AFIT/GE/ENG/92D-04

An Investigation of Structural Locality in the Memory Referencing Behavior of Computer Programs

THESIS

Michael E. Bletzinger
GS-12

AFIT/GE/ENG/92D-04

AFIT/GE/ENG/92D-04

An Investigation of Structural Locality in the Memory Referencing Behavior of Computer Programs

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Michael E. Bletzinger, B.S.

GS-12

December, 1992

Acknowledgements

I would like to thank my thesis advisor, Lt. Col. Hobart, for providing me with his guidance, patience, and wisdom throughout the course of this research. I would also like to thank my family for their encouragement and support as well as the free food and lodging. I would like to thank my fiance, Diane, for her help, prayers, understanding, and support. Her presence made this thesis possible. Finally, this thesis is proof that God answers prayers and performs miracles.

Michael E. Bletzinger

Table of Contents

List of Figures

List of Tables

Abstract

The nature of structural locality as defined by same stack distance access is investigated in this thesis. The question is whether structural locality can be characterized as an inherent and predictive type of behavior. The parameter used to characterize structural locality was the distribution of the lengths of same stack distance runs. First-time memory referencing behavior was also characterized using the distribution of the lengths of new reference runs.

The results revealed that structural locality is strongly influenced by a program's design, and phase of execution. Similar results were also found for first-time memory referencing behavior although this behavior was found to be more inherent than structural locality. Entropy measurements revealed that the predictiveness of structural locality is influenced by program design. The entropy measurements also showed that first-time referencing behavior was more predictable than structural locality.

A Markov model was developed to capture the characteristics of structural locality and first-time memory referencing behavior. Trace synthesis demonstrated some success in reproducing run length distributions when the model had enough states to encompass the entire distribution of same stack distance and new reference run lengths. Entropy measurements on the synthesized traces showed that the predictiveness of structural locality was not solely due to the same stack distance behavior.

An Investigation of Structural Locality
in the Memory Referencing Behavior of Computer Programs

I. Introduction

1.1 Foreword

In the drive toward faster computers, technology has given most of its advances in speed to the computer's central processing unit (CPU) rather than to its memory. This has forced the computer architect to become more creative with memory. He or she has to balance the choices of speed, size and cost, to create a complex memory subsystem where the computer's entire memory appears to be as fast as its speediest, smallest, and most expensive unit. Unfortunately, unlike other things in nature, memory referencing behavior, the way in which software programs access memory, is difficult to model. Thus, the computer designer has had to rely on educated assumptions to design the memory subsystem. This thesis is an investigation of a new type aspect of memory referencing behavior that will add more knowledge to those assumptions and will help define memory referencing behavior mathematically.

1.2 Describing Memory Referencing Behavior

Software programs do not reference memory locations uniformly. Instead the references usually occur in clusters. This is both a solution and a problem to the computer architect. It is a solution because those clusters do not use much space and so can be put in the fastest memory. It is a problem because this clustering effect, called locality, is neither deterministic nor completely random. This makes locality difficult to model. Memory referencing behavior exhibits three types of locality:

Spatial Locality is the tendency for consecutive memory references to be "close" to each other in terms of their location in memory.

Temporal Locality is the tendency of a subsequent reference to be one of those used in the recent past.

Structural Locality is the tendency for consecutive memory references to fall into particular order patterns. For instance, if locations A, E, and W had been consecutively referenced in the past, then when A is referenced again there is a good chance that E and W will be referenced next.

1.3 Purpose of this Thesis

Spatial and temporal localities have been the subject of much research. Computer design innovations such as caches and prefetching strategies have been developed to take advantage of these localities. Structural locality is a more recent characterization and so has not been the subject of much research. Although all three of these localities are straightforward, they have not been quantified so that memory referencing behavior can be modeled. This is the purpose of this thesis:

To characterize and model structural locality in memory referencing behavior

The investigation in this thesis involves measuring structural locality in address traces. An address trace is a record of the order in which memory was referenced while the computer program was executing. The results from these measurements are used to refine an existing analytical model.

This research follows the research in memory referencing behavior done by Hobart (Hob89). Hobart observed that address traces contained large groups of consecutive references which had the same temporal distance indicating that they were being referenced again in the same order. He concluded that these references constituted a memory structure; thus this behavior could be used to measure structural locality. Hobart's results are discussed in more detail in the next chapter. This

research attempts to see if the structural locality behavior observed by Hobart can be predicted. The assumption is made that after a number of consecutive references occur with the same temporal distance, it can be predicted with some confidence that the subsequent references will also have the same temporal distance. In other words, the behavior can be characterized as containing a constant state for some finite duration. The behavior can then be incorporated into a memory referencing behavior model.

The refined model which incorporates this structural locality behavior must be tested to address its stability and validity. The stability of the model is determined by the consistency of the structural locality behavior throughout an address trace. The validity of the model is determined by the scope of the behavior across the spectrum of different types of software. Both of these issues are addressed by determining if the behavior falls in one or more of the following categories:

Inherent Behavior is memory referencing behavior that is the same regardless of what program the computer is executing. This would be due to characteristics of the machine code, in that regardless of the program, the machine code is always compiled to execute certain branches or loops, or move data in certain ways that causes a specific memory referencing behavior. It could also be due to widely used programming structures and algorithms exhibiting a common behavior.

Program Specific Behavior is memory referencing behavior that depends on the way a program is designed to execute. A program's data and control structures would cause certain characteristics in memory referencing behavior.

Local Behavior is memory referencing behavior that depends on phases and transitions during a program's execution. A program's phase of execution would then dictate the behavior.

1.4 Outline of the Thesis

The subsequent chapters are organized as follows:

Chapter Two gives the context of the thesis as well as the definitions and concepts used in this paper.

Chapter Three describes the techniques that were developed to measure structural locality.

Chapter Four summarizes the results of the locality measurements.

Chapter Five discusses the refinements to the memory referencing behavior model based on the results of the locality measurements and addresses the issues of the validation and the stability of the model.

Chapter Six discusses the conclusions drawn from this thesis and their application to computer architecture design. This chapter also discusses recommendations for follow-on research.

1.5 The Address Traces Used to Measure Behavior

The address traces used for this research came from two sources. Both of these sets of traces were collected by altering the microcode of the computer to store each reference that is called. Hobart collected address traces for his dissertation using custom microcode on a Texas Instrument Explorer II workstation (Hob89). He traced a wide variety of software. The programs were all written in Lisp and were divided by Hobart into two categories.

Symbolic Programs are programs which used symbolic processing and data driven behavior. These included applications which performed qualitative reasoning, theorem proving, as well as expert systems, compilers, and symbolic computation programs.

Numeric Programs are programs which primarily used data structures such as arrays, matrices, hash tables, and records rather than the list structures used by the symbolic programs. These included applications that performed numeric computations and circuit analysis.

These categories were chosen because the focus of Hobart's research was to find the differences in memory referencing behavior between normal computational software and symbolic software used

Table 1.1. Explorer Trace Descriptions

Program Name	Application	Category
BIASLisp	Circuit Analysis	Numeric
Boyer	Theorem Prover	Symbolic
Compile Buffer	Lisp Compiler	Symbolic
FFT	Fast Fourier Transform Computation	Numeric
GLISP	Expert Systems Tool	Symbolic
QSIM	Qualitative Reasoning	Symbolic
Reducer	Symbolic Computing	Symbolic
TMYCIN	Expert System Tool	Symbolic

in artificial intelligence. Hobart eliminated most of the references which were unique to the Explorer operating system by modifying the code; commenting out print statements and redirecting the input away from the keyboard (See Table 1.1 for a description of the traces). Most of the traces had 1 to 2 million references. Hobart extracted 150,000 reference samples from each trace to do his analysis. This sample size was kept for this research so that the results could be compared. The traces were captured during the computational phase of the programs to prevent distortions due to operating system disk and user interface I/O routines. The references in these traces are virtual addresses and are independent of the architecture of the TI Explorer. See Chapter 6 of (Hob89). Hobart's research revealed that two of the programs showed more than one phase. For the Compile program these phases were within the trace. Two samples, Compile-RB representing the read buffer phase and Compile-STR representing the streaming phase of the two pass compiler, were extracted from the Compile trace and used in the set. GLISP had phases large enough so that two traces could be collected. The GLISP-Comp trace is GLISP compiling an expert system. GLISP-Pay is GLISP running an expert system that performs financial calculations. The latter trace was an attempt by Hobart to trace a data processing type program. Unfortunately, this trace is not pure data processing since GLISP frequently consults its expert system data to run the program.

Agarwal, Sites, and Horowitz also used altered microcode to trace programs on a VAX 8200 processor (ASH86). As in the Explorer traces, these ATUM (for address tracing using microcode) traces are virtual addresses. See Table 1.2 for a description of these traces. Unlike the Explorer

Table 1.2. ATUM Trace Descriptions

Acronym	Description	Category
DEC0	DECSIM, a behavioral simulator	Symbolic
FORA	Fortran compiler	Symbolic
FORF	Fortran compiler	Symbolic
FSXZZ	File check program	Numeric
IVEX	Interconnect Verify, checks net lists in a VLSI chip	Symbolic
LINP	LINPACK, a numerical benchmark	Numeric
LISP	LISP runs of BOYER, the theorem prover	Symbolic
MACR	An assembly level compiler	Symbolic
MEMXX	Description not available	
PASC	PASCAL compiler of a microcode parser	Symbolic
SAVEC	Part of a C compiler	Symbolic
SPIC	SPICE simulating a 2-input tri-state NAND buffer	Numeric
UE02	UNIX emulator	Numeric

traces, these traces contain operating system routines at an average user to system ratio of 4 to 1. They also contain stack references which also are not present in the Explorer traces. Little is known about the programs that produced the ATUM traces, but they are generally accepted as representing typical memory referencing behavior. Table 1.2 also categorizes the traces by their description using Hobart's criteria for symbolic and numeric workloads.

1.6 Other Resources

The research described in this thesis was done on a Sun SPARC®station computer that was connected to a file-server network. The analysis routines used in this research were written in C and compiled on a GNU C compiler. The thesis is written in LaTeX. The graphs were plotted using GNUPlot.

II. Context of the Research

2.1 Introduction

This chapter outlines the past efforts of research into memory referencing behavior. First, some terms which are used in this area of research will be defined. The discussions of related research efforts will be organized by the techniques used and the concepts defined.

2.2 The Anatomy of Computer Memory

Memory in a computer is designed to be hierarchical with the smallest and fastest memory at the top and the slowest and largest memory at the bottom. Most computers have a cache at the top level followed by main memory and then disk storage. A program is stored on the disk drive in sets of consecutive addresses called blocks. As the program starts running, some of the blocks get loaded into main memory and are organized as pages. How and under what circumstances these pages get loaded is determined by the operating system of the computer. As the program calls the references it needs from main memory, they and some of their neighbors get loaded into the cache on the chance that they will be used by the program again. Here are some concepts and definitions used in describing memory referencing behavior:

virtual memory Virtual memory is a technique used so that software does not have to incorporate the implementation of the computer, on which it is running, into its design. This technique allows software programs to see references in a single address space independent of the various storage mediums. The program refers to a large "virtual" address space to get its data. The operating system then determines from this virtual address whether the reference is in main memory or on the disk. It retrieves the reference by converting the virtual address to a physical address.

page A page of memory is a set of references brought from disk to memory upon a page fault.

page fault Page faults (or page exceptions) occur when the program refers to a reference that is not in the main memory. In order to get the reference, the operating system has to load that page from the disk. Page faults slow down the execution of programs and the rate of their occurrance is a measure of the efficiency of the memory management system.

cache hit A cache hit occurs when a reference is found in the cache. A cache miss occurs when the reference is not in the cache and has to be retrieved from main memory. Cache hit rates and miss rates are used as performance measures for memory implementations.

replacement policy Sooner or later the main memory fills up with pages that have been used by the programs the computer has been running. In the same way, the cache also fills up with references. The replacement policy of a computer decides which pages or references can be replaced by the currently needed data without slowing down the program with page faults and cache misses. These policies try to exploit temporal locality.

prefetching strategy Prefetching strategies take advantage of spatial locality by retrieving not only the needed page or reference but also a few of its neighbors called, a prefetch block.

address traces This is the type of data used to research memory referencing behavior. The computer or the operating system is altered to record the virtual address of each reference as a program is running. Information on the type of reference (instruction fetch, read operand, or write operand) is also recorded.

All of the design decisions made on a memory subsystem revolve around locality. Caches are useless without enough temporal locality, and prefetching strategies will degrade performance if there is not enough spatial locality. In all, good memory subsystem design depends on a sound understanding of memory referencing behavior.

2.3 *The Search for a Model of Memory Referencing Behavior*

Research into memory referencing behavior has proceeded on two levels. The first level is page fault analysis where attempts have been made to model the paging behavior of programs. A more recent effort has been at the cache level where modeling efforts have centered on cache hit and miss rates. The research in this thesis centers on the cache, but since the principles of locality apply at both levels, a lot of page fault analysis research helps in interpreting the results in this thesis. The following paragraphs summarize research done in these areas and focus on efforts that have an impact on the research in this thesis.

2.3.1 Page Fault Analysis Denning developed a model called a working set using page fault analysis (Den68). This model is widely accepted today and is based on a program's tendency at the beginning of its execution to cause an initial set of pages to be loaded. After this working set is loaded, the program does most of its referencing from these pages. This model shows up as a large number of page faults at the start of a program. After the working set is in memory, the page fault rate reduces significantly. Lewis and Shedler used some sophisticated statistical techniques to model page fault behavior (LS73). They measured, from address traces, the interval between page faults and the page fault counts. After considerable analysis, they decided that a two-state semi-Markov model best fit the spectrum of page fault intervals and counts.

2.3.2 LRU Stack Algorithm A good technique for measuring temporal locality is to derive a temporal distance string from the address trace using the least recently used (LRU) stack model. An LRU stack is an algorithm which sorts incoming references or pages by the time they were last referenced. So, the most recent reference is at the top of the stack, and the least recent reference is at the bottom. The temporal distance is how far a reference is from the top of the stack plus one, which corresponds to the number of unique references that have been called since the last time the reference was used. If a reference has never been called before, it can be assigned a temporal distance of 0. Lewis and Shedler decided upon the semi-Markov model after they analyzed some

of the distance strings they computed from page number reference traces. Spirn showed how these distance strings could be used to validate memory referencing behavior (MRB) models (Spi77). He showed how "goodness" in a model could be demonstrated by comparing temporal distance histograms and auto-correlograms derived from actual trace data with curves generated by the model. Wong and Morris used address traces to examine the hit rate of a finite LRU stack versus the size of the stack (WM88). They plotted the hit rate versus size curve for various address traces, and then they fitted a curve to the hit rate function with a formula derived from a Markov chain. They demonstrated how benchmark address traces could be synthesized using the hit rate functions of several traces.

2.3.3 Markov Models Although a number of models have been developed to describe memory referencing behavior, the emphasis in this thesis will be on the Markov state model since it is the focus of this research. Spirn suggested that the two-state Markov model developed by Lewis and Shedler could both describe and model memory referencing behavior at the paging level (see Figure 2.1). The two states in the figure represent two different memory referencing behaviors. The state where referencing is occurring within a working set has a low page fault rate. In the other state, a transition is occurring to a new working set and so the page fault rate is much higher. Hobart also used a two-state model to describe individual memory referencing as opposed to page referencing (Hob89). His old state represented the re-referencing of memory addresses, and the new state represented the referencing of memory addresses that had not been previously referenced. Hobart pointed out that the new state was where memory referencing behavior was mostly influenced by spatial locality, and the old state was where temporal locality dominated the behavior. Hobart derived the transition probabilities for the model from a number of numeric and symbolic program address traces. He showed how the differences in transition probabilities could be accounted for by the nature of the program that generated the trace. He also demonstrated how the model could be used to evaluate cache architectures. Agarwal, Horowitz, and Hennessy

used a Markov chain to model the cache miss rate. The references that concern this model are initially loaded during a transition in a program's execution (AHH89) (Their model is shown in Figure 2.2). In this model, the $R1$ represented the initial loading of a single reference. The $R2$ state represented the initial loading of the second consecutive reference and so on. The transition probability fln is thus the probability that a cache miss would occur after the initial loading of a reference. They observed that these initially loaded references belonged in two categories: singular references which were loaded intermittently between calls to references already in the cache, and nonsingular references which were loaded consecutively and are sequentially addressed. They noted that a two-state model separating the singular and nonsingular references is a good approximation of this behavior since the nonsingular references seemed to be memoryless.

2.3.4 Measuring Locality Part of the problem in modeling memory referencing behavior is measuring the degree of the different localities. Spirn showed that temporal distance histograms could be used to indicate the amount of temporal locality in a particular trace. A large percentage of references at the smaller distances would indicate temporal locality. Bunt and Murphy developed a metric for measuring locality based on the Bradford-Zipf distribution (BMM84). They contend that previous applications of this distribution, the number of books in a library versus the number of times they were used, are very similar to memory referencing behavior. Their locality measurement is based on two parameters: The number of references to a particular page and the number of consecutive references to a particular page. Both metrics indirectly measure temporal locality.

2.3.5 Structural Locality As mentioned earlier, structural locality is a fairly new concept which has not been the focus of much research. Matthew J. Thazhuthaveetil and Andrew R. Pleskun introduced the concept as a characteristic of symbolic program memory referencing behavior(TP87). They collected data by monitoring the accessing of LISP lists by "primitive" functions such as *car* and *cdr* . They generated list sets from the resulting data. After analyzing these sets, they found that a few large and long-lived sets generated most of the references and exhibited a large amount

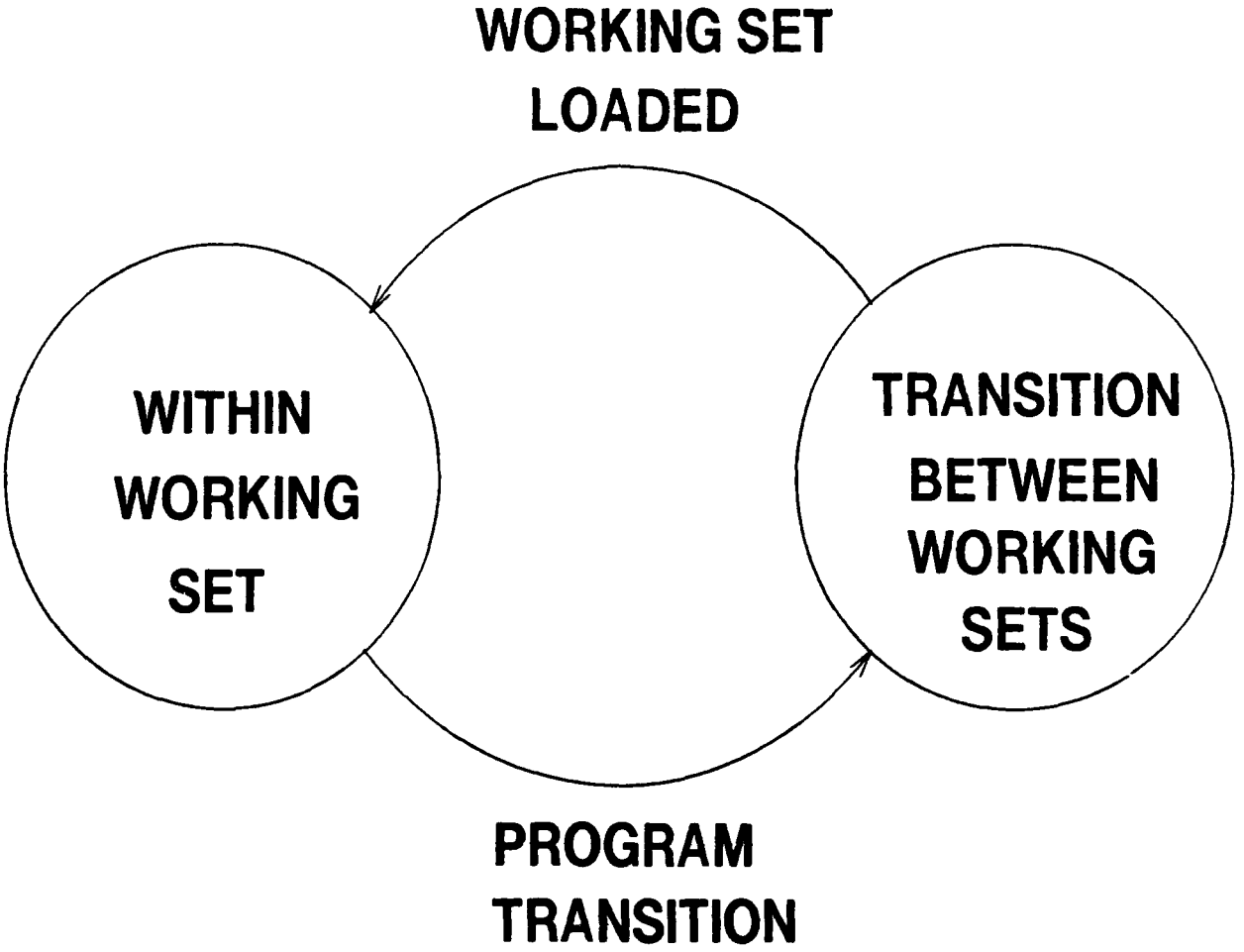


Figure 2.1. Lewis and Schedler's MRB Model

of temporal locality. Thazhuthaveetil and Pleskun suggested that the behavior of these large sets meant that there is a structural aspect to memory referencing behavior in symbolic programs. Hobart was able to define a metric for structural locality using the temporal distance string. He showed that while access to any reference that is already in the stack indicates a degree of temporal locality depending on its temporal distance, consecutive references which have the same temporal distance are being referenced again in the same order. This indicates that they constitute some sort of structure. Hobart represented structural locality by including an additional transition probability in his Markov model from the trace data: the probability of an old—old transition with the same

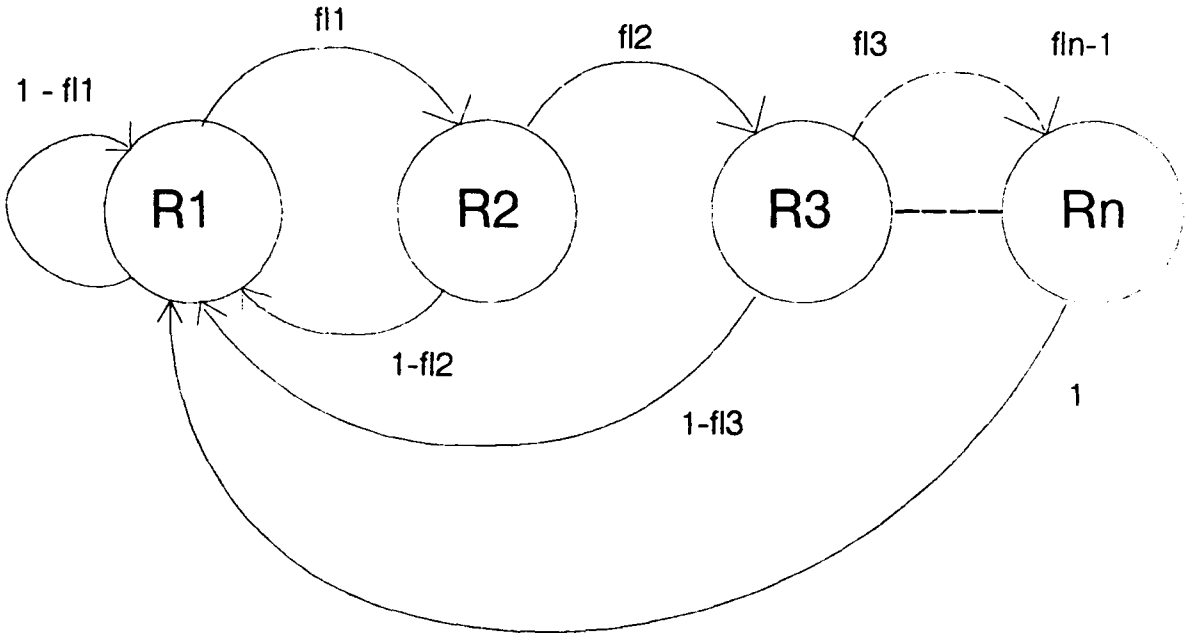


Figure 2.2. Agarwal's MRB Model

temporal distance (see Figure 2.3). He found structural locality in all of the address traces he tested. These results will be discussed in more detail in a later section.

2.3.6 Variations in Locality Another aspect of memory referencing behavior that makes it difficult to model is that the behavior varies as a program progresses. The locality depends on the type of behavior the program is currently displaying, e.g. a looping construct or matrix multiplication. Madison and Batson used temporal distance strings to show how temporal locality occurred in intervals (MB76). They extracted bounded locality intervals (BLI) from distance strings. A bounded locality interval shows the life of a particular working set of references. The BLI would be described by its activity set of references and its lifetime at the head of the LRU

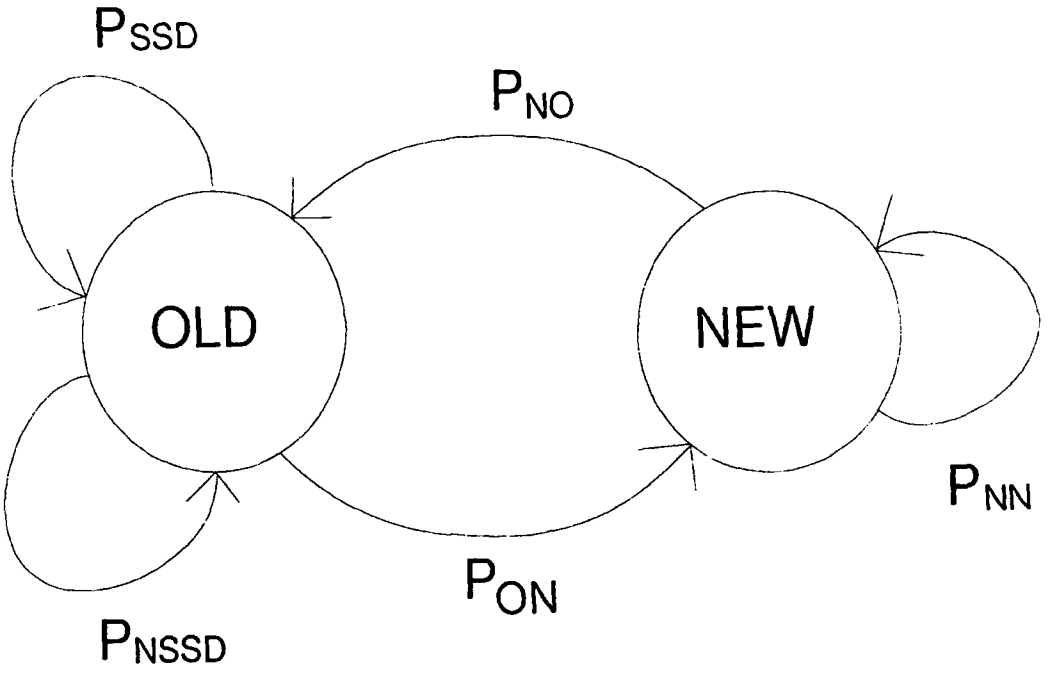


Figure 2.3. Hobart's MRB Model

stack. Madison and Batson demonstrated how a hierarchy of BLIs could be developed from an address trace using its distance string. Masuda and Fin pointed out that the BLI model did not fit well with the accepted notions about locality (Mas83). They showed a new model for locality sets that was based on the source program. They defined locality contours that were derived from the looping structures in the source code. These contours were marked by the compiler so that they showed up on the address trace. Masuda and Fin found that the contours correlated well with the BLI's but had less of a hierarchy. Murphy and Bunt used their locality measurement technique to observe the locality in working set phases and transitions between working sets (MB88). The phases and transitions were derived from the traces using a technique developed by S. Kurten

(Kur86). Transitions were recognized in the traces as page fault clusters and phases as page fault lulls. Murphy and Bunt found no difference between the locality in the phases and locality during transitions. The locality of the phases and transitions were both higher than for the overall program which they attributed to the "locality disrupting effects of aggregation (MB88)". This meant that the locality present within a phase or transition tended to be masked as the measurement broadened in scope to include more phases and transitions.

2.3.7 Apportioning Locality A program references data for different purposes. The program fetches instructions from memory to find out what to do next. It fetches data from memory and writes data to memory depending on the instructions. Many researchers have separated the address trace into instruction, read data, and write data traces on the idea that the localities that characterize the overall trace stem from the behavior of these extracted traces. Murphy and Bunt separated address traces into instructions and data strings and measured the locality in each. They found that the instruction traces had a stronger tendency for reusing references than the overall trace. This tendency toward reuse, i.e. temporal locality, was weaker in the data traces. Hobart found that instruction traces had the strongest presence of structural and temporal locality of all of the extracted traces as well as the overall trace.

2.3.8 Cache Performance Modeling A.J. Smith developed the use of trace driven simulations to measure the performance of a cache architecture design. This method enjoys wide spread use, although it slows down the design process. Smith tried to speed up this process by attempting to model referencing behavior in terms of cache performance parameters such as the miss rate (Smi78). Agarwal, Horowitz, and Hennessy followed up Smith's research by developing a model that predicted the miss rate based on such memory referencing behavior phenomena as start-up effects, non-stationary behavior, and intrinsic and extrinsic interference (AHH89). Fricker and Robert also developed an analytical cache model (FR91). Their model, which was based on the source referencing behavior of the benchmark Kernel-3 of Lawrence Livermore, described memory

referencing behavior by an elementary reference followed by a sequence of repeated references based on the elementary reference and a locality variable. Fricker and Robert contend that this model gives the right amount of randomness without destroying the locality of the behavior being modeled.

2.4 *Hobart's Work*

The research done by Hobart serves as a starting point for this thesis. The MRB model, that is the focus of this research, is a refinement of the model that he investigated in his dissertation. As mentioned earlier, Hobart derived transition probabilities for a Markov state model (See Fig. 2.3). The transition probabilities were derived from an address trace using its temporal distance string. A new reference is defined to have 0 temporal distance, while an old reference has a temporal distance that is less than or equal to the size to which the LRU stack has grown at this time. For example, consecutive references with distances 0 and 3 would indicate a new–old transition, and consecutive references with distances of 4 and 13 would indicate an old–old transition that is not the same stack distance. The former would contribute to P_{NO} and the latter to P_{NSSD}. Hobart analyzed the traces of eight different workloads. From these workloads he found a strong presence of structural locality especially when the traces were separated into instruction, read data, and write data traces. From this data, Hobart concluded a need for further refinements in his model to include structural locality behavior. The remaining chapters will explain these refinements.

III. Structural Locality Measurement Methods

3.1 Introduction

This chapter discusses the techniques that were developed to measure structural locality. The first section defines structural locality so that it can be measured. The rest of the chapter outlines three techniques that measure different aspects of structural locality. As shown in Chapter 4, the combined results of these three types of measurements help to quantify and characterize the structural locality in any address trace. Chapter 5 will then use these measurement techniques on traces synthesized from an MRB model to show how well the model captures the characteristics of structural locality.

3.2 Defining Structural Locality

Structural locality describes the referencing that occurs among the locations that constitute a structure such as a code segment, loop, matrix, or linked list. For instructions, these structures are accessed by referencing their locations often in the same order. This behavior shows up in a temporal string as consecutive references with the same stack distance and so will be called SSD access. Data structures are less often accessed this way. These structures can also be referenced in reverse of the previous access which would show up as consecutive references with decrementing stack distances. This type of access will be referred to as reverse access. Furthermore, accesses to structures such as matrices could alternate between column and row accesses causing the consecutive stack distances to follow equation 3.1. This will be called transverse access.

$$S = (I \bmod L) * L + \frac{I}{H} \tag{3.1}$$

S is the stack distance.

I is the ith consecutive reference.

L is the length of the dimension (row or column) of the matrix previously traversed.

H is the dimension presently being traversed.

Hobart's research showed that a large proportion of references in the temporal strings had the same stack distance as their neighbors. This led to his definition of structural locality as the SSD type of access. The other two types of accesses discussed are also present in the data used in this research. However, the measurements by Hobart show that the SSD access is the dominant form of structural locality (Hob89). Therefore structural locality will be measured and modeled by considering only SSD accesses.

3.3 The Run Distributions Method

Using this SSD access definition, structural locality can now be quantified in a trace by measuring the proportion of consecutive references with the same temporal distance. The set of consecutive references with the same stack distance is called a run. The length of the run is determined by the amount of looping in the instructions or the size if the data arrays or lists. A run that consists of consecutive references with the same stack distance of something other than 0 is an "SSD reference run." If the first reference after an old reference run has a stack distance of 0, the run is considered "terminated by a new reference." In the same way, if the first reference after an old reference run has a stack distance other than 0, then the run is "terminated by an old reference."

A run that consists of consecutive references with a stack distance of 0, is considered a "new reference run" since references are assigned a stack distance of 0 the first time they are referenced. The distribution of these runs will be referred to as first-time referencing behavior. Agarwal, Horowitz, and Hennessy also discuss first-time referencing behavior in terms of run distributions but their definition of a run is somewhat different (AHH89). They realized that consecutive new references stem from different sources. The runs consist of interleaved streams of instructions as

well as data from the stack and the heap. To counter this interleaving, Agarwal et al sorted the references by their block addresses. They also stipulated that the consecutive references had to be contiguous in memory so that they could then assume that the resulting runs were the initial access of a structure. The new references runs defined here still contain this interleaving and are not necessarily sequentially addressed so the assumption that these runs are the initial access of a structure is not valid.

Each temporal string has specific distributions of runs of different lengths. Run distributions with a large percentage of long runs have more structural locality or first-time referencing behavior.

The chi square test was used to compare run distributions (SW88). Two versions of the test were used. The first (3.2) compares an actual distribution against the expected distribution. The second (3.3) compares two actual distributions.

$$\chi^2 = \sum_i \frac{(N_i - n_i)^2}{n_i} \tag{3.2}$$

N_i the number of actual occurrences of event i.

n_i the number of expected occurrences of event i.

$$\chi^2 = \sum_i \frac{(S_i - R_i)^2}{R_i + S_i} \tag{3.3}$$

R_i the number of actual occurrences of event i in the first distribution.

S_i the number of actual occurrences of event i in the second distribution.

Event i is defined in this research as a run of length i. To lend significance to this test, two additional parameters are coupled with χ^2. These are ν the degrees of freedom in the distributions, and $Q(\chi^2|\nu)$ the chi square probability function. The degrees of freedom in a distribution equal the number of different events minus one. The chi square probability function is the probability

that the sum of the squares of ν random normal variables of unit variance will be greater than χ^2. $Q(\chi^2|\nu)$ is thus the probability that the two distributions were drawn from the same set. If this probability is close to zero, then the differences between the two distributions are significant. This test is used as a "goodness of fit" test to match distribution functions with actual data. In this research, the test is used to quantify the differences in the run distributions of different traces. The expected distribution is developed from the combined distributions of all the traces in a set using the equations 3.5 and 3.5. The "likeness" among the traces in the set can be determined by testing each of the traces' run distribution with this expected distribution. A $Q(\chi^2|\nu)$ other than zero would indicate some commonality between the shapes of the histograms. Of course, this does not prove that the distributions are the same. It does, however, give information that can be used to compare various samples of traces. As will be seen in Chapter 4, the resolution of this test can be altered by adjusting the bins which group the different run lengths. This gives information about the shape of the histograms that is not weighted by the run lengths. This means that an occasional extra long run, which sometimes appears in a trace, will not distort the data as it would for other tests.

$$Q(L) = \frac{\sum_{i=1}^{N} \frac{O_{i_L}}{T_i}}{N} \tag{3.4}$$

$$P_E(L) = Q(L) * T_A \tag{3.5}$$

$P_E(L)$ is the expected probability of the occurrence of a run of length L.

$Q(L)$ is the average normalized occurrence of runs of length L.

O_{i_L} is the number of occurrences of runs of length L in trace distribution i.

N is the number of traces in the set.

T_i is the total number of runs in trace distribution i.

T_A is the total number of runs of the actual distribution.

The chi-square test shows the amount of difference between distributions but it does not reveal where those distributions differ. For this, the difference, $D(L)$, between the combined cumulative distribution histogram, and the individual cumulative histogram, was computed and plotted (equation 3.6). As can be seen, negative $D(L)$s would indicate that the individual distribution has a smaller fraction of runs of length L than the combined distribution and positive $D(L)$s would indicate a larger fraction.

$$D(L) = \frac{O_L}{T} - Q(L) \tag{3.6}$$

3.4 The Entropy Method

Hammerstrom and Davidson proposed using entropy to estimate the information content of address traces (HD77). They derived a method of transforming an address trace into program graphs of instruction and data references. Then they calculated the entropy of each of the program graphs. In this research, a much simpler application of entropy was developed to determine the predictiveness of structural locality. Entropy is a measure of the randomness of a sequence of symbols (Sha48). The first step was to transform the temporal string of a trace into a sequence of symbols that illuminate structural locality. Two different sets of symbols were used. The first set, called structural entropy, assigned the symbol "1" to each pair of references with the same stack distance, and the symbol "0" to each pair of references that did not have the same stack distance. The second set, called state entropy, assigned a "1" to each old reference and a "0" to each new reference. Since the size of both our symbol sets is 2, the maximum entropy possible in this application is $\log_2(2) = 1$. Just as with Hammerstrom and Davidson's research, the N-gram method was used to estimate entropy. This method is outlined by Shannon (Sha50). Entropy is estimated by calculating the probability of the occurrence of blocks of symbols. These blocks

overlap in the sequence. For example, for a block size of 2 symbols, the first and second symbol would be block 1 and the second and third symbol would be block 2. The probabilities of the occurrence of these blocks are fed into equation 3.7:

$$\begin{aligned} F_N &= -\sum_{i,j} p(b_i, j) \log_2 p_{b_i}(j) \\ &= -\sum_{i,j} p(b_i, j) \log_2 p(b_i, j) + \sum_i p(b_i) \log_2 p(b_i) \end{aligned} \tag{3.7}$$

N is the order of entropy and the size of the block of symbols.

b_i is a block of $N-1$ symbols.

j is an arbitrary symbol following b_i.

$p(b_i, j)$ is the probability of the occurrence of N-gram b_i, j.

$p_{b_i}(j)$ is the conditional probability of the occurrence of symbol j after the block b_i.

The second line of equation 3.7 is the computational form for estimating entropy since this is simply the summation of the contributions of the N-gram occurrence probabilities subtracted from the summation of the contributions of the $N-1$-grams occurrence probabilities. As larger and larger blocks of symbols are examined, F_N gets closer to the absolute entropy H as seen in equation 3.8.

$$H = \lim_{N \to \infty} F_N \tag{3.8}$$

An example of these calculations is found in Appendix A. The problem with using structural entropy as a measurement tool is that it will illuminate any pattern of symbols in a sequence. A pattern of consecutive "1" symbols indicating a run would lower the entropy of the trace as expected, but so will a pattern of consecutive "0" symbols. An additional ratio, Hobart's P_{SSD},

which consists of the number of same stack distances divided by the number of references, can be used to indicate whether the entropy stems from the presence of or the lack of locality.

3.5 Conclusions

Each of these measurements serve a different purpose in this research. Run distributions are used to characterize structural locality and new referencing in the temporal string of the different traces. The chi-square test is used to compare these amounts between traces. The transition probabilities also serve as a structural locality measure and are used to develop a model for the structural locality behavior. Structural entropy is used to show the predictability of the structural locality and the variations in structural locality within the traces. State entropy shows the predictability of the first-time referencing behavior.

IV. Structural Locality Measurement Results

4.1 Introduction

This chapter discusses the results of the structural locality measurements. The focus of these measurements is to determine how well the locality can be modeled, which will be refered to as the characteristics of structural locality. This determination can be made by answering the following questions:

Is Structural Locality Predictable? The assumption is that structural locality can be described by the structure access behavior of repeated referencing in the same order as described in Chapter 3. Whether this type of access behavior is dominant enough to allow structural locality to be predicted by a model has to be determined.

Is Structural Locality Inherent or Program Specific? This will determine the scope of the MRB model. The hope is that one model can be used to describe the memory referencing behavior no matter what type of program is being run.

Is Structural Locality Program Specific or Local? This will determine how stable the MRB model is. The hope is that the behavior being modeled will remain constant over multiple locality phases and transitions.

These questions should be answered in terms of cache performance. Specifically, is structural locality inherent enough to consistently predict cache performance for a wide range of workloads? However, to do this, spatial and temporal locality would also have to be incorporated into the MRB model so that cache performance can be measured on a synthesized trace produced by the model. Incorporating these other two localities into the model is beyond the scope of this research. Instead, these questions will be answered by comparing the measurements of the different traces.

The Explorer and ATUM traces detailed in Chapter 1 are measured to determine relative natures of structural locality and new referencing. In this way, the differences among the traces

and between the sets are determined. The differences between the sets are especially important to determine if the measurements will be corrupted by distortions in the traces caused by things such as the collection technique and architectural differences of the host computers. The predictability issue is addressed next. Structural locality is then characterized by the run disributions of data sets decreasing in scope with the inherency issue addressed first, then program specific behavior, and finally the local behavior.

4.2 Characterizing Structural Locality and First-Time Referencing Behavior

The measurement of structural locality is still relative since no benchmarks for structural locality have been developed. Therefore, the measurements in this research are based on the "normal" combined distribution that was derived for each set using the method in equations 3.5 and 3.5.

The cumulative histograms of the combined Explorer run distributions are shown in Figures 4.1 and 4.2. These are analyzed in this section to define the normal behavior of the set of Explorer traces. As can be seen, the histograms of the new run distributions and the SSD run distributions are similar. The histograms of the instruction traces show a far greater percentage of long runs than the rest of the traces. The overall new run distributions had more singular references than any of the other new run distributions and also more long runs than the read, write, and data distributions. This correlates with the findings of Agarwal, Horowitz, and Hennessy since the overall new run distribution could be categorized as containing a combination of singular and non-singular references. The write new run distribution contained the largest ratio of short runs. In the SSD histograms, the write distribution had the most singular references and the read distribution had the longer runs. The overall and data histograms were between the two. Figures B.11 through B.20 show the differences from normal for each trace in the Explorer set. These figures support Hobart's findings on the structural locality differences between symbolic and numeric programs.

As can be seen, the FFT and BIASLisp traces have a larger proportion of longer runs in the instruction traces and a larger proportion of short runs in the read, write, data, and overall traces. This difference is also reflected in the new run distributions.

Figures C.11 through C.20 show how the ATUM traces differ from their combined distributions. Here, with the exception of the LINP trace, no traces stood out, although it did seem that quite a few traces e.g. MACR had a smaller proportion of singular references.

4.3 Comparisons between ATUM and Explorer traces

Another necessary item to resolve are the differences between the ATUM and Explorer trace sets. To do this, the normal distributions of the two sets of traces were compared. Figures 4.3 and 4.4 show how the Explorer set differed from the ATUM set. As can be seen, the Explorer set has shorter instruction and write runs and longer read, data, and overall runs. From these results, it could be concluded that the Explorer set contained mostly symbolic programs characterized by short runs in the instruction traces and long runs in the data traces, and the ATUM set contained mostly numeric programs which have long instruction runs and short data runs. However as shown in Table 1.2, the ATUM set contains primarily symbolic traces.

