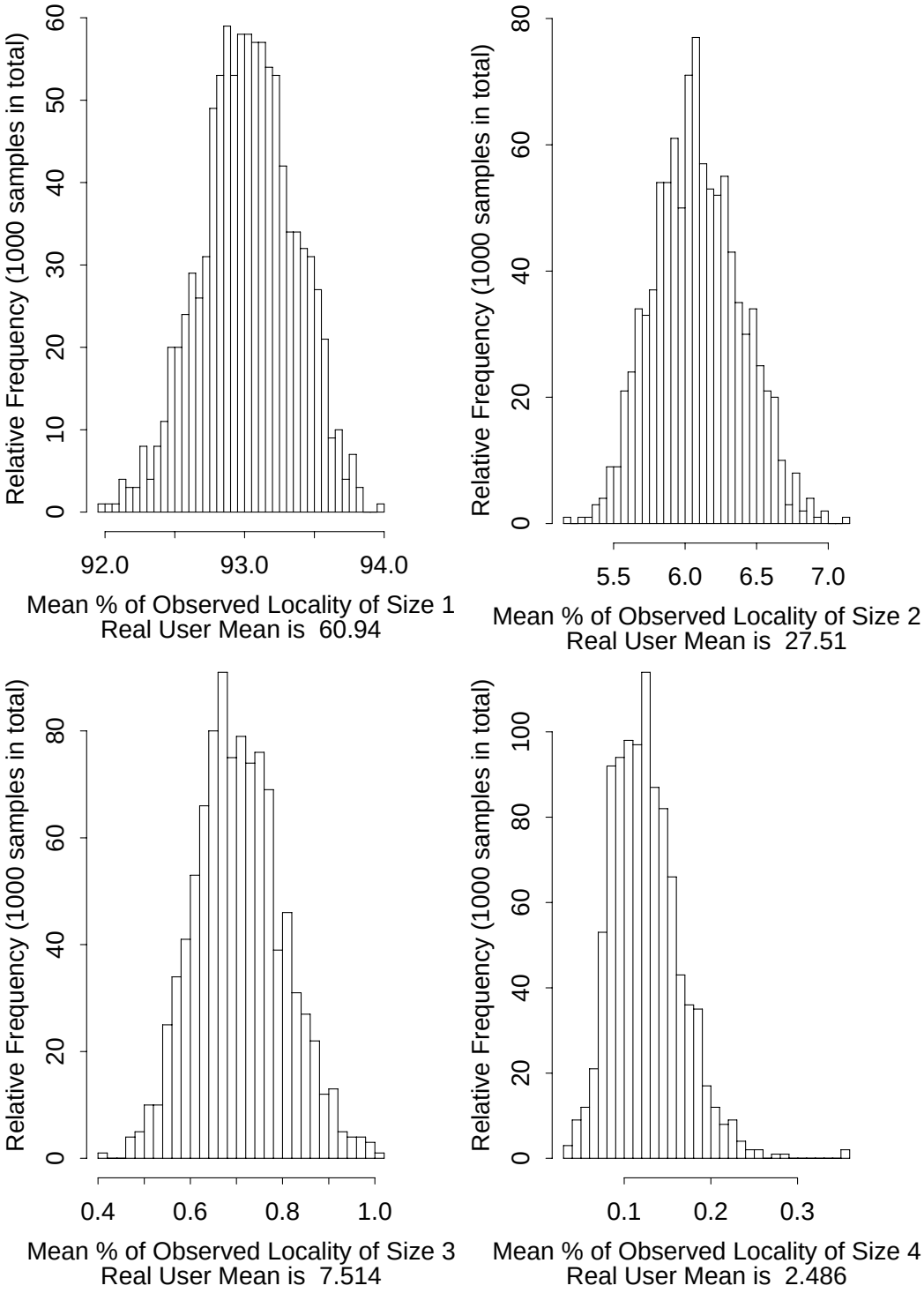


Figure 11: Relative frequency distributions of $\overline{L}_l(u=p)$ for l = 1 - 4.