To further explore this, cumulative differences were plotted between the BIASLisp, FFT, Compile-RB, Dec0.0, Lisp.0, and Linp.0 traces. BIASLisp, FFT, Dec0.0, and Lisp.0 are the deviant traces in their respective sets with the first two displaying numeric characteristics and the latter two displaying symbolic characteristics. The other two traces were included as controls. The results are shown in Figures D.1 to D.10 and D.11 to D.20. As can be seen, the symbolic ATUM traces still had a larger percentage of short runs and singular references than the Explorer set, although not as much as the control trace. Consequently, the ATUM traces showed more of a difference with the Explorer set than with the ATUM set. On the other hand, the numeric Explorer traces were closer to the ATUM set for most of the distributions, although the ATUM set had a much larger

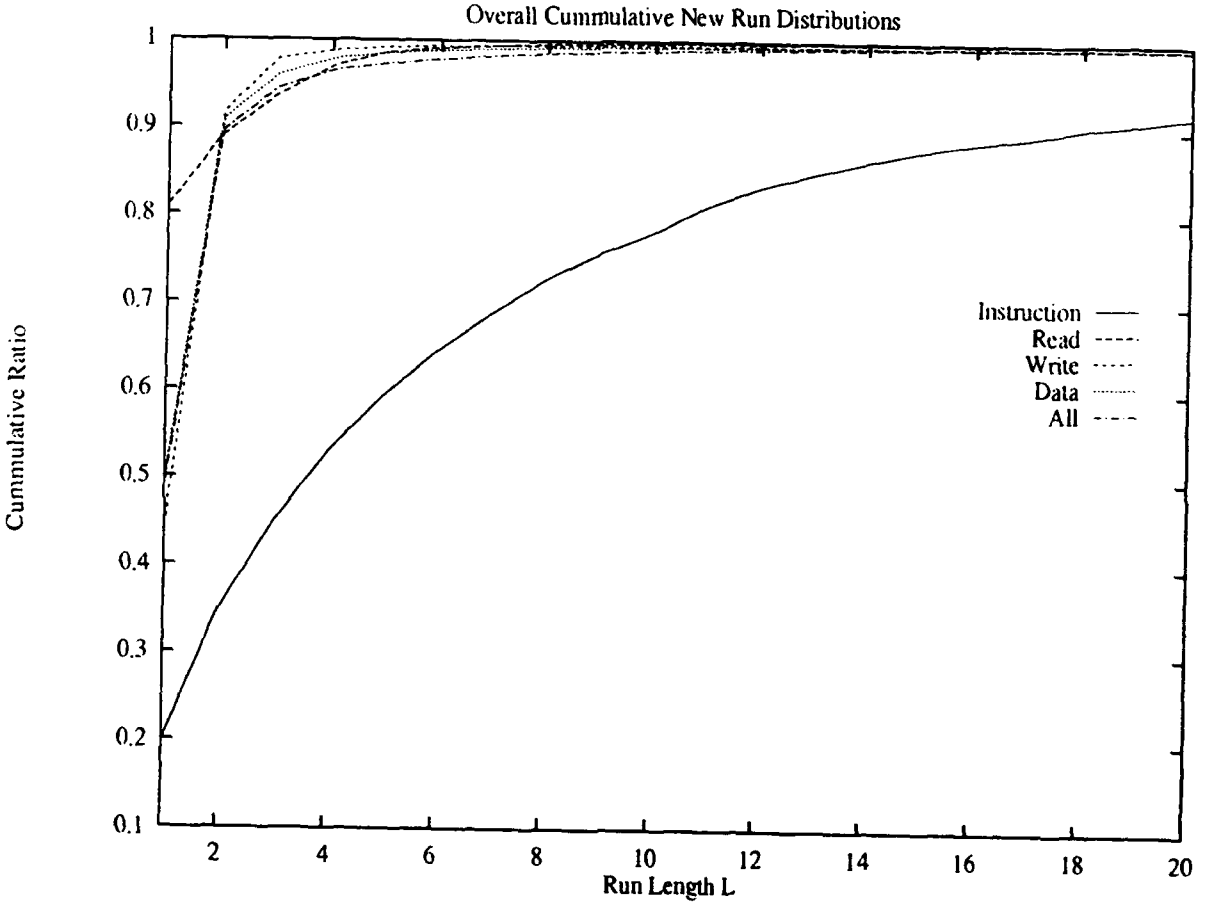


Figure 4.1. Cumulative Distribution of New Runs for the Explorer Traces

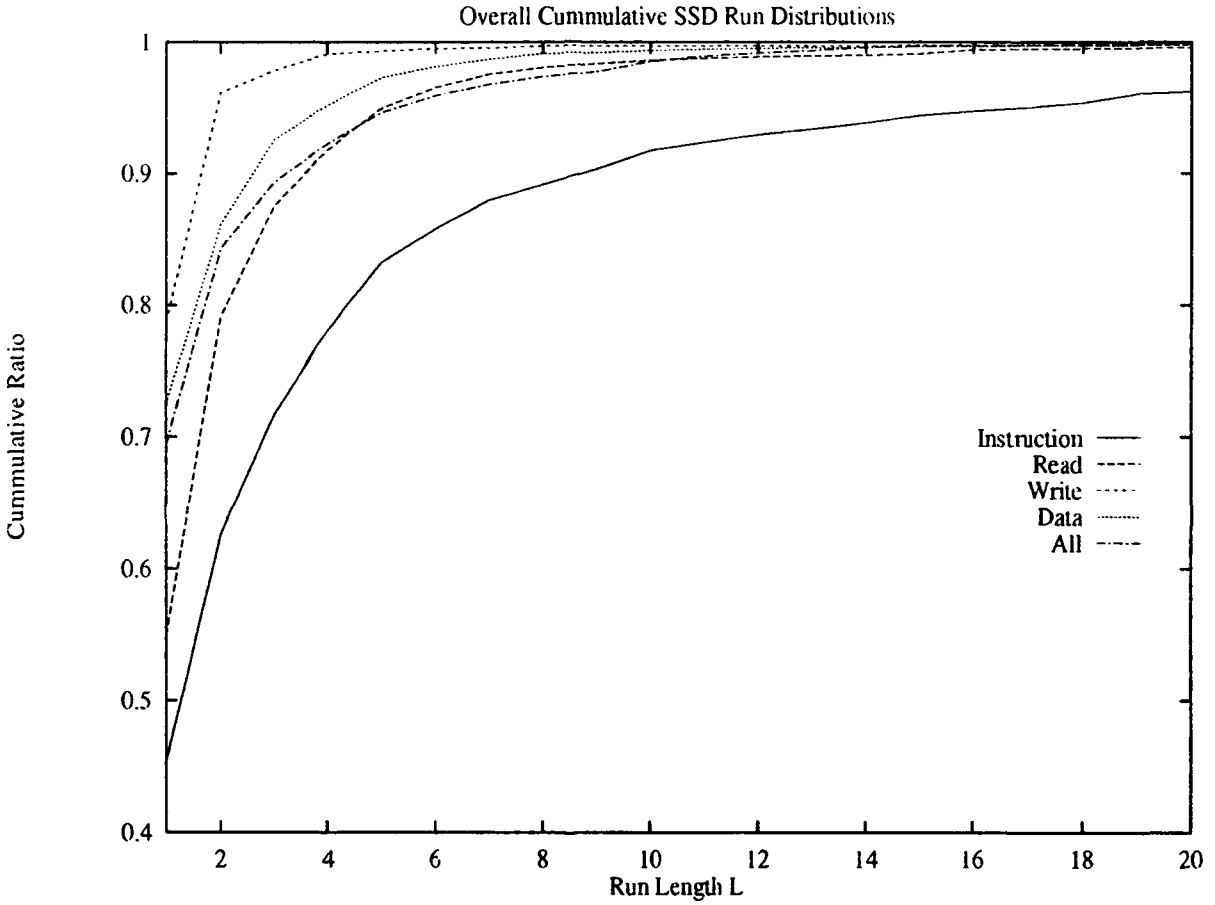


Figure 4.2. Cumulative Distribution of SSD Runs for the Explorer Traces

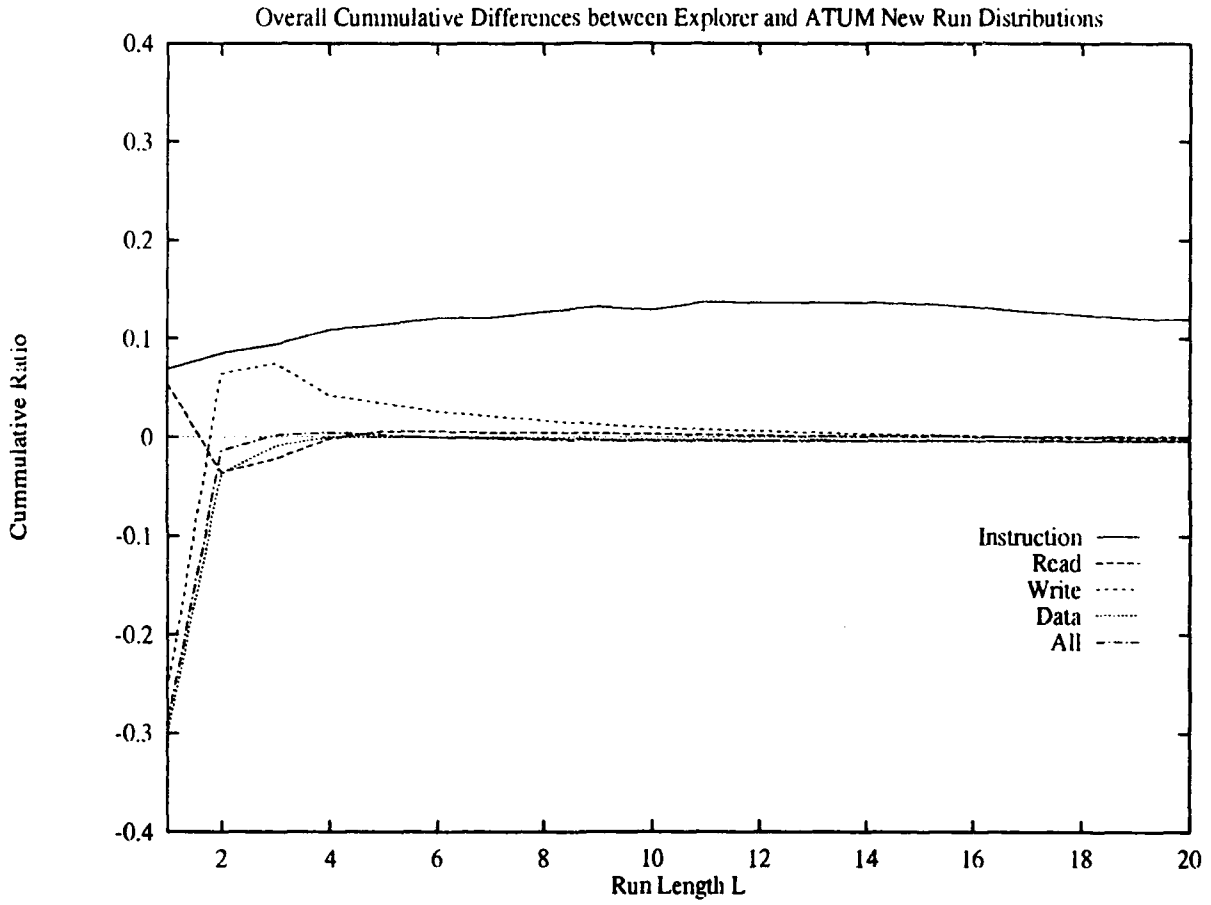


Figure 4.3. Differences in the Cumulative Distribution of New Run Lengths between the Explorer Set and the ATUM Set

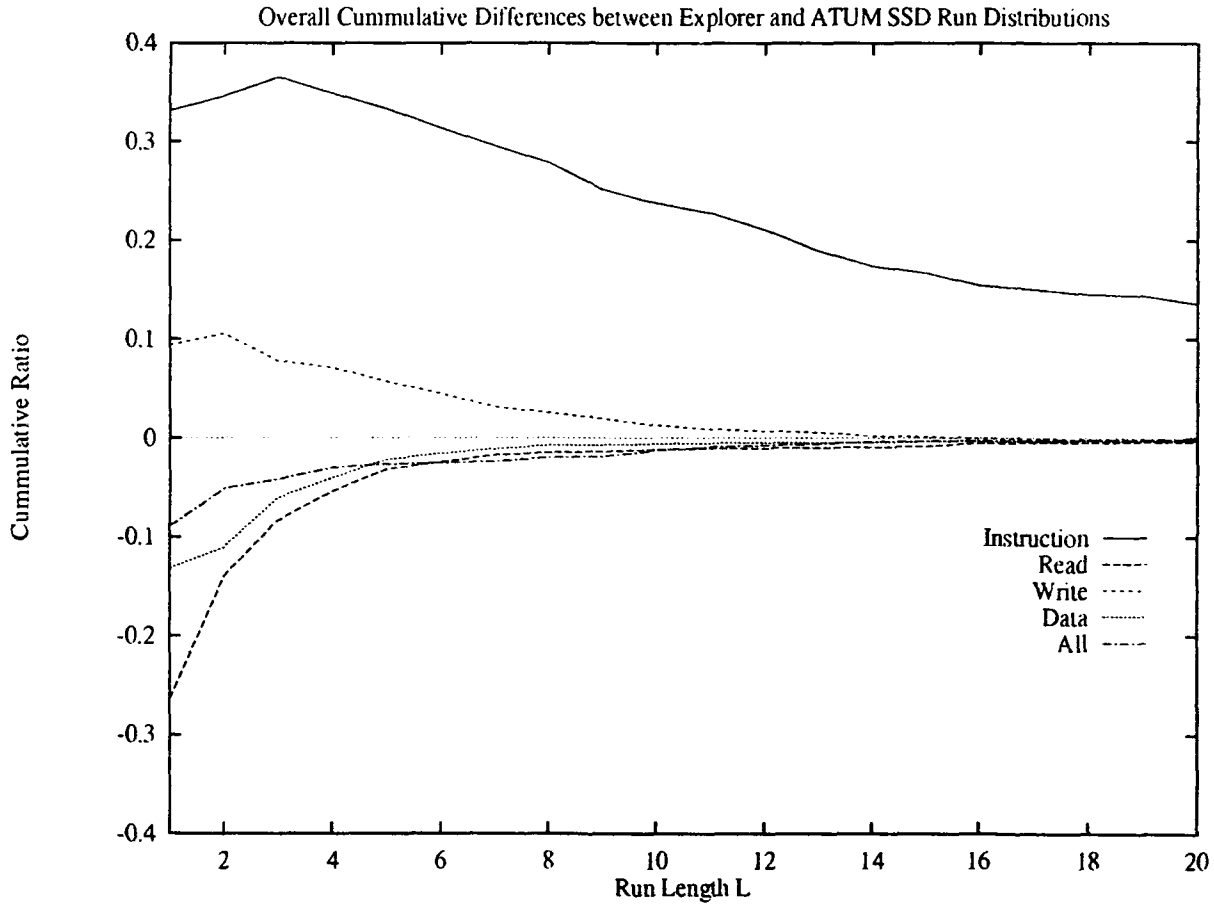


Figure 4.4. Differences in the Cumulative Distribution of SSD Run Lengths between the Explorer Set and the ATUM Set

proportion of singular references in the read, data and overall traces and longer runs in the write traces. In summary, the Explorer set shows more of a symbolic type of behavior than the ATUM set. This may be explained in two ways. First, the ATUM set does have more numeric traces than the Explorer set which could have affected its distributions. Second, the Explorer operating system is designed to run AI programs, which means that the compiler for the Explorer could emphasize the symbolic behavior in the Explorer traces, while the VAX compilers and assemblers could emphasize numeric behavior in the ATUM traces.

4.4 Predictability

The structural and state entropy of the Explorer traces was measured to determine the predictability of structural locality. The entropy was estimated to the 20th order to reveal how much of the past is needed to predict future behavior. For state entropy, the results are in Figures B.1 to B.5. As can be seen, the entropy for the instruction traces stays constant as the order increases showing that only the latest reference is needed to predict whether the next reference will be new or old. The entropy level is set at the first order indicating that one of the symbols (which in this case is the old references) dominate the traces. The rest of the traces show a steadily decreasing entropy as the order increases with the exception of some of the write traces. This shows that an increasing history of previous references will only marginally improve the predictability. The results for structural entropy (Figures B.6 to B.10) tell a different story. Here the entropy varies widely out to about the 16th order where the entropy curve flattens out. This shows that whether the next reference will have the same stack distance or not can best be determined from a history of about 16 previous references. These results show that the same stack distance behavior is not as predictive as the transitions between new and old references. The excessive structural entropy in the lower orders could point to a more complicated structure access than the SSD access which is captured by the entropy measurement. The results for the ATUM traces, which are found in Figures C.1 to C.10, are similar except for the structural entropy of the instruction traces which

had less variation. The low entropy in the LINP trace as compared to the other traces suggests a relationship between structural entropy and program complexity, since the LINPACK benchmark which generated this trace consists of a simple loop.

4.5 Program Dependence

The variations in the structural entropy curves in the previous section seem to suggest that the distribution of SSD run lengths in a trace is not inherent. This section will attempt to determine the significance of these variations. If the chi-square test indicates that the run distributions of each individual trace have no likeness to the combined distribution of all the traces, then the distribution of SSD run lengths in memory referencing behavior is not inherent. All the traces within each set were combined as explained in Chapter 3. The chi-square test was used to determine the difference in shape between the histograms of the combined and individual distributions. For this test, the combined distribution was the "expected" distribution and each individual distribution was an "actual" distribution. The chi-square test results for the Explorer traces are shown in Table 4.1. The χ^2 probability function was uniformly 0.0 for all the traces except the new run instruction traces. This indicates that with the exception of the new run instruction traces, the chi-square test rejects the hypothesis that there is a relationship between the individual distributions and the combined distributions. Thus, according to this test, the distribution of SSD run lengths in the traces is not inherent.

Hobart's transition probabilities (P_{SSD}) in the instruction traces are so high that they would seem to suggest that some inherency exists in the distribution of SSD run lengths present, but even in this case, the distribution of SSD runs varies. The chi-square test looks at all of the runs equally. The transition probabilities on the other hand are dominated by the SSD transitions, so that once a run is considered "long" its actual length does not really matter. To reflect this in the chi-square test, a "long run bin" was established in the histogram which contained all of the runs with lengths

greater than L. L was then varied to see at what length $Q(\chi^2|\nu)$ would no longer be zero. The results showed that $Q(\chi^2|\nu)$ was uniformly 0.0 even down to two degrees of freedom which had one bin for singular or -SSD references and one bin for non-singular or SSD references. Another binning scheme that produced unsatisfactory results had the bin size increasing exponentially. For example, for a base 3 binning scheme, the first bin would have runs of 1, the third bin would have runs of length 2 and 3, and the second bin would have runs of lengths 4, 5, 6, 7, 8, and 9. Here also $Q(\chi^2|\nu)$ was uniformily zero even as the exponential base was increased until the degrees of freedom reached 2. The only binning scheme that had non-zero $Q(\chi^2|\nu)$ was linearly based, with bins of an equal size. The results are shown in Figures 4.5 and 4.6. The new run distributions showed some likeness in all types of traces, although the data and overall traces didn't show likeness until a bin size of 72 and the write traces had only a slight $Q(\chi^2|\nu)$ at a bin size of 38. On the other hand, the SSD run distributions did not show any likeness except for the write traces at a bin size of 80. This suggests that the distribution of SSD run lengths is less inherent than the distribution of new run lengths.

The ATUM set showed similar results as shown in Table 4.2. As shown in Figures 4.7 and 4.8, the ATUM set showed more likeness than the Explorer set for all of the types of traces except the SSD run distributions for the instruction trace. Two subsets were chosen out of the ATUM set to see if the set results could be improved. Subset 1 consisted of the IVEX, LISP, and FSXZZ traces which registered a non-zero $Q(\chi^2|\nu)$ for some of the bin sizes in the data and overall traces. Subset 2 added DEC0 which registered a non-zero $Q(\chi^2|\nu)$ with IVEX in the read trace, and FORF which registered a non-zero $Q(\chi^2|\nu)$ in the overall trace. The results are shown in Figures 4.9 and 4.10. They show an improvement in $Q(\chi^2|\nu)$ suggesting that these traces share some commonality in their SSD run distributions.

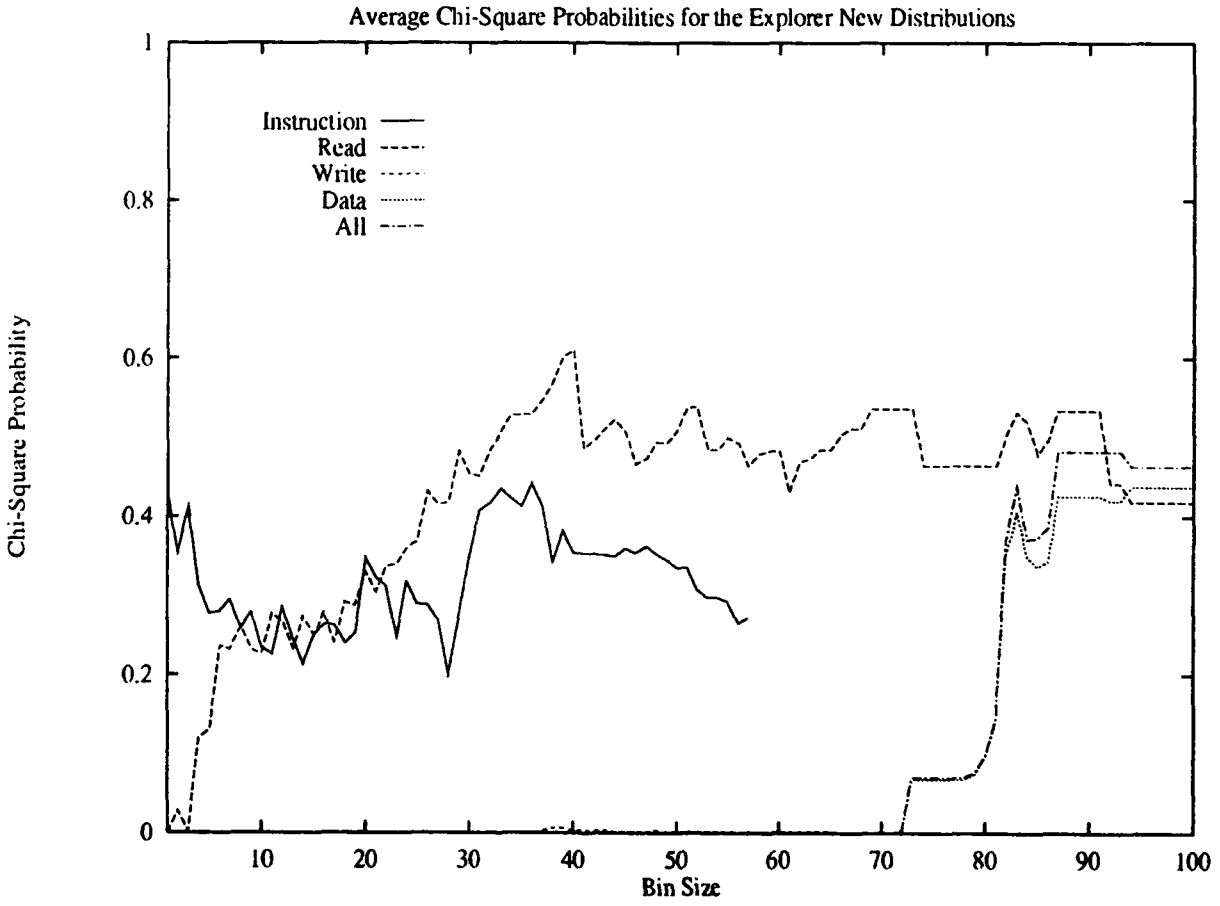


Figure 4.5. Chi-Square Probability versus Bin Size for the Cumulative Distribution of New Run Lengths for the Explorer Traces

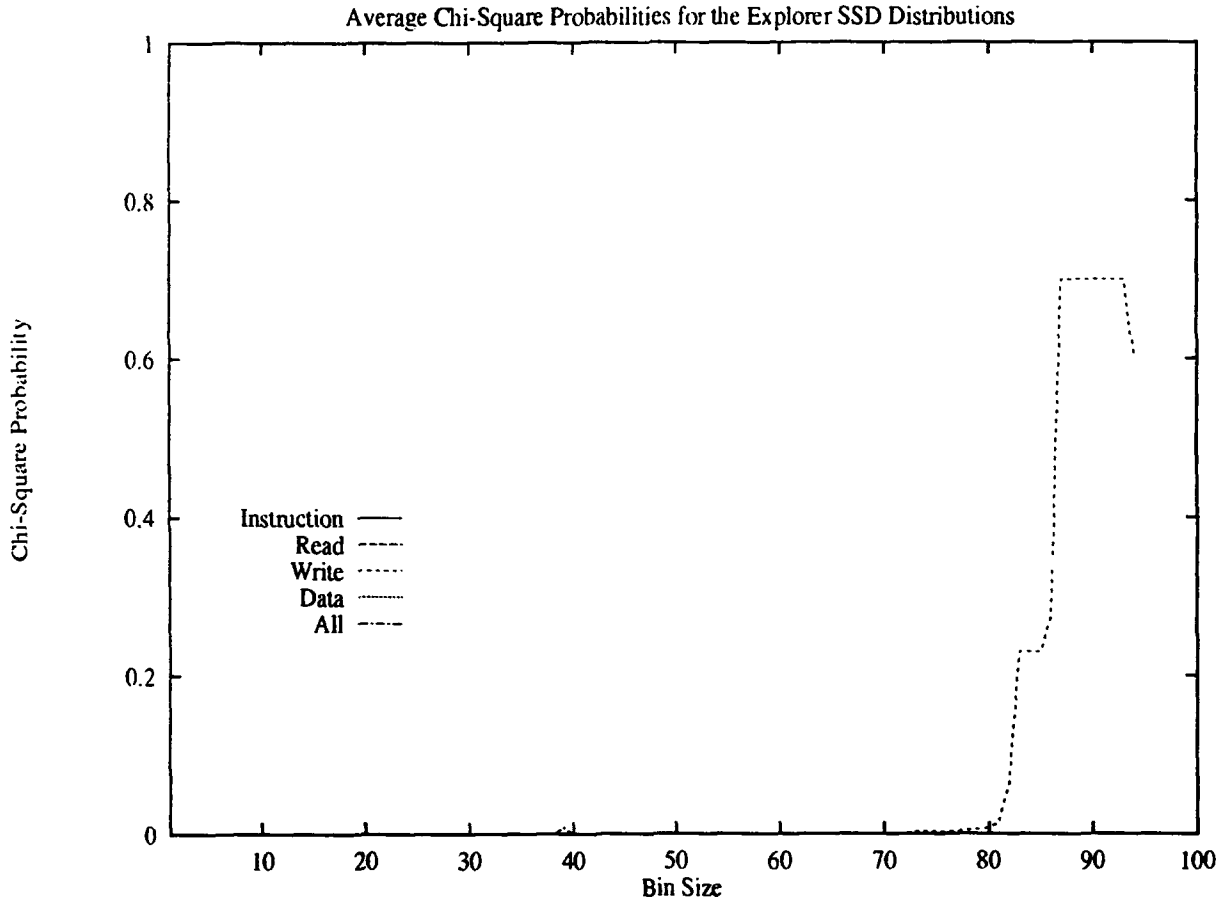


Figure 4.6. Chi-Square Probability versus Bin Size for the Cumulative Distribution of SSD Run Lengths for the Explorer Traces

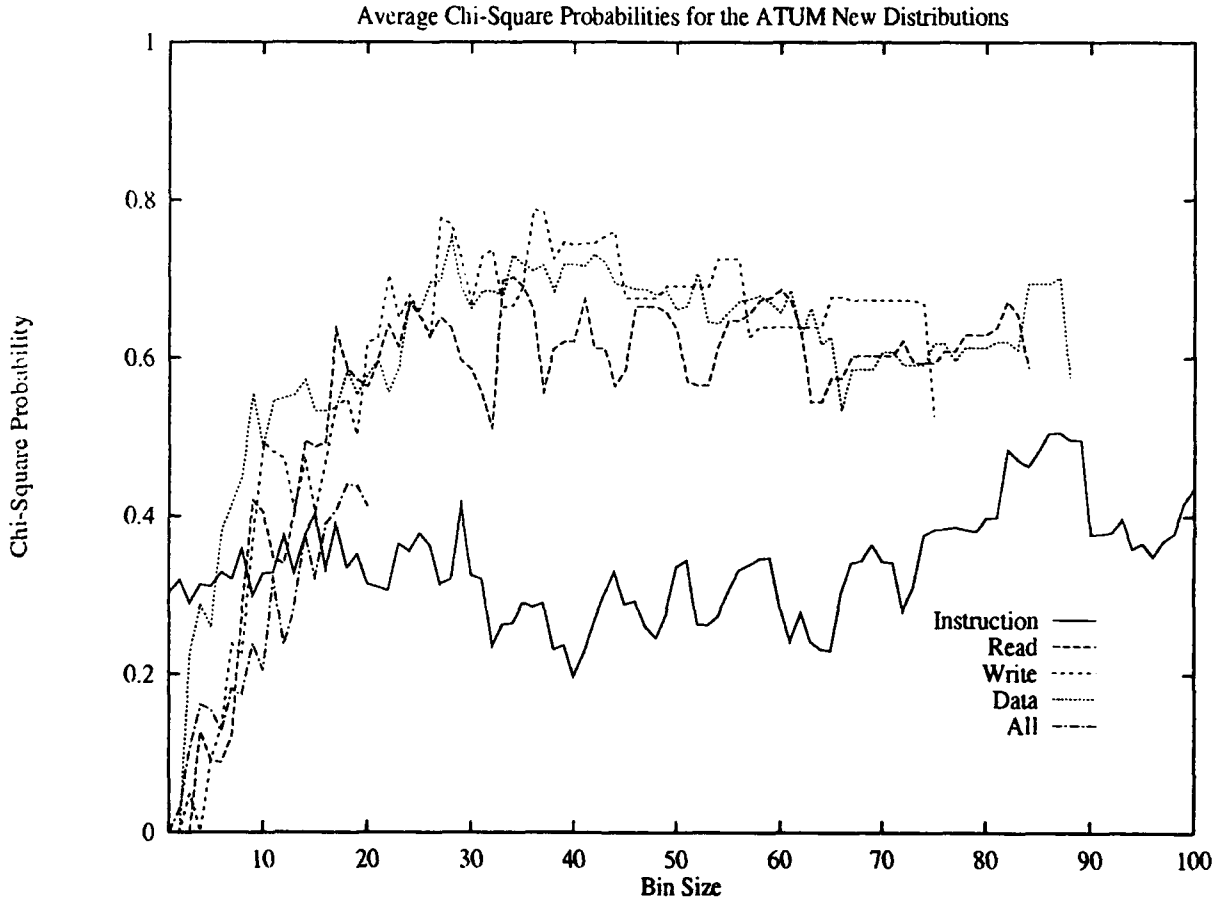


Figure 4.7. Chi-Square Probability versus Bin Size for the Cumulative Distribution of New Run Lengths for the ATUM Traces

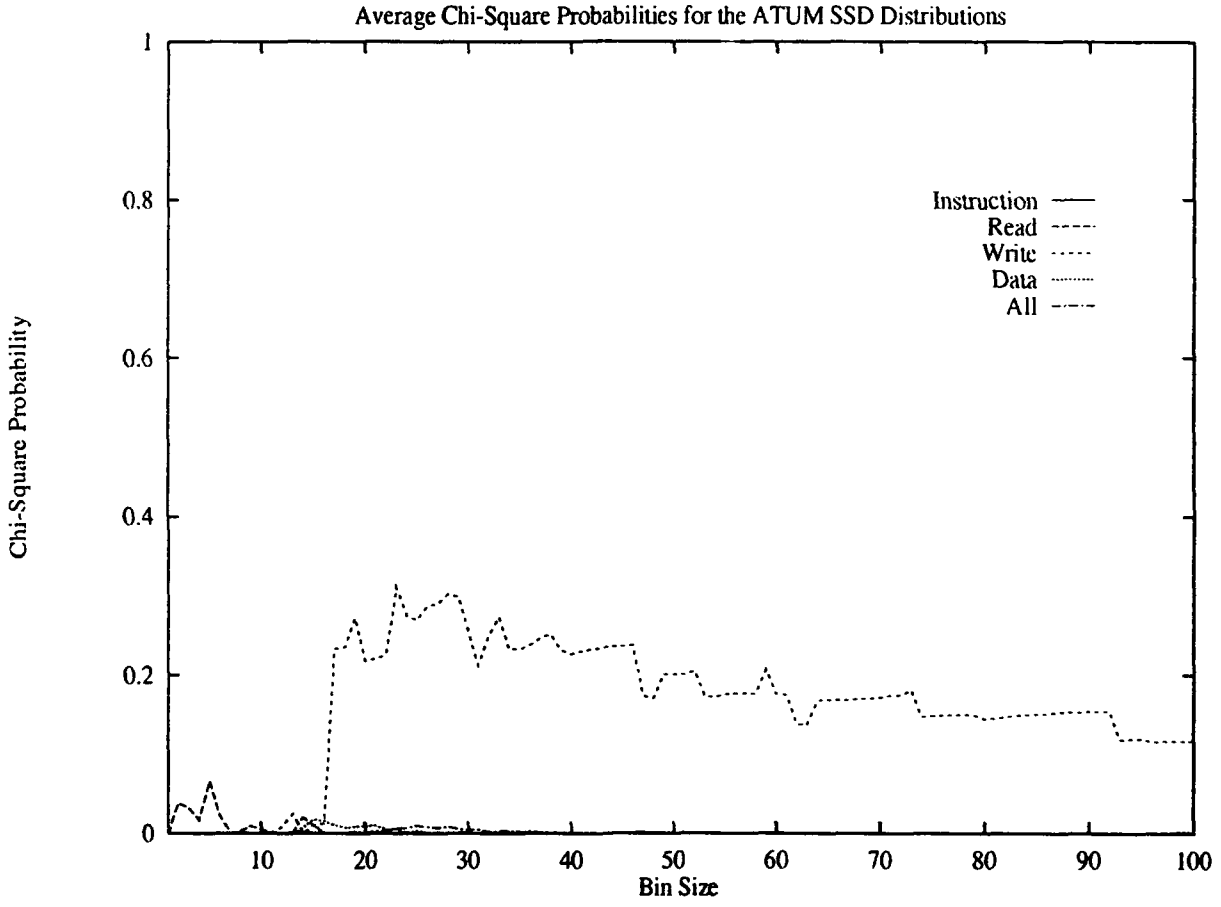


Figure 4.8. Chi-Square Probability versus Bin Size for the Cumulative Distribution of SSD Run Lengths for the ATUM Traces

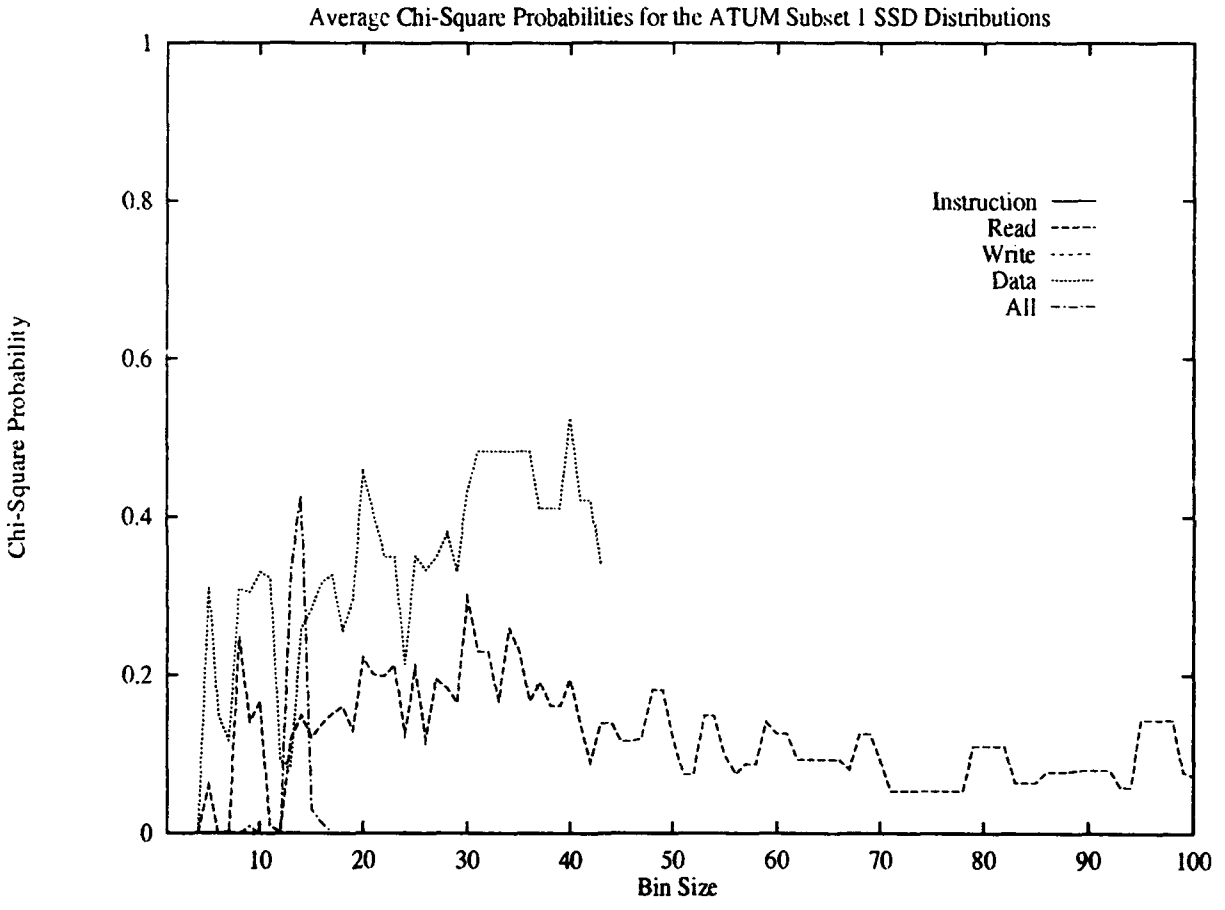


Figure 4.9. Chi-Square Probability versus Bin Size for the Cumulative Distribution of SSD Run Lengths for Subset 1 of the ATUM Traces

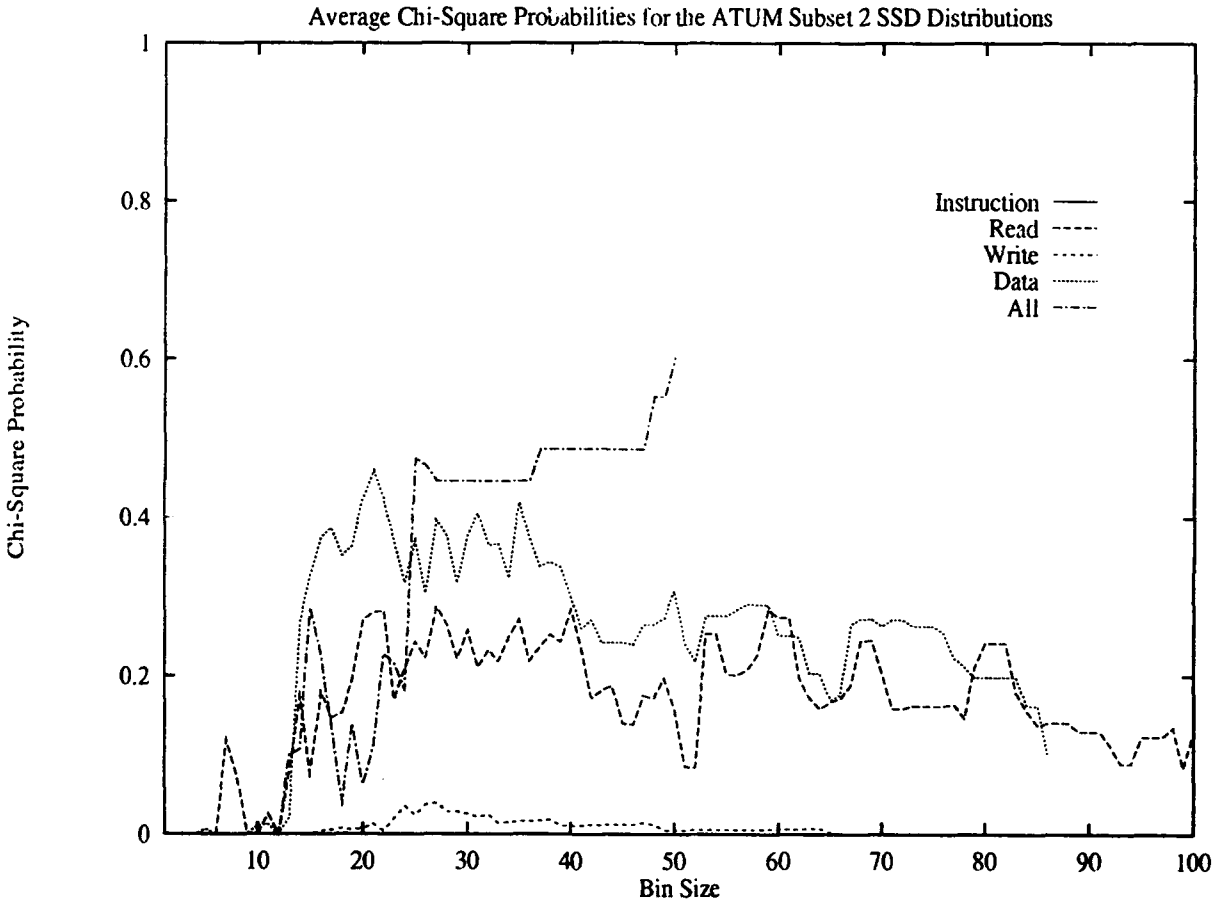


Figure 4.10. Chi-Square Probability versus Bin Size for the Cumulative Distribution of SSD Run Lengths for Subset 2 of the ATUM Traces

Table 4.1. Explorer Traces Overall Chi-square Test Results (Inter-Trace)

Trace Type	New Runs Average χ^2	ν	$Q(\chi^2\|\nu)$	SSD Runs Average χ^2	ν	$Q(\chi^2\|\nu)$
Instruction	86.14	69	0.428	9186.46	201	0.000
Read	1591.65	55	0.000	16,778.90	91	0.000
Write	4290.45	51	0.000	4536.18	57	0.000
Data	3045.30	80	0.000	11,810.71	93	0.000
All	3131.30	84	0.000	29,563.92	83	0.000

Table 4.2. ATUM Traces Overall Chi-square Test Results (Inter-Trace)

Trace Type	New Runs Average χ^2	ν	$Q(\chi^2\|\nu)$	SSD Runs Average χ^2	ν	$Q(\chi^2\|\nu)$
Instruction	184.41	168	0.303	5940.78	681	0.000
Read	1946.42	65	0.000	7212.32	236	0.000
Write	1627.26	55	0.000	3942.14	107	0.000
Data	1888.95	72	0.000	9353.96	107	0.000
All	1548.95	48	0.000	16262.7	146	0.000

4.6 Variability Within Programs

The next step was to combine multiple samples of each trace. A zero $Q(\chi^2|\nu)$ would indicate that the distribution of SSD run lengths has nothing to do with the static characteristics of the program, but is dependent on the local phases and transitions within the trace. This was only done with the Explorer traces since the ATUM traces were too short to extract multiple samples. Each Explorer sample contained 450,000 references which were sufficient to characterize the memory referencing behavior for all but the Compile trace for the metrics used (Hob89). These samples were used in the testing discussed in the previous sections. New 450,000 reference samples were extracted consecutively starting with the first reference. The chi-square test would define local variations as variations between the 450,000 reference samples.

The measurements with the chi-square test are binning characteristics in that they cannot describe the type of variations within the traces. To determine where these local variations occurred, 4th order structural entropy estimation measurements were computed and plotted. The 4th order estimation was used because the results were smoother than the lower order estimations.

The entropy was measured for every 10,000 references in a trace. This window is the same size as the reference window Agarwal, Horowitz, and Hennessy used in their analytical cache model measurements. They determined the size of their window, which they called the time granule τ, by observing the cache miss rate at the start of the program. They decided on 10,000 because this was the smallest size that did not cause large variations in the miss rate at start up (See Appendix A in (AHH89)).

Figures E.1 through E.9 show the intra-trace chi-square probabilities and Figures E.10 through E.18 show the 4th order structural entropy versus time in references. The Explorer set could be subdivided by comparing these graphs. The GLISP-Pay, Compile, and QSIM did not have non-zero $Q(\chi^2|\nu)$ until a bin size of 12 up to 43. These traces also showed major phase variations in their 4th order structural entropy plots. The other traces had non-zero $Q(\chi^2|\nu)$ for bin sizes down to 1 and 2. FFT did have a few large variations in structural entropy, but apparently these did not affect the distributions. GLISP-Comp also had large variations, but these variations were reoccurring with a period close to the 450,000 reference sample size which could account for the low non-zero $Q(\chi^2|\nu)$ bin size. QSIM also had reoccurring variations of the same magnitude as GLISP-Comp but with a much larger period (1,300,000 references) which led to a large non-zero $Q(\chi^2|\nu)$ bin size. BIASLisp, Boyer, Reducer, TMYCIN had less 4th order variations in entropy and small non-zero $Q(\chi^2|\nu)$ bin sizes. In summary, the distribution of SSD run lengths does depend on the phases and transitions of a programs execution, although programs with only a few phases, or with periodic phases, show a tendency towards a fixed SSD run length distribution.

4.7 Conclusions

The questions asked at the introduction of this chapter can now be answered. Structural locality needs a history of about 16 previous references to be predictable. This does depend on the type of program that generated the trace. The distribution of new run lengths is much more

predictable as was seen by comparing the state entropy with the structural entropy measurements. The distribution of new run lengths was also more inherent than the distribution of SSD run lengths, although overall, neither behavior had more than a slight chance of inherency. The ATUM set had several traces that seemed to share the same distribution of SSD run lengths, and some of the Explorer traces also seemed to have a constant distribution of SSD run lengths between the 450,000 sized samples. A correlation also existed between the likeness among the samples and the variation in 4th order structural entropy which suggested that the distribution of SSD run lengths varied with major phases and transitions of the programs. In summary, the measurements do not support the idea of being able to model structural locality with a fixed distribution of run lengths. How well this model does match this behavior is answered in the next chapter.

V. *Memory Referencing Behavior Model*

5.1 Introduction

This chapter explains the proposed refinements to the MRB model. The refinements are proposed and correlations are made to the results in Chapter 4. Then two validation techniques are proposed and findings are discussed.

5.2 Proposed Refinements to the Memory Referencing Behavior MRB Model

As was discussed in Chapter 2, Hobart's original model described memory referencing behavior in terms of two states; the "new" state where previously unreferenced memory locations are being referenced, and the "old" state where memory locations are being re-referenced (See Figure 2.3). To model structural locality, Hobart split the old-old transition in the model to differentiate the old-old transitions with the same stack distance. As was discussed in Chapter 1, the intent of this research is to determine if structural locality behavior can be modeled as constant state for some finite duration. A certain amount of memory has to be added to the model in order to predict when this constant state behavior occurs. The memory is provided by adding a chain of states to either side of the model. These chains represents bulk server Markov chains with a queue size equal to the server size. The last state in this chain would then be the constant state, and all of the other states in the chain would be transition states which predict the constant state.

Figure 5.1 represents a model with the longest markov chains possible. If a constant state exists for structural locality behavior, then the transition probabilities P_{NND_L} and P_{SSD_L} , would eventually converge to a constant value as L increases. Figures 5.2 to 5.5 show large variations in the transition probabilities for the combined Explorer traces as well as some individual traces. The transition probabilities never converge. An example of the computation of these probabilities is found in Appendix A.

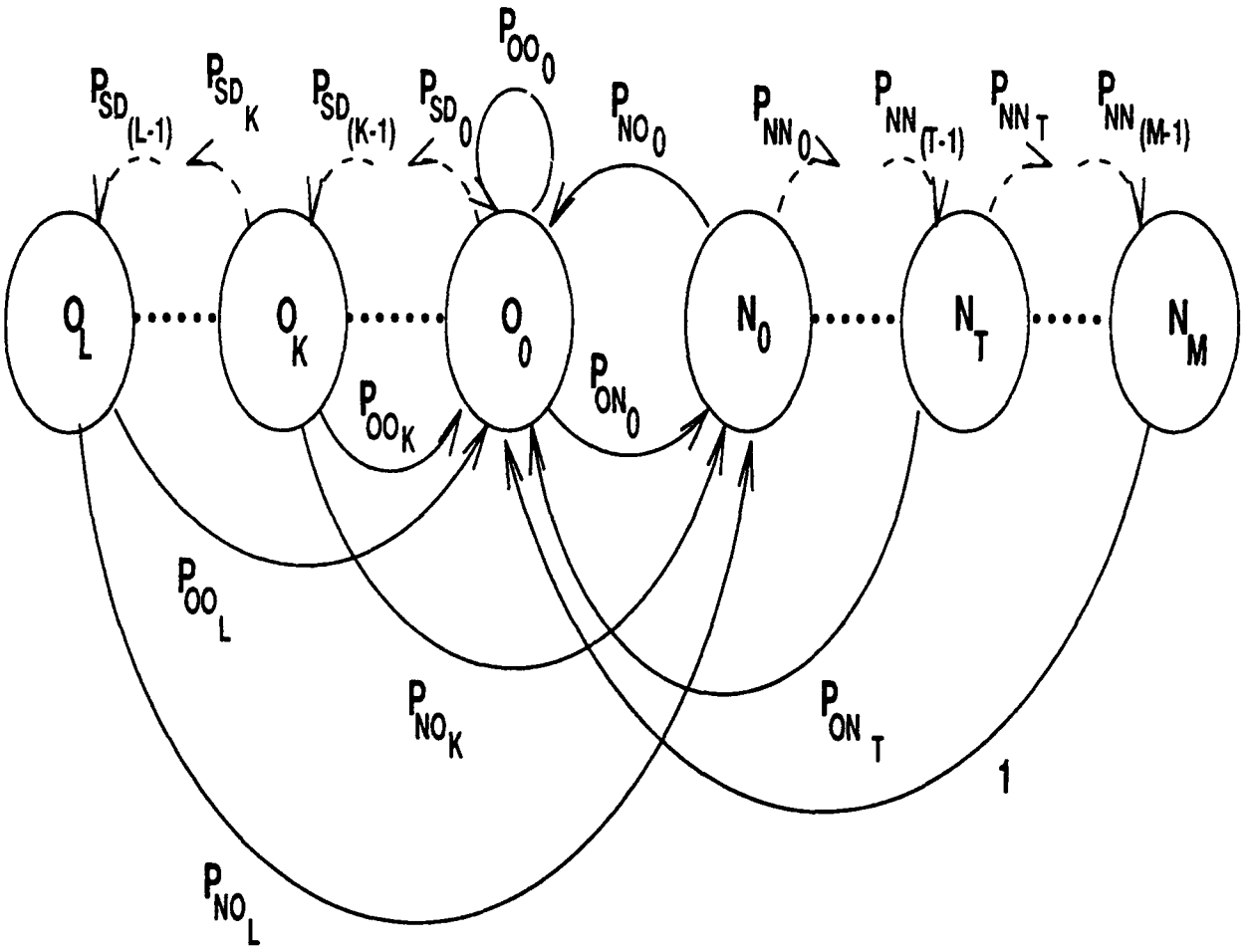


Figure 5.1. MRB Model with the Number of States Equivalent to the Longest SSD and New Reference Run Length

These results do not support the idea of a constant state in structural locality behavior. They confirm the conclusions in the previous chapter which call into question the stability and scope of these refinements. However, developing this model could reveal some additional characteristics of structural locality behavior that have not been revealed by the measurements already discussed.

5.3 The Number of States to Add to the MRB Model

The number of states to be added to the MRB model remains an open question. The structural entropy measurements in the previous chapter would support adding 16 additional states to either

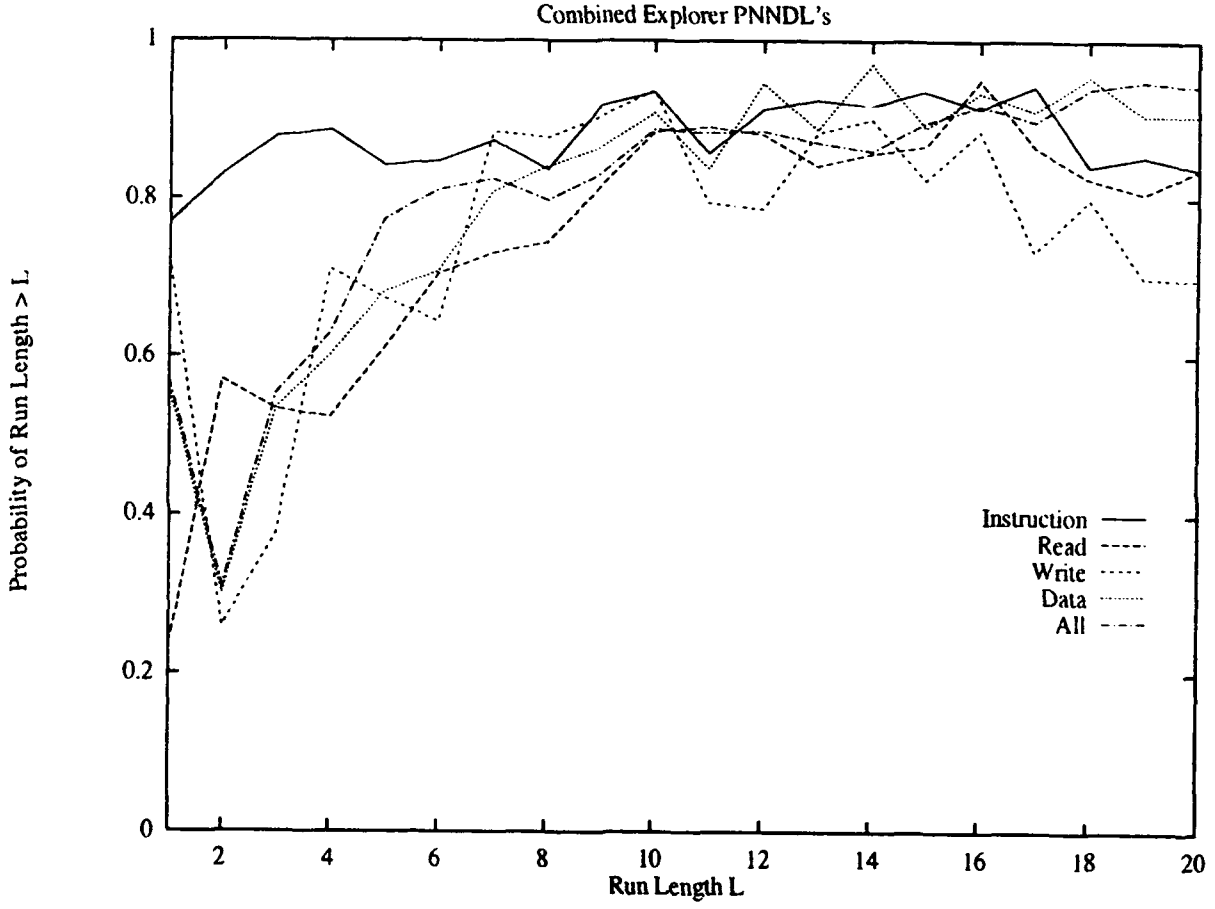


Figure 5.2. Explorer Traces Combined PNNDLs

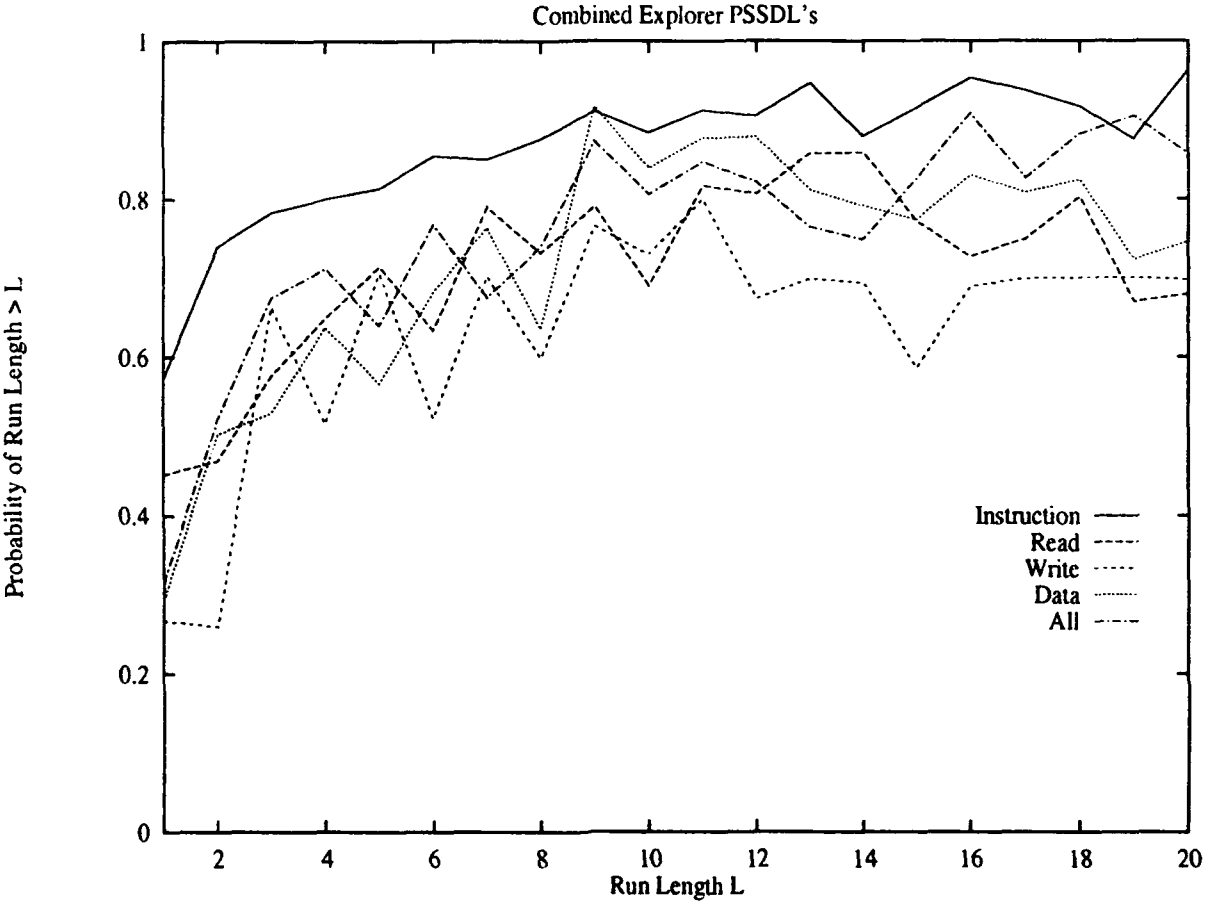


Figure 5.3. Explorer Traces Combined PSSDLs

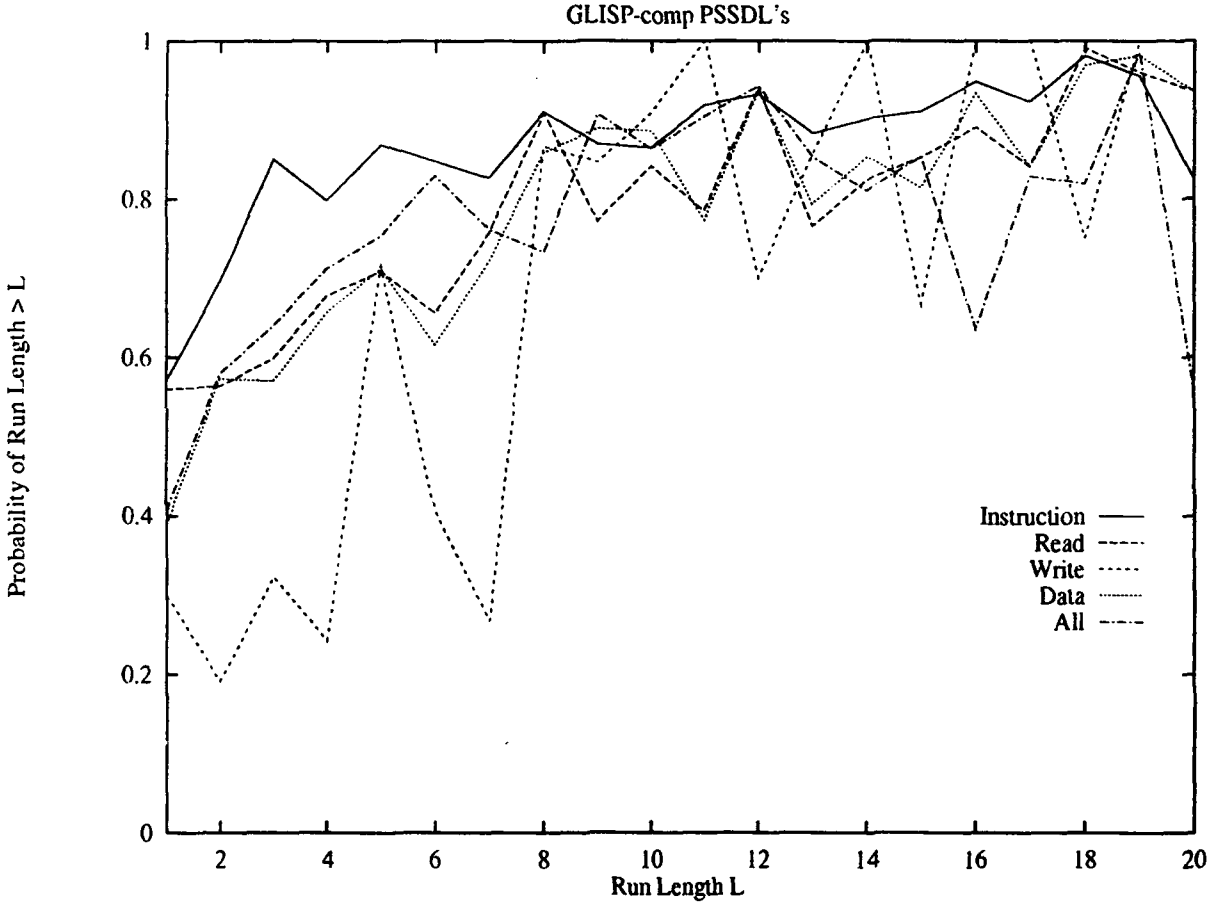


Figure 5.4. GLISP-Comp PSSDLs

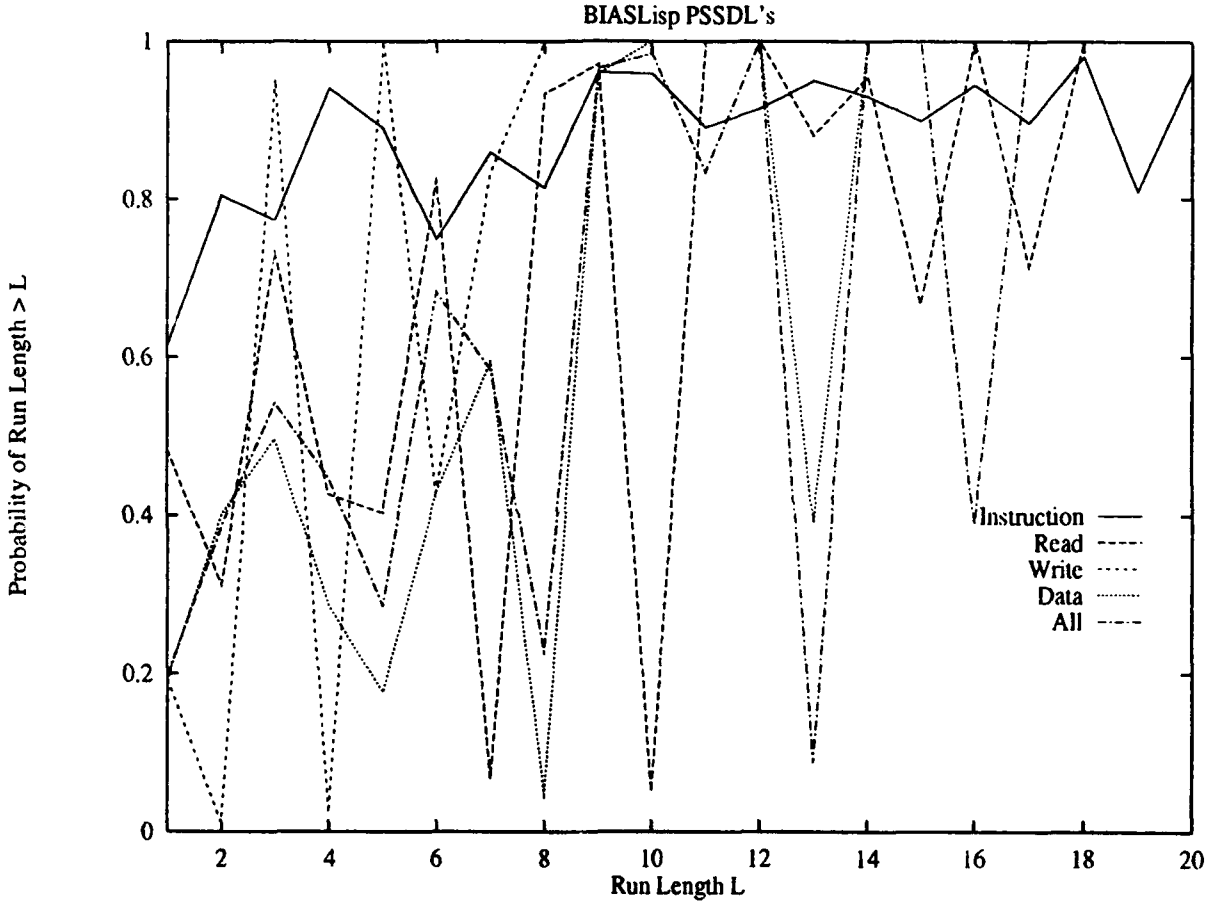


Figure 5.5. BIASLisp PSSDLs

or both sides of the model. The best approach to resolve this issue is this proposed "general case MRB" model (See Figure 5.6). As can be seen, the parameters L and M describe how many states are added to each side. These two parameters can thus be adjusted to fit the structural locality characteristics of any trace. The transition probabilities for this model are computed as before with the exception of the transition probabilities of the L and M states. Since these states are the constant states, their probabilities have to model the distributions of runs with lengths greater than L or M. The average new run length w_N is given by.

$$w_N = (1 - P_{NN_M}) * \sum_{j=0}^{\infty} P_{NN_M}^j = \frac{P_{NN_M}}{1 - P_{NN_M}} \tag{5.1}$$

Where P_{NN_M} is the probability of a new-new transition given that there have been at least M consecutive new-new transitions as shown in Figure 5.6. Likewise, the average SSD run length is given by

$$w_O = (1 - P_{SD_L}) * \sum_{j=0}^{\infty} P_{SD_L}^j = \frac{P_{SD_L}}{1 - P_{SD_L}} \tag{5.2}$$

The rest of the transition probabilities for the constant states L and M are then derived from the average and a ratio of O_{OT} and O_{NT} which are the number of old and new terminated SSD runs.

$$P_{NO_M} = 1.0 - P_{NN_M} \tag{5.3}$$

$$P_{OO_L} = \frac{O_{OT}}{O_{OT} + O_{NT}} * (1.0 - P_{SD_L}) \tag{5.4}$$

$$P_{ON_L} = 1.0 - P_{SD_L} - P_{OO_L} \tag{5.5}$$

The other transition probabilities for the model are computed the same as the transition probabilities in the previous markov chain model 5.1.

The state equations of the model can be derived by partitioning the model (Kle75). Equation 5.6 is from partition 2 (Figure 5.6), equation 5.7 is from partition 1, and equation 5.8 is from partition 3. Equation 5.9 states that the sum of all the state probabilities should equal one.

$$\sum_{i=0}^{L} P_{O_i} P_{ON_i} = \sum_{j=0}^{M} P_{N_j} P_{NO_j} \tag{5.6}$$

$$P_{O_{K-1}} P_{SD_{K-1}} = \sum_{i=K}^{L} P_{O_i} (P_{OO_i} + P_{ON_i}) \tag{5.7}$$

$$P_{N_{T-1}} P_{NN_{T-1}} = \sum_{j=T}^{M} P_{N_j} P_{NO_j} \tag{5.8}$$

$$\sum_{i=0}^{L} P_{O_i} + \sum_{j=0}^{M} P_{N_j} = 1.0 \tag{5.9}$$

P_{O_i} is the state probability of the old state i which relates to the probability of the occurrence of a SSD run of length i or longer.

P_{N_j} is the state probability of the new state j which relates to the probability of the occurrence of a new run of length j or longer.

L is the number of old states in the model.

M is the number of new states in the model.

P_{SD_i} is the probability of an old-old transition with the same stack distance from the old state i.

P_{OO_i} is the probability of an old-old transition with a different stack distance from the old state i.

P_{ON_i} is the probability of an old-new transition from the old state i.

P_{NN_j} is the probability of an new-new transition from the new state j.

P_{NO_j} is the probability of an new–old transition from the new state j.

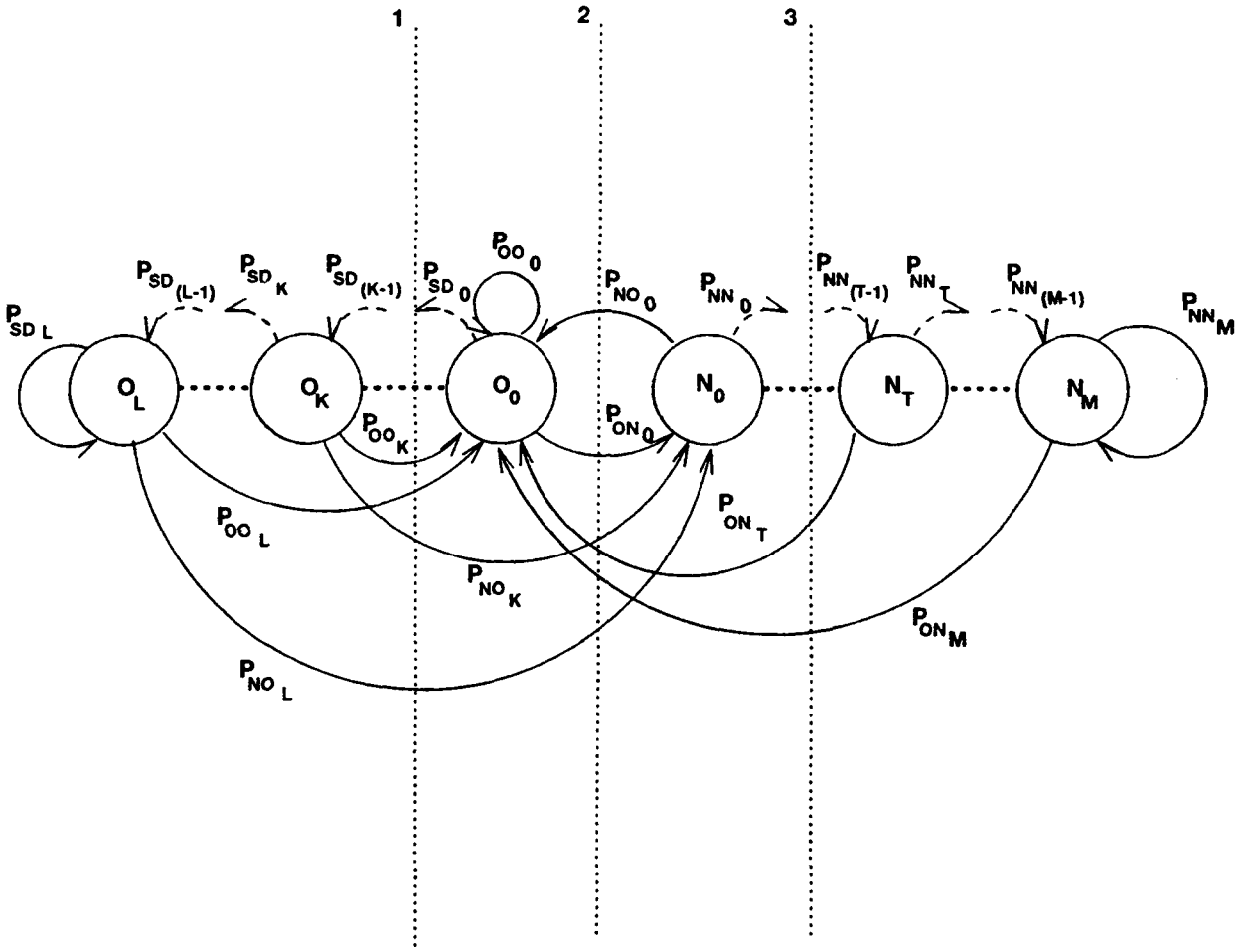


Figure 5.6. General Case Markov MRB Model

5.4 The MRB Model and Entropy Estimations

This MRB model implicitly contains an entropy prediction. Two orders of estimation can be computed. The first order would be computed using only the state probabilities. The second order is calculated by a summation of the products of the state probability and corresponding transition probability. To calculate the state entropy, the products are divided into two sets which are then summed. The products are placed in sets depending on whether the transition probability of the product ends up in a new or old state. Structural entropy is calculated in a similar fashion.

5.5 *Validation Techniques*

The standard method to validate MRB models is to synthesize a trace using the model. Then the synthesized trace and an actual trace can be run through a trace-driven cache simulator, to compare the cache performance results. The problem with this approach is that all of the types of localities would have to be incorporated into the model which could obscure the validation of the structural locality aspect of the model. Spirn demonstrated a validation approach which compared locality measurements of the synthesized trace with the results from the actual trace (Spi77). The measurements developed in this research suggest a similar approach with two different methods. The first method would take advantage of the entropy estimation aspect of the MRB model. This estimate could be compared with the actual entropy measured from the trace, although the estimation order N has to be taken into account. Since the additional states provide the MRB model with memory, M correlates with the order of structural entropy and L correlates with the order of entropy in the new reference runs. The state probability equations for the MRB model are complex but once they have been derived it is possible to write programs that will compute the error in entropy for models with a range of Ls and Ms. The second approach involves synthesizing a primitive form of distance string which would contain the structural locality characteristics of the trace being modeled. The measurements of the previous chapter are then performed on the trace and compared with the results from the actual trace. Since these measurements are solely concerned with same stack distances and zero stack distances, actual temporal distances do not have to be included in the trace.

5.6 *Synthesis Results*

Traces were synthesized from models with a variety of Ls and Ms. The traces that were modeled, shown in Table 5.1, represent the range of structural locality behaviors measured in Chapter 4. The chi-square test results are shown in Tables 5.2 through 5.5. The results demonstrated that

Table 5.1. Behavior Characteristics of Modelled Traces

Trace	Structural Locality Characteristics
Boyer	Short runs and a high structural entropy
IVEX	One of the ATUM traces which showed commonality in the SSD run distributions
QSIM	SSD Run distributions influenced by major changes in phase
Reducer	SSD Run distributions consistent throughout the trace and low structural entropy

Table 5.2. Boyer Synthesized Trace Chi-square Probabilities

		Instructions		Read		Write		Data		Overall	
M	L	New	SSD	New	SSD	New	SSD	New	SSD	New	SSD
0	0	0.647	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	4	0.849	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	9	0.862	0.000	0.000	0.000	0.000	0.761	0.000	0.000	0.000	0.000
0	15	0.862	0.000	0.000	0.000	0.000	0.761	0.000	0.000	0.000	0.000
2	4	0.772	0.000	0.000	0.000	0.865	0.000	0.000	0.000	0.000	0.000
2	15	0.440	0.000	0.000	0.000	0.866	0.587	0.000	0.000	0.000	0.000
4	4	0.879	0.000	0.563	0.000	0.620	0.000	0.788	0.000	0.694	0.000
4	15	0.537	0.000	0.778	0.000	0.621	0.589	0.841	0.000	0.679	0.000
15	15	0.433	0.000	0.572	0.000	0.621	0.589	0.961	0.000	0.572	0.000
15	19	0.433	0.000	0.601	0.410	0.621	0.589	0.963	0.015	0.572	0.000
15	24	0.433	0.000	0.601	0.410	0.621	0.589	0.963	0.015	0.572	0.000
19	19	0.238	0.000	0.601	0.410	0.621	0.589	0.907	0.012	0.572	0.000
19	24	0.238	0.016	0.601	0.410	0.621	0.589	0.907	0.012	0.572	0.000

the deciding factor was run length. The Explorer traces with the shorter runs had a more successful synthesis. The IVEX trace was also more successful because of the shorter runs. From this it becomes apparent that the variations in the distributions of run lengths require more states than the Ls or Ms tested here. In fact, successful synthesis may require that the models have enough states to encompass all of the run lengths of the actual distribution. A chi-square test was also run on traces synthesized from the combination of the traces which had some commonality in their run length distributions. These results showed a moderate success in synthesizing the distribution of new run lengths, but no success in synthesizing the distribution of SSD run lengths. This corroborates the results in the previous chapter which showed that these programs tend to be more similar in their new run length distributions than in their SSD run length distributions.

Figures 5.7 through 5.14 show the structural entropy of the synthesized traces. The results for all of the traces have the same trend. Those traces, such as the IVEX and Reducer instruction traces which have a low and steady entropy, had a synthesis which followed fairly closely. The rest

Table 5.3. IVEX Synthesized Trace Chi-square Probabilities

		Instructions		Read		Write		Data		Overall	
M	L	New	SSD	New	SSD	New	SSD	New	SSD	New	SSD
0	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.668	0.000	0.000
0	15	0.000	0.000	0.000	0.998	0.000	0.763	0.000	0.758	0.000	0.591
2	4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	15	0.000	0.000	0.000	0.995	0.000	0.679	0.000	0.772	0.000	0.721
4	4	0.000	0.000	0.000	0.000	0.008	0.000	0.000	0.000	0.000	0.000
4	15	0.000	0.000	0.000	0.995	0.015	0.704	0.000	0.796	0.000	0.710
15	15	0.042	0.000	0.924	0.997	0.673	0.702	0.897	0.790	0.002	0.653
15	19	0.037	0.000	0.937	0.695	0.684	0.587	0.894	0.870	0.002	0.590
15	24	0.007	0.000	0.936	0.616	0.682	0.521	0.898	0.874	0.003	0.565
19	19	0.110	0.000	0.979	0.693	0.935	0.531	0.994	0.874	0.003	0.641
19	24	0.067	0.000	0.979	0.613	0.933	0.461	0.995	0.878	0.006	0.617

Table 5.4. QSIM Synthesized Trace Chi-square Probabilities

		Instructions		Read		Write		Data		Overall	
M	L	New	SSD	New	SSD	New	SSD	New	SSD	New	SSD
0	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	4	0.005	0.000	0.000	0.000	0.000	0.032	0.000	0.000	0.000	0.000
0	9	0.001	0.000	0.000	0.000	0.000	0.478	0.000	0.000	0.000	0.000
0	15	0.001	0.000	0.000	0.000	0.000	0.467	0.000	0.000	0.000	0.000
2	4	0.004	0.000	0.000	0.000	0.000	0.007	0.000	0.000	0.000	0.000
2	15	0.001	0.000	0.000	0.000	0.000	0.293	0.000	0.000	0.000	0.000
4	4	0.001	0.000	0.000	0.000	0.000	0.021	0.000	0.000	0.000	0.000
4	15	0.000	0.000	0.000	0.000	0.000	0.410	0.000	0.000	0.000	0.000
15	15	0.917	0.000	0.890	0.000	0.918	0.422	0.966	0.000	0.000	0.000
15	19	0.982	0.000	0.717	0.000	0.919	0.445	0.966	0.000	0.000	0.000
15	24	0.972	0.000	0.725	0.000	0.924	0.528	0.951	0.000	0.000	0.000
19	19	0.993	0.000	0.167	0.000	0.861	0.358	0.957	0.000	0.000	0.000
19	24	0.978	0.000	0.170	0.000	0.867	0.437	0.952	0.000	0.000	0.000

Table 5.5. Reducer Synthesized Trace Chi-square Probabilities

		Instructions		Read		Write		Data		Overall	
M	L	New	SSD	New	SSD	New	SSD	New	SSD	New	SSD
0	0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0	15	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	4	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2	15	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	4	0.001	0.000	0.078	0.000	0.000	0 000	0.000	0.000	0.000	0.000
4	15	0.593	0.000	0.029	0.000	0.000	0.000	0.000	0.000	0.000	0.000
15	15	0.953	0.000	0.995	0.000	0.000	0.000	0.000	0.000	0.000	0.000
15	19	0.981	0.078	0.996	0.000	0.000	0.000	0.000	0.000	0.000	0.000
15	24	0.956	0.178	0.999	0.000	0.000	0.000	0.000	0.000	0.000	0.000
19	19	0.996	0.089	0.996	0.000	0.000	0.000	0.000	0.000	0.000	0.000
19	24	0.999	0.222	0.999	0.000	0.000	0.000	0.000	0.000	0.000	0.000

of the traces, which had high entropy in the lower orders, showed the synthesis deviating from the actual trace anywhere from the 2nd to 5th order. Models with more than 16 SSD states did not show any significant improvement. This suggests that the predictiveness measured with structural entropy may come from a more complex type of referencing than the SSD access assumed for this research and that after a threshold of about 16 states, any additional states will not improve the model. It is interesting to note that most of the synthesized traces experienced a drop in entropy at the higher orders, indicating that the model captures some high order type of predictiveness in addition to the SSD runs.

In summary, the entropy results were the opposite of the chi-square test results. The chi-square test favored those traces with shorter runs which the models could encompass. The entropy results favored traces with low structural entropy resulting from long runs. The results strengthen the case made in Chapter 4 that structural locality is difficult to model using only the SSD access. The next chapter will put all of these results in perspective.