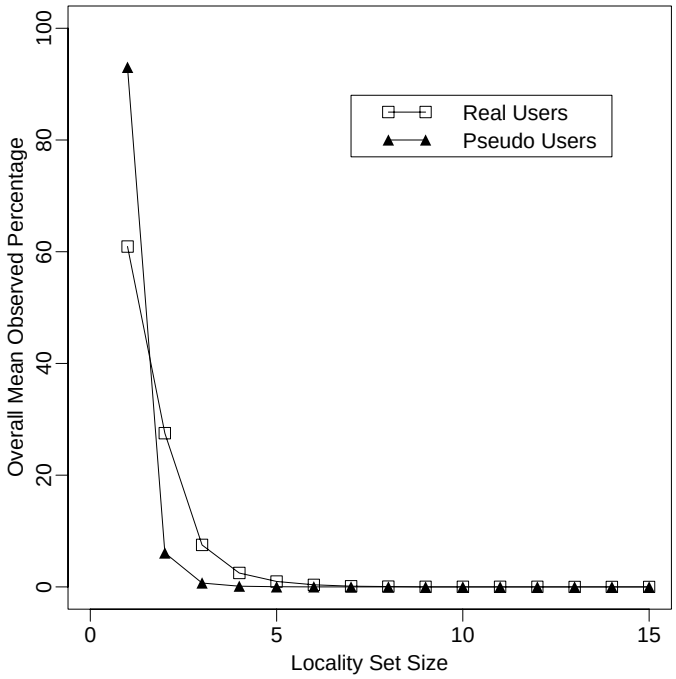


Figure 12: $\overline{L}_l(u=p)$, $\overline{L}_l(u=r)$ versus locality set size.

Size l	$\overline{L}_l(u=r)$	Probability Test	Value
1	60.937	$P[\overline{L}_1(u=p) \leq 60.94]$	0.
2	27.515	$P[\overline{L}_2(u=p) \geq 27.52]$	0.
3	7.514	$P[\overline{L}_3(u=p) \geq 7.51]$	0.
4	2.486	$P[\overline{L}_4(u=p) \geq 2.49]$	0.
5	0.965	$P[\overline{L}_5(u=p) \geq 0.97]$	0.
6	0.368	$P[\overline{L}_6(u=p) \geq 0.37]$	0.
7	0.117	$P[\overline{L}_7(u=p) \geq 0.12]$	0.
8	0.065	$P[\overline{L}_8(u=p) \geq 0.07]$	0.
9	0.009	$P[\overline{L}_9(u=p) \geq 0.01]$	0.004
10	0.008	$P[\overline{L}_{10}(u=p) \geq 0.01]$	0.002
11	0.004	$P[\overline{L}_{11}(u=p) \geq 0]$	0.004
12	0.008	$P[\overline{L}_{12}(u=p) \geq 0.01]$	0.001
14	0.002	$P[\overline{L}_{14}(u=p) \geq 0]$	0.

Table 8: The $\overline{L}_{r,l}$ values and the probability tests for the respective locality set sizes and their corresponding values. The test is the probability that the mean locality measure of $\overline{L}_l(u=p)$ is equal to or more extreme than $\overline{L}_l(u=r)$. It involves the tail of the sampling distribution on which the value of $\overline{L}_l(u=r)$ is located (see Figure 11 for the location of the value of $\overline{L}_l(u=r)$ in the sampling distribution for $l=1,2,3,4$).

Method

In order to verify this claim, we conduct two selection experiments. In the *locality-selection* experiment, we identify those users who made use of the history tool. Then, for these user's session, we select only those activities appearing in phases and count the number of activities that are formed using a history tool (i.e., $h_{locality}$) . This selection experiment would select a percentage of the activities in a user's session that is equal to $R_{locality}$. In the *random-selection* experiment, we randomly select the same fraction of the activities in the same user's sessions and count the number of activities that are formed using a history tool (i.e., h_{random}). Then, we determine $H_{locality}$ and H_{random}. $H_{locality}$ is the percentage of the activities appearing in phases that are formed with a history tool:

$$H_{locality} = \frac{h_{locality}}{h_{total}} \text{ x } 100\%$$

H_{random} is the percentage of the randomly selected activities that are formed using a history tool:

$$H_{random} = \frac{h_{random}}{h_{total}} \text{ x } 100\%$$

Note, the value of H_{random} is equal to $R_{locality}$ because a random selection of a percentage of the activities in a user's session (i.e., $R_{locality}$) would also randomly select a similar percentage of the activities formed using a history tool. Thus, the value of h_{random} is:

$$h_{random} = \frac{R_{locality} \; x \; h_{total}}{100}$$

If Claim 2 is false, then activities formed using history tools would be equally likely in phases as well as transitions. That is, the percentage of activities in a user session formed with a UNIX history tool within phases $H_{locality}$ would be equal to H_{random} and $R_{locality}$.