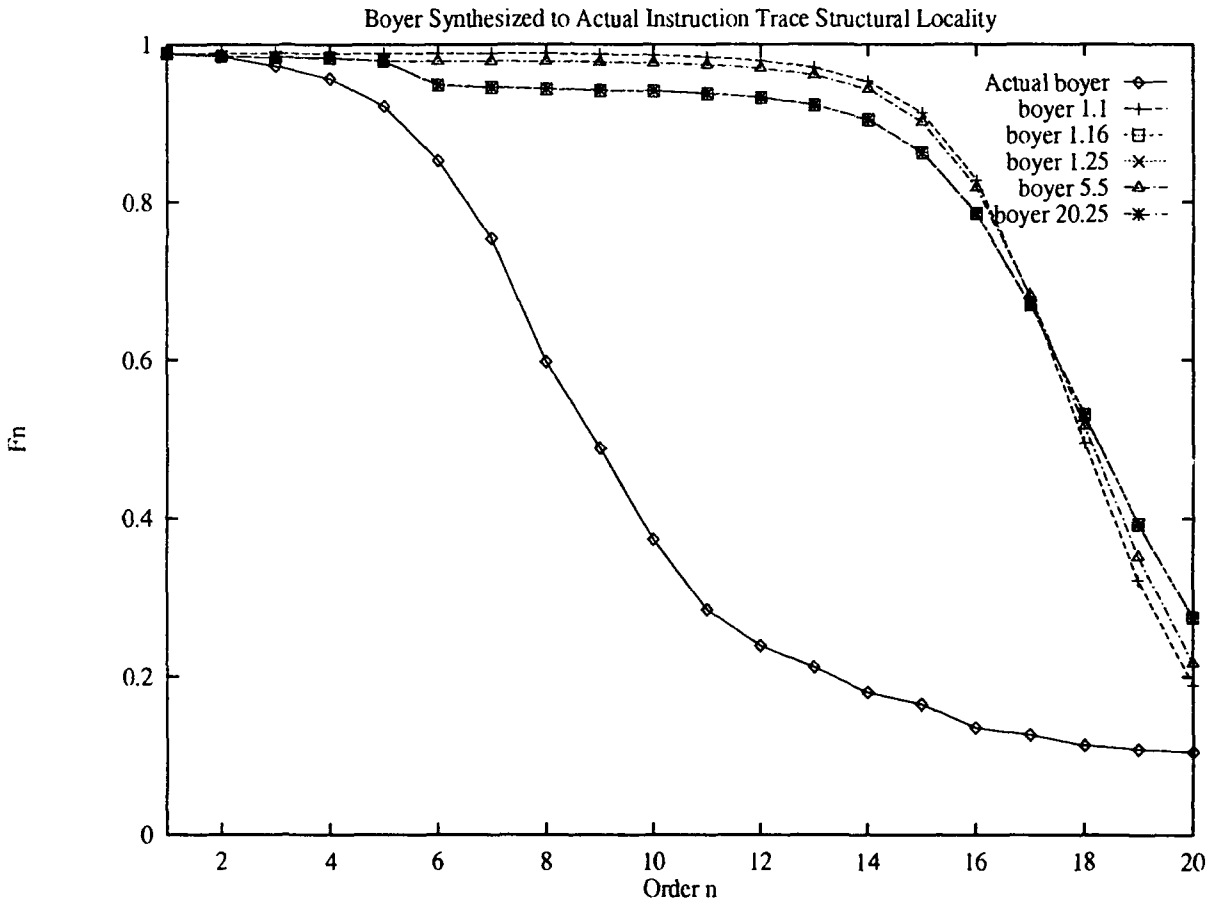


Figure 5.7. Boyer Synthesized Instruction Traces Structural Entropy

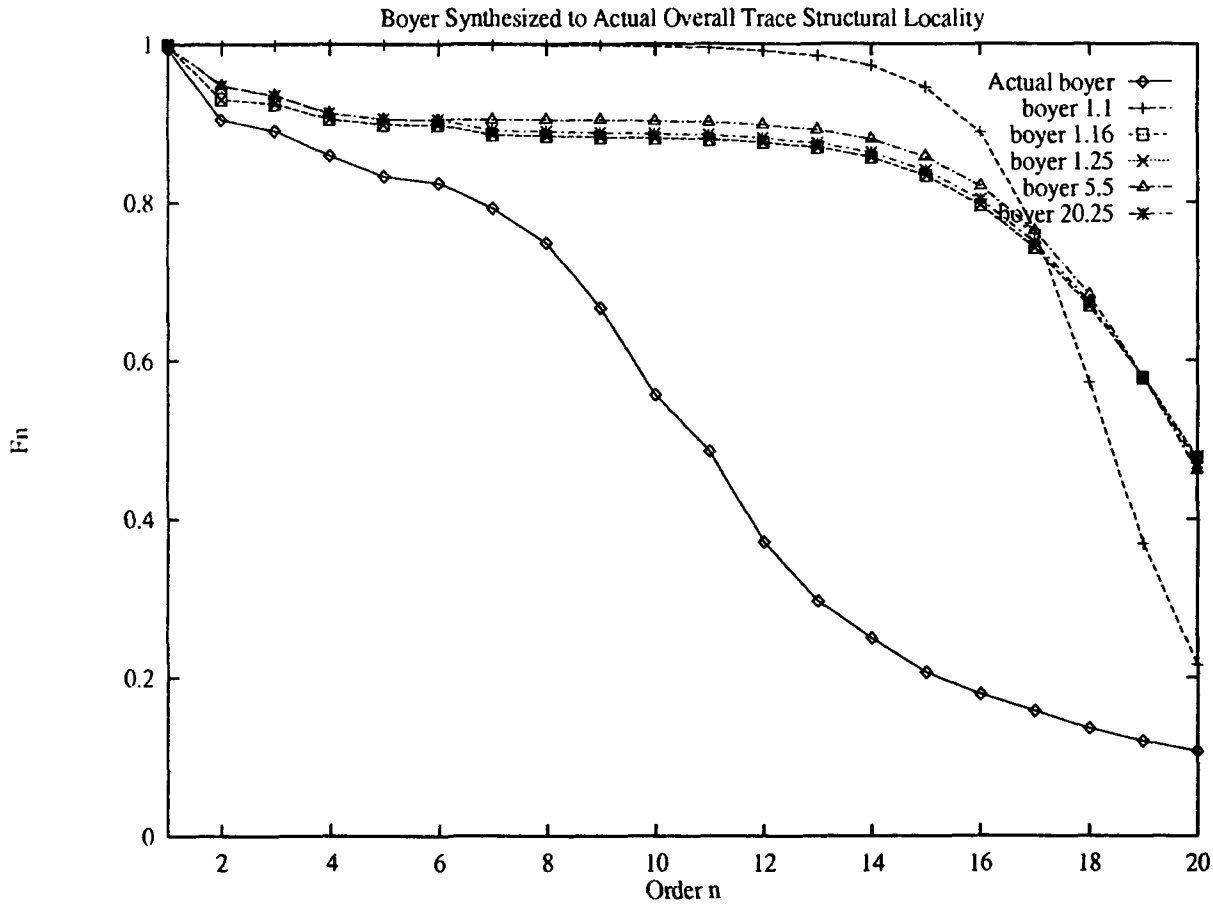


Figure 5.8. Boyer Synthesized Overall Traces Structural Entropy

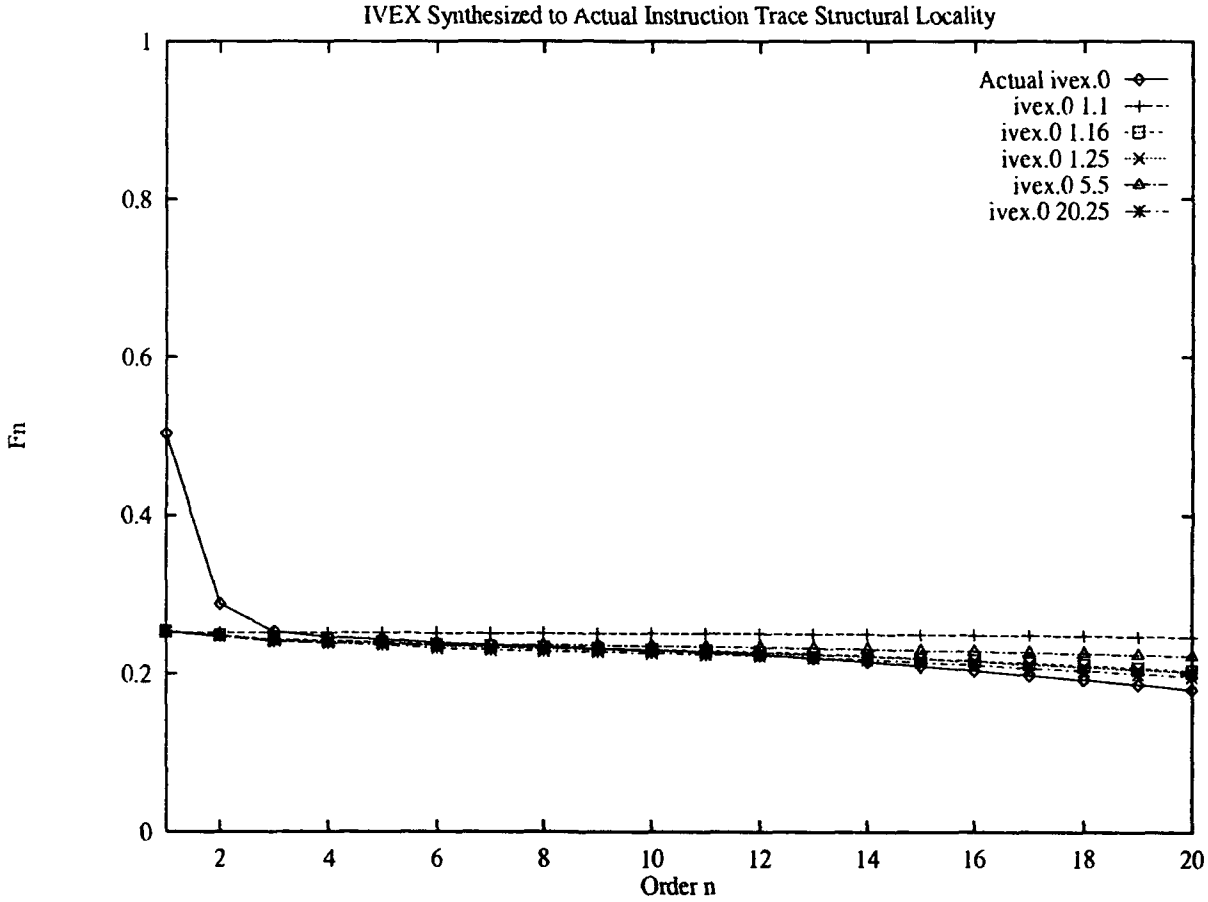


Figure 5.9. IVEX Synthesized Instruction Traces Structural Entropy

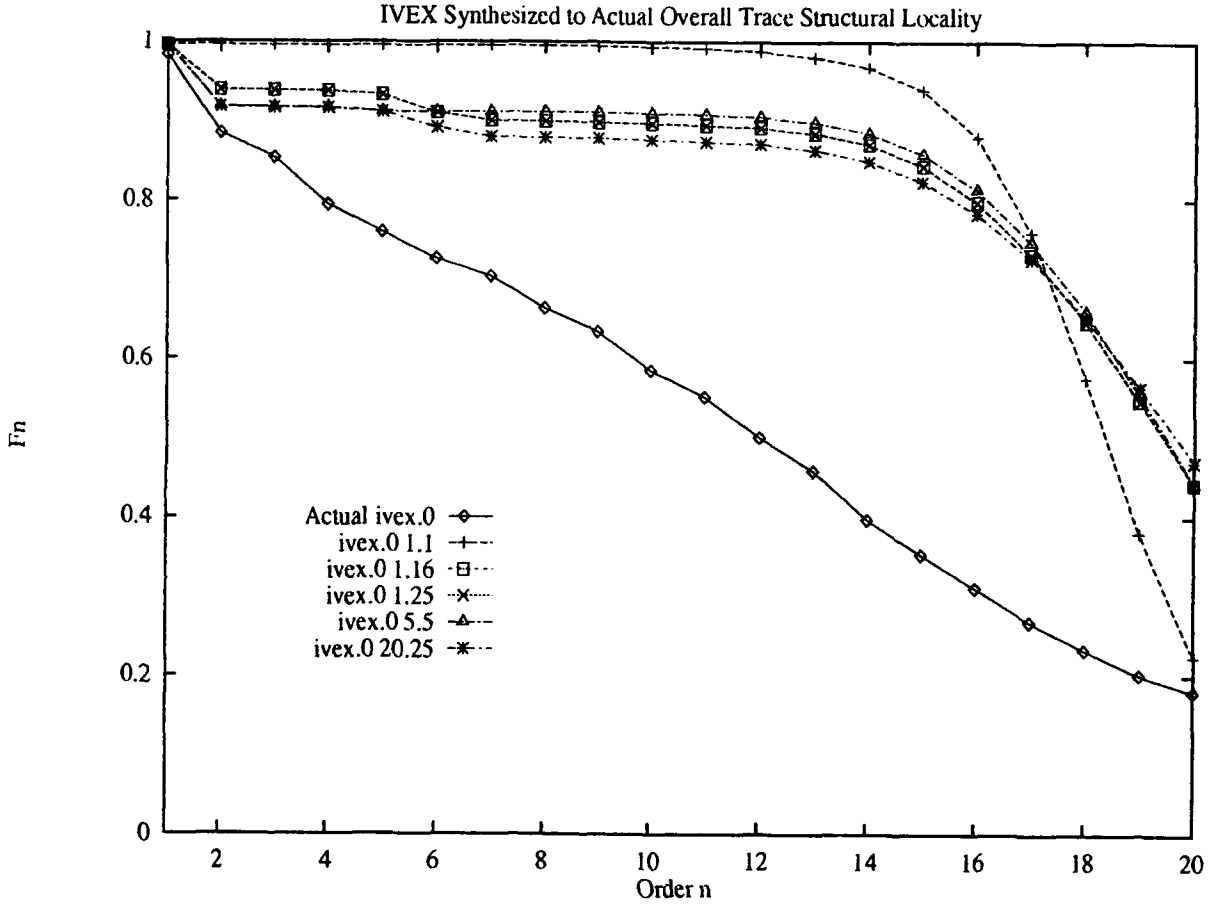


Figure 5.10. IVEX Synthesized Overall Traces Structural Entropy

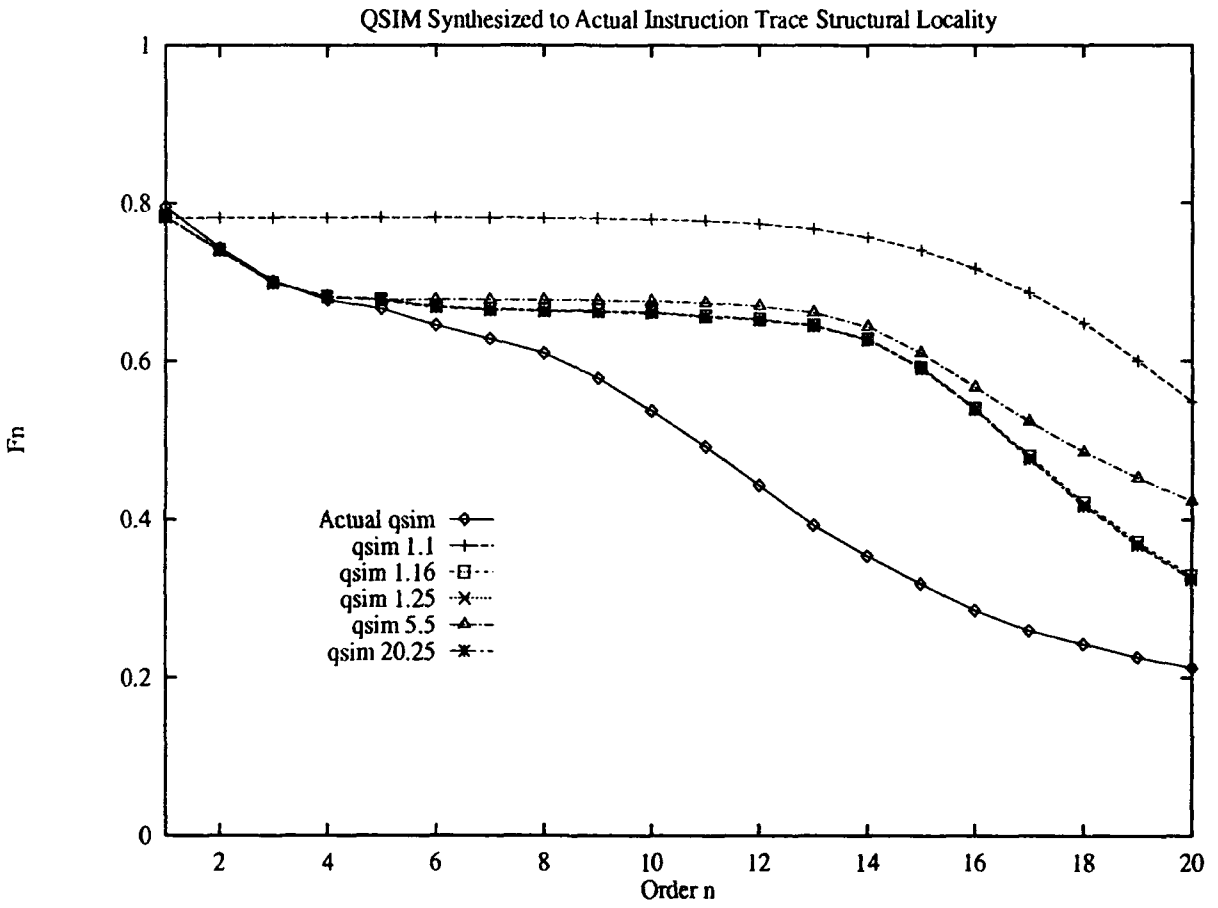


Figure 5.11. QSIM Synthesized Instruction Traces Structural Entropy

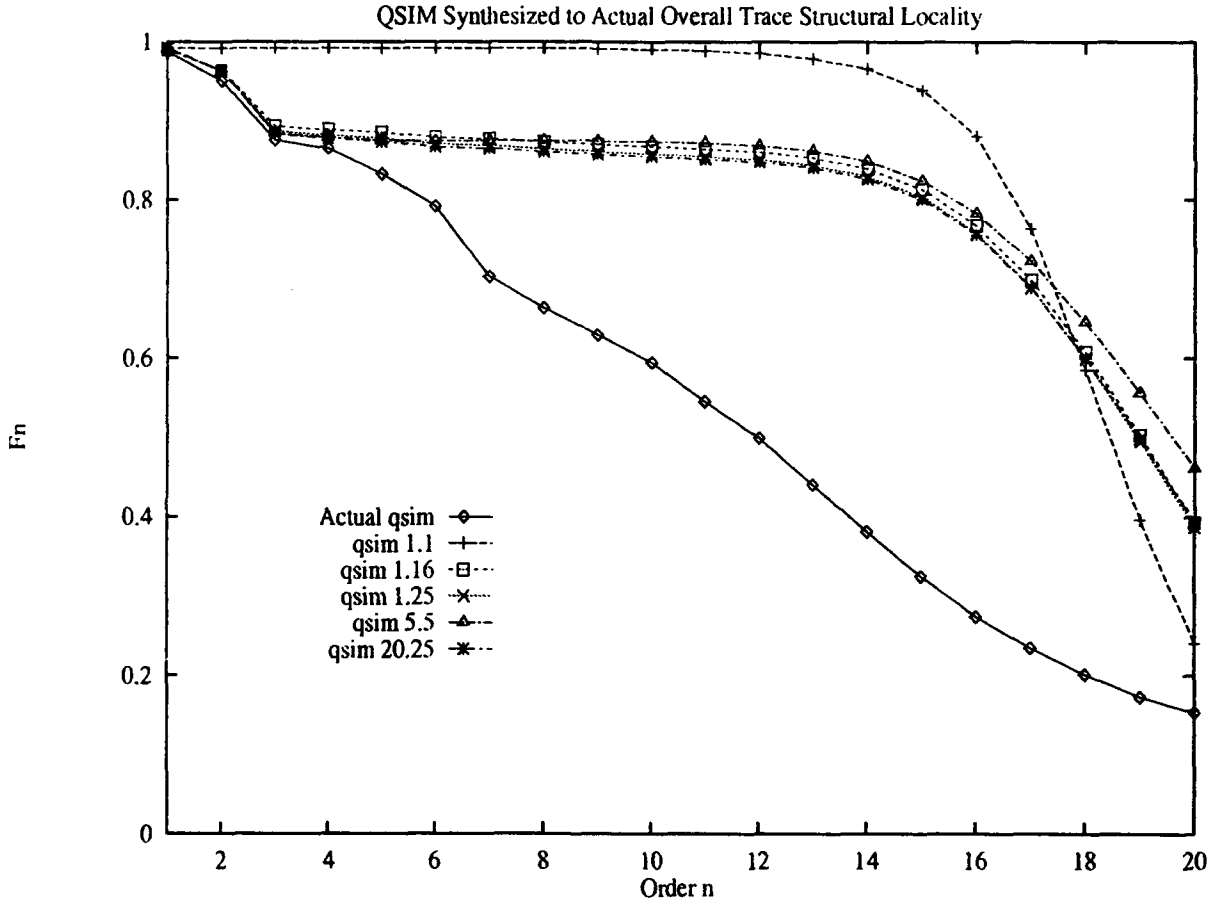


Figure 5.12. QSIM Synthesized Overall Traces Structural Entropy

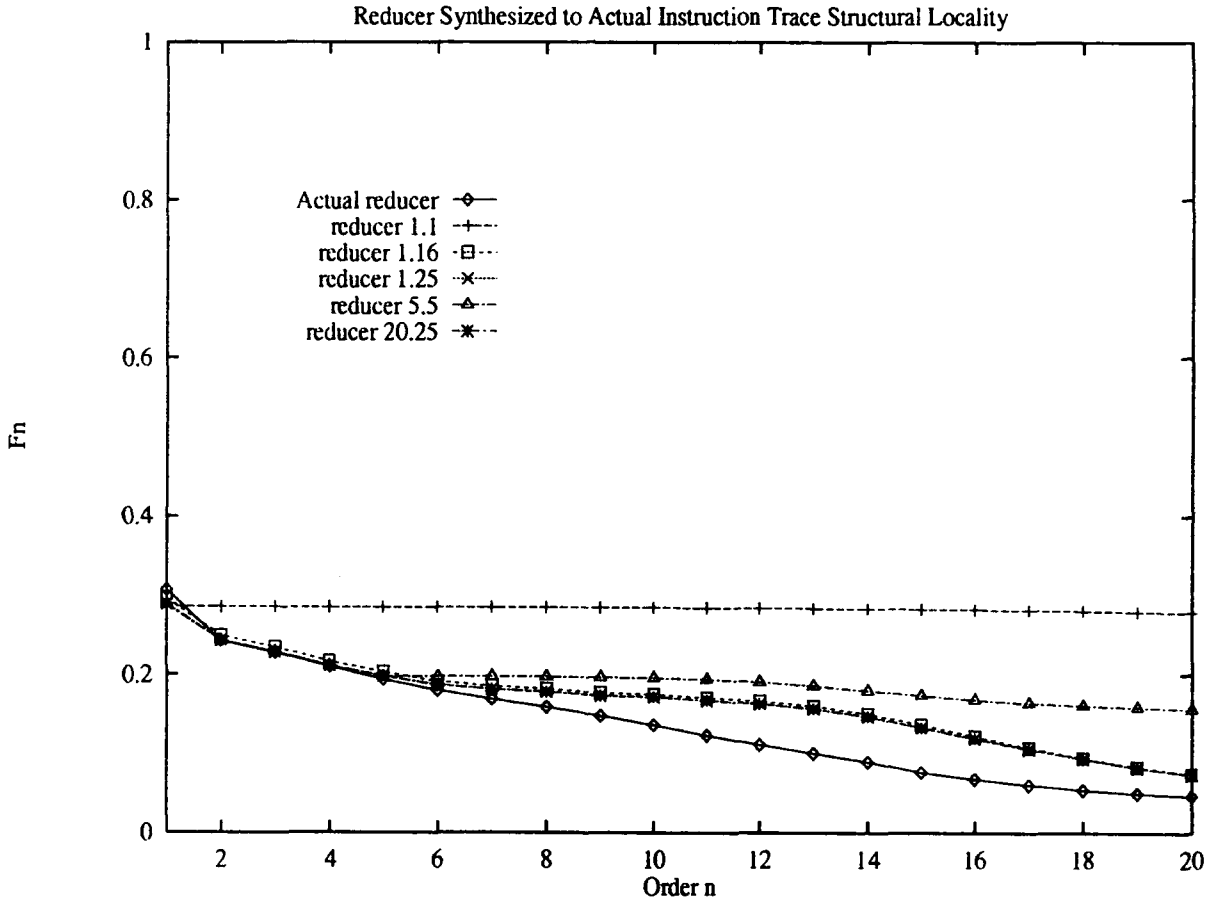


Figure 5.13. Reducer Synthesized Instruction Traces Structural Entropy

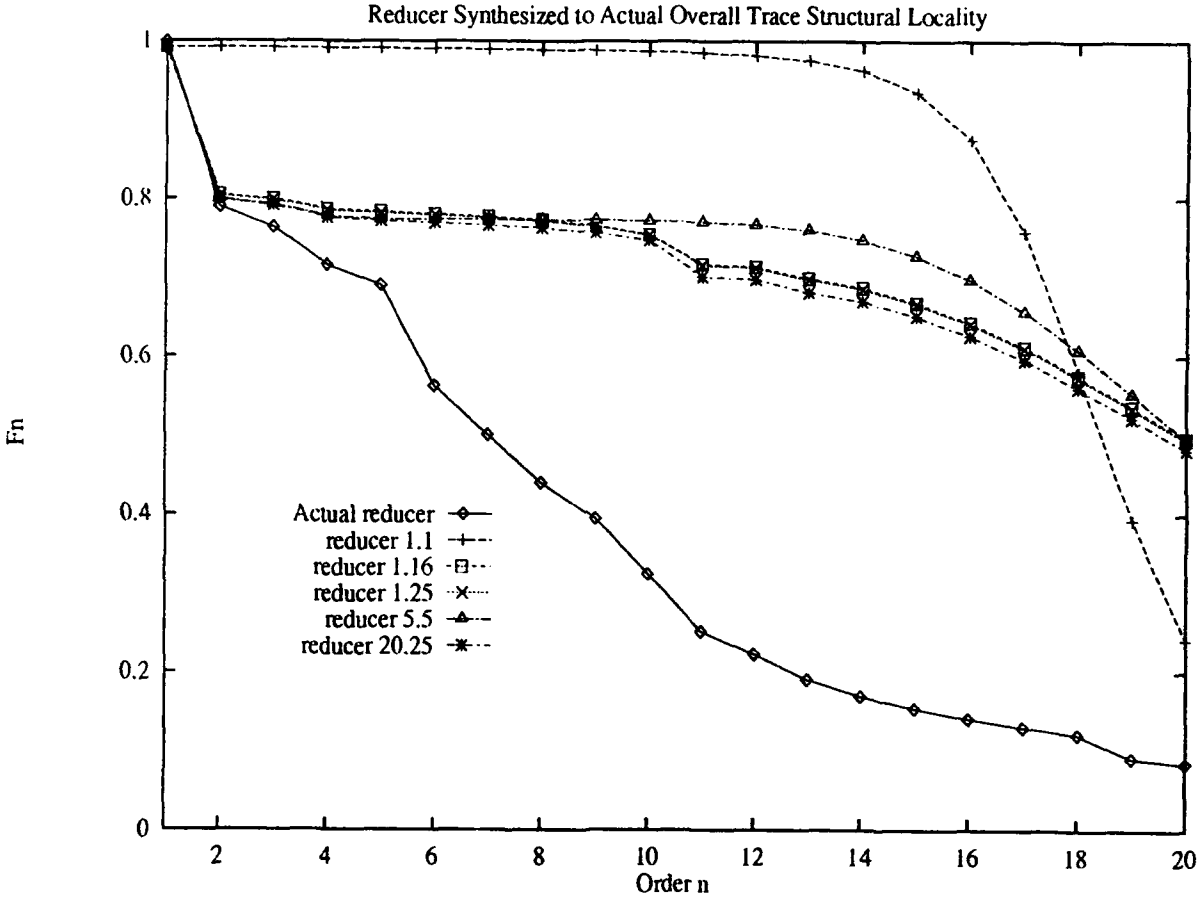


Figure 5.14. Reducer Synthesized Overall Traces Structural Entropy

VI. Conclusions

6.1 The Findings of this Research

The measurements in this research have quantified and characterized the structural locality first suggested by Thazhuthaveetil and Pleskun. They have also expanded Hobart's as well as Agarwal, Horowitz, and Hennessy's findings about the nature of structural locality and first-time referencing of memory addresses. Basically, the measurements indicate that the distribution of SSD run lengths is strongly influenced by program design as well as phases and transitions in a program's execution. The distribution of new reference run lengths is not influenced as much. This greatly complicates modeling of SSD run lengths with a Markov chain, although trace synthesis using the model did show some promise if the model had enough states to encompass all of the SSD run lengths. The ATUM set contained a number of traces that showed some commonality in their SSD and new run length distributions. A trace synthesis of the combination of these traces showed some success in reproducing the distribution of new run lengths but no success in reproducing the distribution of SSD run lengths. The structural entropy curves showed that structural locality was not very predictive until the 16th order, although some of the traces with long SSD runs were more predictive. Trace synthesis using the models was unable to reproduce the structural entropy curves beyond the 5th order showing that this predictiveness was not solely due to the SSD type of access. The state entropy curves had lower entropy and less variation in entropy among the traces showing that the first-time referencing of memory locations is more predictive than structural locality.

6.2 Usefulness of this Research

The techniques used in this research can be applied to classify the traces used in trace-driven simulations so that variations in cache performance, that are caused by the program specific characteristics of the traces, can be identified. This could then lead to the design of memory systems which can be tailored to a particular class of programs. The measurements also show the

amount of history needed to optimally model the stack distance. This knowledge could be used to design dynamic structural locality caches (SLCs) such as the one suggested in (Hob89) which would prefetch a data structure out of the LRU cache. The size of the prefetch block would be determined by the hit rate of the SLC which is a good mechanism to predict the presence of a run.

6.3 Suggestions for Further Research

Follow on research should develop an MRB model to characterize the structural locality quantified in this research. A possible is model shown in Figure 6.1. In this model each state represents a group of run lengths. The new transition probabilities P_{RSO_L} and P_{RSN_L} are added to the model to show the next reference staying within the run group. This exploits the commonality observed during the chi-square test when the run lengths were grouped. It also allows the model to encompass all of the run lengths in a distribution. In addition, this research could be continued in two areas.

6.3.1 Adding Spatial and Temporal Locality to the MRB Model This research has only been concerned with structural locality. In order to have a complete memory referencing behavior model, the relationship between structural, spatial, and temporal locality has to be determined. Specifically, the interdependence of the temporal distance of the reference after a run and the length of the run should be determined. Additionally, a study of the relationship between the spatial locality window found by Hobart and the length of the new runs would help to further characterize memory referencing behavior. Characterizing the relationship between temporal and structural locality will improve the MRB model by providing a way to predict new states and the end of runs. Characterizing the relationship between spatial and structural locality will help determine if any relationship exists between new reference runs and SSD runs.

6.3.2 Tying the Structural Locality Measurements to Cache Performance and Program Structure Additional research efforts should attempt to correlate program performance characteristics

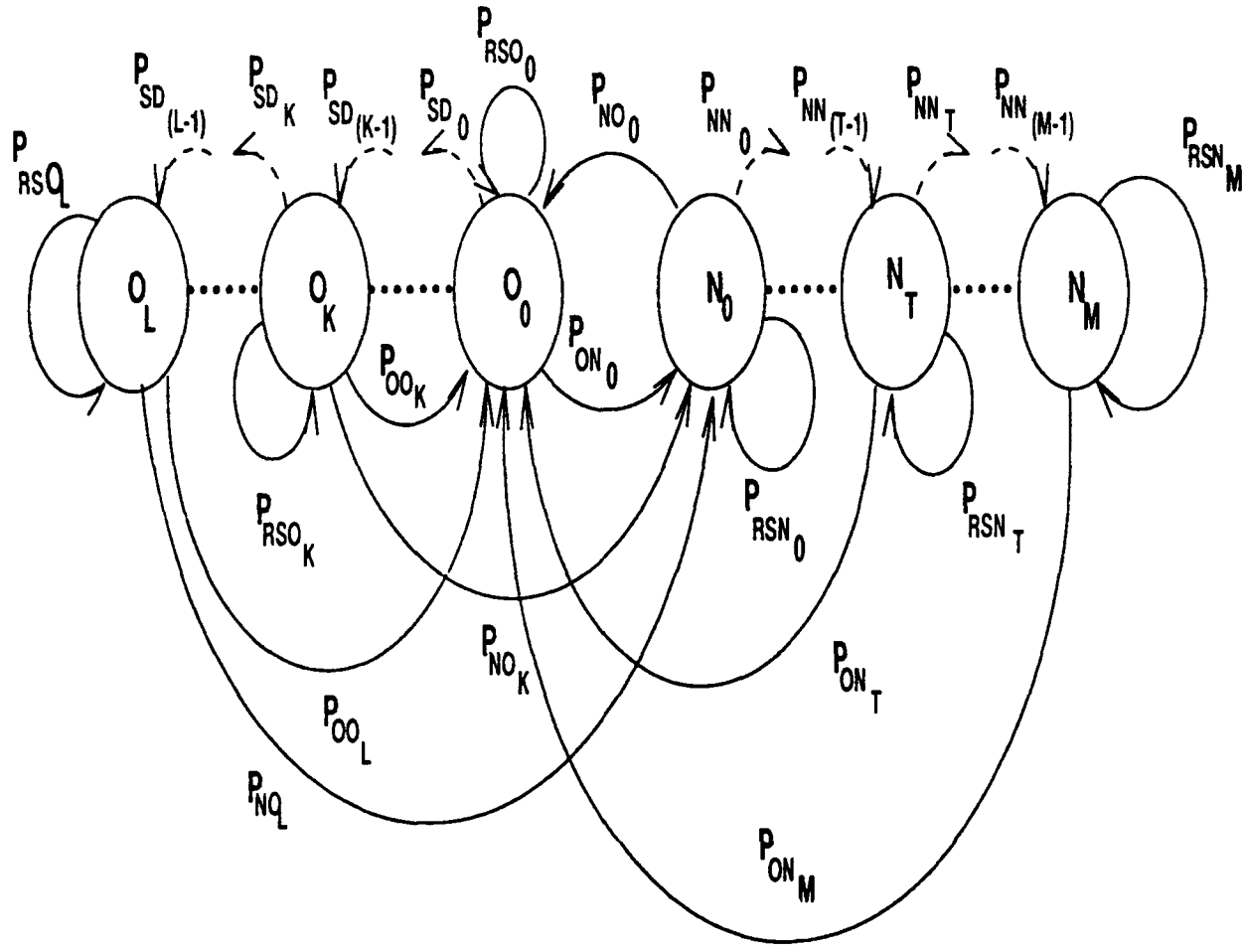


Figure 6.1. MRB Model with States Representing Run Length Groups

such as complexity metrics and cache performance statistics to the locality characteristics measured in this thesis. Running traces which display differing amounts of structural locality, through a trace simulator would be a straightforward way of relating structural locality to cache performance. Program complexity could also be related to structural locality using metrics. Masuda and Fin's technique of trace marking by the compiler described in section 2.3.6 suggests an interesting avenue. Benchmarks which contain pure versions of different program and data structures could be traced. The traces could be marked by the compiler to indicate which references came from what structure. The structural locality can then be directly correlated with the structures in the benchmarks, providing a complete picture of the relationship between memory referencing behavior

and program structure. This will also determine whether a relationship exists between program complexity and structural locality.

6.4 Conclusions

This research further refined the MRB model to better capture the nature of structural locality by modeling the inherent locality in memory referencing behavior. However, the results of this research have shown that many structural locality characteristics in memory referencing behavior vary across programs and even between phase of a programs execution. This suggests that to better exploit the localities of memory references, memory systems should be designed in such a way that they dynamically characterize and adapt the particular locality characteristics present during a programs's execution phase.

Appendix A. *Sample Calculations*

A.1 Introduction

The following is a sample of the calculations used to perform structural locality measurements on a temporal string.

A.2 The Temporal String

Table A.1 shows the temporal string used for the sample calculations. It also shows the symbolic representations of the string used for the entropy calculations. Note that each symbol in the structural entropy string represents the relationship between the current reference and the previous reference.

A.3 Transitional Probability Calculations

The transitional probabilities are calculated using equations A.2.

$$P_{SSD_l} = \frac{O_{L_l}}{O_{L_l} + O_{ZT_l} + O_{NT_l}} \quad \text{(A.1)}$$

$$P_{ON_l} = \frac{O_{ZT_l}}{O_{L_l} + O_{ZT_l} + O_{NT_l}} \quad \text{(A.2)}$$

$$P_{OO_l} = 1 - P_{SSD_l} - P_{ON_l} \quad \text{(A.3)}$$

$$P_{NN_l} = \frac{O_{NN_l}}{O_{NN_l} + O_{OT_l}} \quad \text{(A.4)}$$

$$P_{NO_l} = 1 - P_{NN_l} \quad \text{(A.5)}$$

(A.6)

O_{L_l} Number of SSD runs longer than length l.

O_{ZT_l} Number of SSD runs of length l with a new termination.

Table A.1. Sample Temporal String

Index	LRU Stack Distance	State Entropy Symbol	Structural Entropy Symbol
1	0	0	
2	0	0	1
3	0	0	1
4	0	0	1
5	0	0	1
6	3	1	0
7	0	0	0
8	0	0	1
9	0	0	1
10	2	1	0
11	2	1	1
12	4	1	0
13	5	1	0
14	6	1	0
15	0	0	0
16	0	0	1
17	1	1	0
18	1	1	1
19	0	0	0
20	1	1	0
21	9	1	0
22	0	0	0
23	0	0	1
24	1	1	0
25	0	0	0
26	0	0	1
27	0	0	1
28	2	1	0
29	2	1	1
30	2	1	1
31	2	1	1
32	2	1	1
33	4	1	0
34	5	1	0
35	6	1	0
36	6	1	1
37	12	1	0
38	12	1	1
39	13	1	0
40	15	1	0
41	15	1	1
42	15	1	1
43	15	1	1
44	15	1	1
45	15	1	1

Table A.2. Run Distributions and Transitional Probabilities

Run Length	O_{OT_l}	P_{NN_l}	O_{ZT_l}	O_{NT_l}	P_{SSD_l}	P_{OC}
1	1	0.833	4	6	0.375	0.375
2	2	0.600	1	3	0.333	0.500
3	2	0.333	0	0	1.000	0.000
4	0	1.000	0	0	1.000	0.000
5	1	0.000	0	1	0.500	0.500
6	1	0.000	0	1	0.000	1.000

Table A.3. Structural Entropy NGram Distributions

NGram	3rd Order	2nd Order	1st Order
0	5	11	21
1	6	10	23
2	6	10	
3	4	12	
4	6		
5	4		
6	4		
7	7		
Total	42	43	44

O_{NT_l} Number of SS runs of length l with an old termination.

O_{NN_l} Number of new runs longer than length l.

O_{OT_l} Number of new runs of length l.

The probabilities for the sample string are shown in Table A.2.

A.4 Sample Entropy Calculation

Table A.3 shows the 1grams, 2grams, and 3grams that were tabulated from the structural entropy sample string in Table A.1. The NGrams are labeled with the rightmost bit the least significant.

Table A.4 shows the calculations for the sample string. The probability summation is the first term of equation A.7. The second term is the probability summation of the previous order. The state entropy calculations use the same algorithm with a different symbol set.

Table A.4. Structural Entropy Calculations

Order	Probability Summation	Entropy
1	-0.9985	0.9985
2	-1.9958	0.9972
3	-2.9687	0.9273

$$F_N = -\sum_{i,j} p(b_i, j) \log_2 p(b_i, j) + \sum_i p(b_i) \log_2 p(b_i) \qquad \text{(A.7)}$$

N is the order of entropy and the size of the block of symbols.

b_i is a block of $N - 1$ symbols.

j is an arbitrary symbol following b_i.

$p(b_i, j)$ is the probability of the occurrence of N-gram b_i, j.

Appendix B. *Explorer Results*

B.1 Introduction

This is the supplementary data for Chapter 4. The graphs are the structural locality measurments performed on the Explorer set of traces.

Figure B.1. State Entropy of the Explorer Instruction Traces

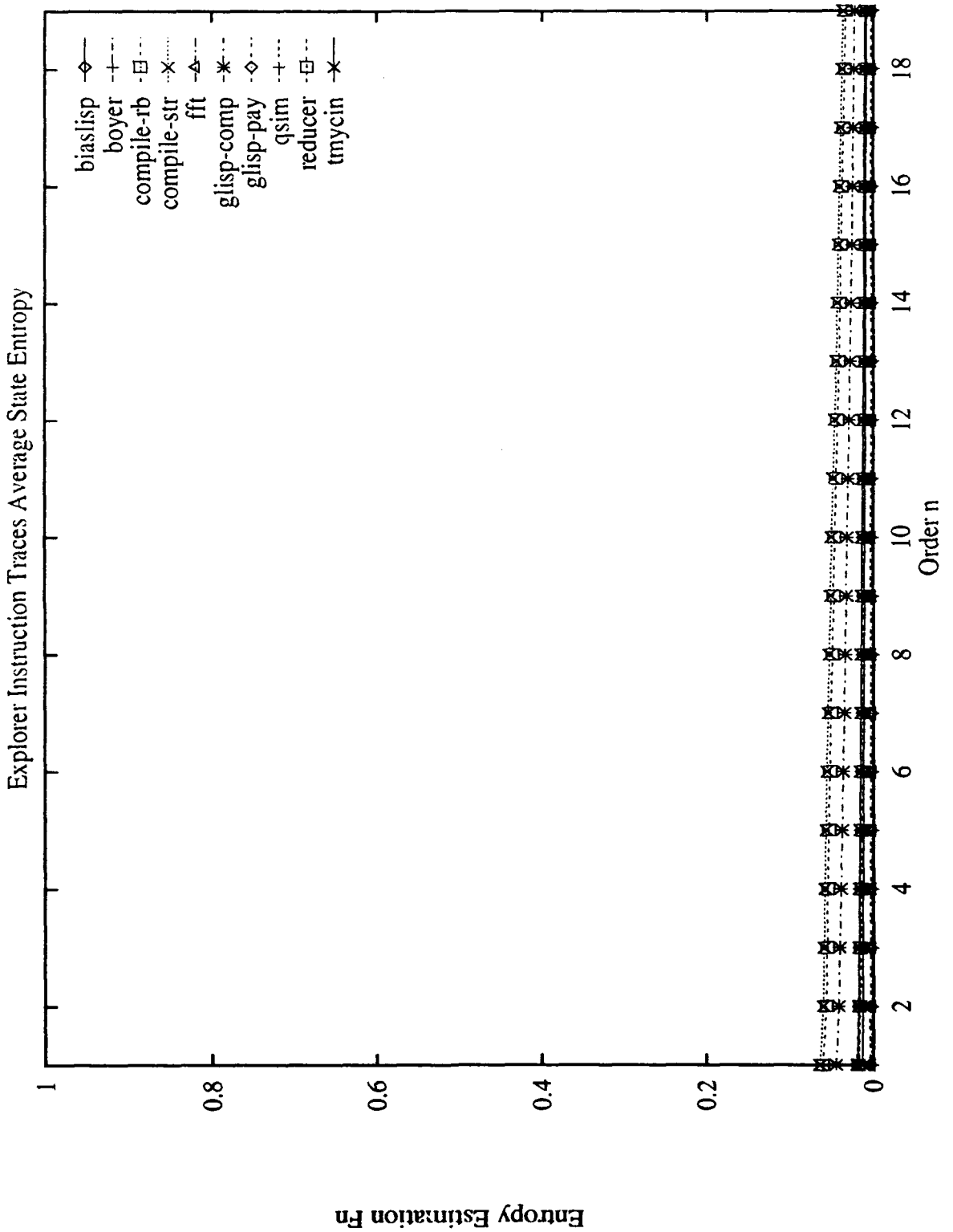


Figure B.2. State Entropy of the Explorer Read Traces

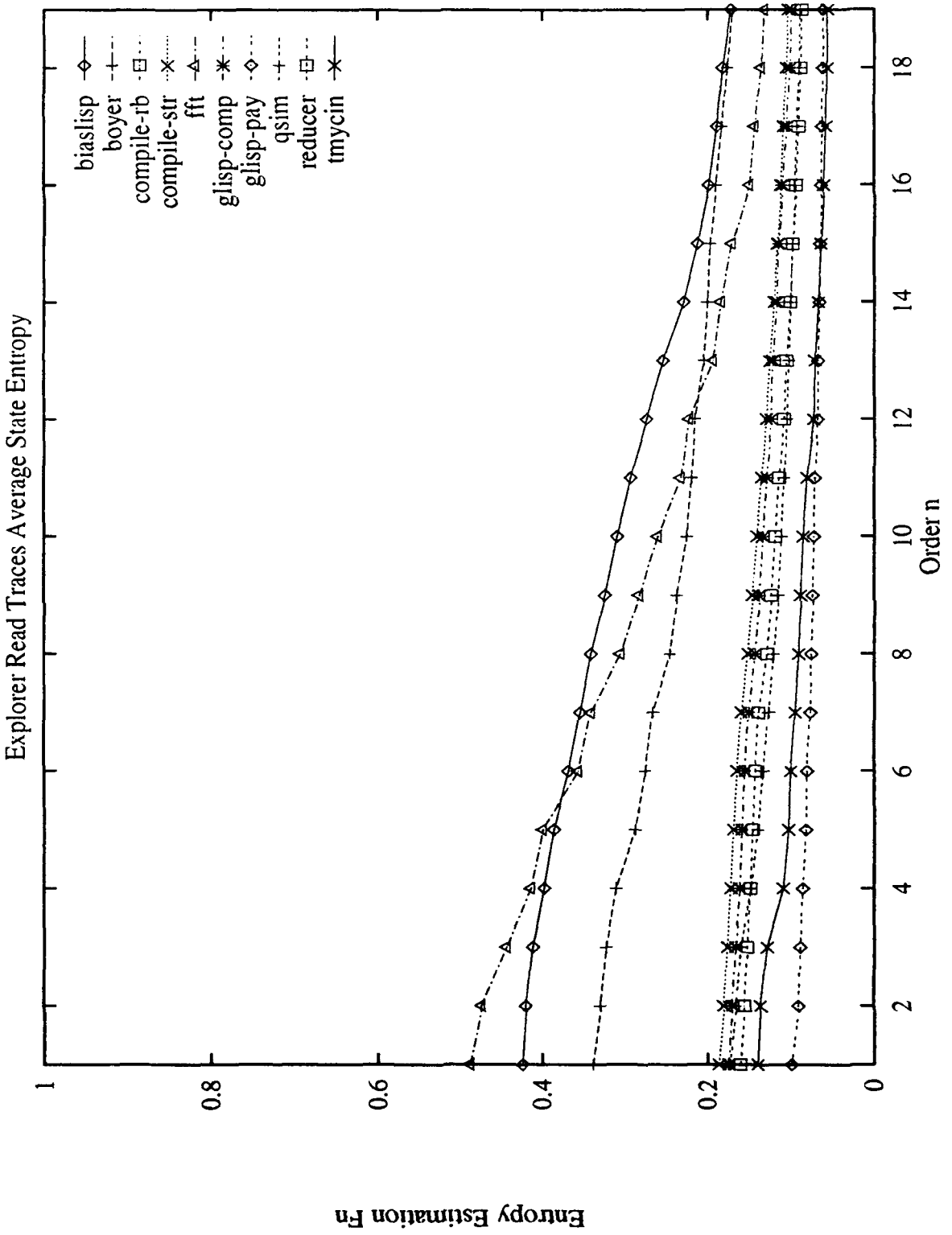


Figure B.3. State Entropy of the Explorer Write Traces

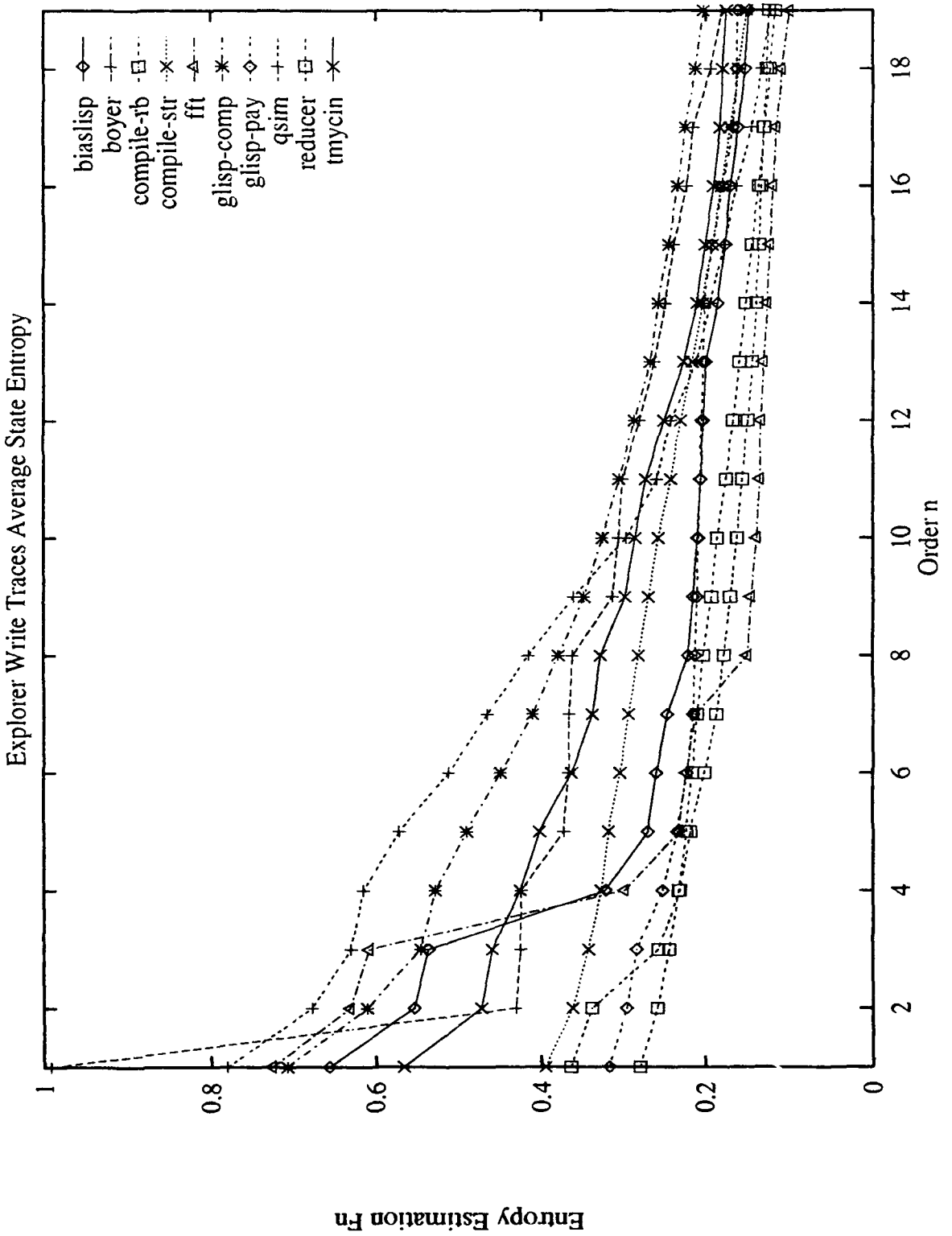


Figure B.4. State Entropy of the Explorer Data Traces

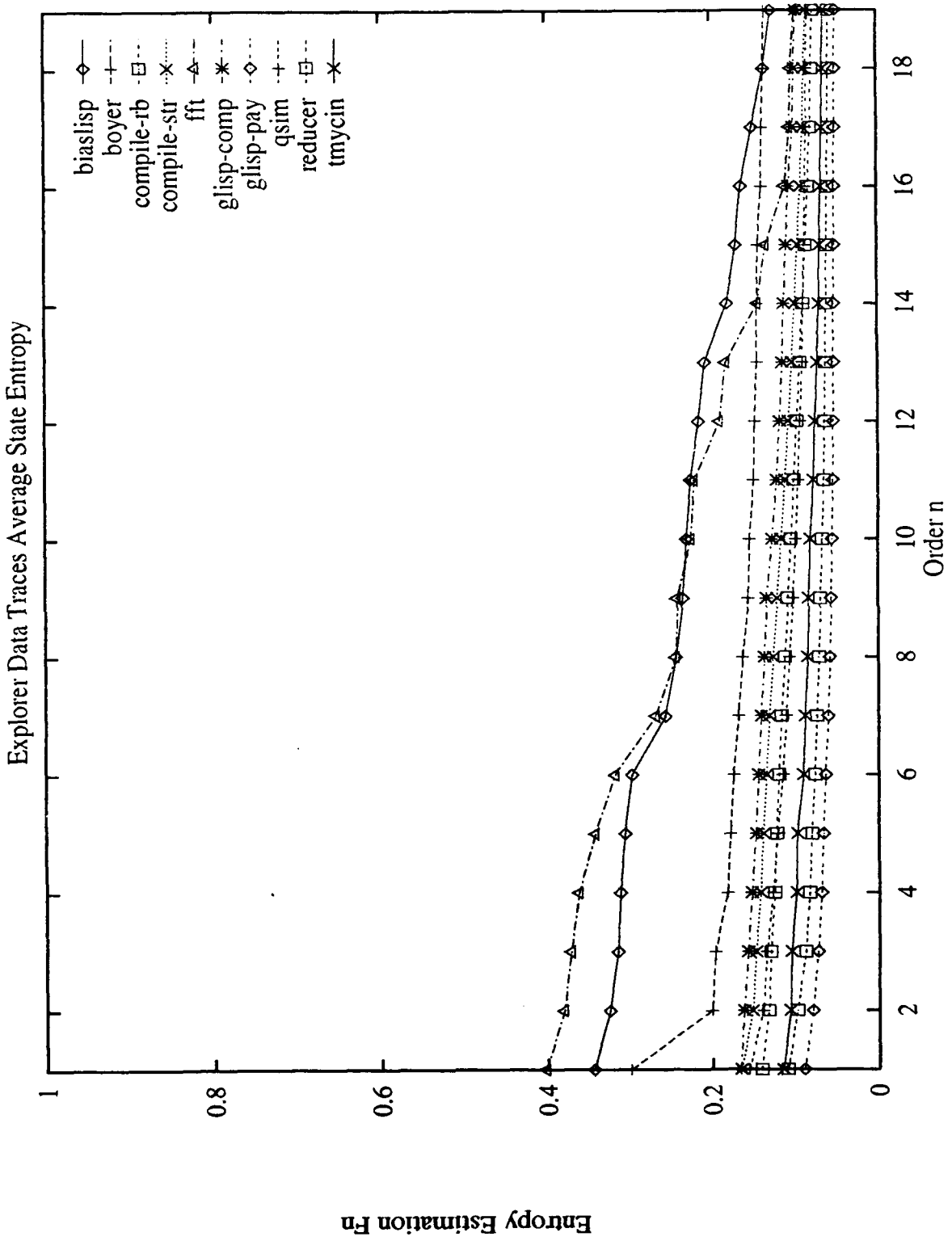


Figure B.5. State Entropy of the Explorer Overall Traces

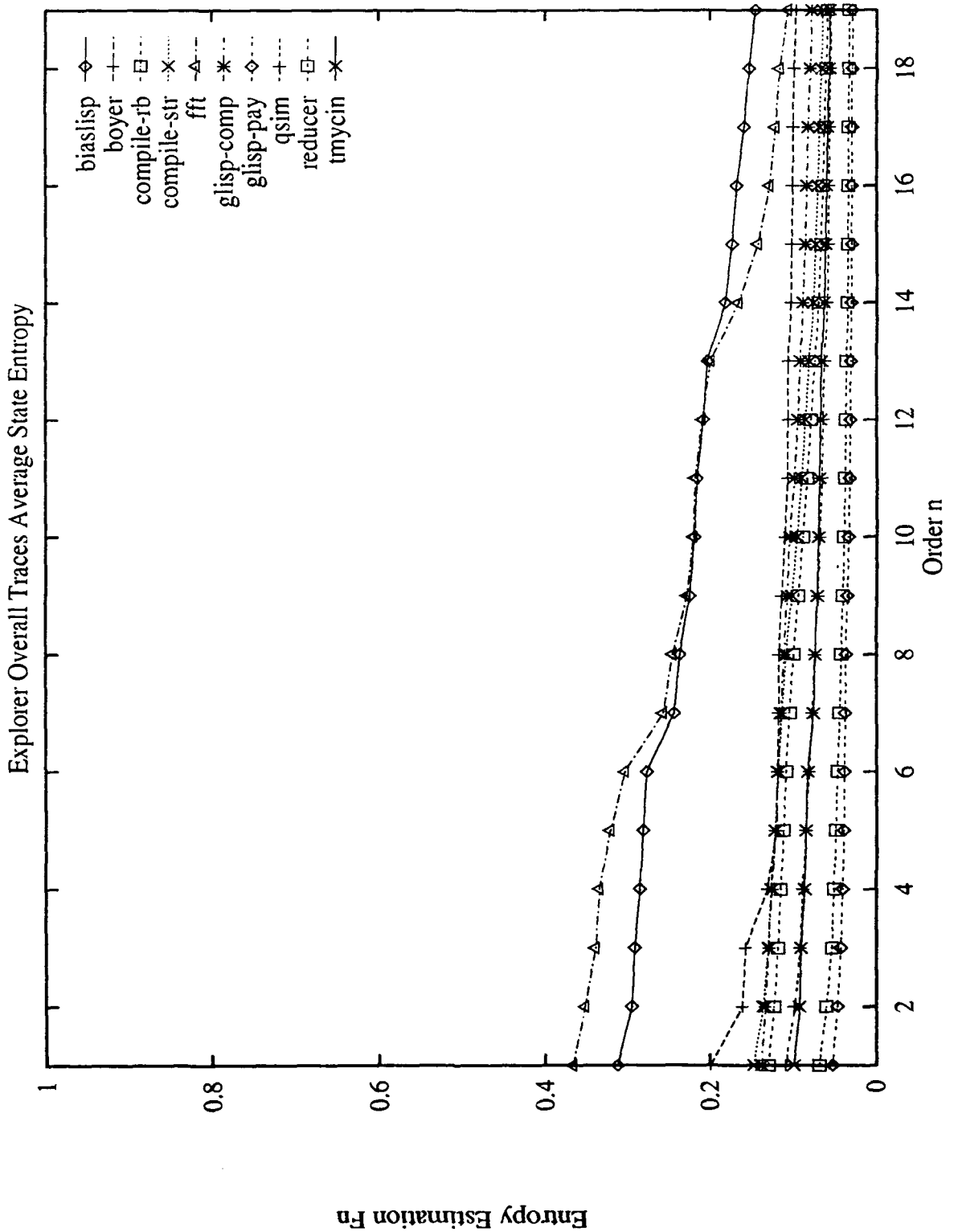


Figure B.6. Structural Entropy of the Explorer Instruction Traces

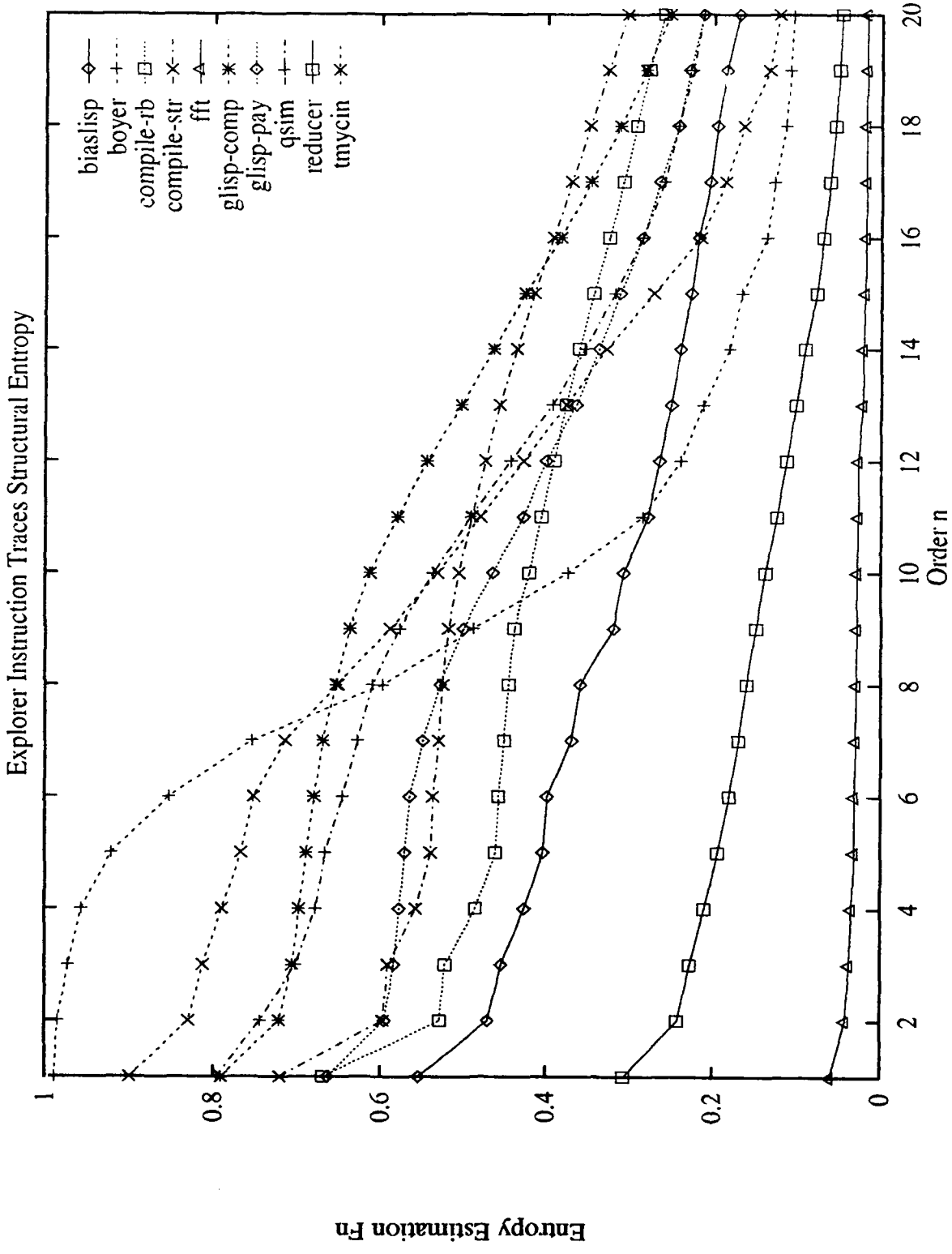


Figure B.7. Structural Entropy of the Explorer Read Traces

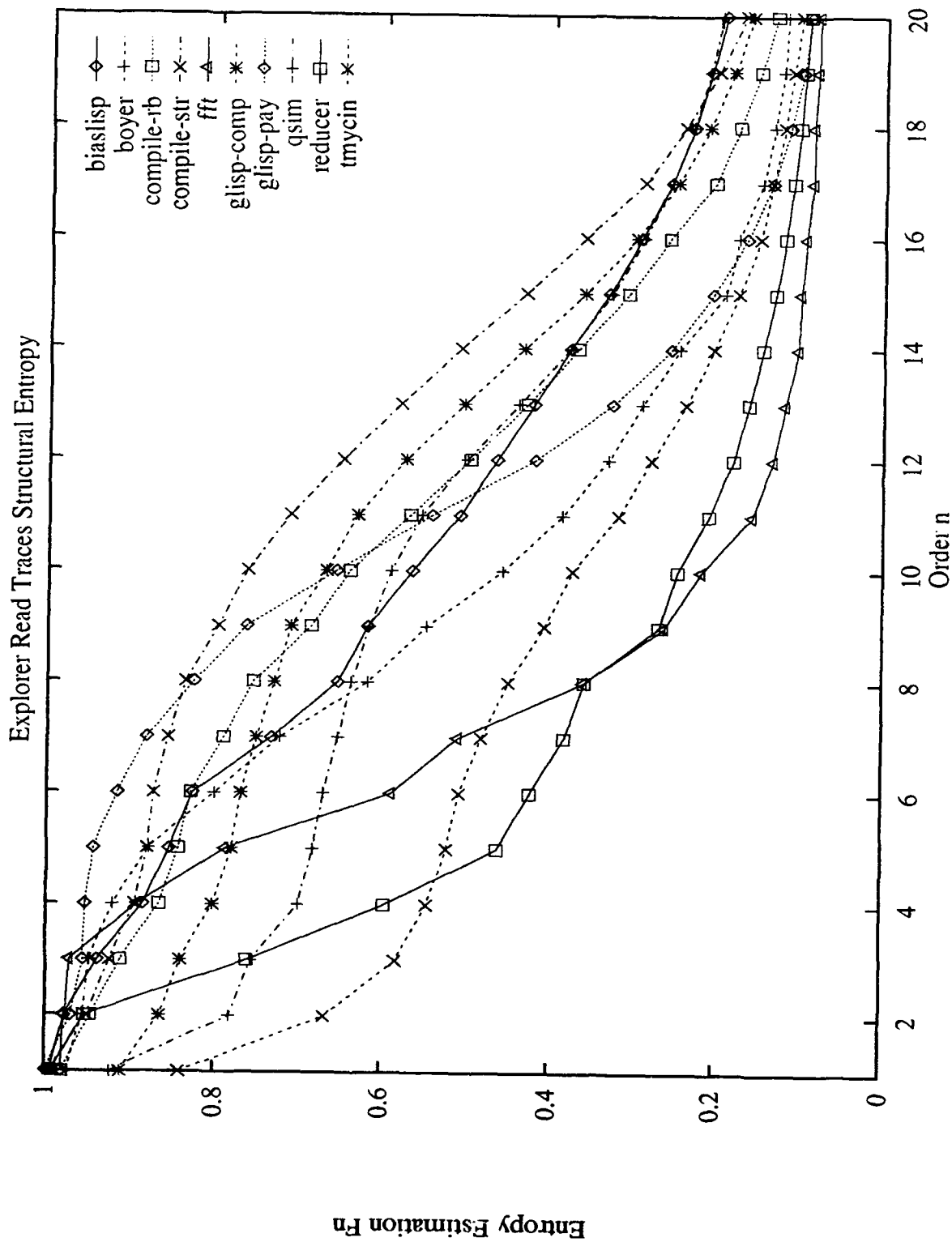


Figure B.8. Structural Entropy of the Explorer Write Traces

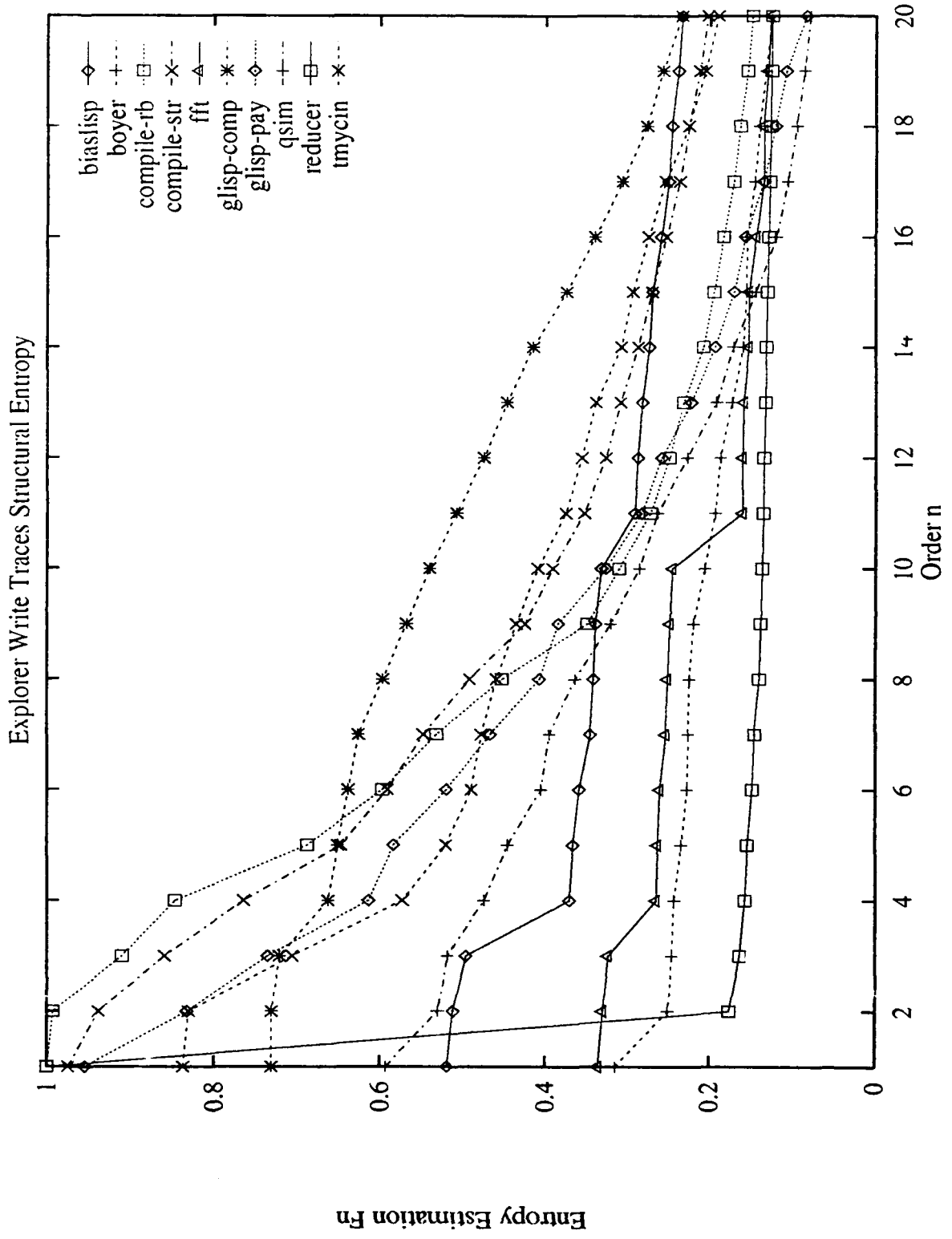


Figure B.9. Structural Entropy of the Explorer Data Traces

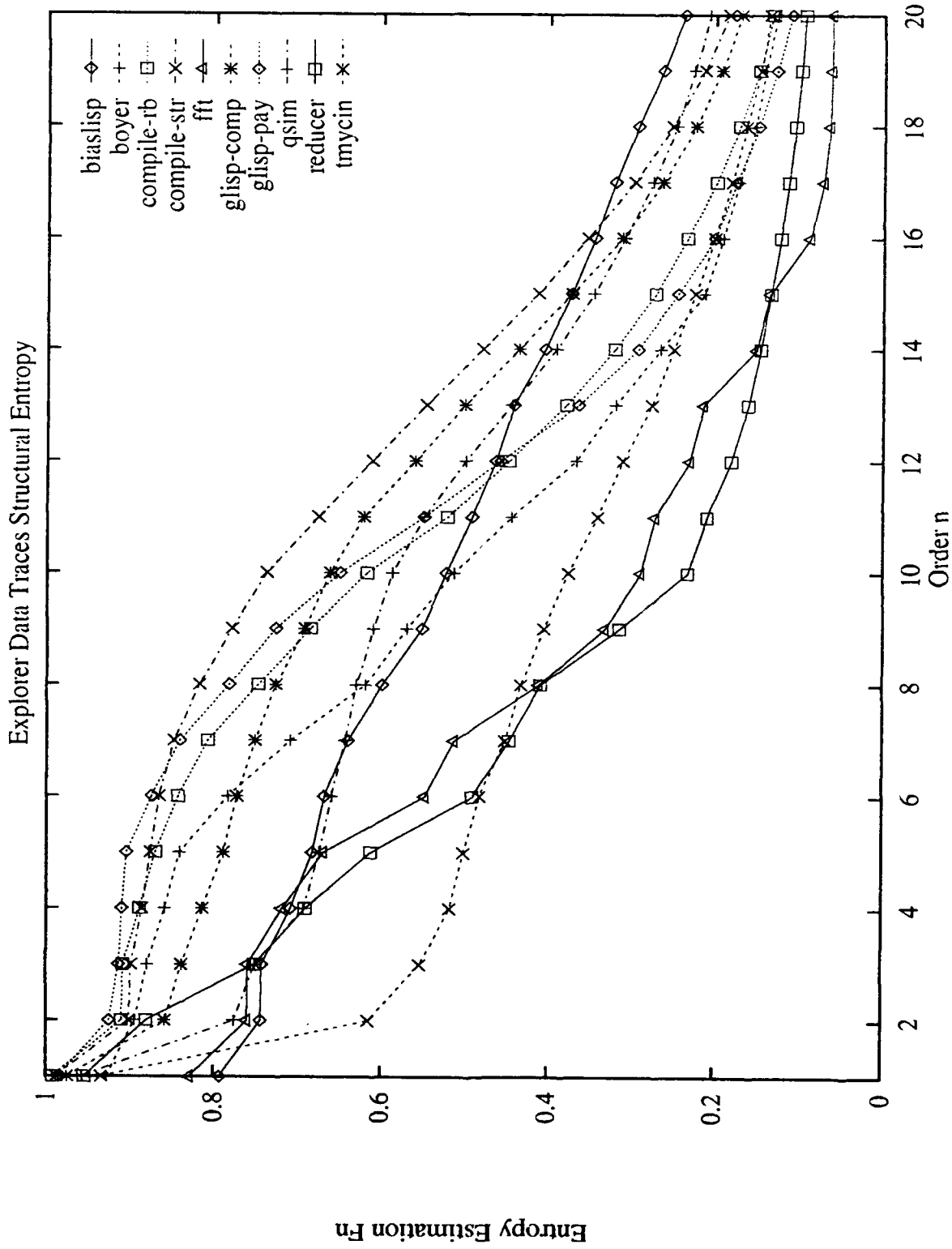


Figure B.10. Structural Entropy of the Explorer Overall Traces

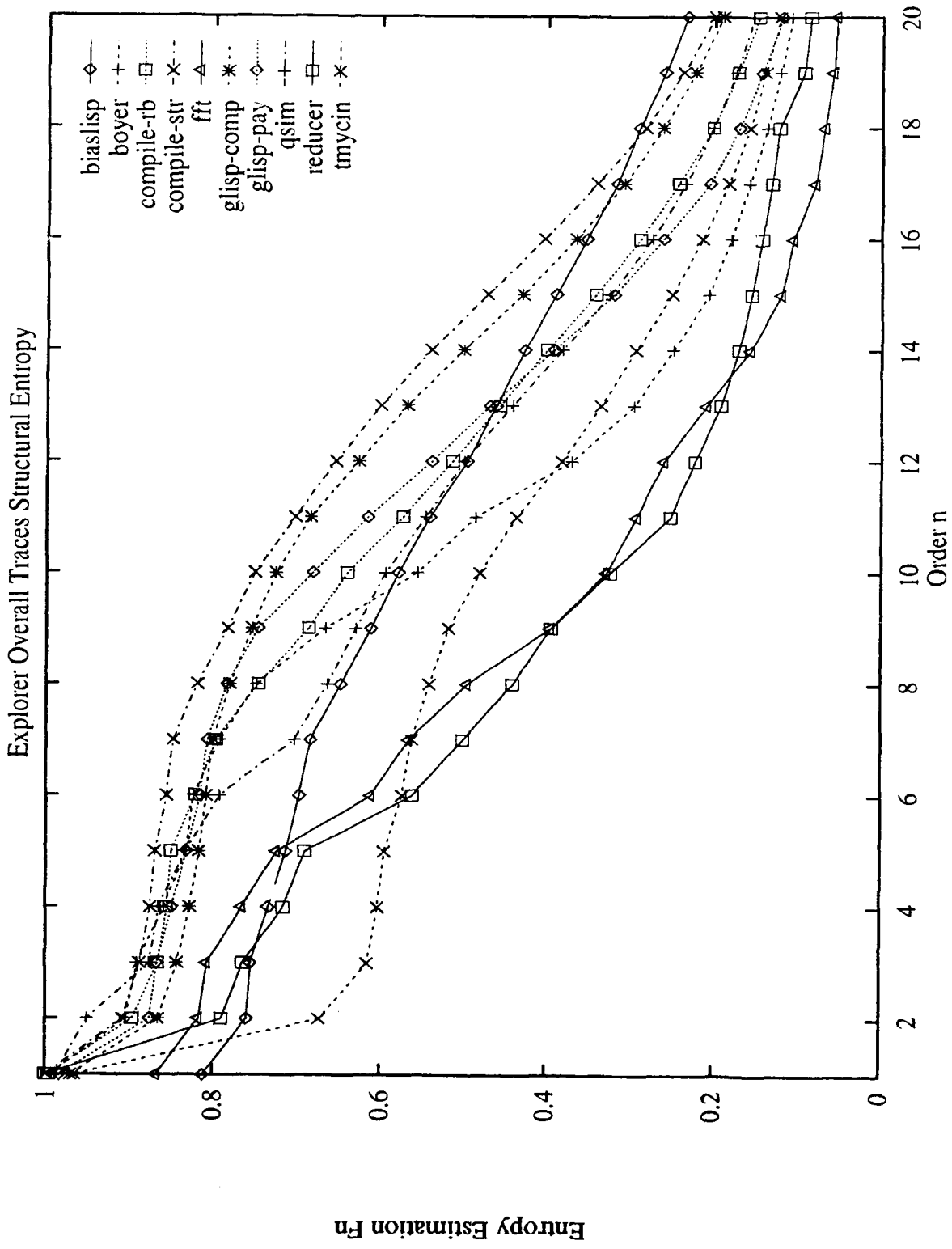


Figure B.11. Differences in the New Distributions of the Explorer Instruction Traces

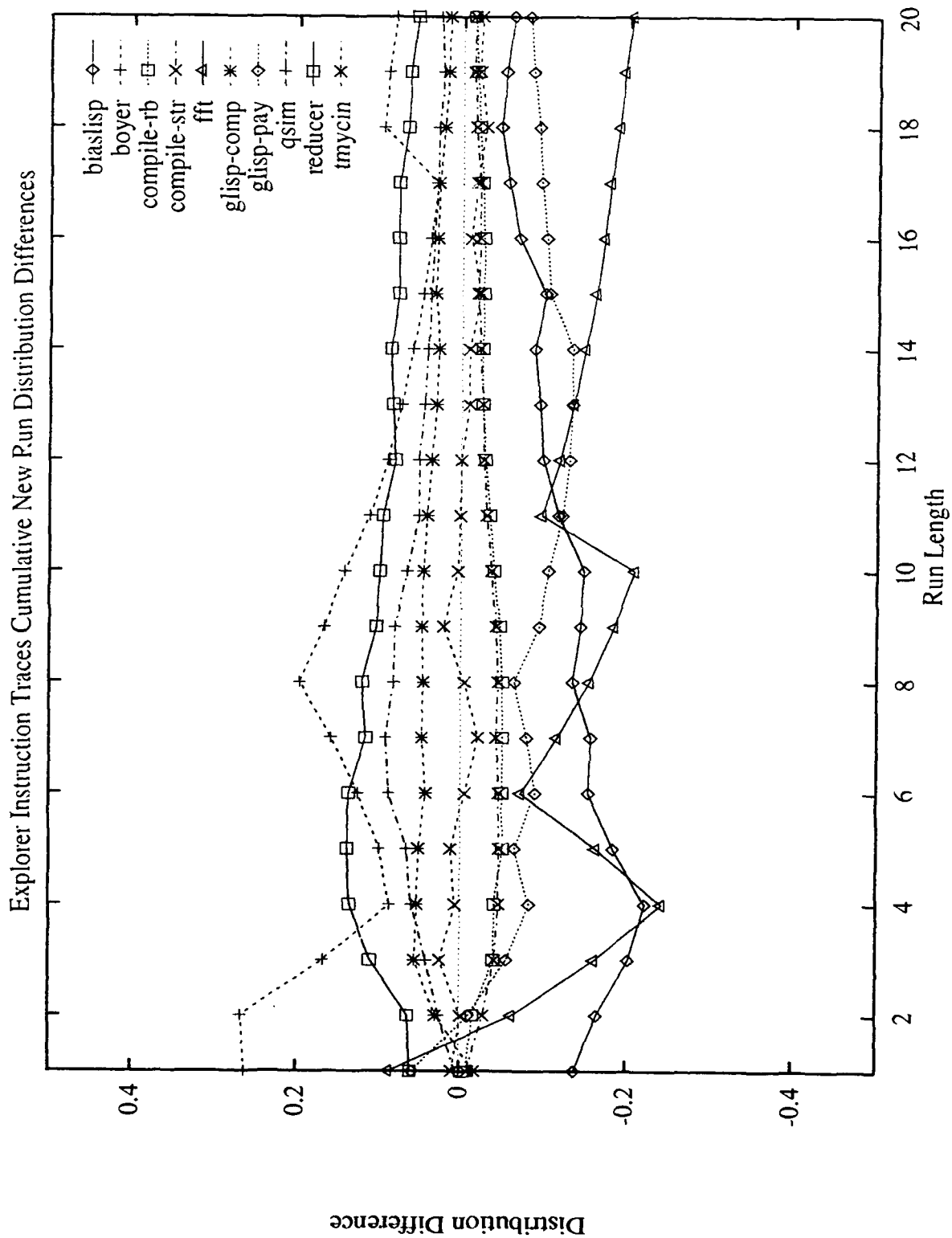


Figure B.12. Differences in the New Distributions of the Explorer Read Traces

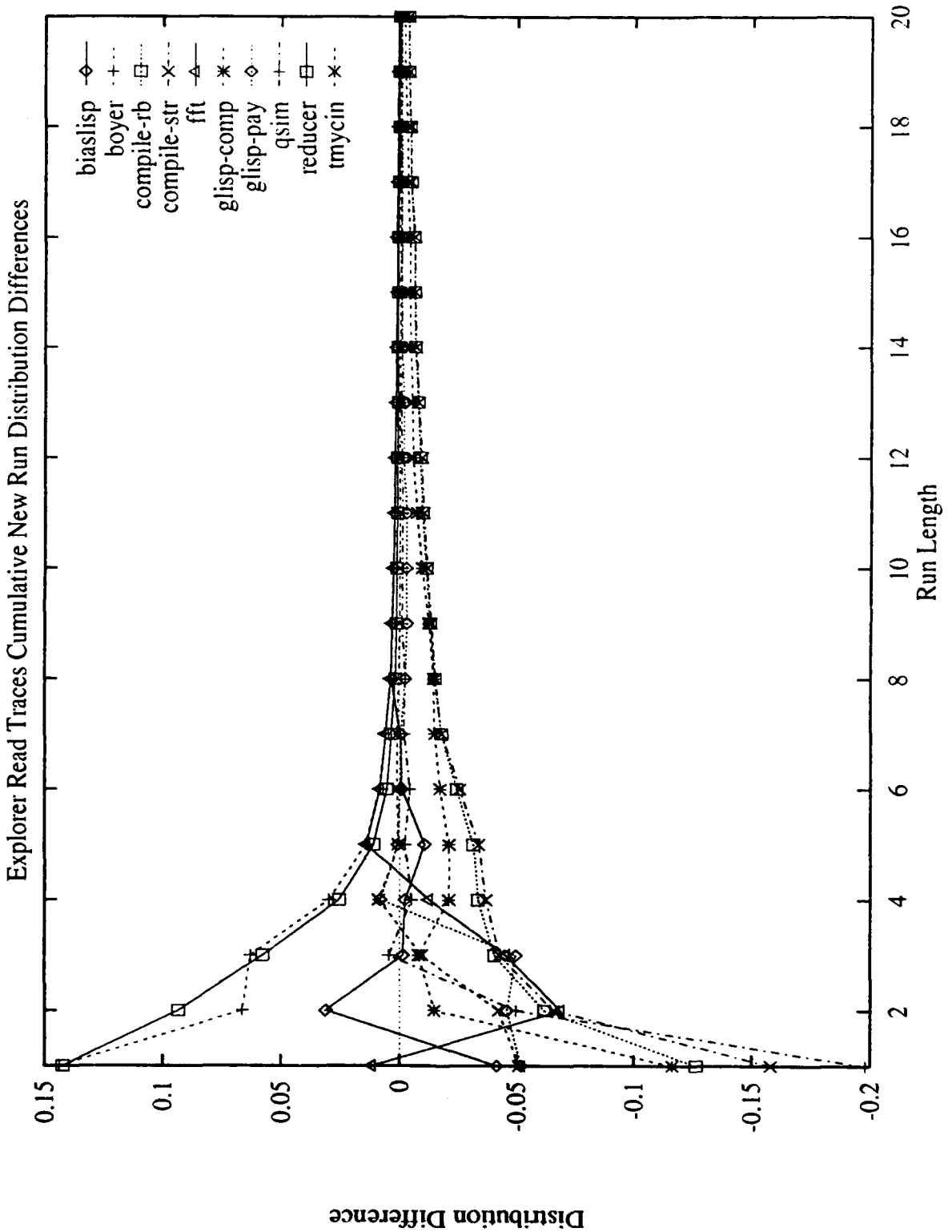


Figure B.13. Differences in the New Distributions of the Explorer Write Traces

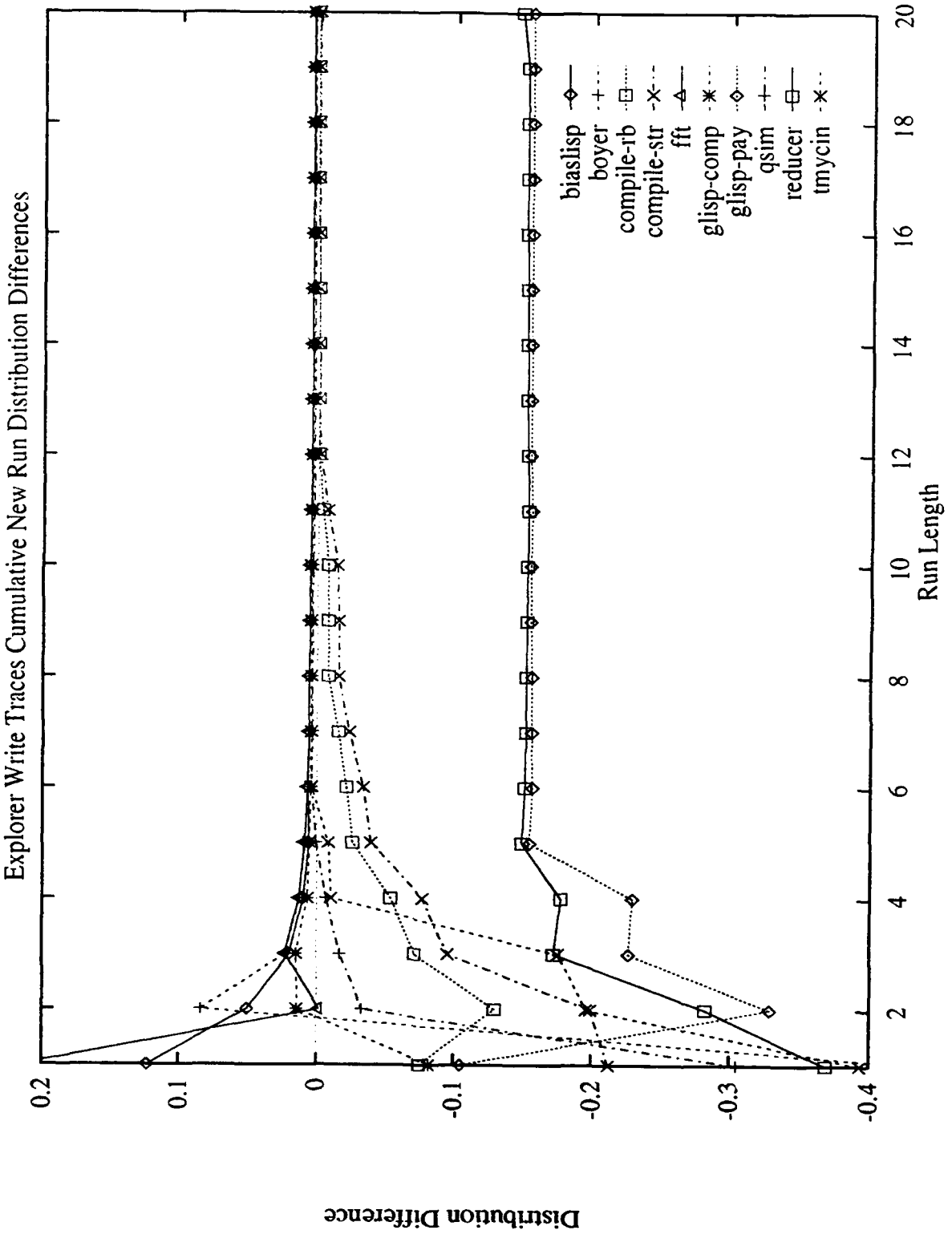


Figure B.14. Differences in the New Distributions of the Explorer Data Traces

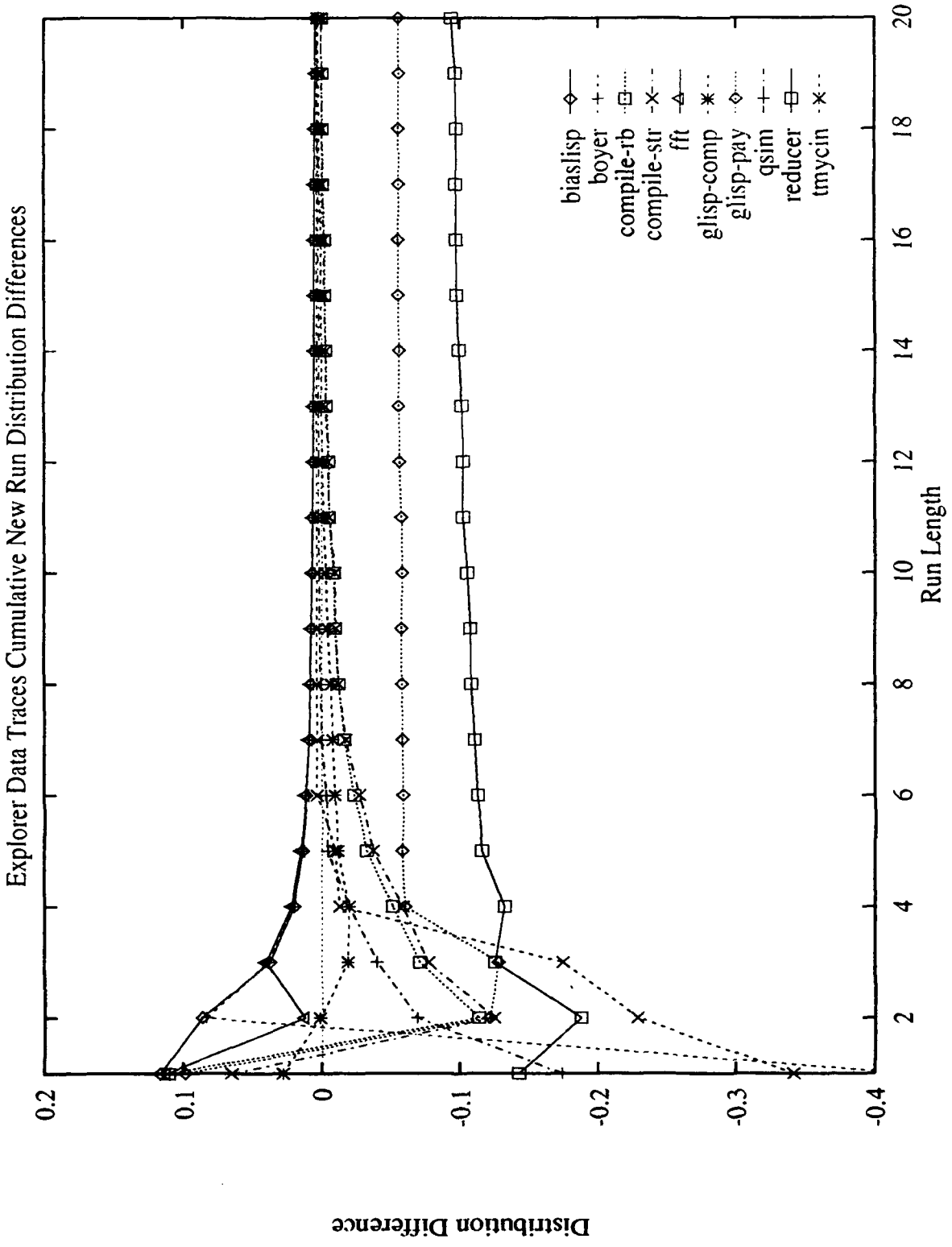


Figure B.15. Differences in the New Distributions of the Explorer Overall Traces

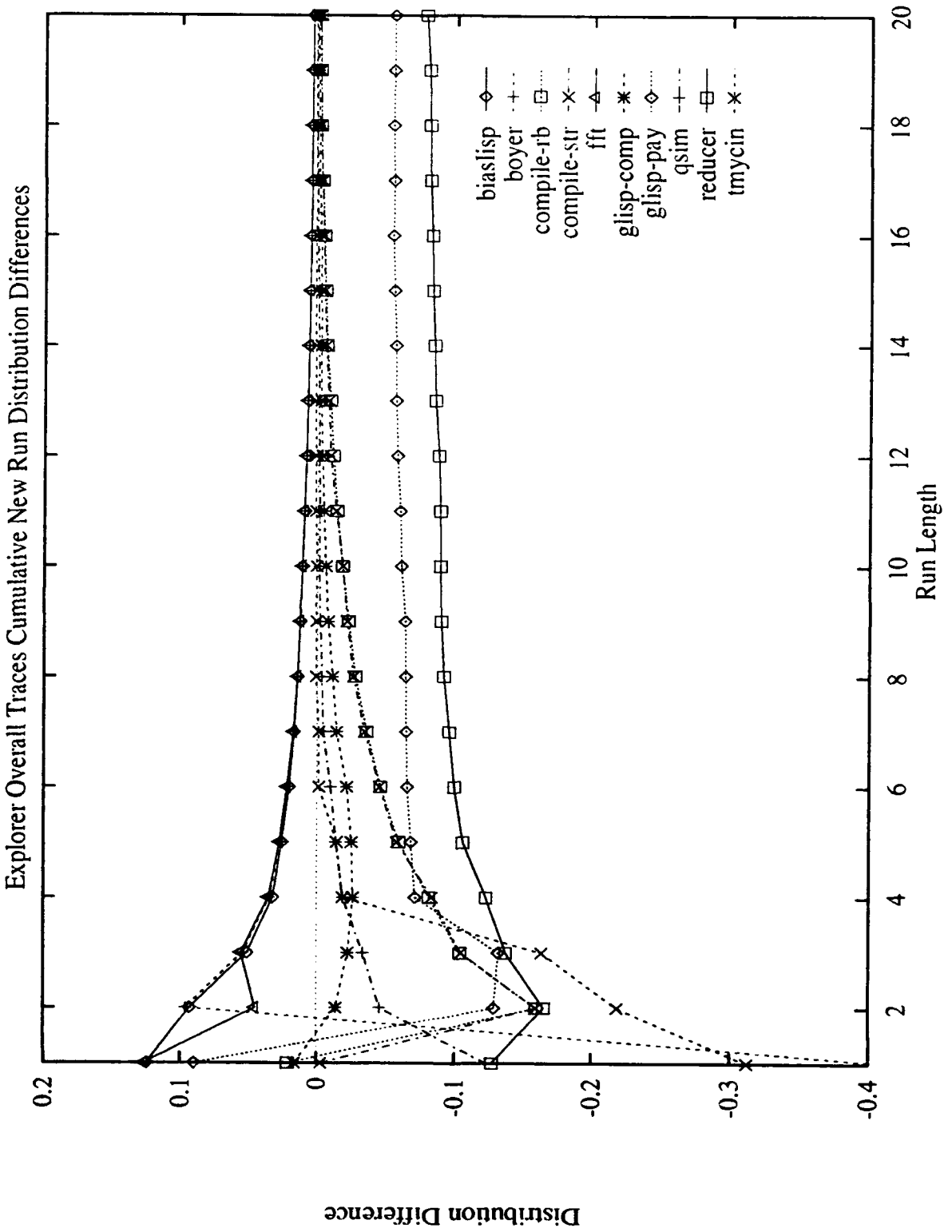


Figure B.16. Differences in the SSD Distributions of the Explorer Instruction Traces

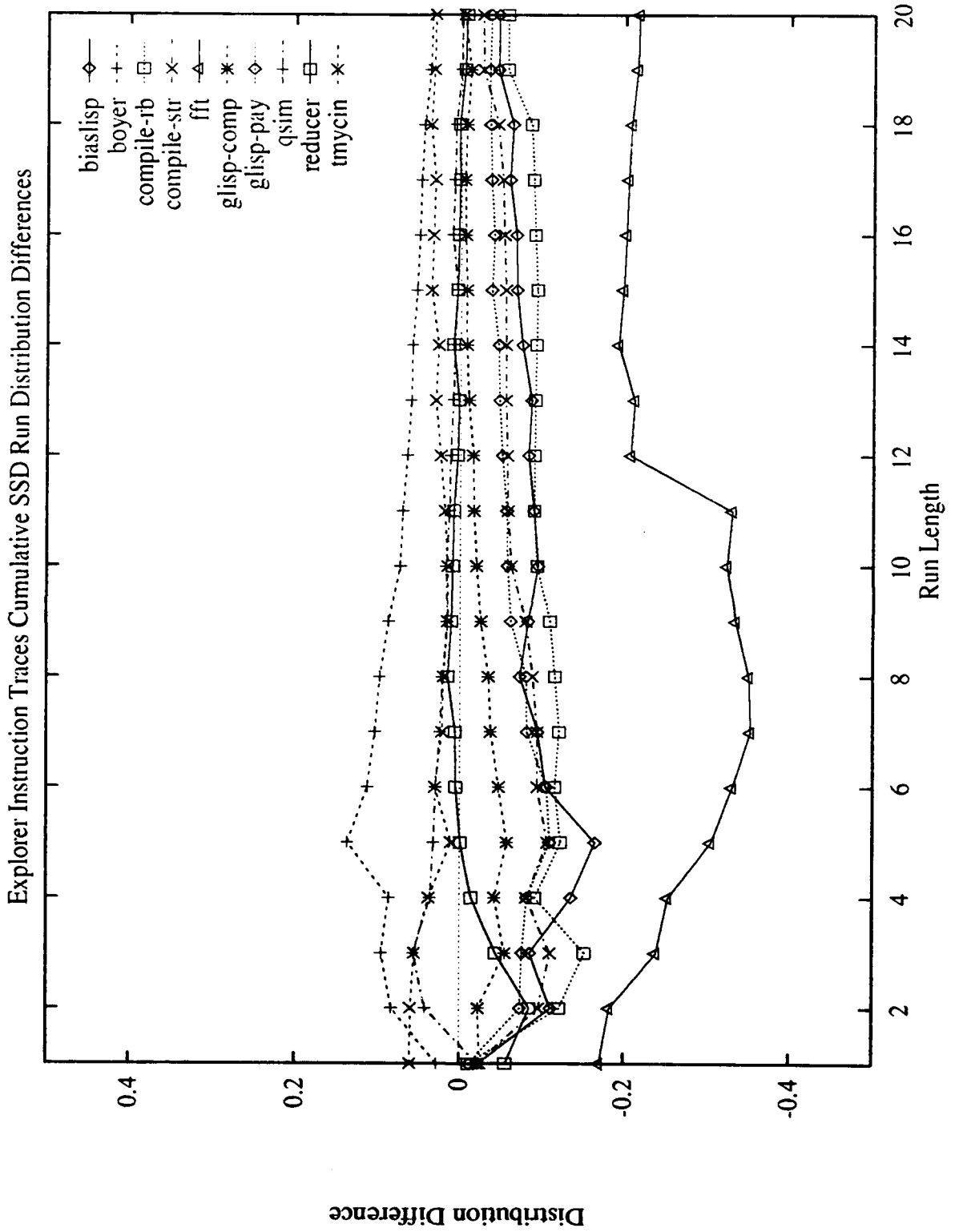


Figure B.17. Differences in the SSD Distributions of the Explorer Read Traces

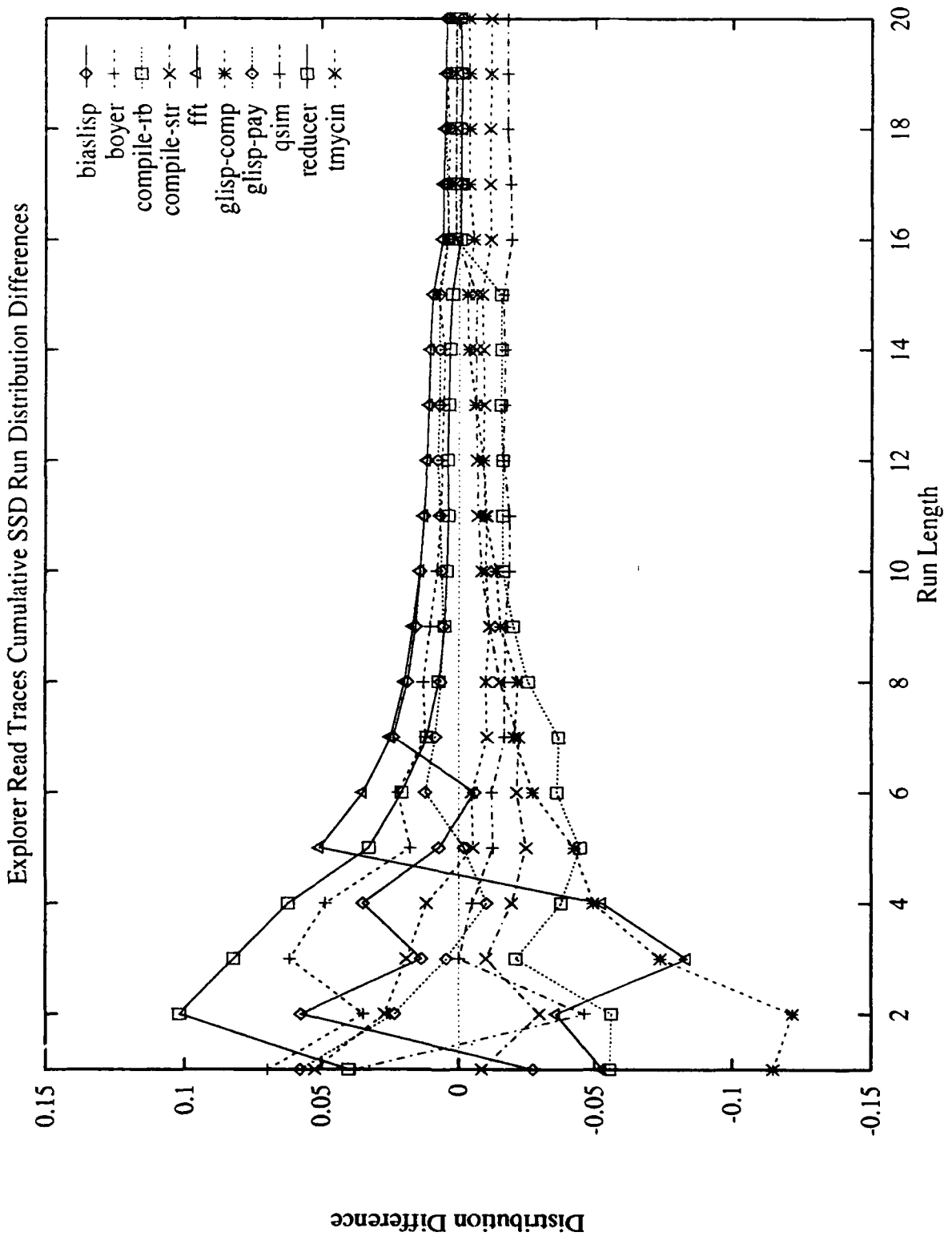


Figure B.18. Differences in the SSD Distributions of the Explorer Write Traces

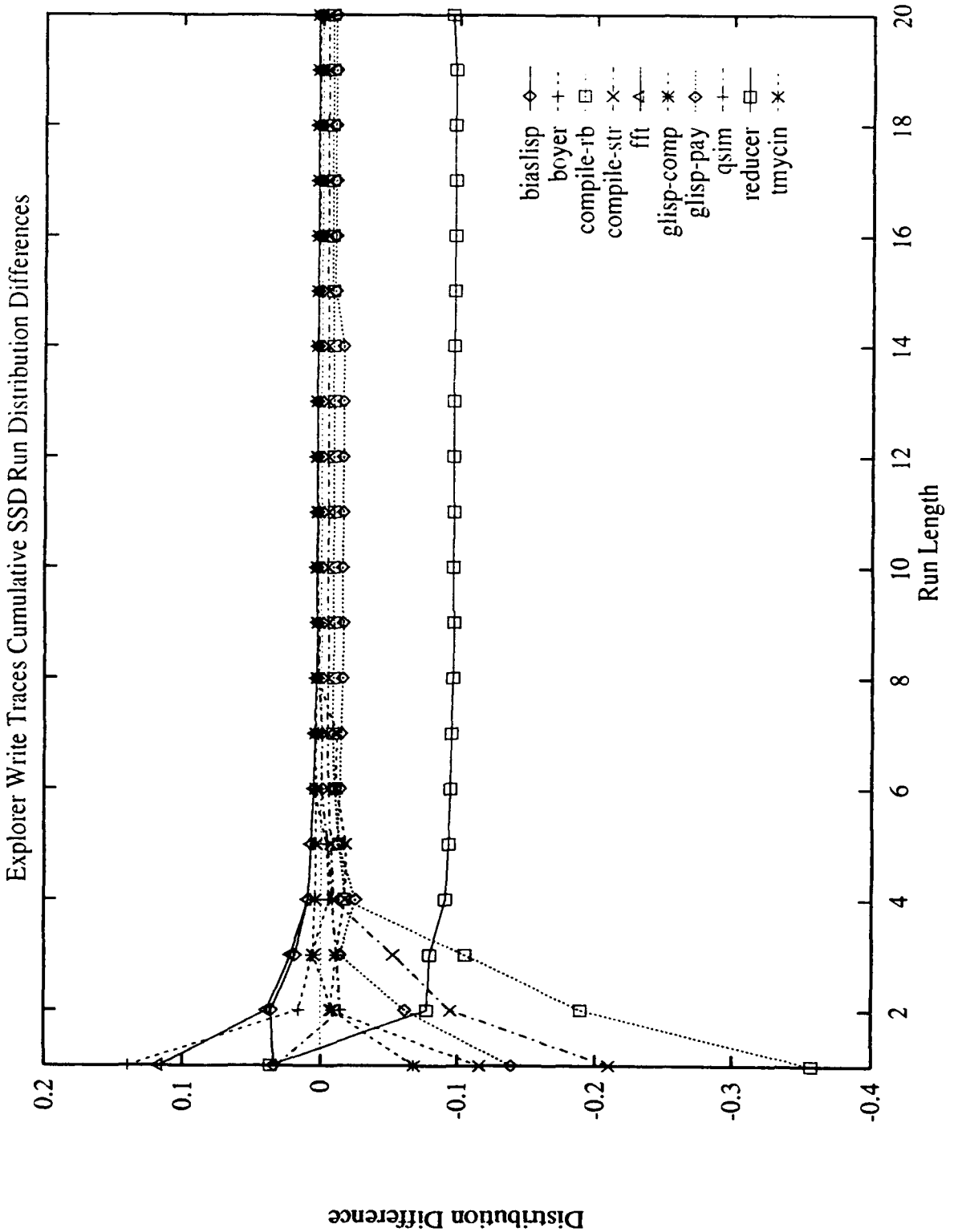


Figure B.19. Differences in the SSD Distributions of the Explorer Data Traces

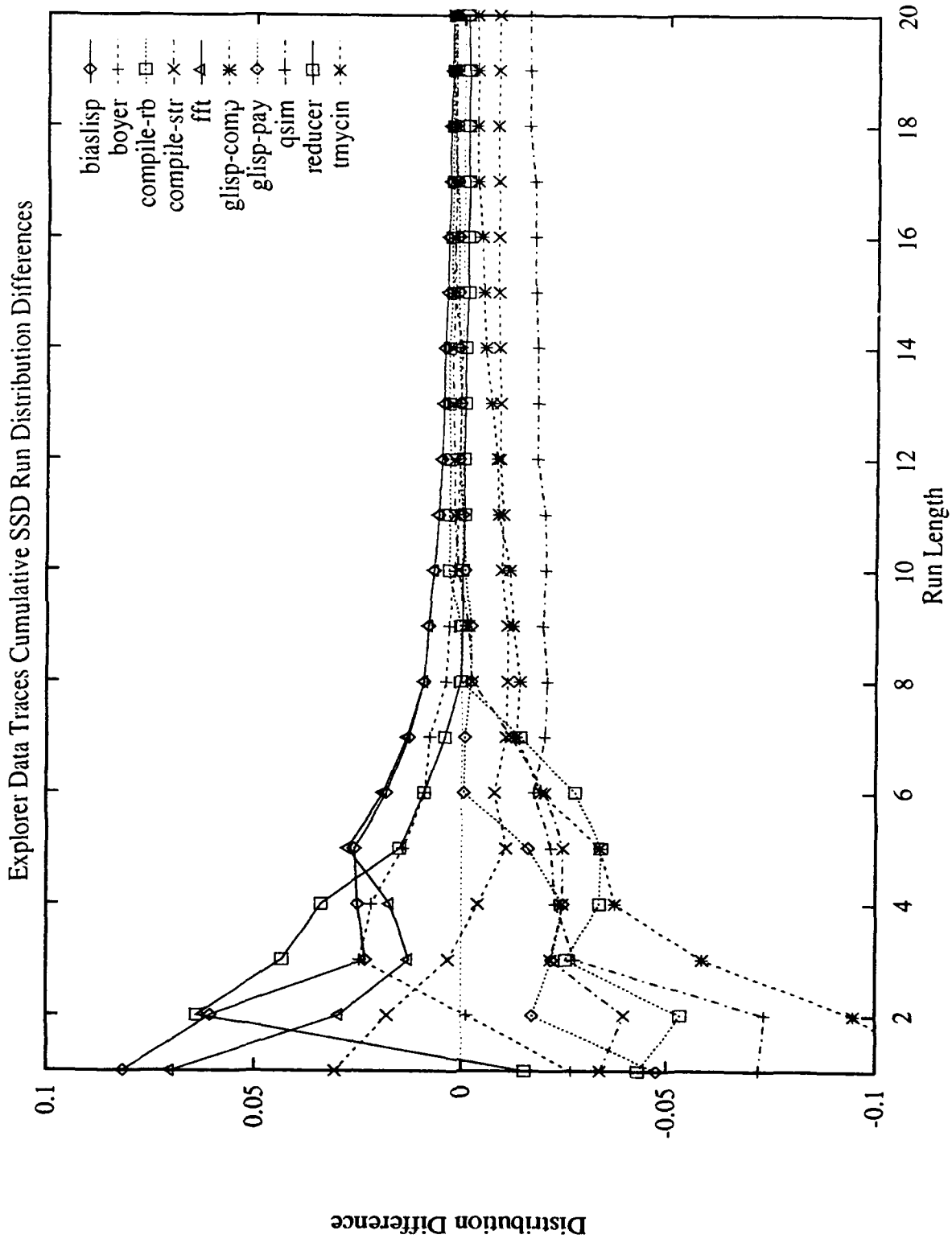
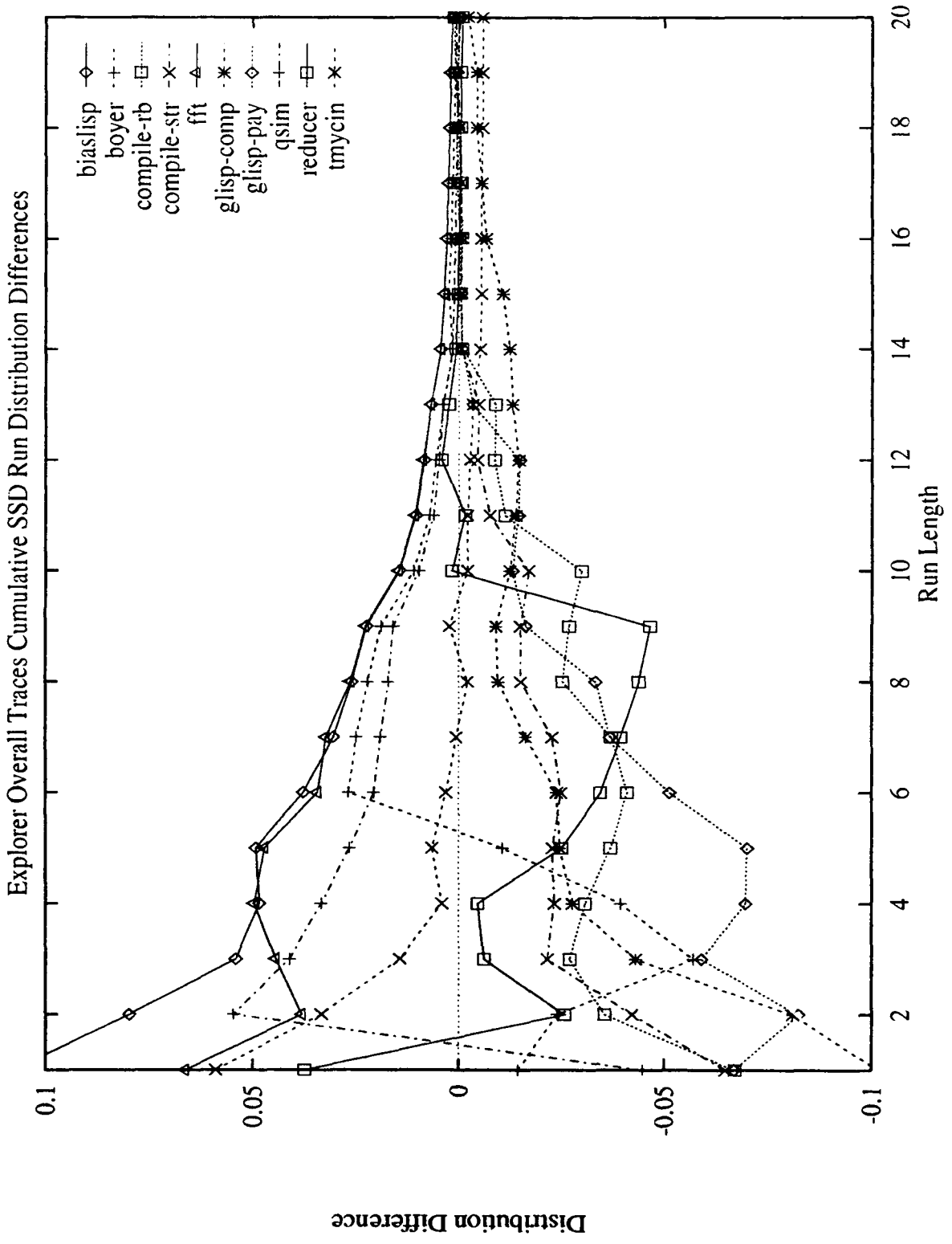


Figure B.20. Differences in the SSD Distributions of the Explorer Overall Traces



Appendix C. *ATUM Results*

C.1 Introduction

This is the supplementary data for chapter 4. The graphs are the structural locality measurments performed on the ATUM set of traces.