However, if Claim 2 is true then the value of $H_{locality}$ would be substantially greater than the value of H_{random} (i.e., $H_{locality}$ is greater than H_{random} which is equal to $R_{locality}$). Thus, $h_{locality}$ should be substantially larger than h_{random}. Note that only 25% of the session activities (i.e., $R_{locality} = 25\%$ which is based on user sessions that involved history use) occur in a phase and less than 4% of the session activities involve history use; both are small percentages (see Table 9). Thus, a valid claim means that the selection systematically locates a large percentage of history use in a quarter of the command lines.

We conducted the locality-selection experiment on the sessions for those users who used the UNIX history tool. For each of these users, we determined the values of $h_{locality}$, h_{total}, h_{random} and $H_{locality}$.

Results and Discussion

A within-subjects ANOVA test comparing h_{random} and $h_{locality}$ yields a significant difference ($F(1,88) = 65.1$, $p = 0$). This confirms our claim that the extent of locality $R_{locality}$ is a better estimator of reuse than the recurrence rate R. On average, 65% of the command lines involving history use are issued within a

User Group	No. of Users	$R_{locality}$ Mean	$R_{locality}$ Std. Err.	% of the History Usage Occurring in a Locality (H) Mean	% of the History Usage Occurring in a Locality (H) Std. Err.	% of the Total Session Involving History Usage Mean	% of the Total Session Involving History Usage Std. Err.
Computer scientists	37	16.3	1.5	60.7	3.8	4.1	.7
Experienced programmers	32	26.5	2.6	64.4	3.7	4.5	.6
Non programmers	9	25.7	5.9	60.4	12.1	4.4	2.1
Novice programmers	11	46.7	5.3	83.1	6.1	2.1	.9
All subjects	89	24.7	1.7	64.8	2.6	4.1	.4

Table 9: The mean and standard error of $R_{locality}$, H and % of total sessions involving history usage. Unlike the $R_{locality}$ values in Table 2, these values are computed from only session traces of history users.

locality (see Table 9). This figure is two and a half times the number of command lines involving history use that are selected by a systematic process (i.e., locality) compared to that which a random process would have selected.

However, on average, 35% of history uses[3] are not found inside a phase. One possible explanation for the unaccounted 35% is that history uses may not be limited to the literal recall of a history item; they may include the modification and recall of parts of previous command lines. If such history features (i.e., modification and partial recall) are used to form a command line, the modified command line would break the locality set. Thus, the locality detection method must be refined to alleviate the restricted definition of recurrences (i.e., literal repetition of previous command lines) so that the repetition of modified command lines is included in a locality set. It is difficult to verify directly our explanation for the missing 35% history usage because our user session traces are not annotated with the history feature that a user used. However, this explanation is indirectly supported by two findings from our studies:

(1) In our exploratory study, we found that *csh* users used word designators and modifiers (see Table 3 in Chapter 4). These features permit the recall and modification of parts of previous command lines.

(2) In this study, we note that 83% of the novice programmers' history uses were accounted for by locality. This group of subjects were students taking an introductory programming course with no previous exposure to UNIX and *csh* history. Thus, their use of history would be limited largely to the simple recall of previous command lines.

[3] Recall, the locality-detection method requires that the composition and makeup of individual items of a locality set remain unchanged throughout the phase. Thus, the 65% of history uses found in phases are the history uses involving the literal reuse of commands.

Study 3B: Locality and Prediction of Reuse Candidates

To corroborate the claim that the performance of the working-set history prediction strategy (observed by Greenberg and Witten (1988b)) is suboptimal, one needs to demonstrate two things. First, user sessions must exhibit poor locality. Second, the performance of the working-set strategy during locality periods is shown to be better than during the entire period of a user's session.

Method

Two versions of each user's session are required: a *full user session S* and a *restricted user session S′*. *S′* contains only command lines in *S* that are part of phases or phase formations. Each *S* and *S′* trace is run through the working-set algorithm, outlined in Figure 13, using various window sizes, $T = 1 .. 10, 20, 30, 40, 50$. For each user and each T, the percentage of time that the next command line is located within the current working set is computed. Also, for each T, the average of this percentage for all users is computed.

Results and Discussion

As revealed in the full-command-lines case, on average, the extent of locality $R_{locality}$ over a full user session *S* is 31% (see Table 10). This measure is substantially less than $R = 74\%$. However, $R_{locality}$ for the locality-only portions of a user session *S′* is 90% and *R* is 88%. By removing command lines appearing in transitions, the recurrence rate *R* and the extent of the locality $R_{locality}$ are much higher. The resulting traces exhibit good locality and high recurrence rate, and more importantly, the two metrics are essentially equivalent.