Figure C.1. State Entropy of the ATUM Instruction Traces

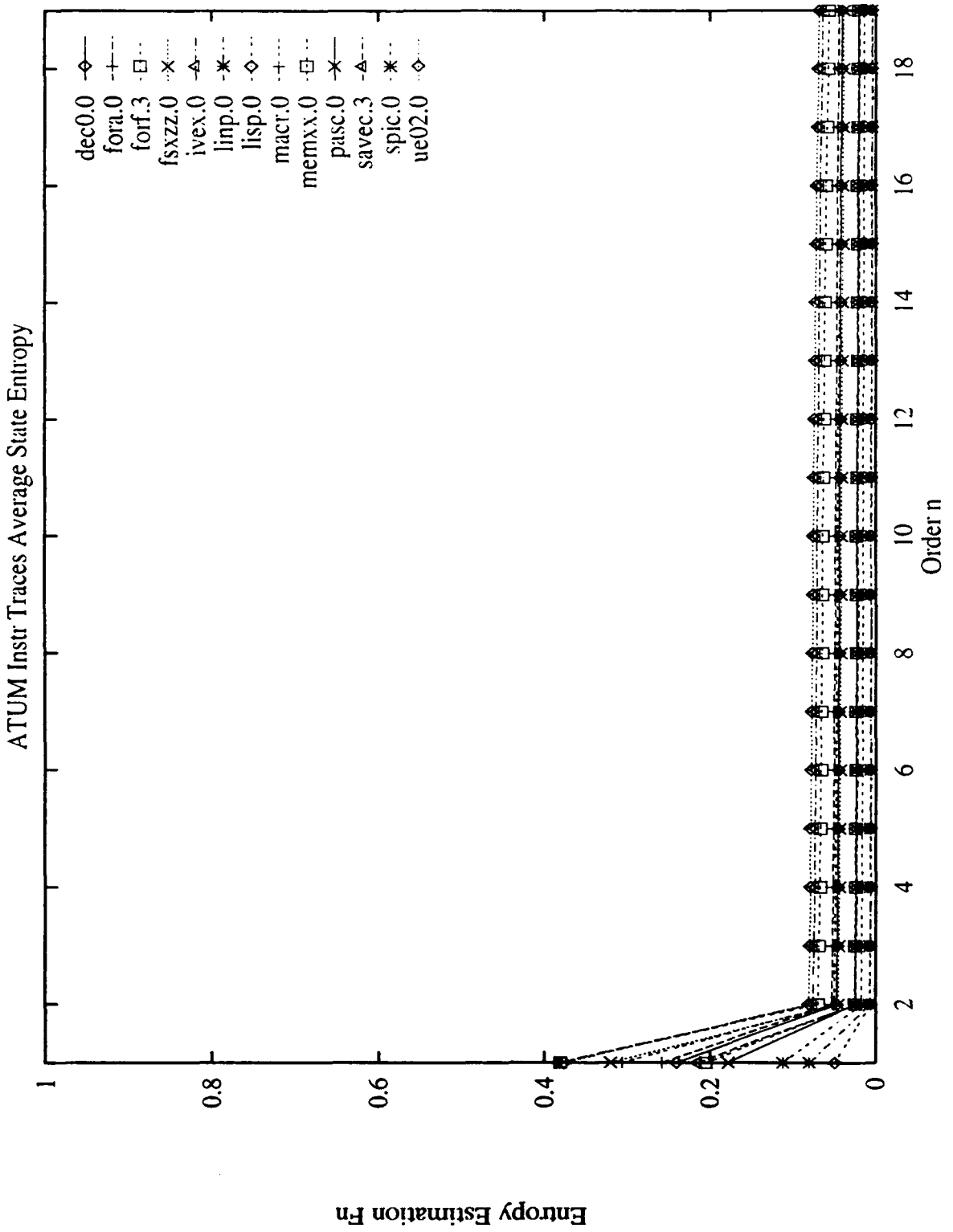


Figure C.2. State Entropy of the ATUM Read Traces

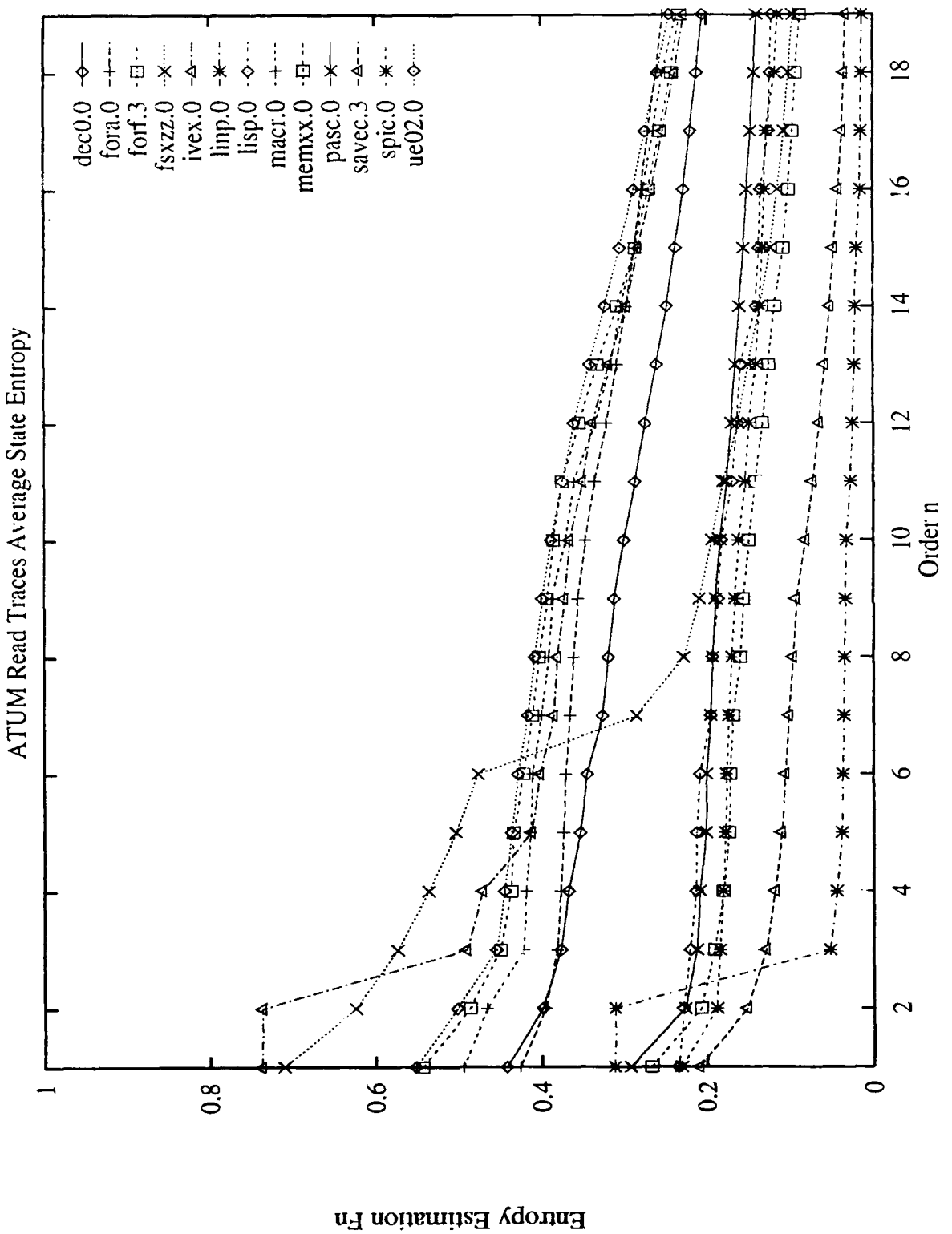


Figure C.3. State Entropy of the ATUM Write Traces

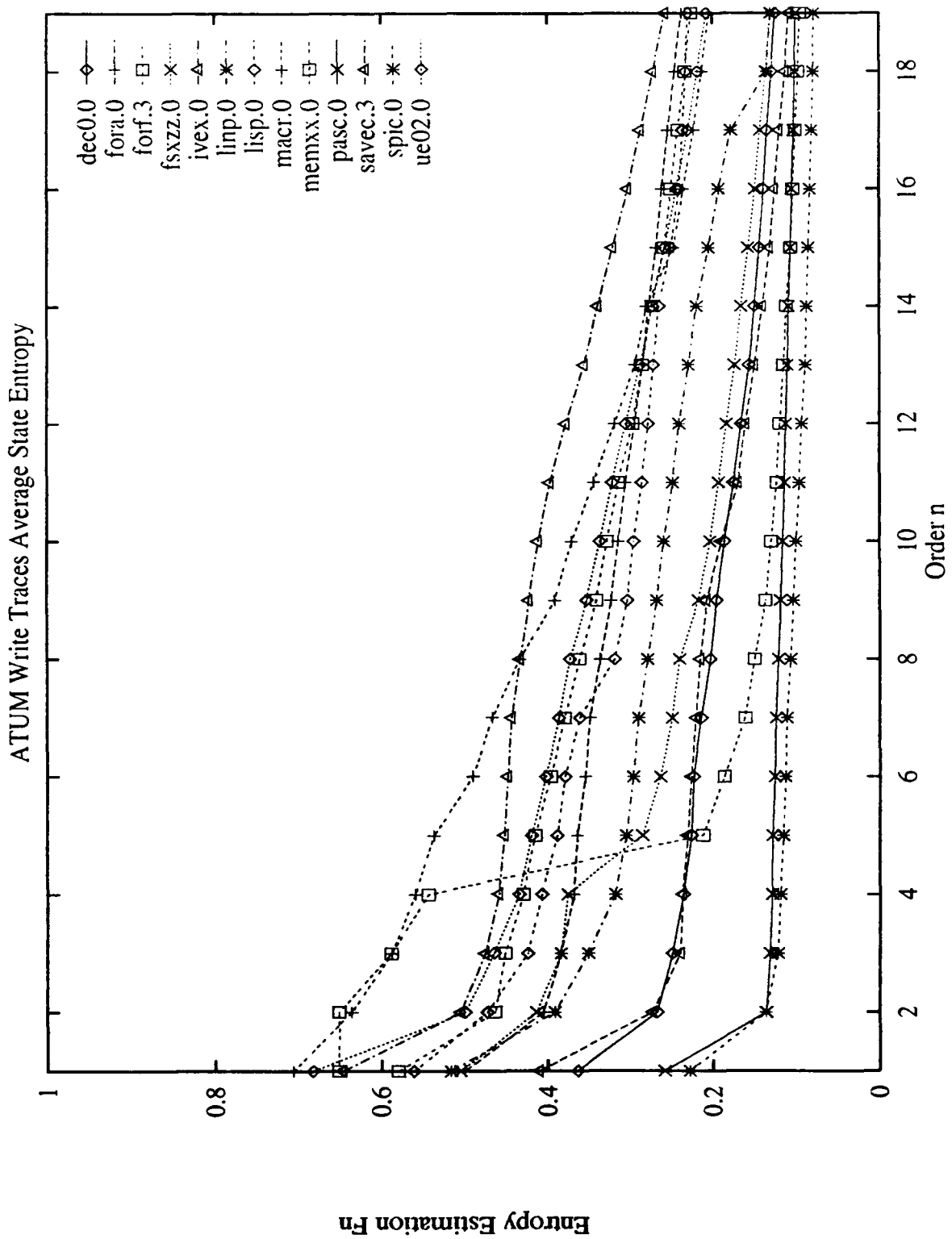


Figure C-4. State Entropy of the ATUM Data Traces

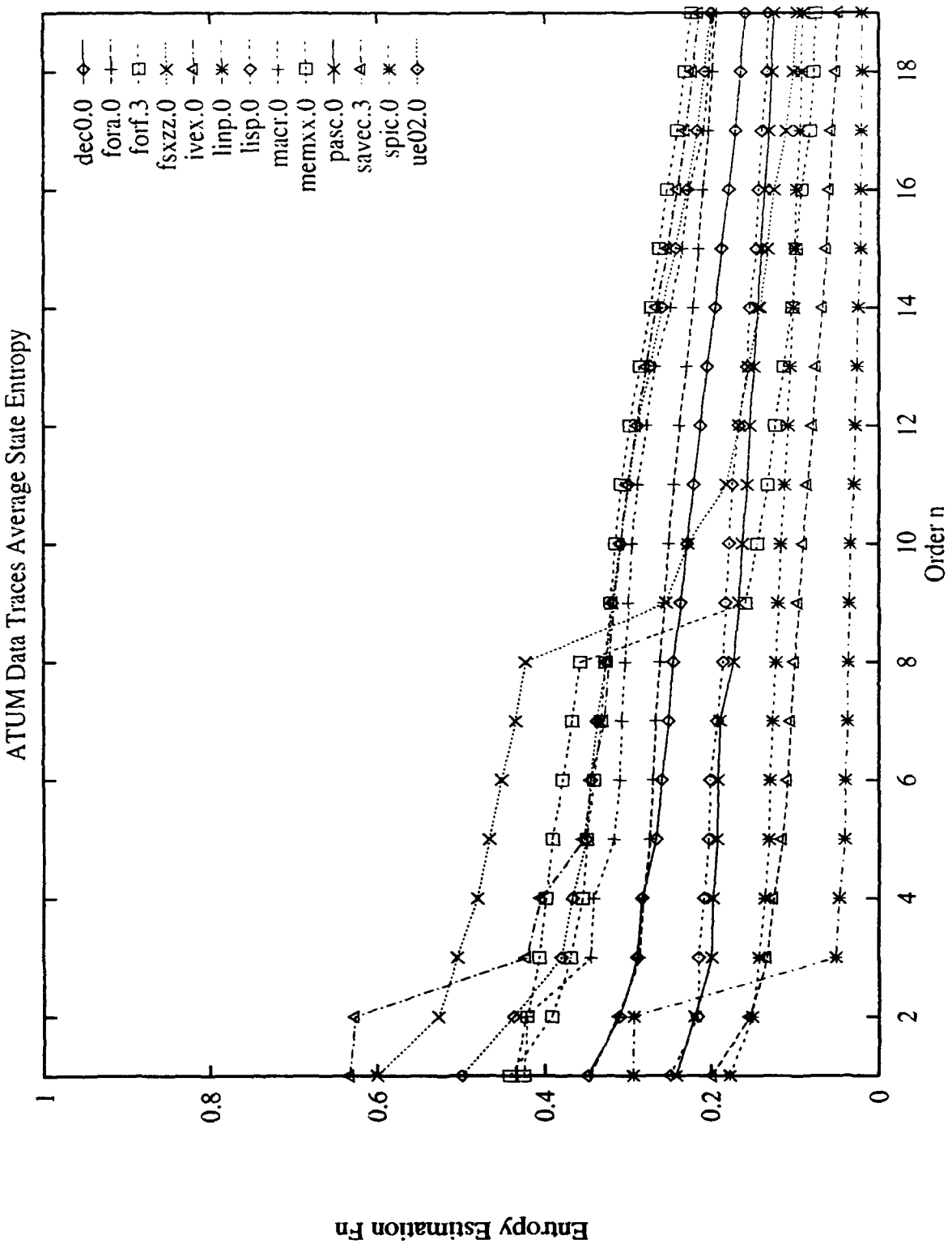


Figure C.5. State Entropy of the ATUM Overall Traces

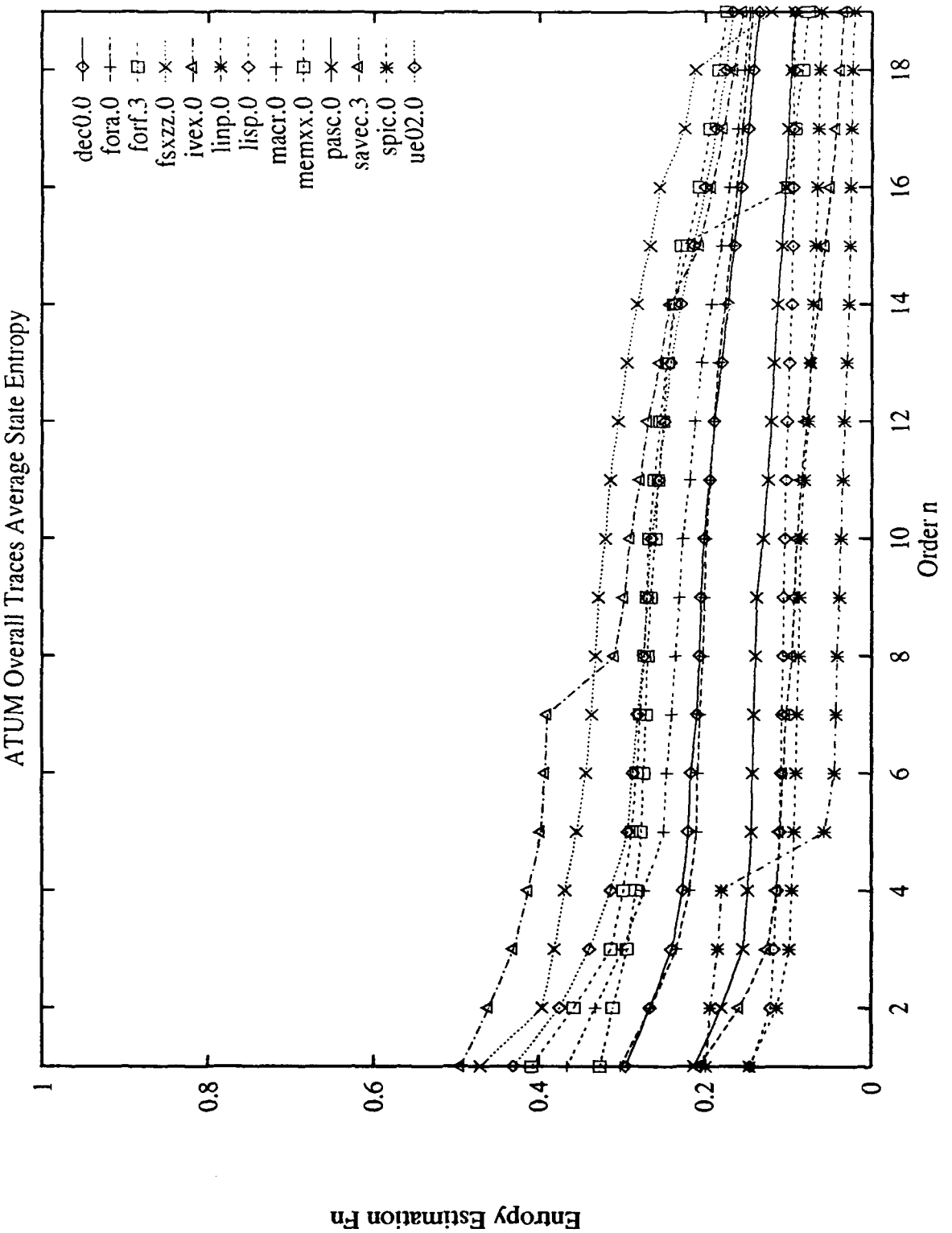


Figure C.6. Structural Entropy of the ATUM Instruction Traces

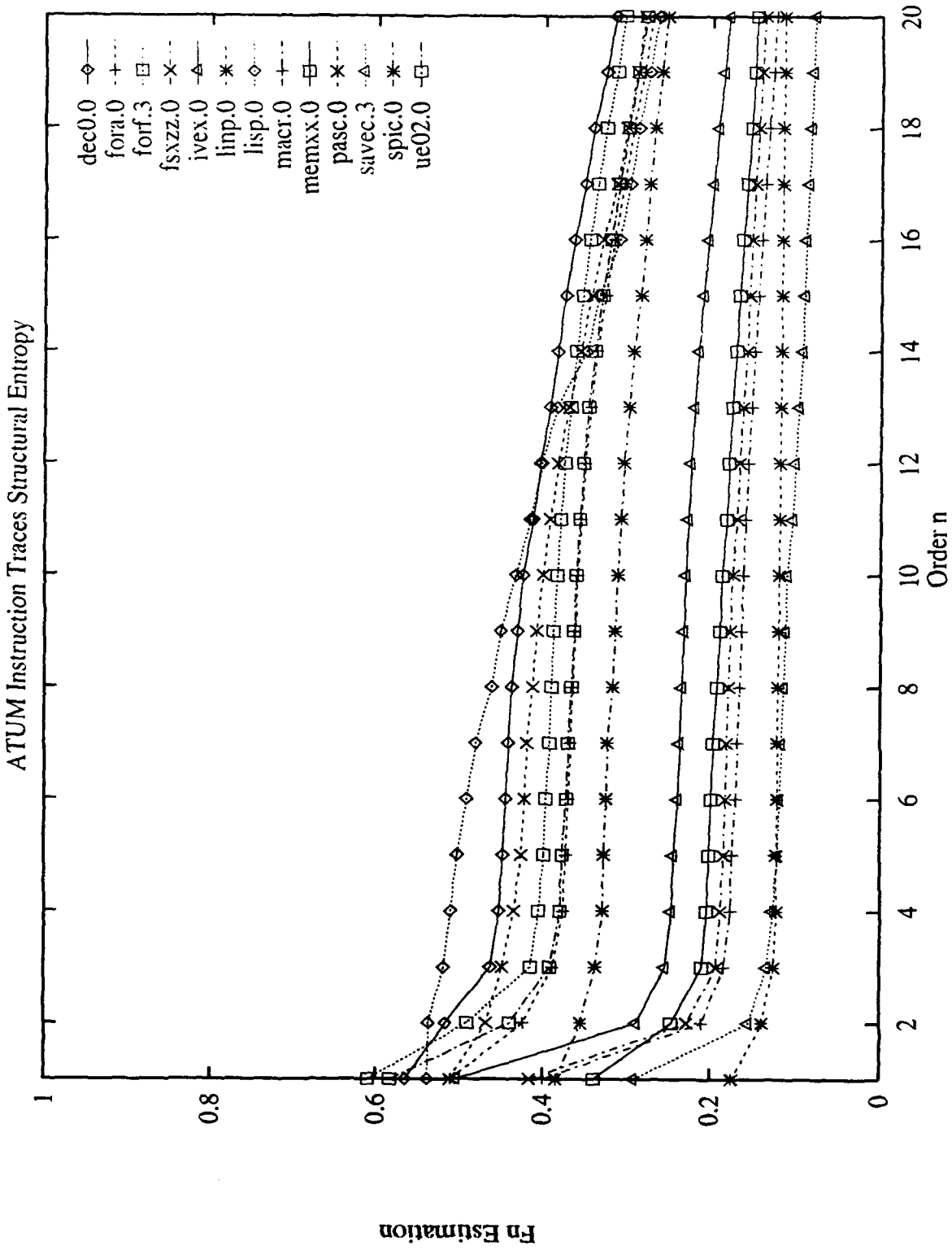


Figure C.7. Structural Entropy of the ATUM Read Traces

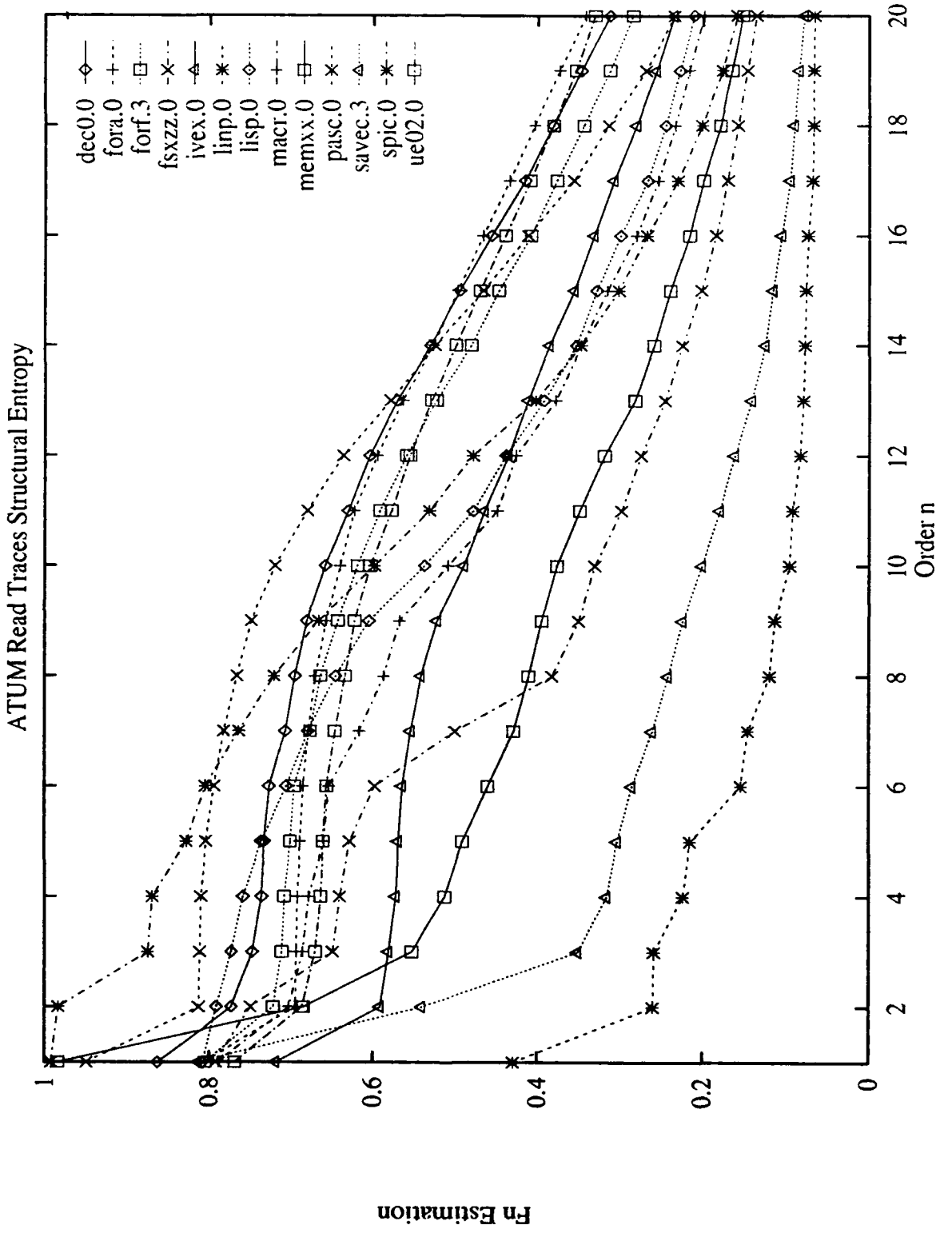


Figure C.8. Structural Entropy of the ATUM Write Traces

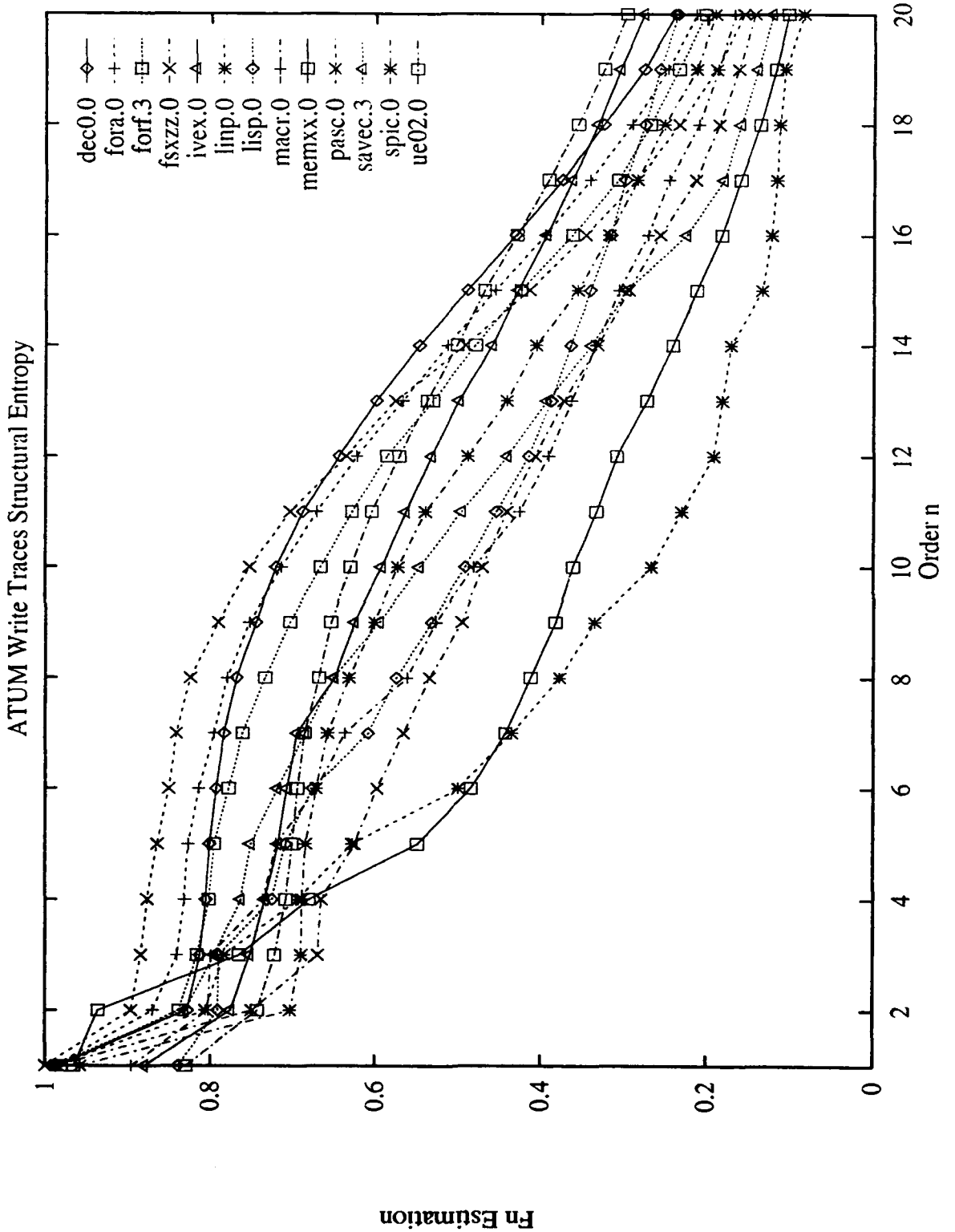


Figure C.9. Structural Entropy of the ATUM Data Traces

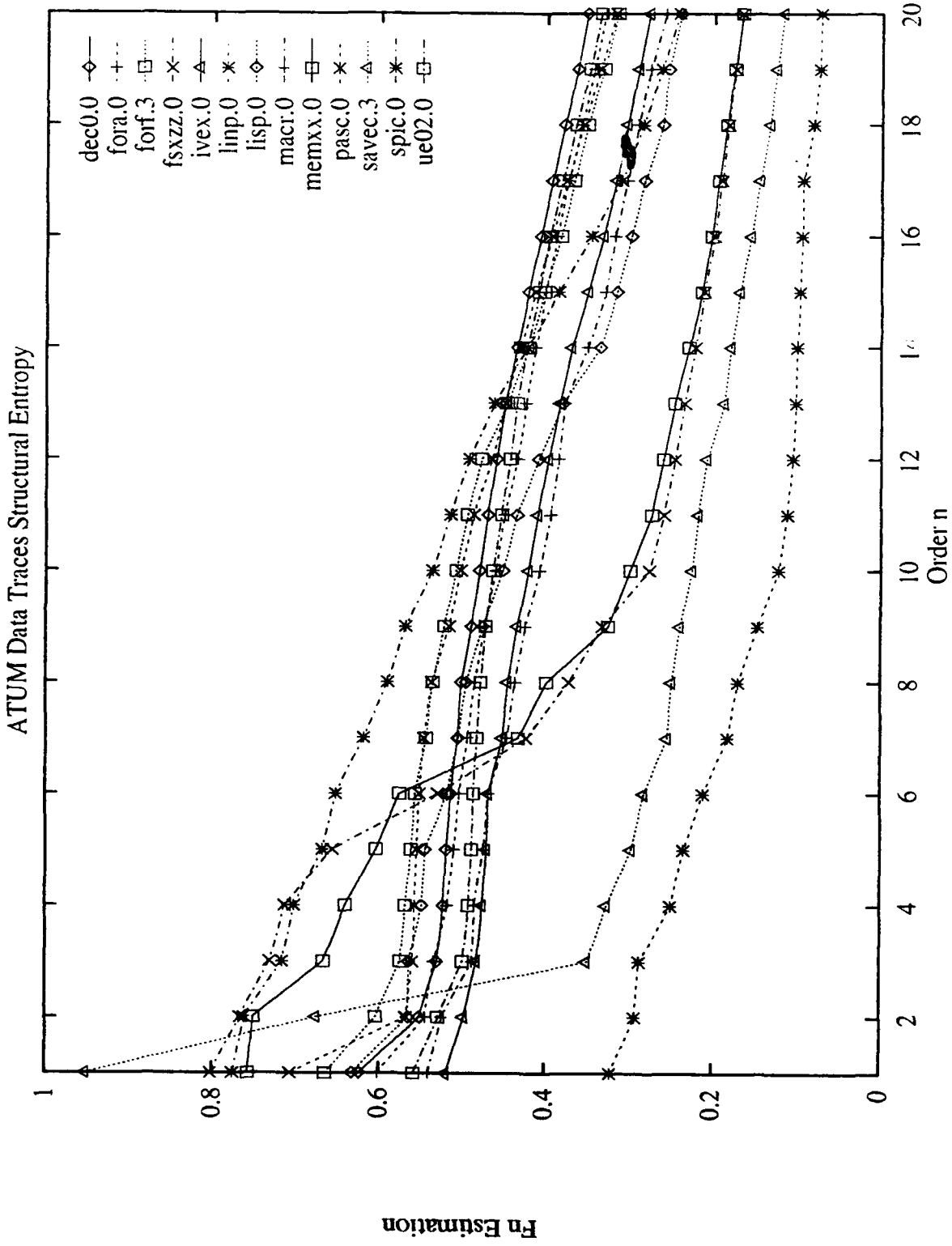


Figure C.10. Structural Entropy of the ATUM Overall Traces

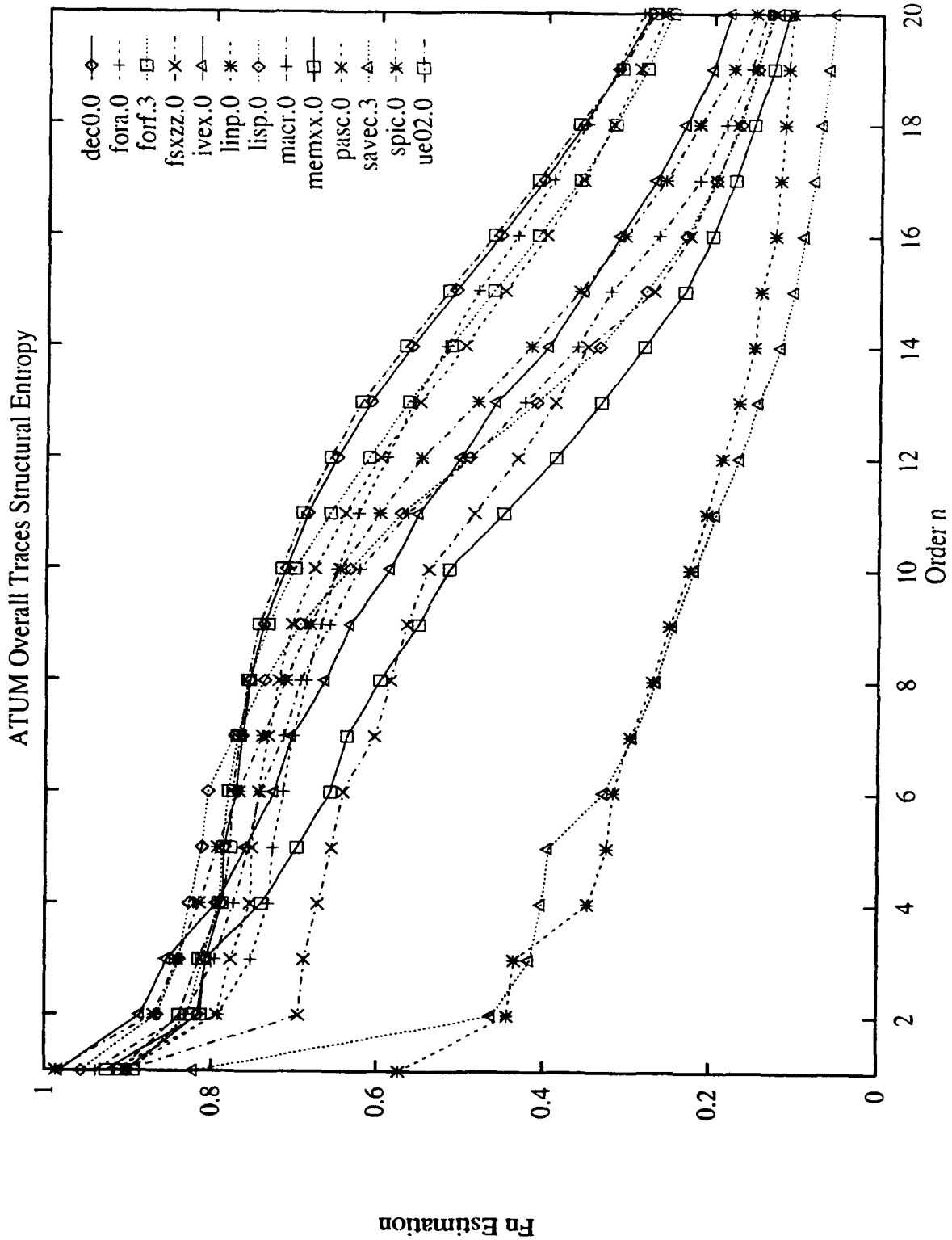


Figure C.11. Differences in the New Distributions of the ATUM Instruction Traces

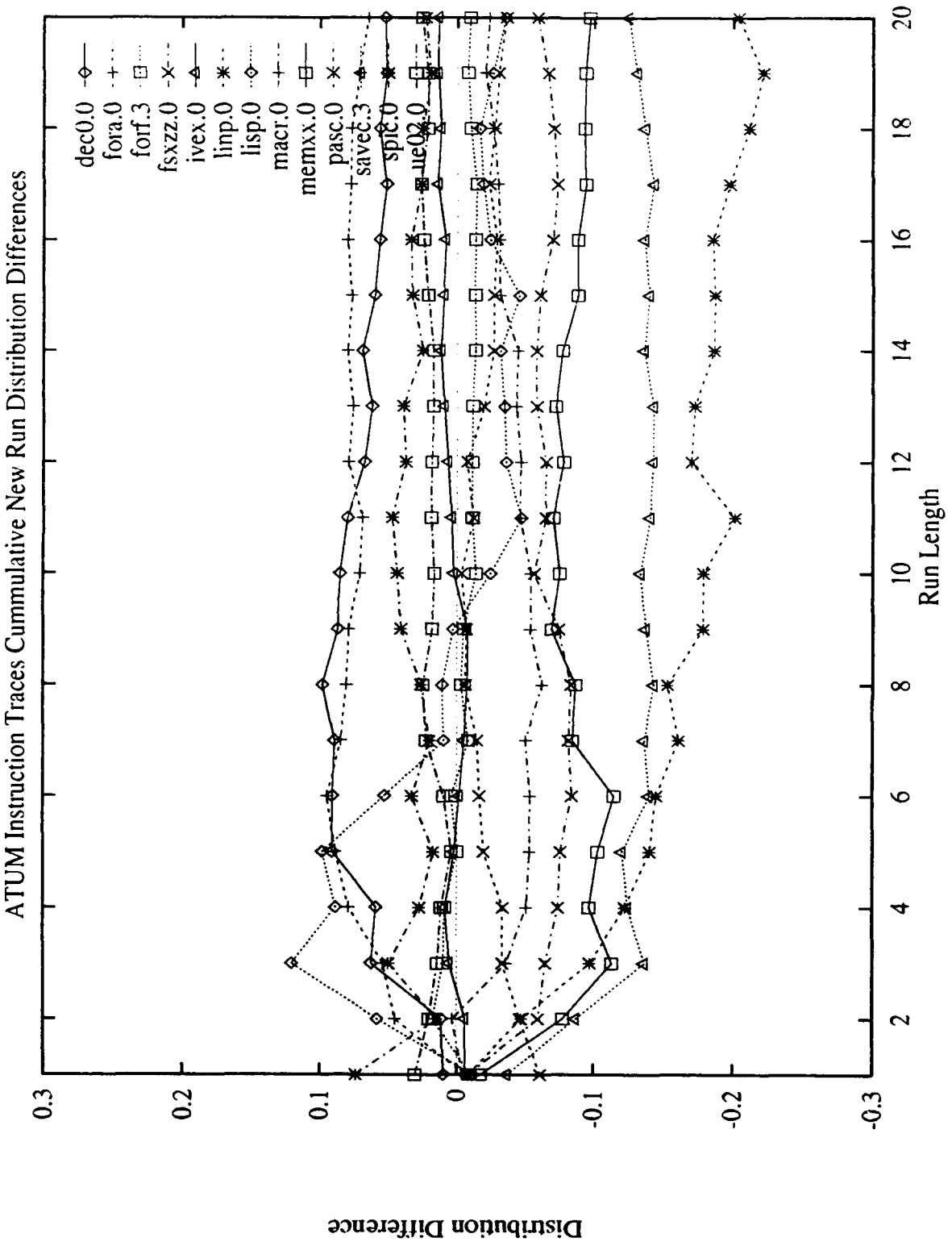


Figure C.12. Differences in the New Distributions of the ATUM Read Traces

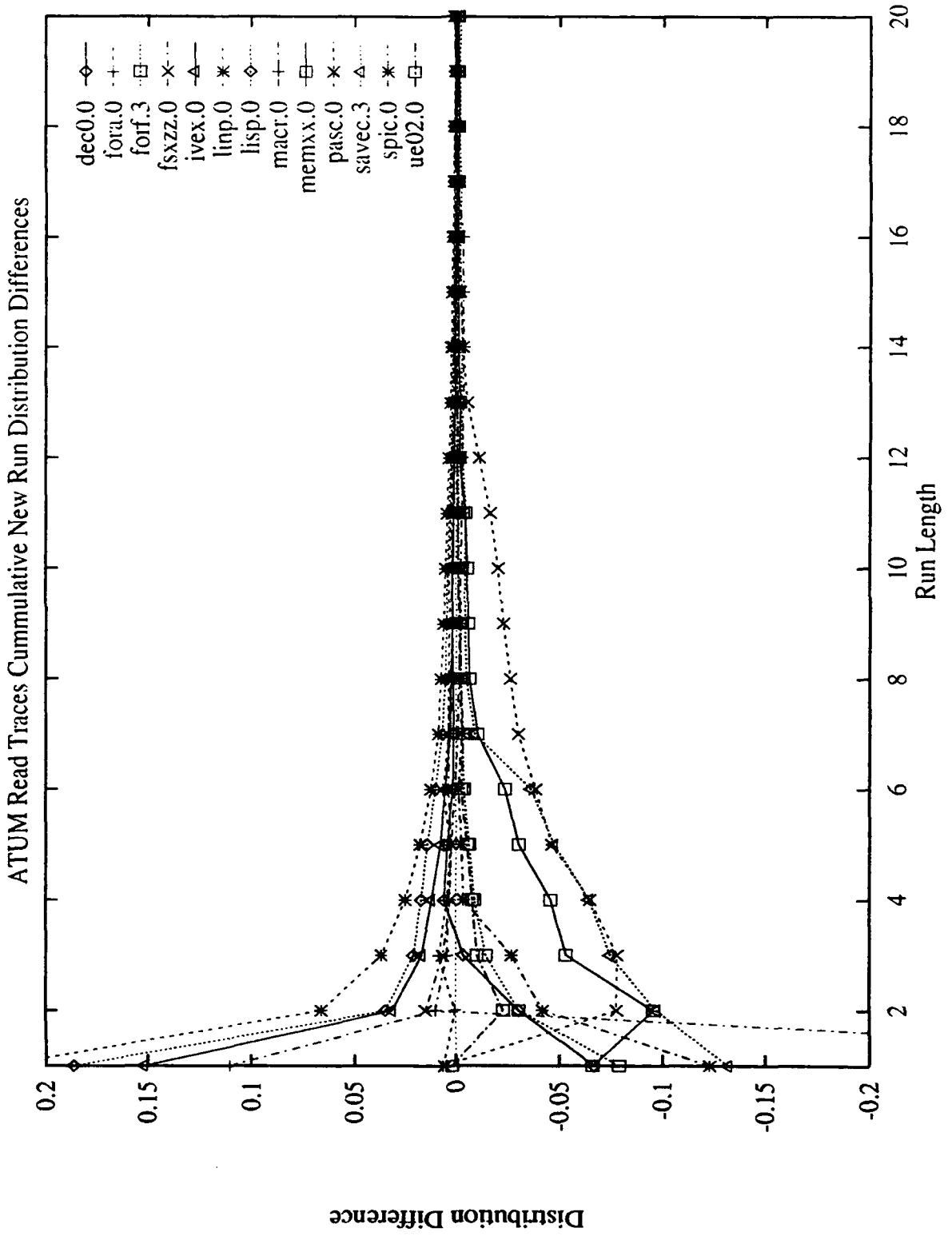


Figure C.13. Differences in the New Distributions of the ATUM Write Traces

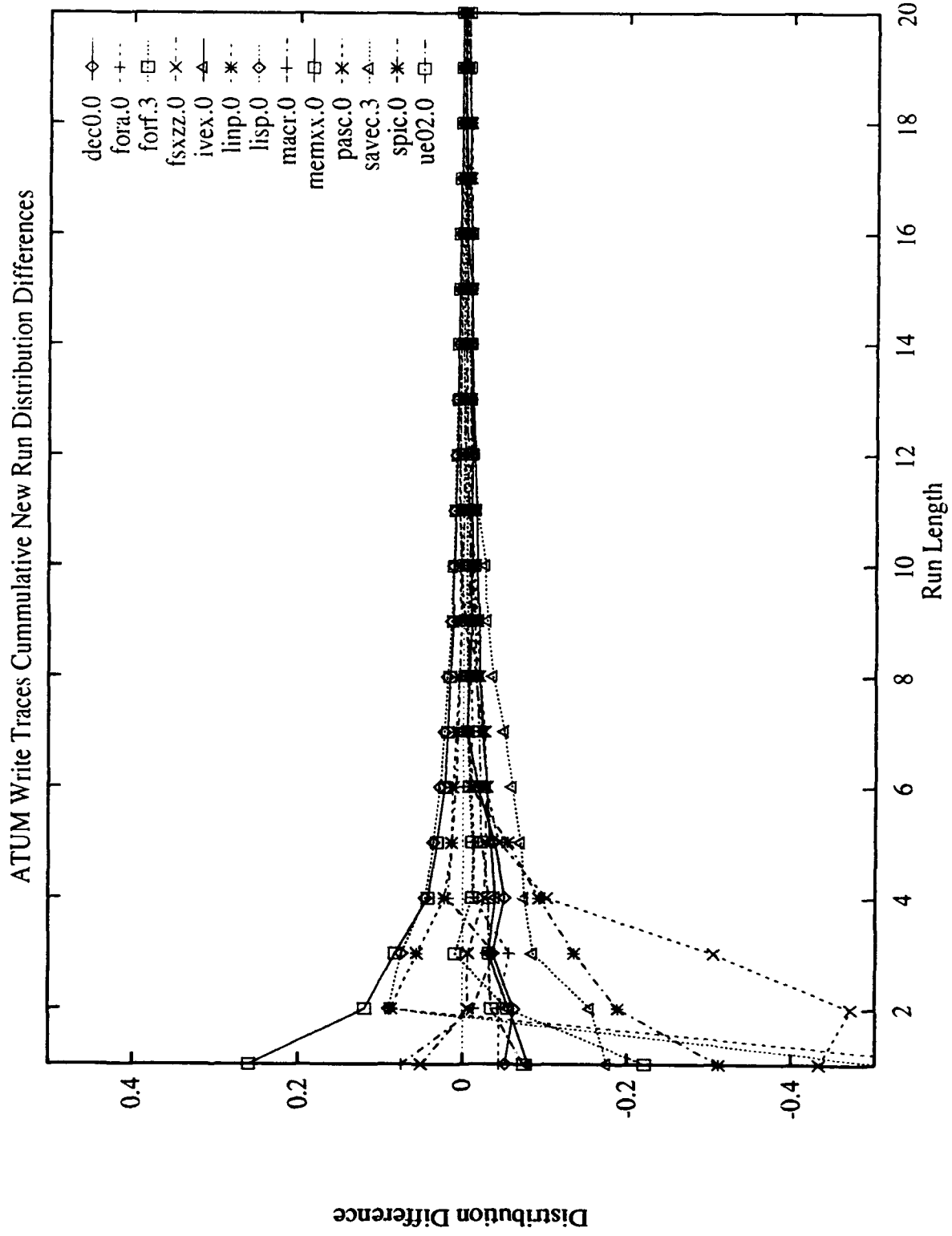


Figure C.14. Differences in the New Distributions of the ATUM Data Traces

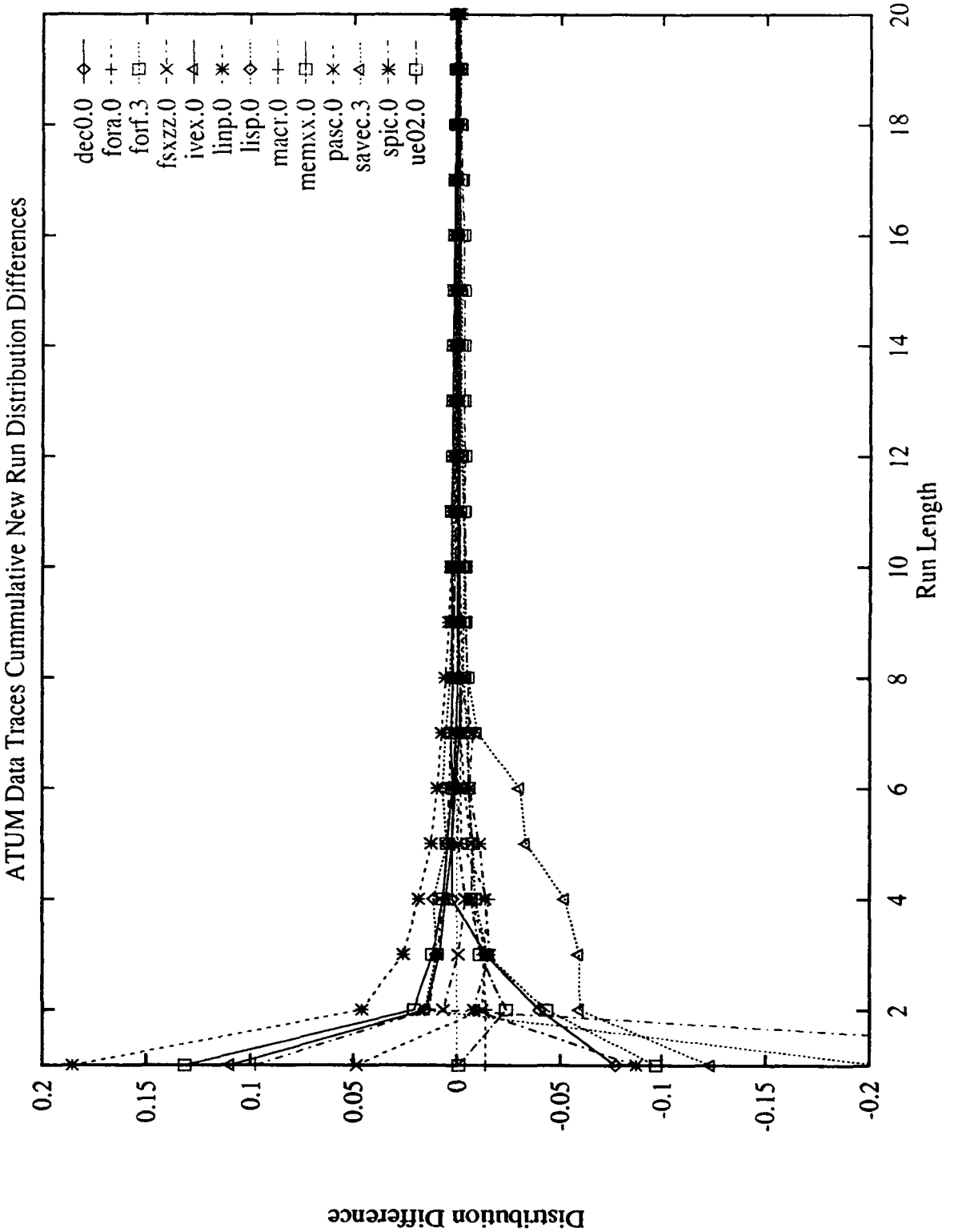


Figure C.15. Differences in the New Distributions of the ATUM Overall Traces

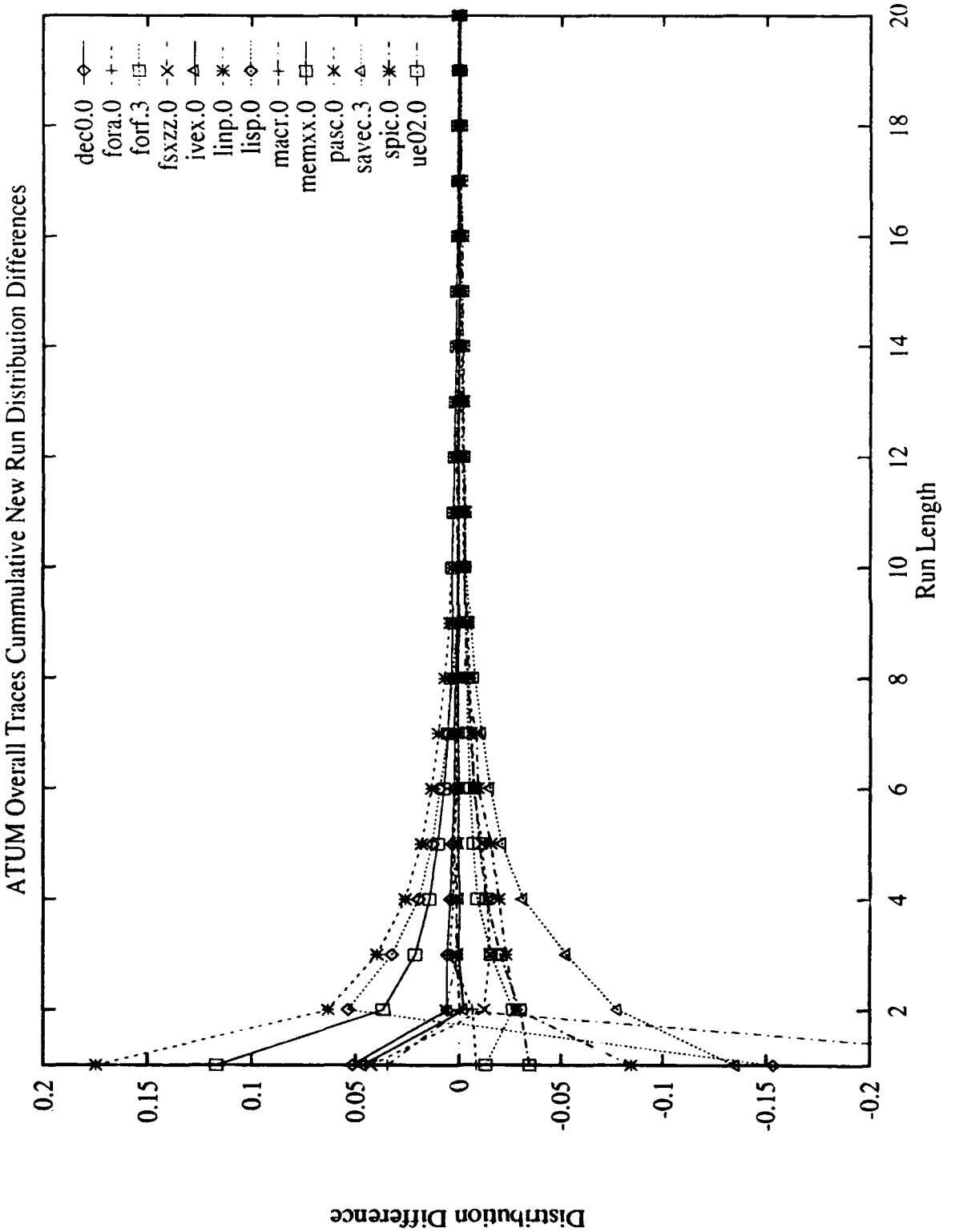


Figure C.16. Differences in the SSD Distributions of the ATUM Instruction Traces

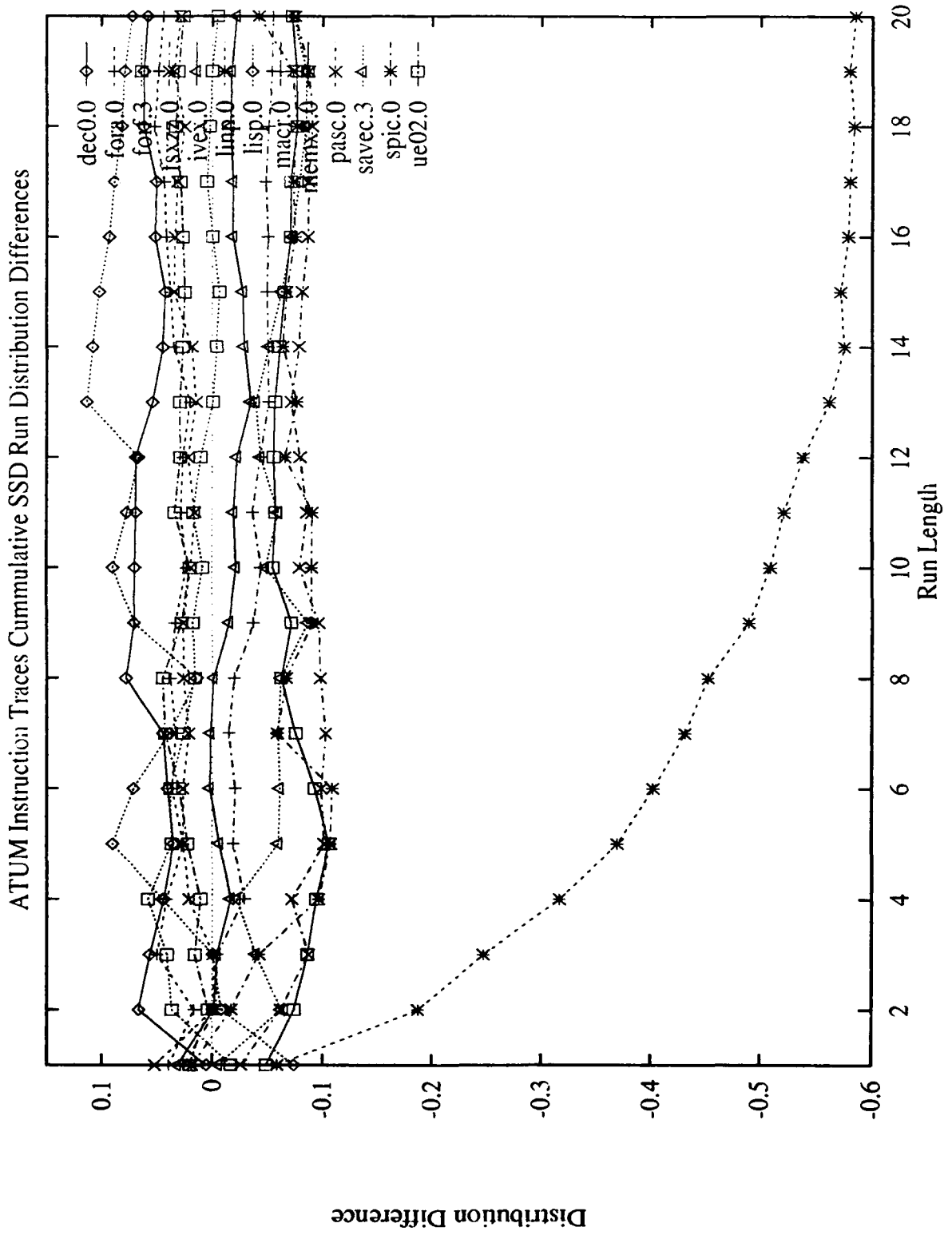


Figure C.17. Differences in the SSD Distributions of the ATUM Read Traces

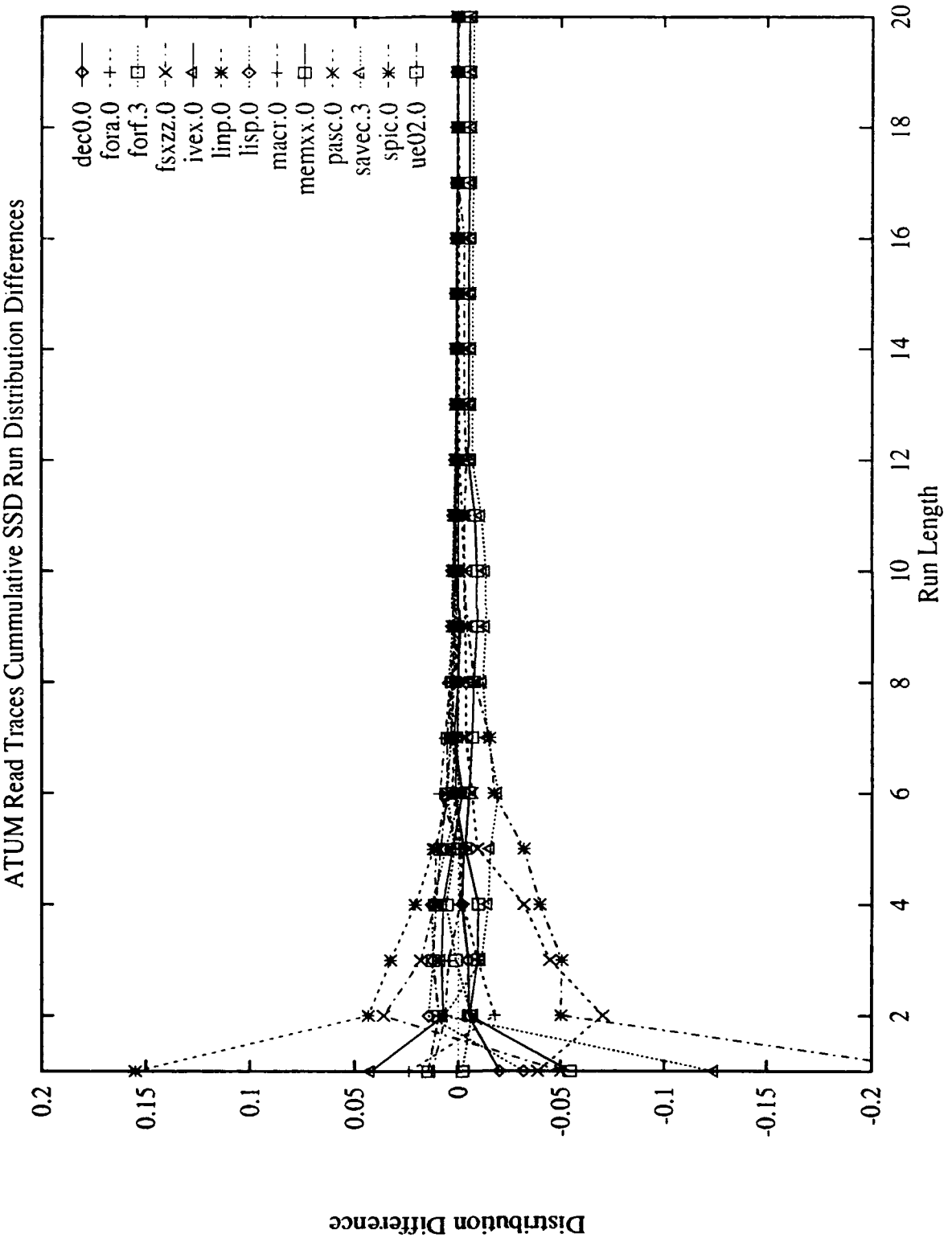


Figure C.18. Differences in the SSD Distributions of the ATUM Write Traces

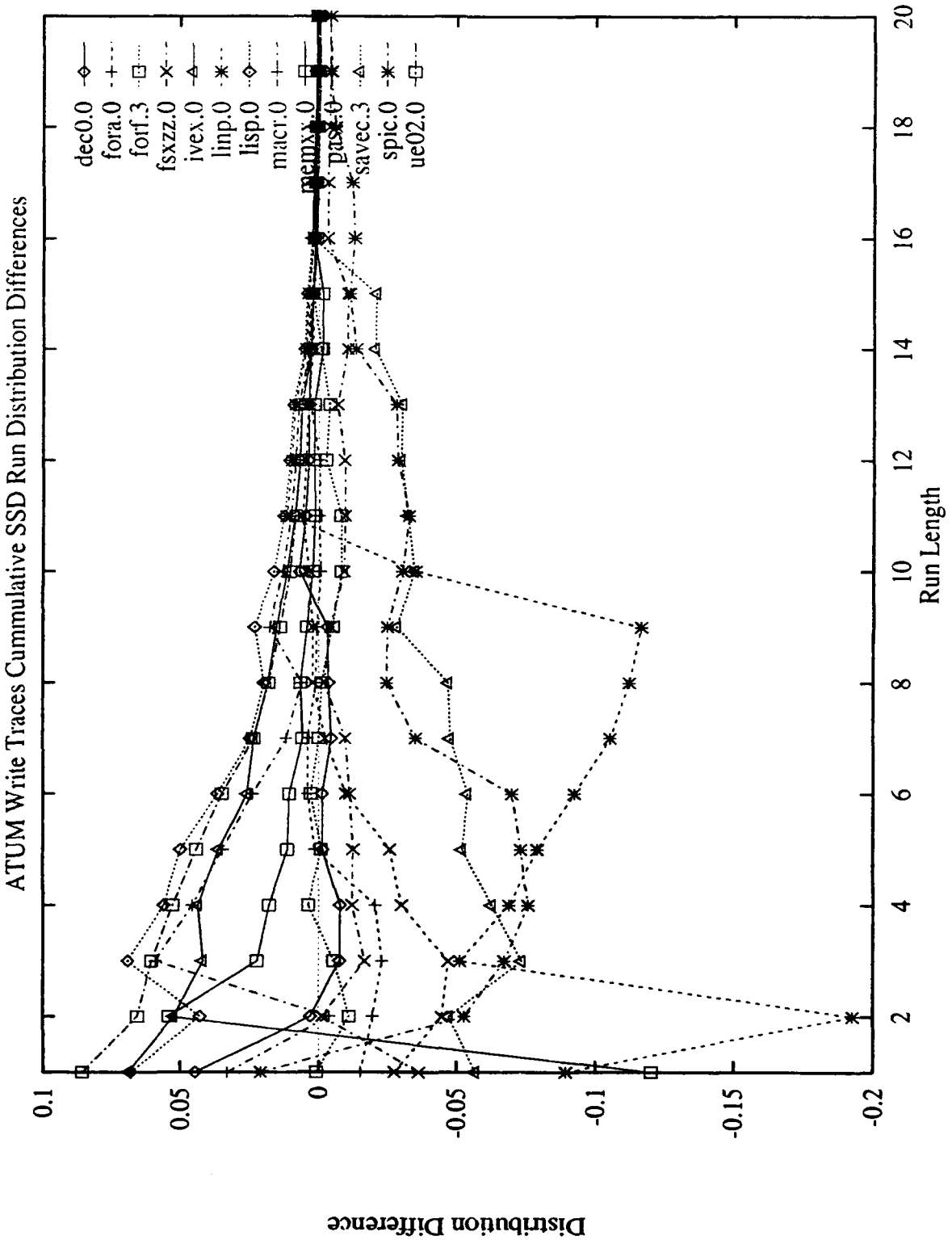


Figure C.19. Differences in the SSD Distributions of the ATUM Data Traces

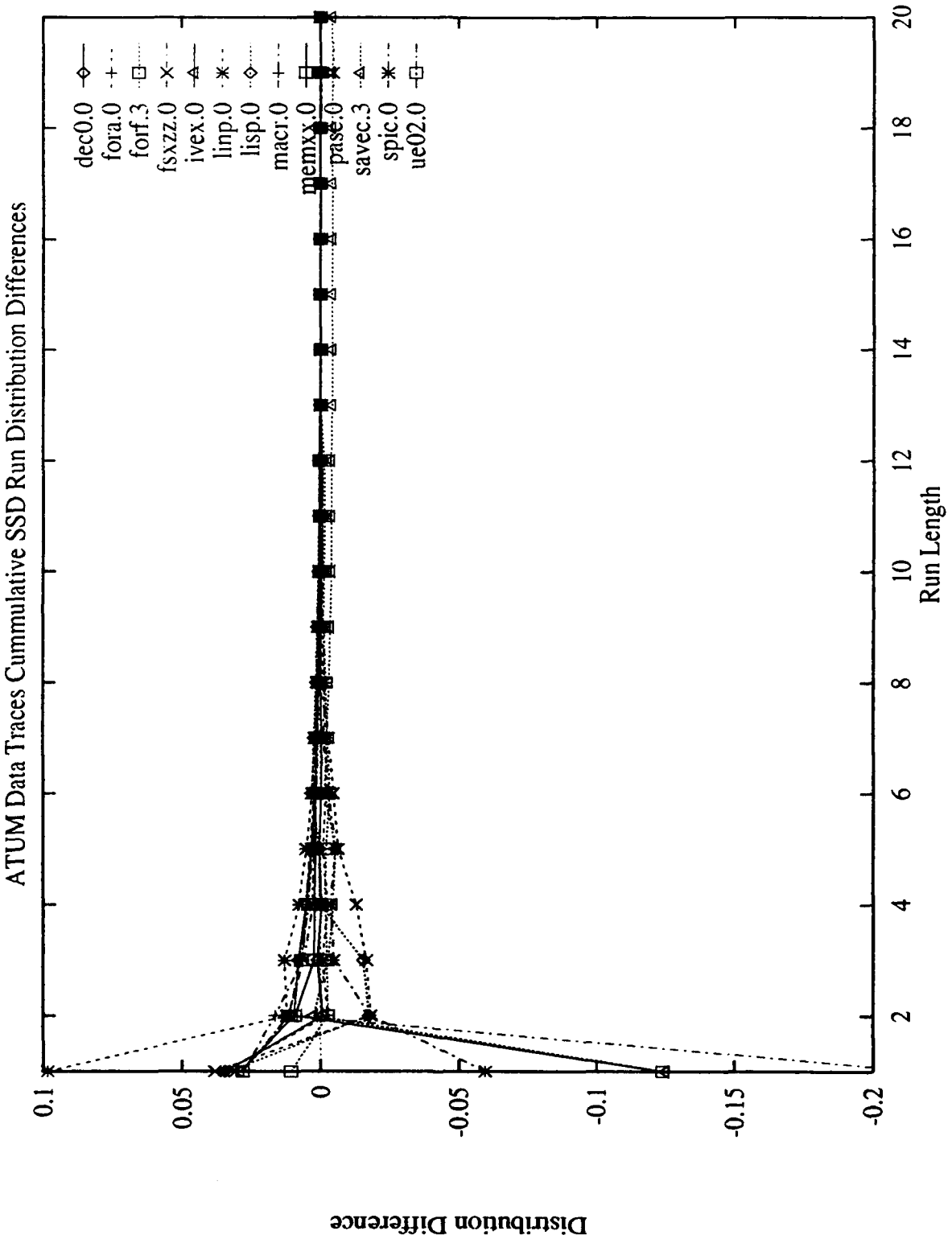
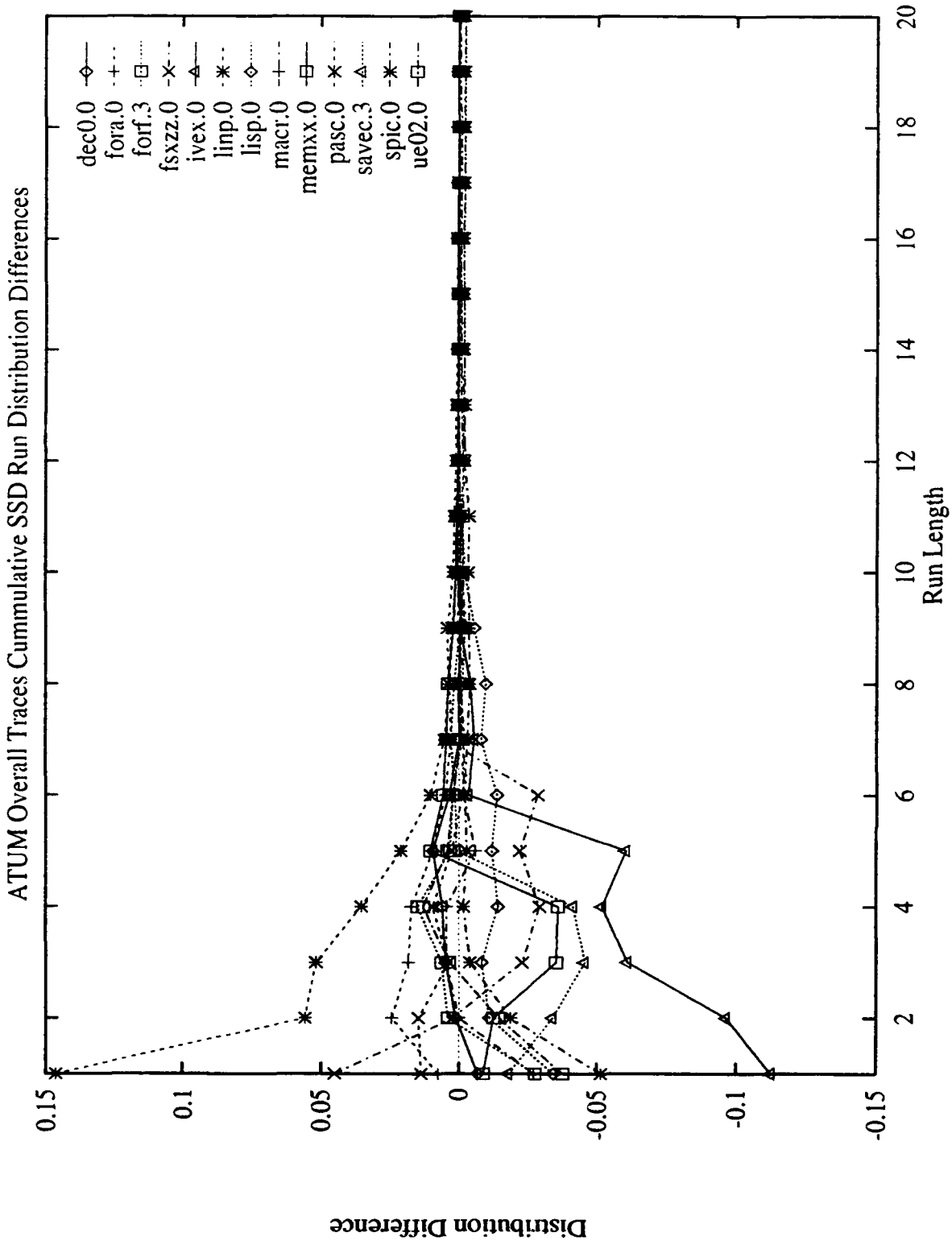


Figure C.20. Differences in the SSD Distributions of the ATUM Overall Traces



Appendix D. *Inter-set Comparision Results*

D.1 Introduction

This is the supplementary data for chapter 4. These curves compare the sample trace distributions from both the ATUM and Explorer Sets with the average distribution of the other set.

Figure D.1. Differences in New Distributions between the ATUM set and some Explorer Instruction Traces

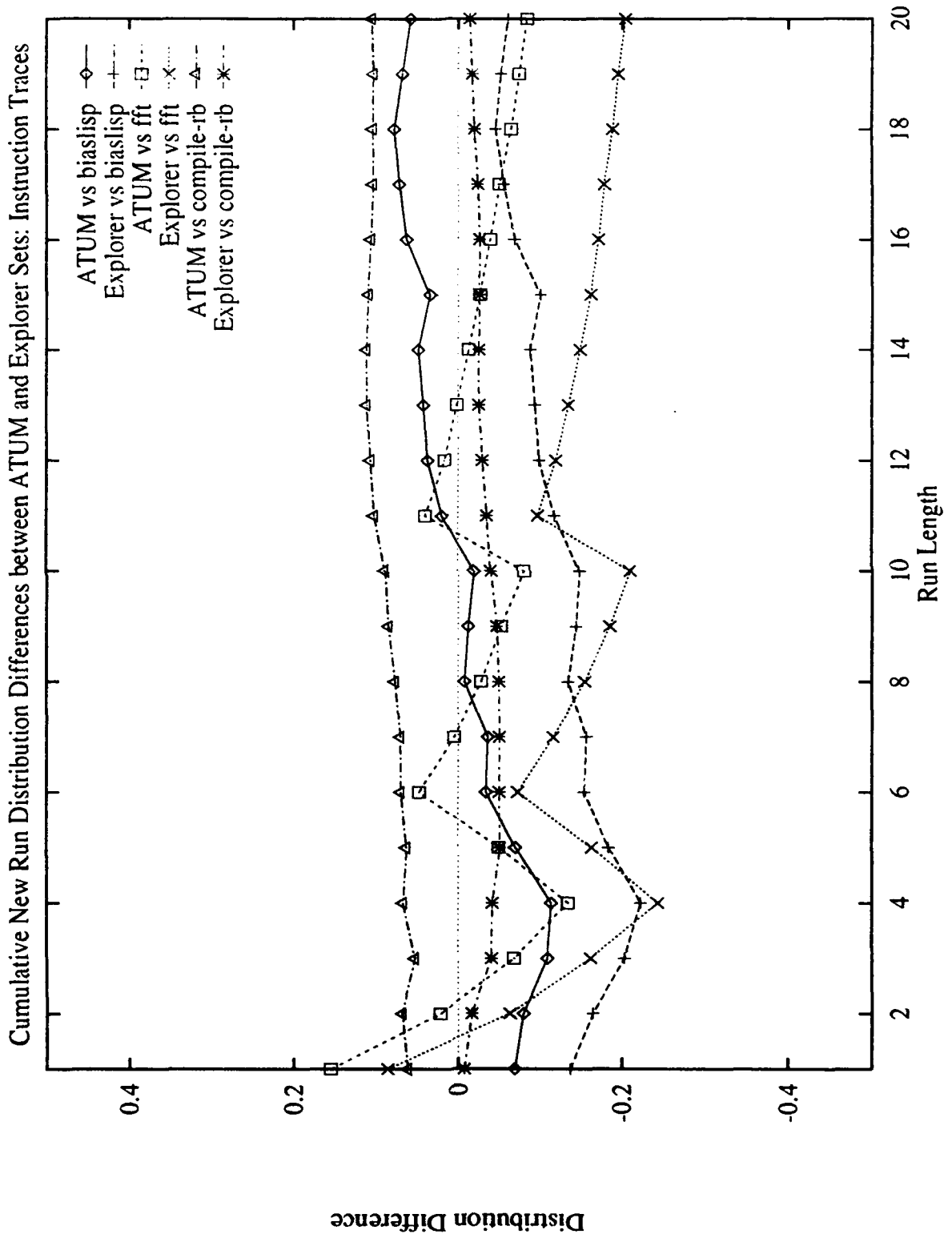


Figure D.2. Differences in New Distributions between the ATUM set and some Explorer Read Traces

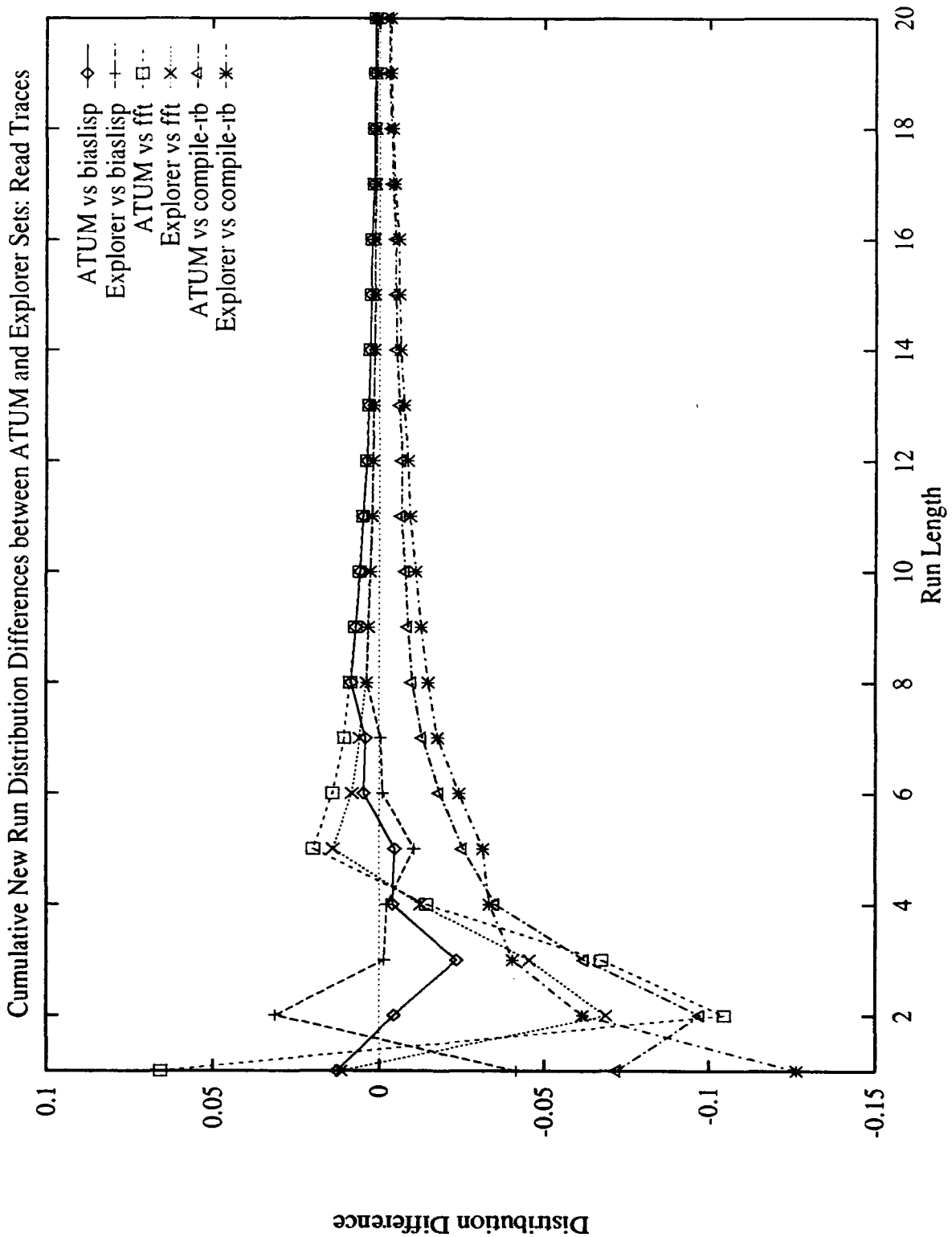


Figure D.3. Differences in New Distributions between the ATUM set and some Explorer Write Traces

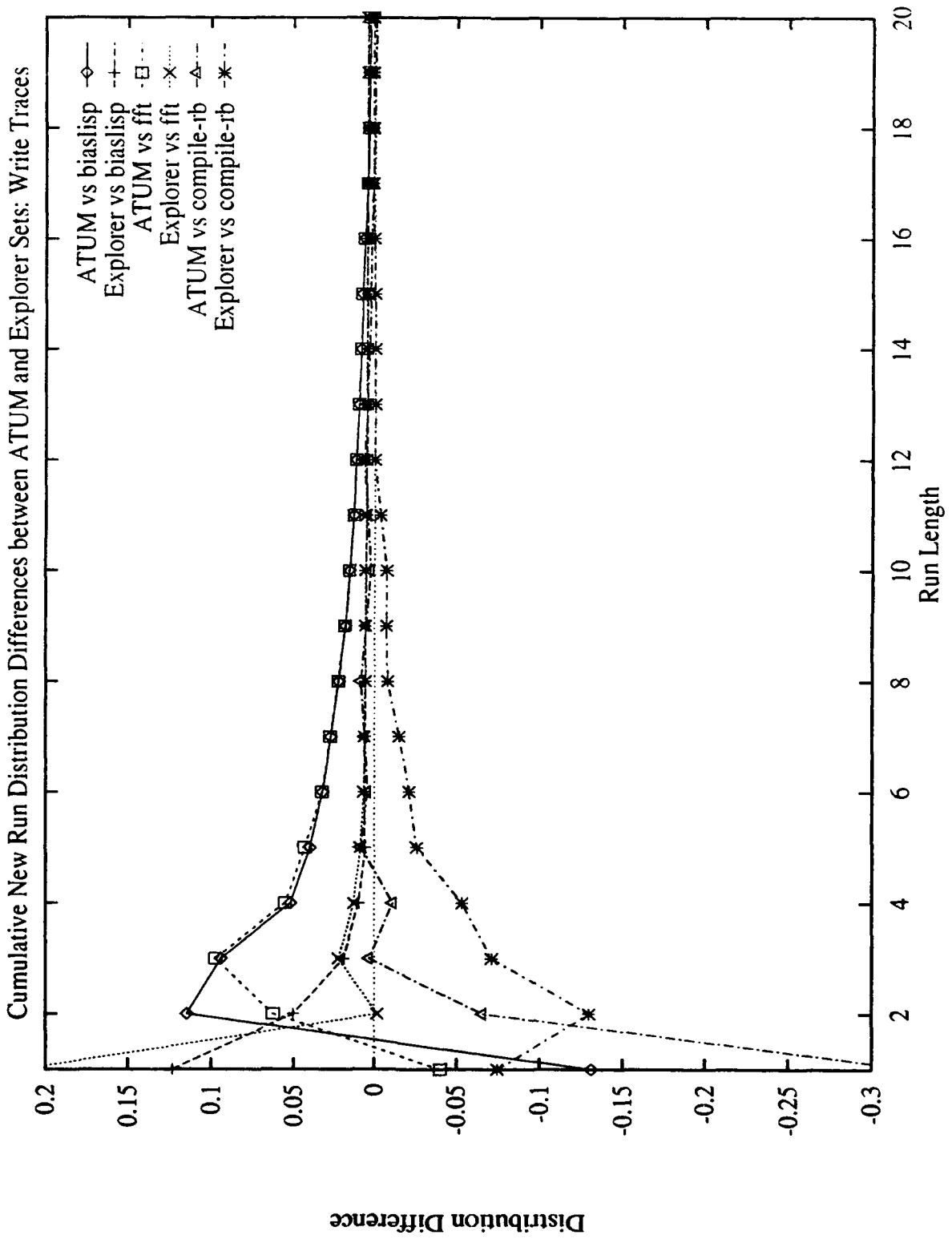


Figure D-4. Differences in New Distributions between the ATUM set and some Explorer Data Traces

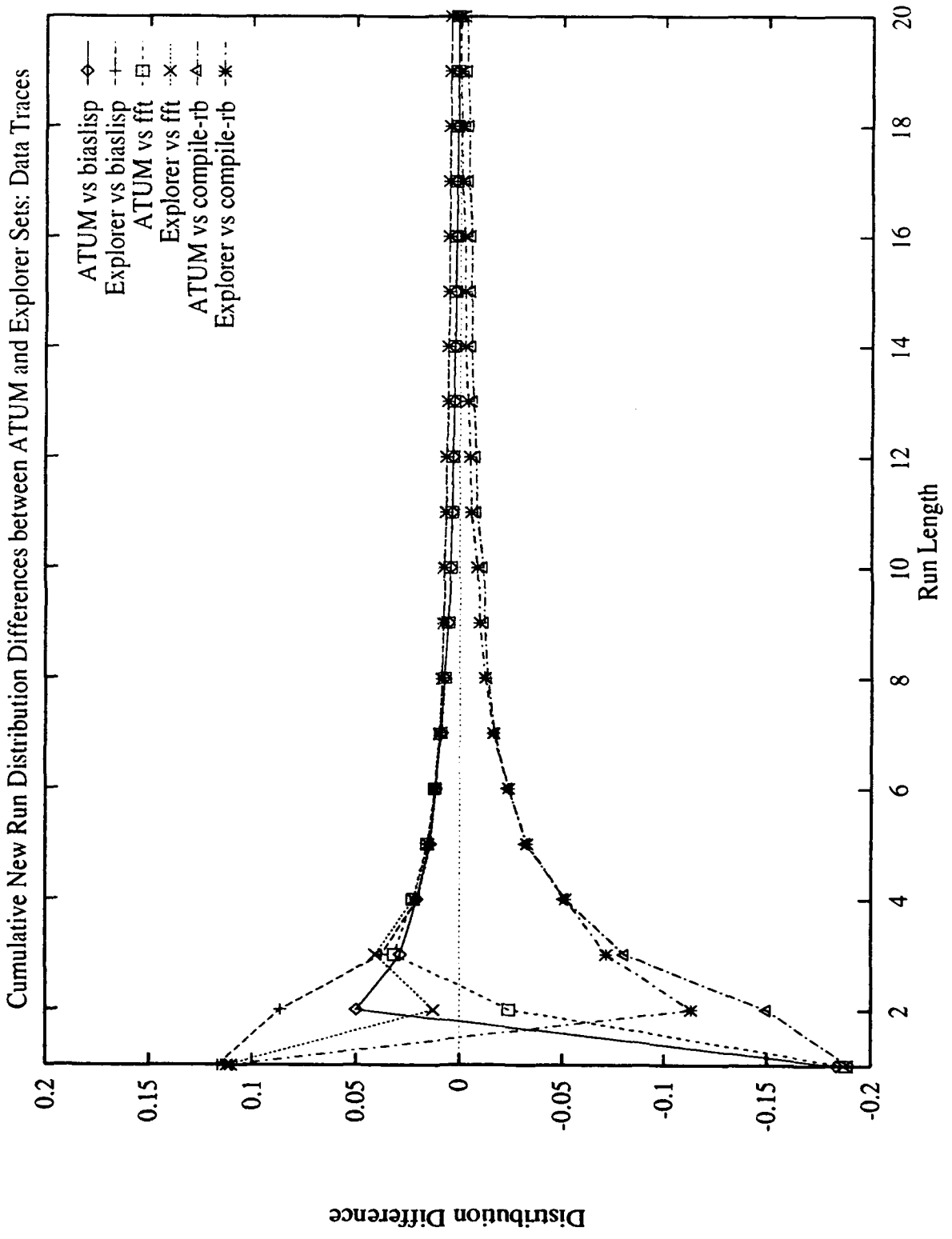


Figure D.5. Differences in New Distributions between the ATUM set and some Explorer Overall Traces

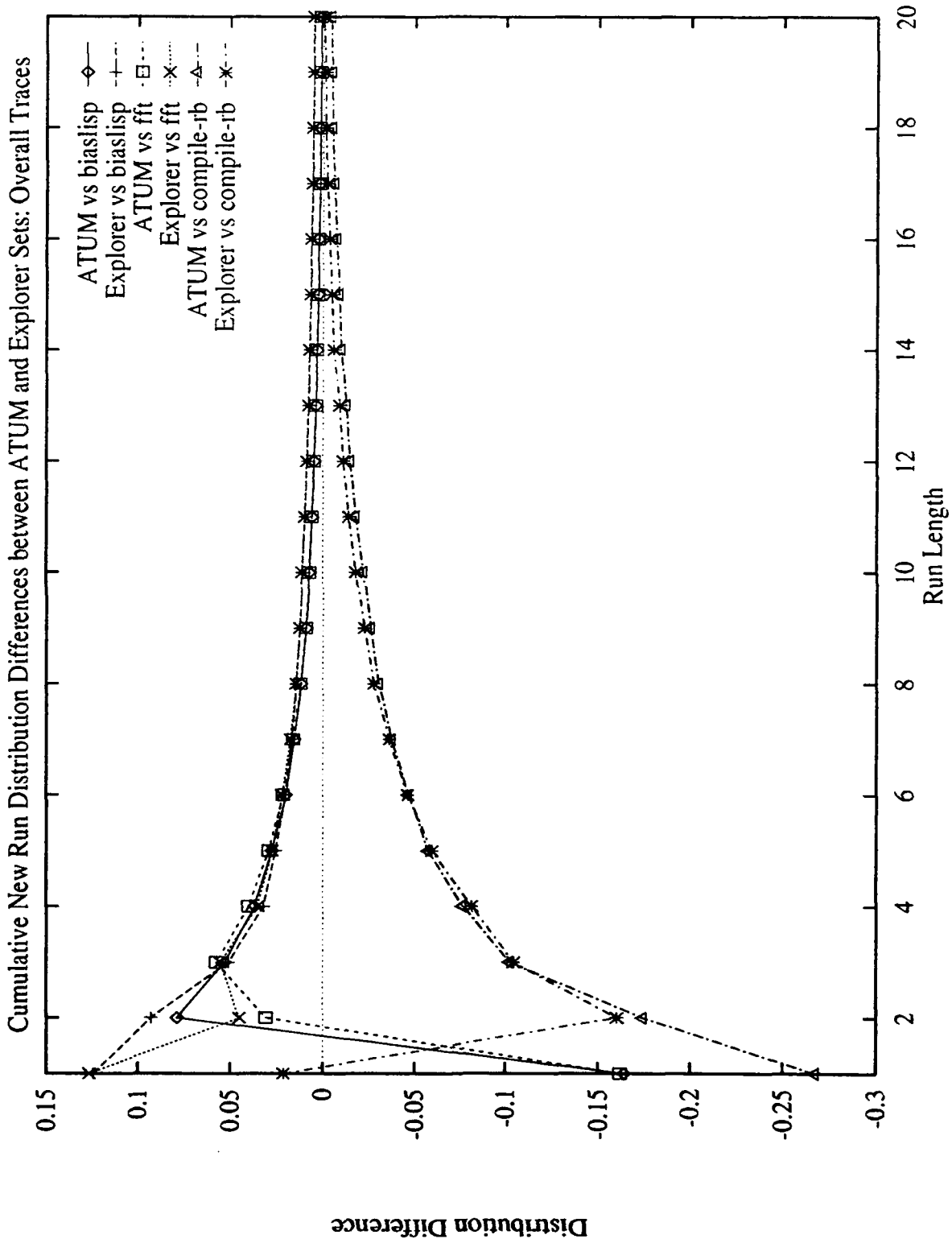


Figure D.6. Differences in SSD Distributions between the ATUM set and some Explorer Instruction Traces

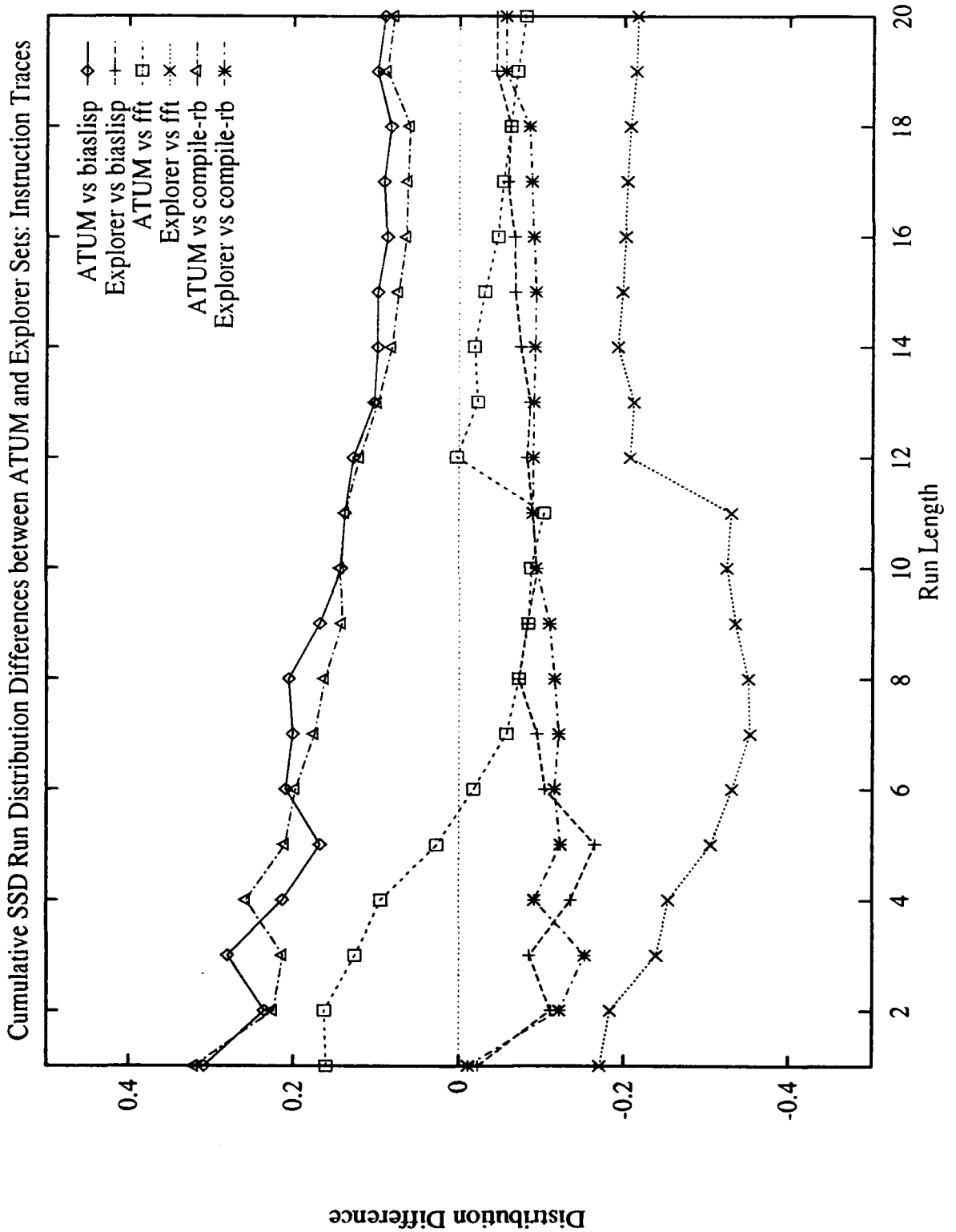


Figure D.7. Differences in SSD Distributions between the ATUM set and some Explorer Read Traces