Figure 14 shows the performance of the prediction strategy for various working-set window sizes when prediction is restricted to locality periods only and when it is performed throughout a session. Hit percentages for the two predictions are tabulated in Table 11. These results corroborate the computer memory research finding which found that when locality is good (see *S′*), so is the performance of the working-set history prediction strategy. Thus, the performance of the working-set history prediction strategy (*S*) is suboptimal[4].

With the restricted user sessions, reuse prediction is, on average, 92% for *S′* compared to 67% for *S* with a window size of $T = 10$. In general, hit percentages for a restricted user session (*S′*) are higher than for a full session *S* at the same recurrence distance T (i.e., $1 \leq T \leq 50$). For T=10, the differential in prediction performance is 26%. Doubling T results in a smaller differential in prediction performance (18%) compared to that obtained for T =10. The incremental gain for T >3 of restricted user sessions diminishes in comparison to full sessions. From the data in Table 11, T=3 appears to be the threshold point.

[4] Another finding of computer memory research is that the working-set policy is the most likely, among nonlookahead policies to generate minimum space-time for any given program. In fact, Denning (1980) reports that experiments with real programs have found that the working-set policy can be run with a single global control parameter value (i.e., T) and deliver performance typically no worse than 10 percent from optimum.

Given:

- an array *cmd_line* which holds n command lines in a session
- an array num_hits to accumulate the number of hits at various distances
- an array T to hold the T values of interest
- an array hit_percent to store the cumulative hit % for a particular T

```
int T[14] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50};

/* scan each item to find its nearest preceding match and note statistics */
for (i = 1 to n)
        for (j = i to 1 by -1)
                if (cmd_line[i] == cmd_line[j]) then {
                        distance = i-j
                        no_hits[distance] = distance +1
                        break;
                }

/* compute the hit_percent for the distances indicated in T */
total = 0
t = 0
for (distance = 1 to n) {
        total = total + no_hits[distance]
        if (distance == T[t]) {
                hit_percent[t] = (total/n) * 100
                t = t + 1
        }
}
```

Figure 13: A working set algorithm, from Greenberg (1988b), for determining hit percentage for window sizes T = 1 .. 10, 20, 30, 40, 50.

User Group	Statistic	R	$R_{locality}$	R'	$R'_{locality}$
Computer scientists	Mean	69.4	17.0	85.7	89.5
	Std. Err.	1.1	1.2	1.0	.6
Experienced programmers	Mean	77.7	26.7	89.2	88.4
	Std. Err.	2.0	2.5	1.3	.6
Non programmers	Mean	68.3	25.0	80.9	87.5
	Std. Err.	1.7	2.8	1.8	.9
Novice programmers	Mean	80.5	50.7	93.1	91.5
	Std. Err.	1.0	2.0	.5	.4
All subjects	Mean	74.6	31.3	88.1	89.6
	Std. Err.	.8	1.5	.6	.3

Table 10: The recurrence and locality metric values for the full user sessions S and the restricted user sessions S' for each user group and the sample as a whole.

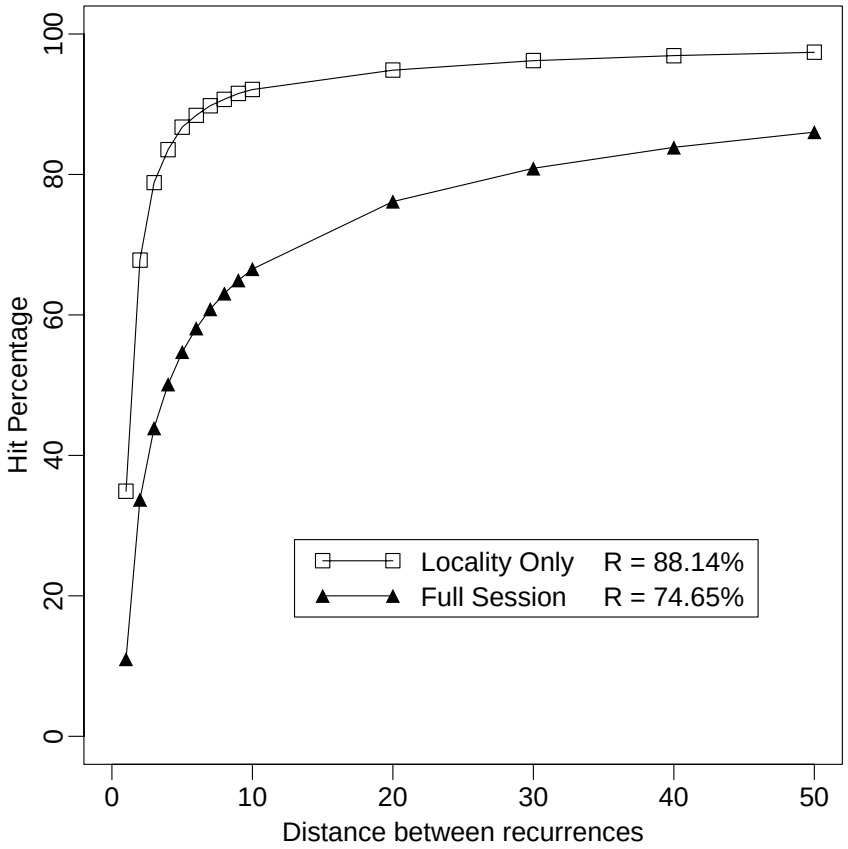


Figure 14: Hit percentage as a function of distance between recurrence.

T	S'	S	$S' - S$
1	34.9	11.0	23.9
2	67.8	33.7	34.1
3	78.8	43.9	34.9
4	83.5	50.1	33.4
5	86.7	54.7	32.0
6	88.4	58.1	30.3
7	89.8	60.8	29.0
8	90.7	63.1	27.6
9	91.5	64.9	26.6
10	92.1	66.5	25.6
20	94.9	76.2	18.7
30	96.2	80.9	15.3
40	96.9	83.9	13.0
50	97.4	86.0	11.4

Table 11: Hit % for various values of T for a full user session (S), restricted user session (S'), and $S' - S$.

These results also demonstrate that a high recurrence rate in command line references is not sufficient for a working-set policy to perform optimally. Rather, the crucial factor is goodness of locality exhibited by command line references. This is consistent with the computer memory research findings [Denning, 1980].

Since users do not exhibit good locality, the working-set history prediction strategy will perform suboptimally. Conditioning techniques, like those proposed by Greenberg and Witten (1988b), may help enhance the prediction of useful history candidates. However, as the above performance analysis clearly demonstrates, history prediction during locality periods are more accurate than during the whole session. Furthermore, the results in Study 3A indicate that 65% of history uses occur within phases.

Concluding Remarks

There are four key results from these studies of locality in command line references. First, locality was exhibited in both the command-lines and command-names cases by all 168 subjects who represent a diverse cross section of UNIX users. Second, locality is a natural consequence of user interactions and is not a randomly occurring behaviour. Third, a substantial percentage of history usages (i.e., 65%) occurred within locality periods. Fourth, unlike program memory references which exhibit good locality (i.e., > 90%), command line references exhibit poor locality (i.e., 31%). This finding has implications for characterizing reuse opportunities and reuse candidates because locality takes into account logical clusters of command lines.

Our study of locality in user interactions along with Greenberg and Witten (1988b)'s study of recency in user interactions, provide an extensive examination of the applicability of computer memory research to the study of user command line reference behaviour and to the performance of the working-set strategy for predicting the (small) set of command line references that a user may need in the near future. However, further research into other aspects of user behaviour are needed before one can conclude that the concepts – working set and locality – are applicable, in general, to other situations in human-computer interaction.

In addition, further refinements to the locality-detection algorithm are required to allow for less restrictive determinations of a locality. The locality-detection technique used in the analyses requires that the references remain unchanged for the duration of a phase. As a result, the pattern matching component is sensitive to slight changes or re-ordering of objects that make up a user reference. The essence of locality is captured by this definition of locality, but in an overly restrictive manner.

An alternative approach for identifying locality, described in Denning (1980), is based on the frequency of segment faults (i.e., transitions are heralded by a series of segment faults occurring in close succession). An examination of this approach of identifying locality, along with a comparison of the results obtained from using this technique against results obtained in our studies, would be informative. This study is left for future research.

In conclusion, there are two possible accounts for poor locality in user interactions. First, poor locality may be due to the fact that users switch between

parallel activities. By separating the different threads of parallel activities, phases would be longer. In this case, the poor locality finding would suggest that user interfaces need to provide tools and resources for managing parallel activities and thereby support for sustaining locality for longer periods. Second, poor locality may be due to the fact that a system imposes fewer constraints on users. As a result, users are not forced to use the same method to accomplish their tasks; they can use different variations which would result in less locality in user interactions. Both accounts are conjectures and must be substantiated through a formal investigation into the cause of poor locality.