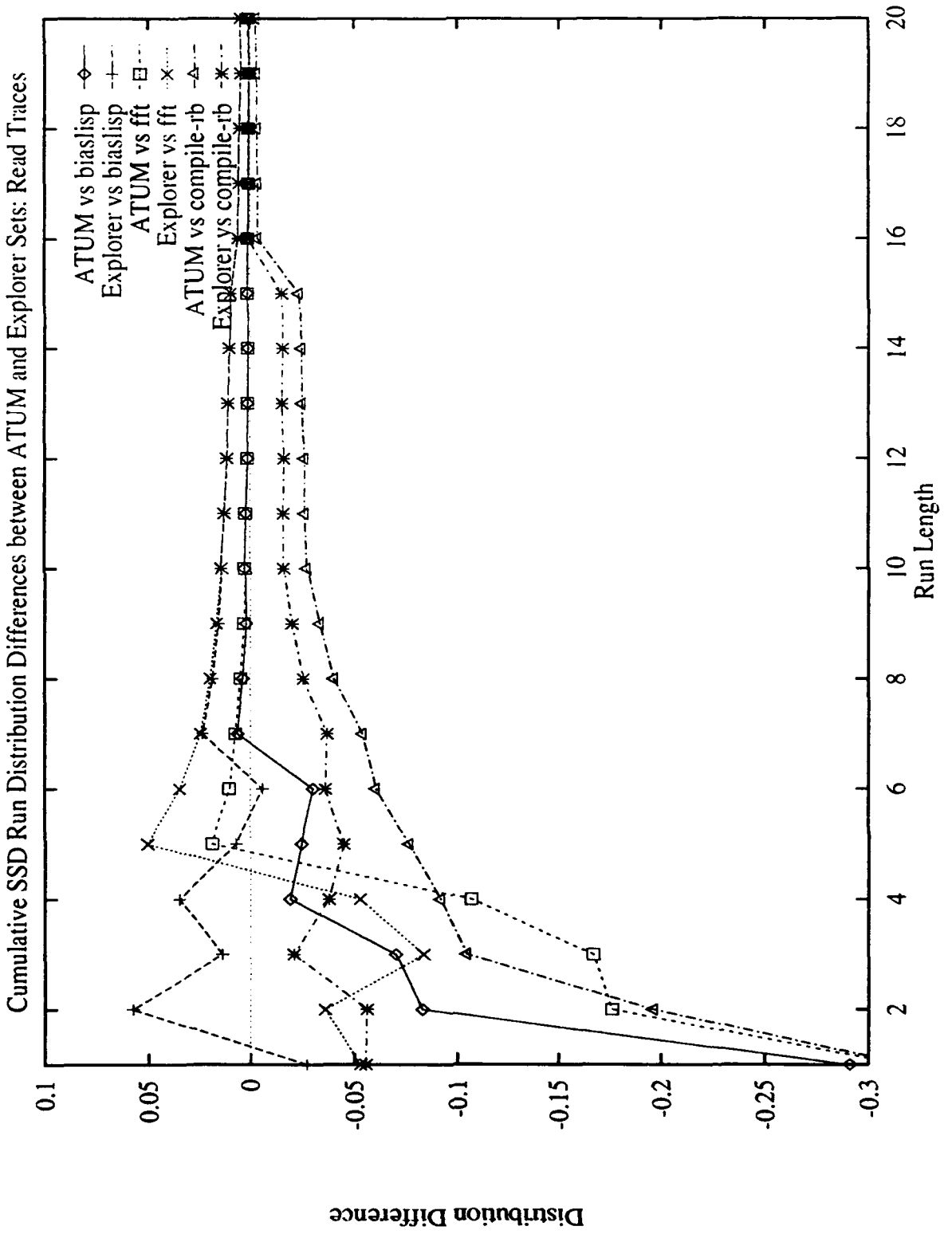


Figure D.8. Differences in SSD Distributions between the ATUM set and some Explorer Write Traces

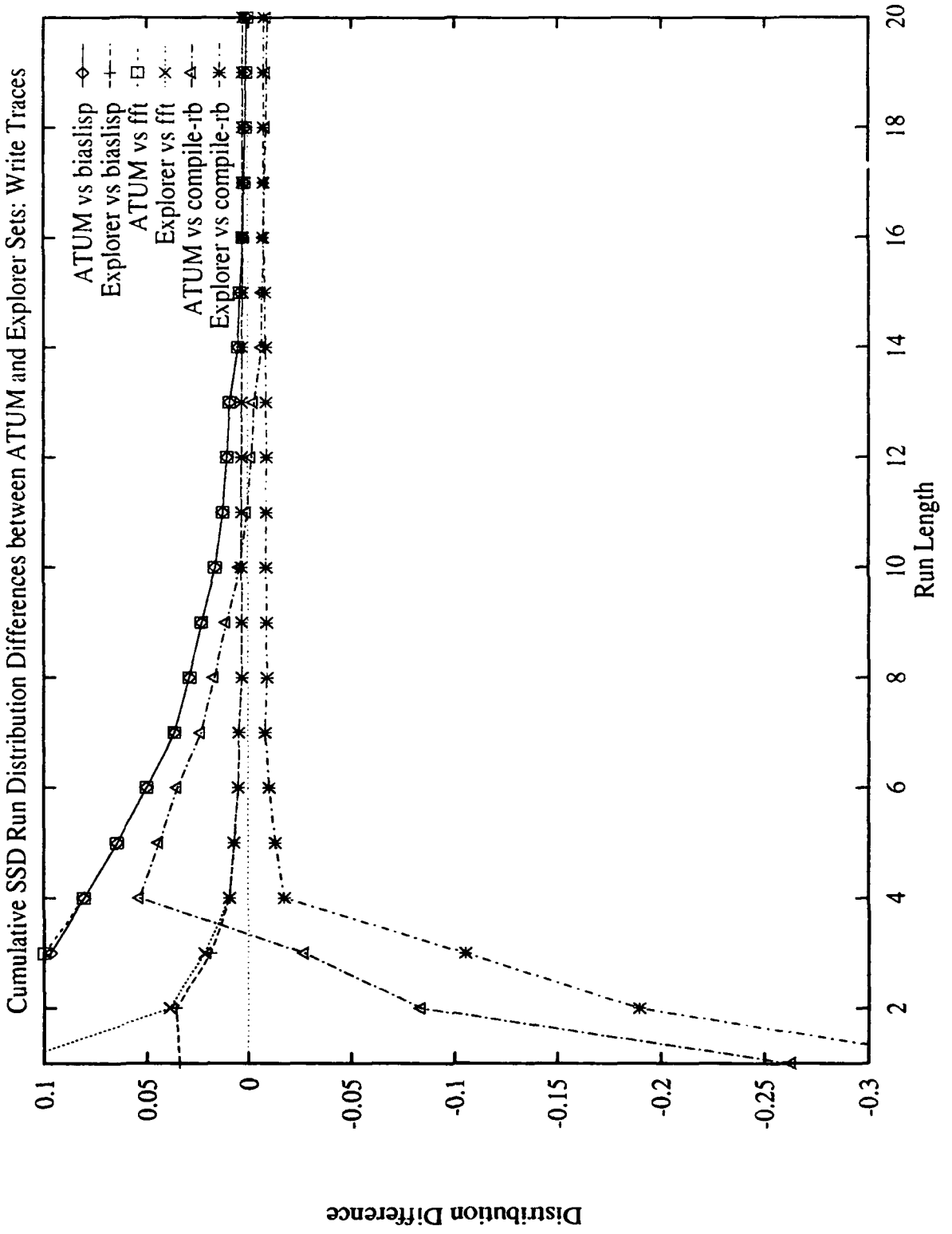


Figure D.9. Differences in SSD Distributions between the ATUM set and some Explorer Data Traces

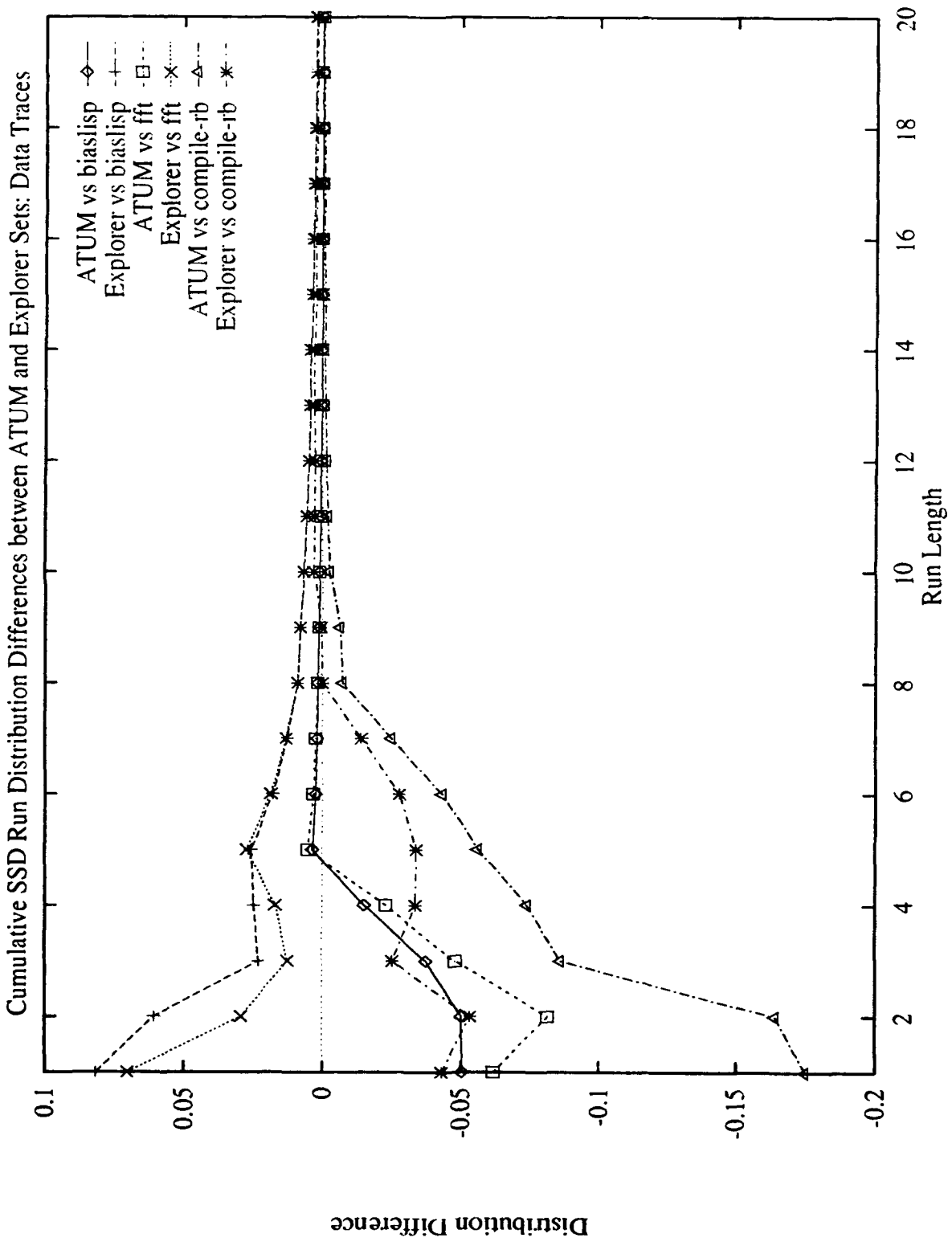


Figure D.10. Differences in SSD Distributions between the ATUM set and some Explorer Overall Traces

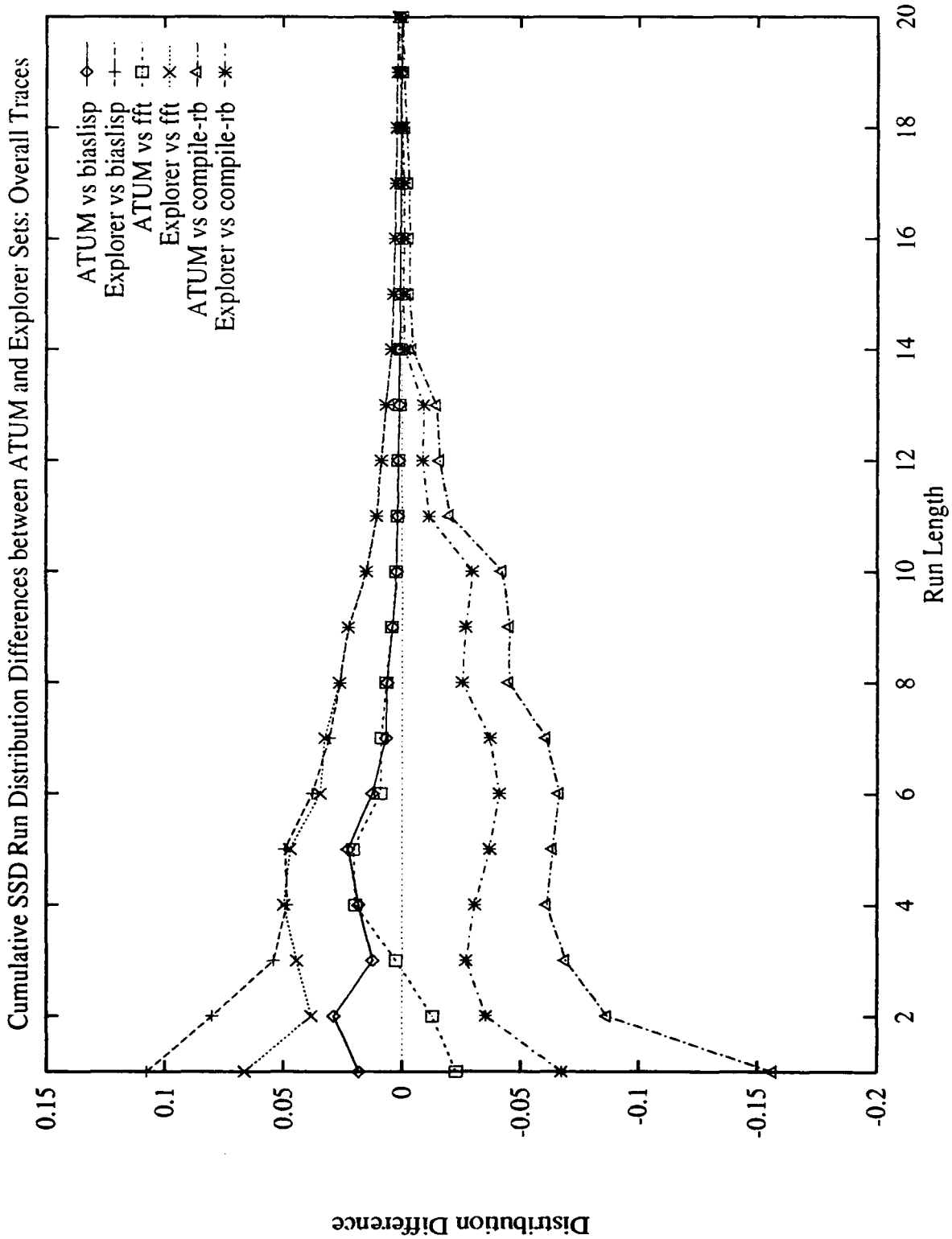


Figure D.11. Differences in New Distributions between the Explorer set and some ATUM Instruction Traces

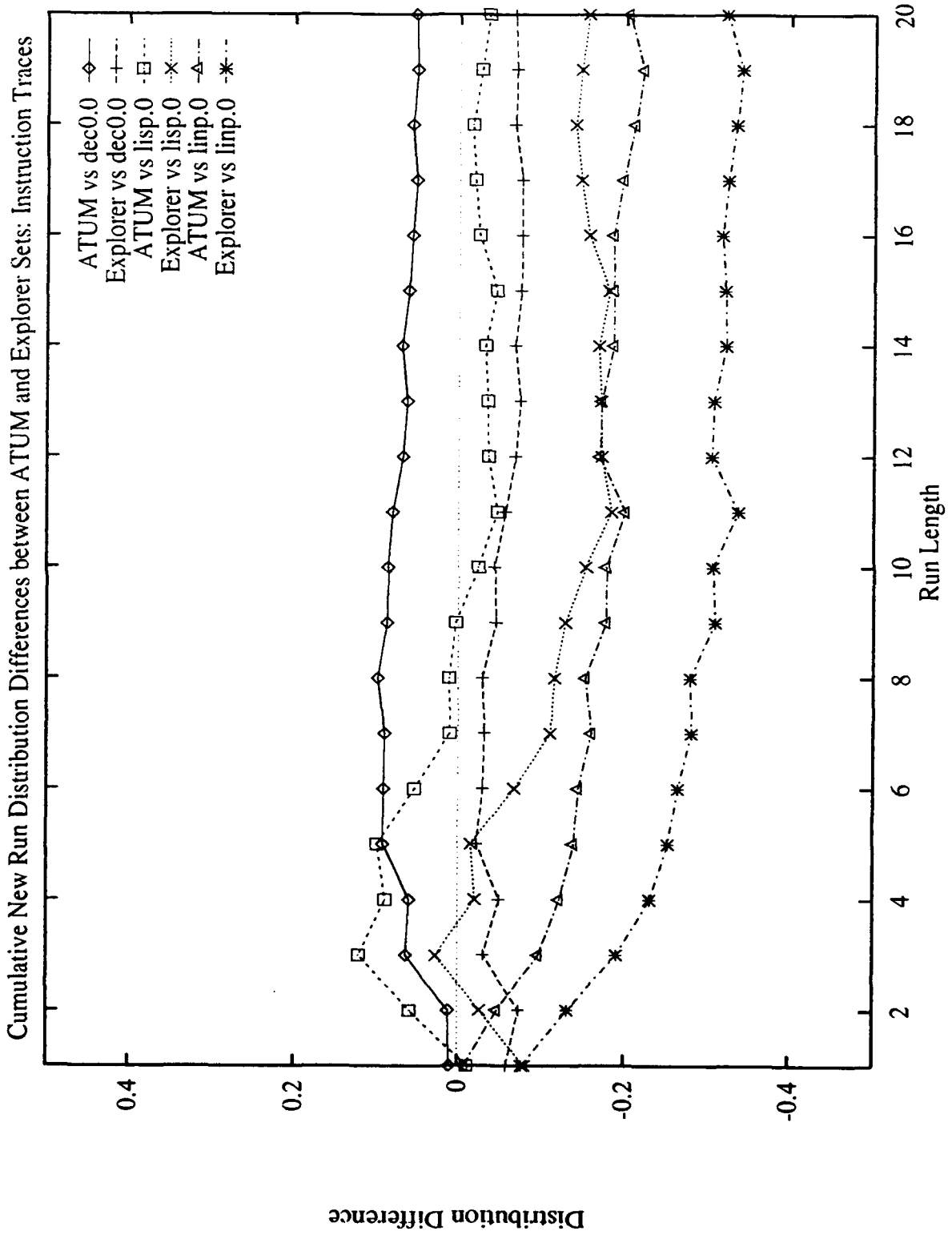


Figure D.12. Differences in New Distributions between the Explorer set and some ATUM Read Traces

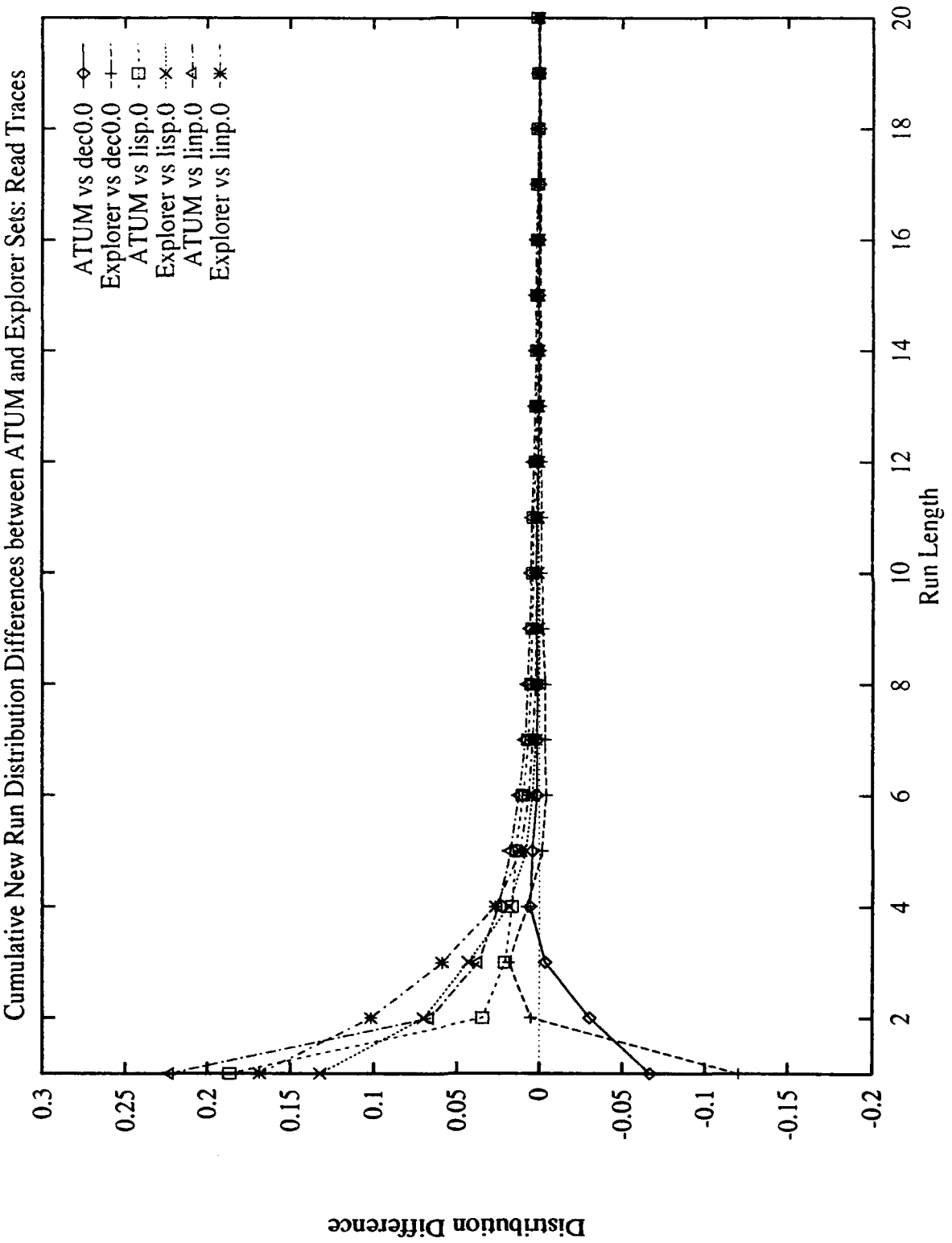


Figure D.13. Differences in New Distributions between the Explorer set and some ATUM Write Traces

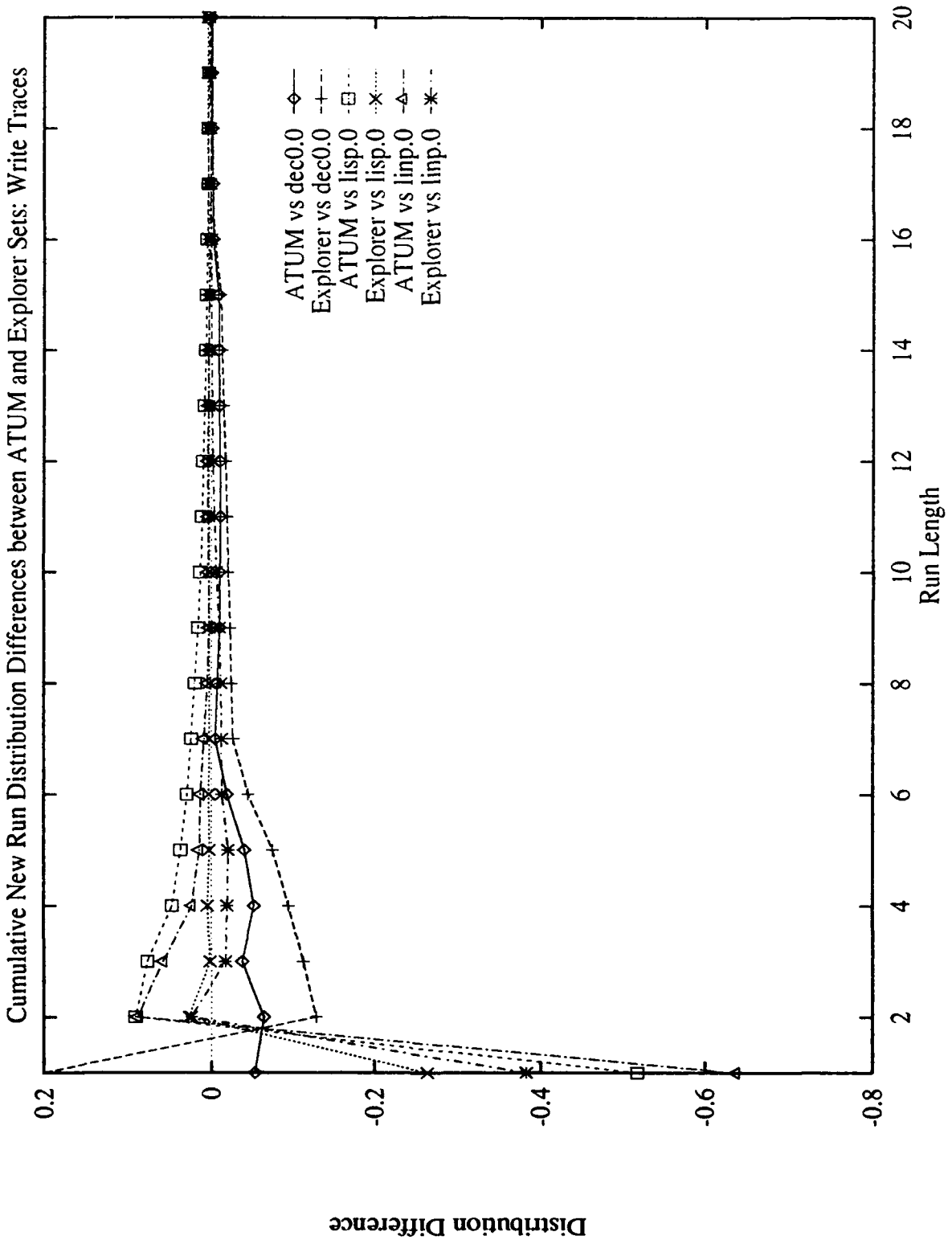


Figure D.14. Differences in New Distributions between the Explorer set and some ATUM Data Traces

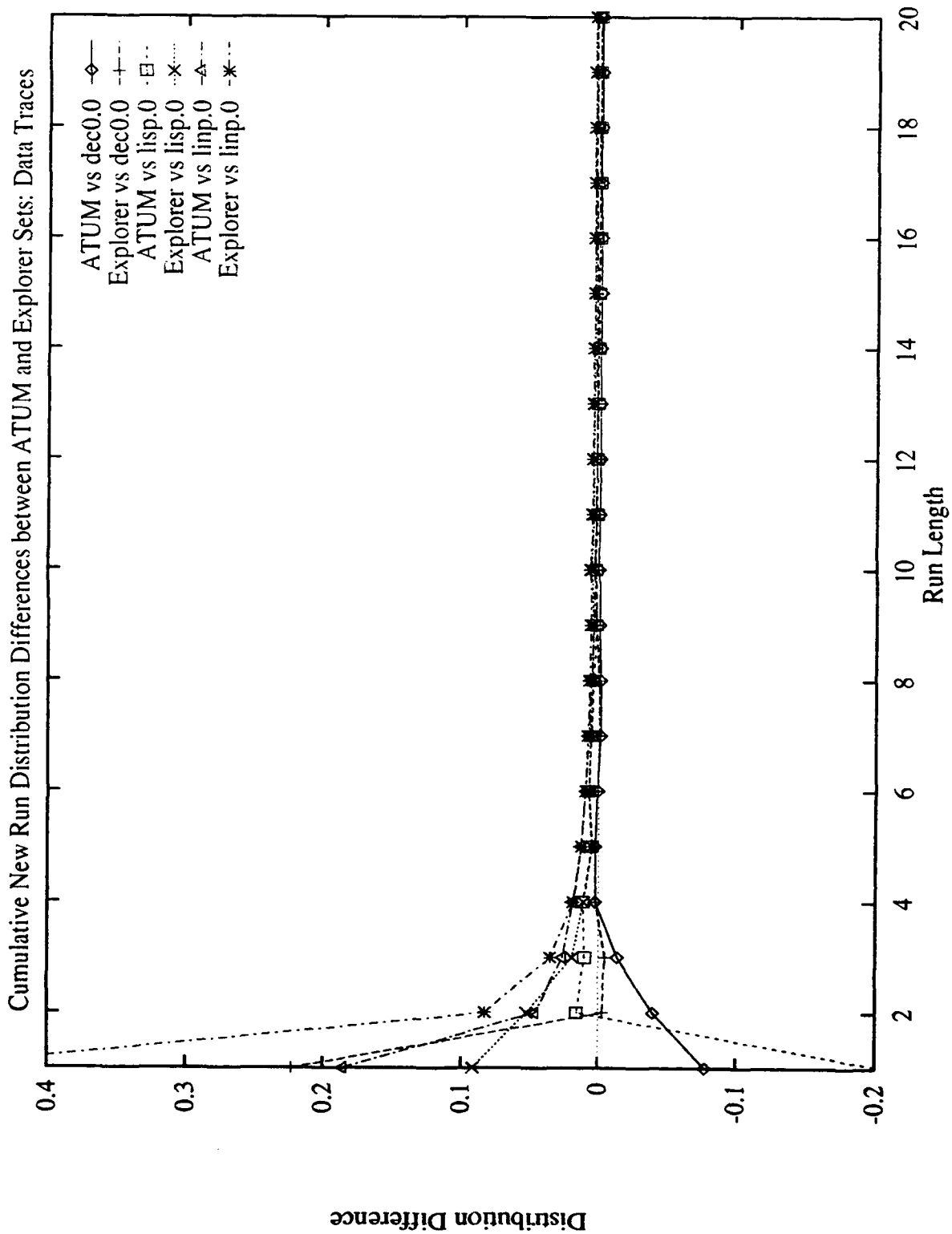


Figure D.15. Differences in New Distributions between the Explorer set and some ATUM Overall Traces

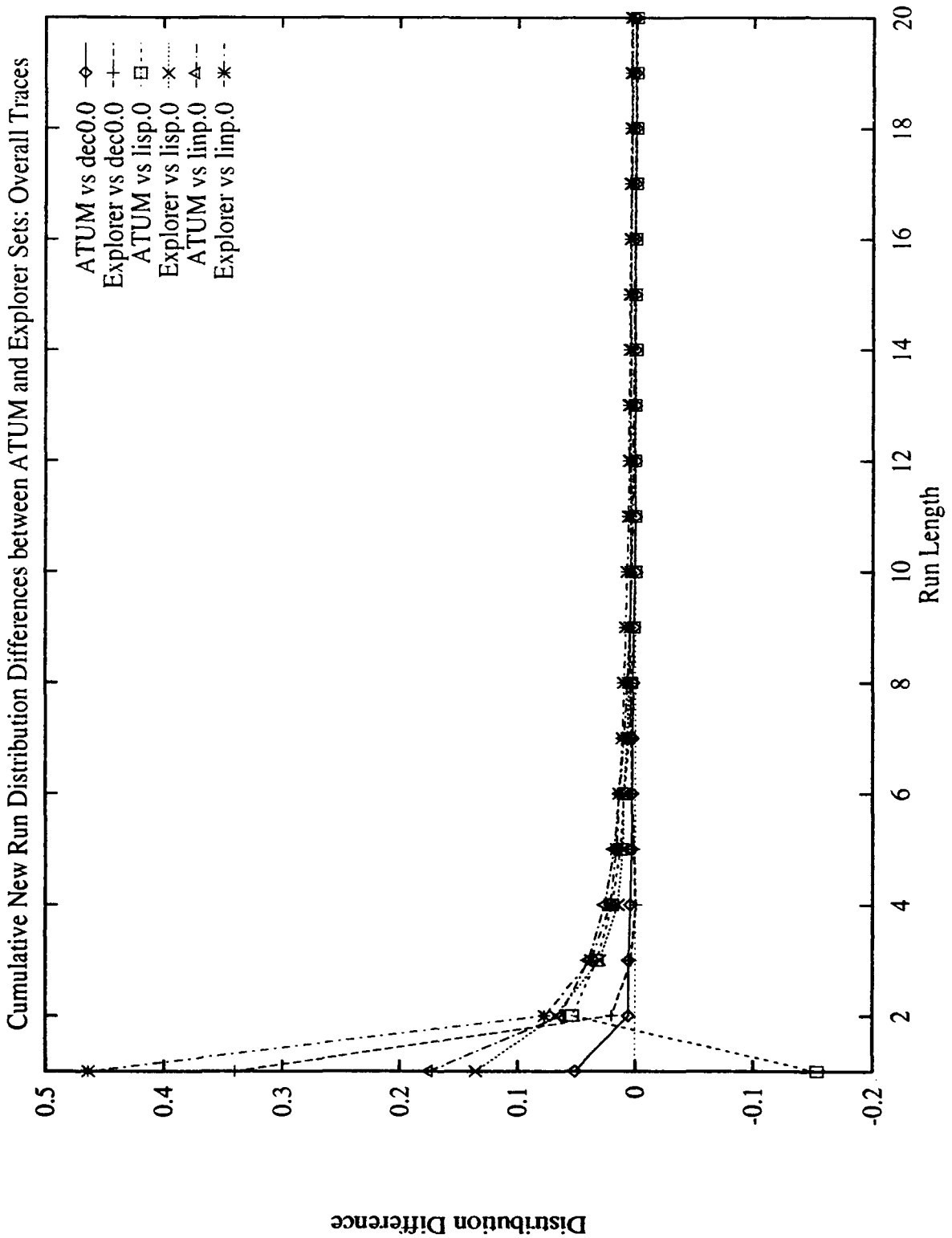


Figure D.16. Differences in SSD Distributions between the Explorer set and some ATUM Instruction Traces

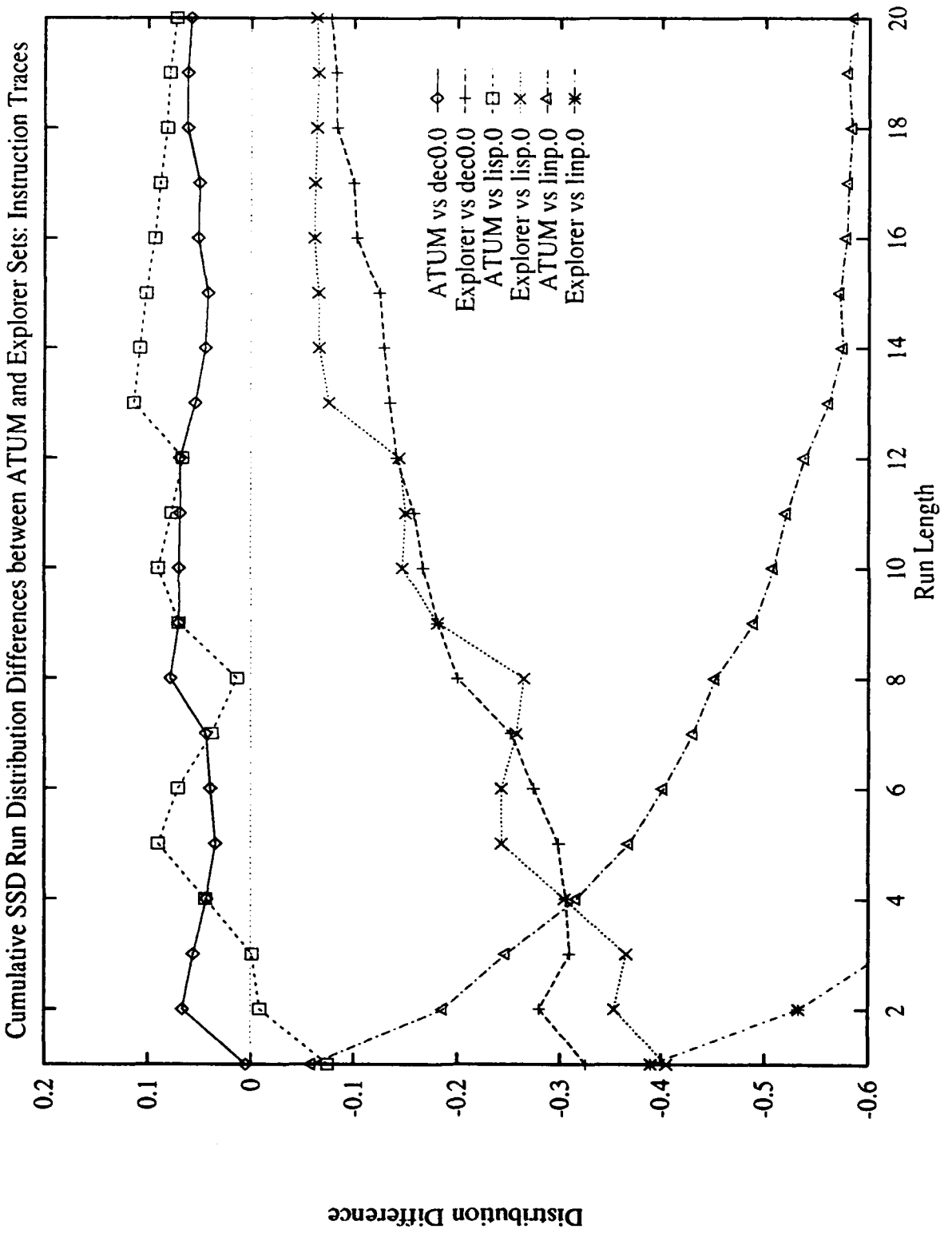


Figure D.17. Differences in SSD Distributions between the Explorer set and some ATUM Read Traces

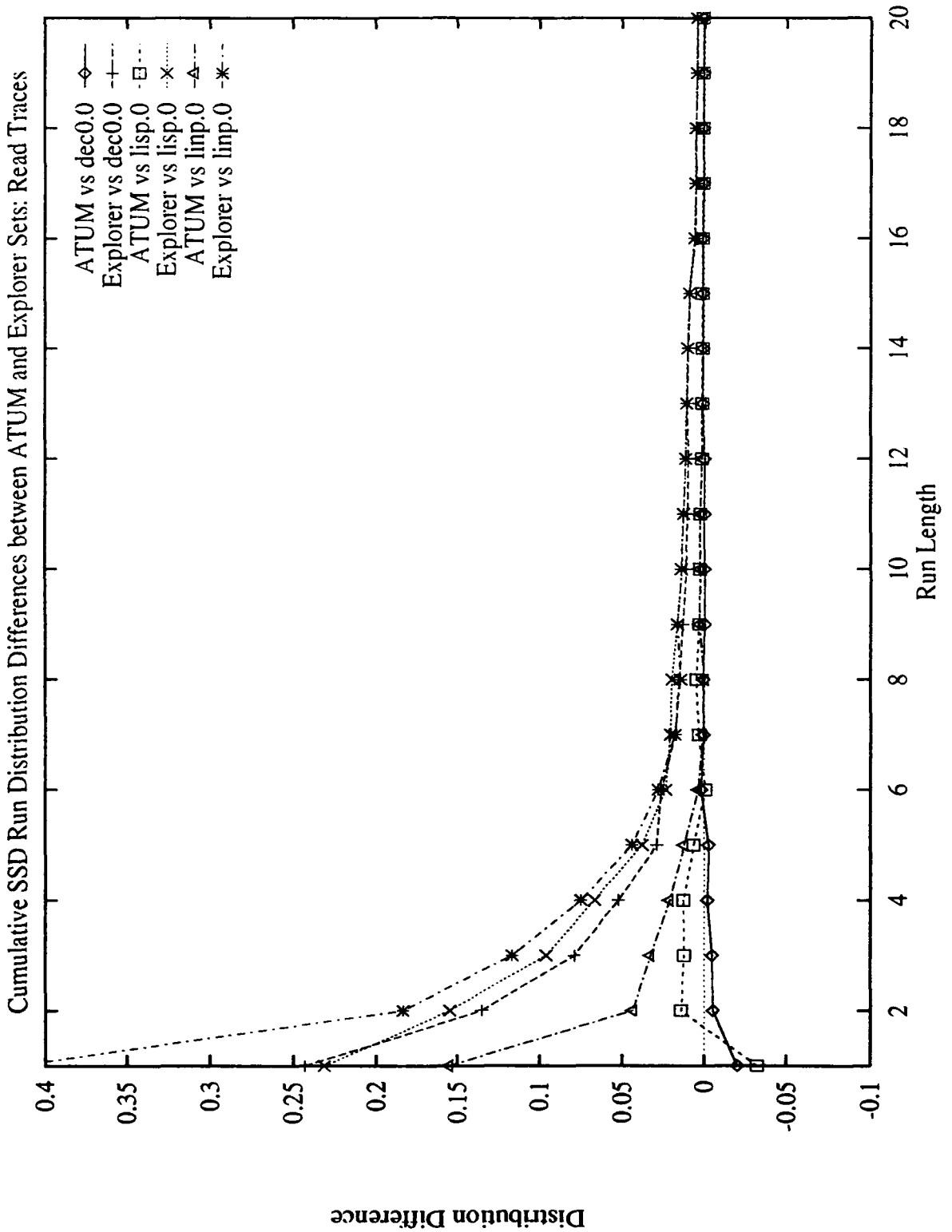


Figure D.18. Differences in SSD Distributions between the Explorer set and some ATUM Write Traces

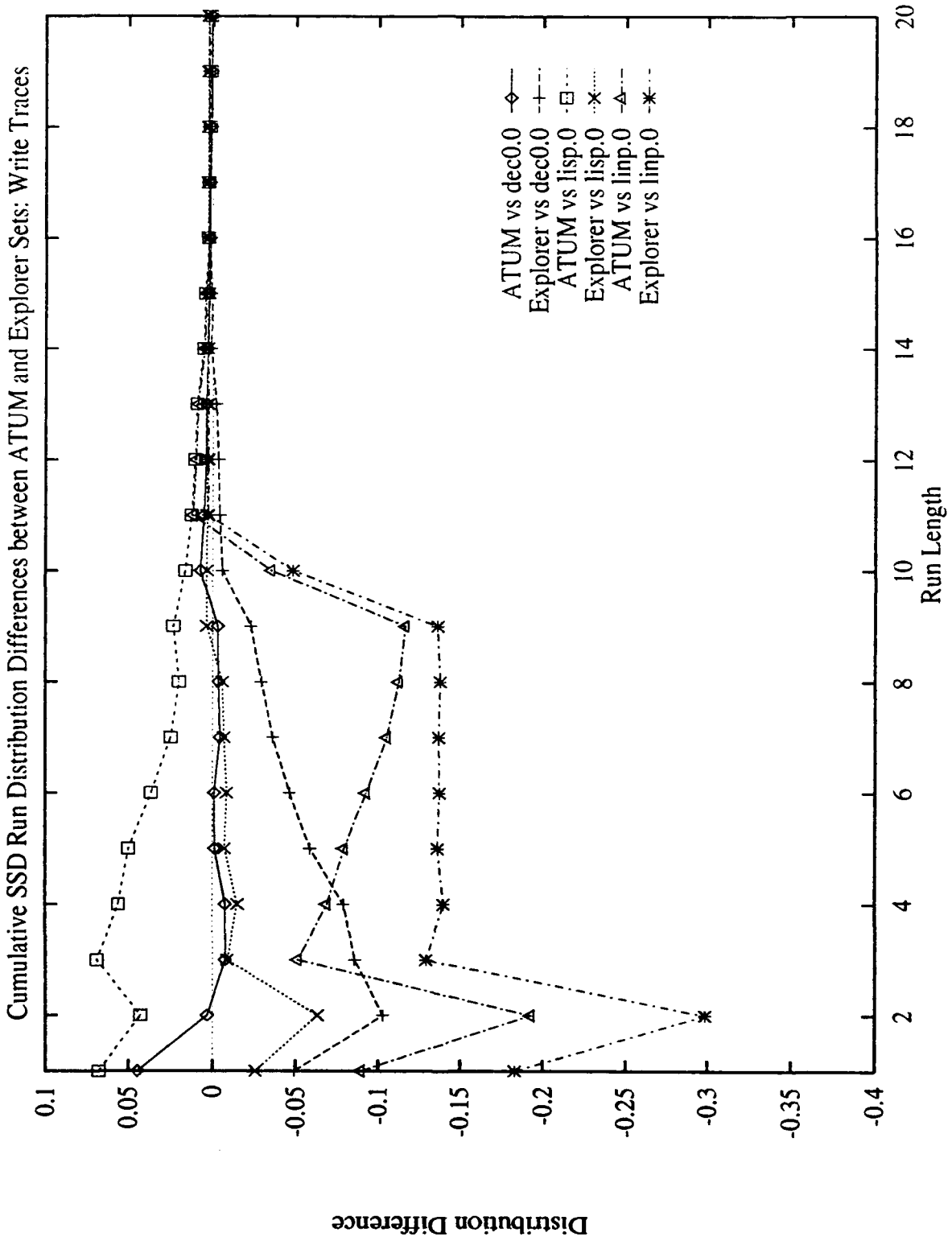


Figure D.19. Differences in SSD Distributions between the Explorer set and some ATUM Data Traces

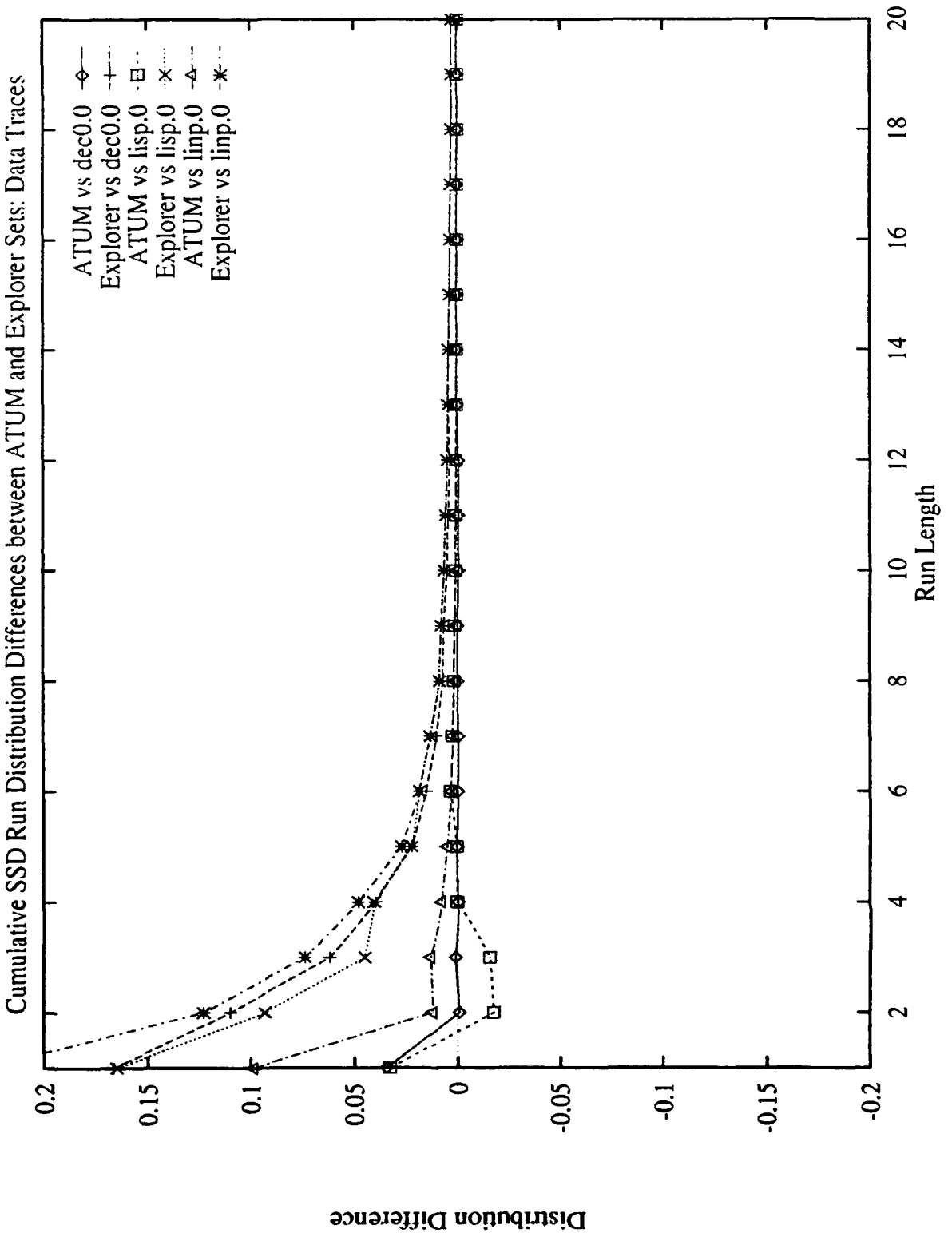
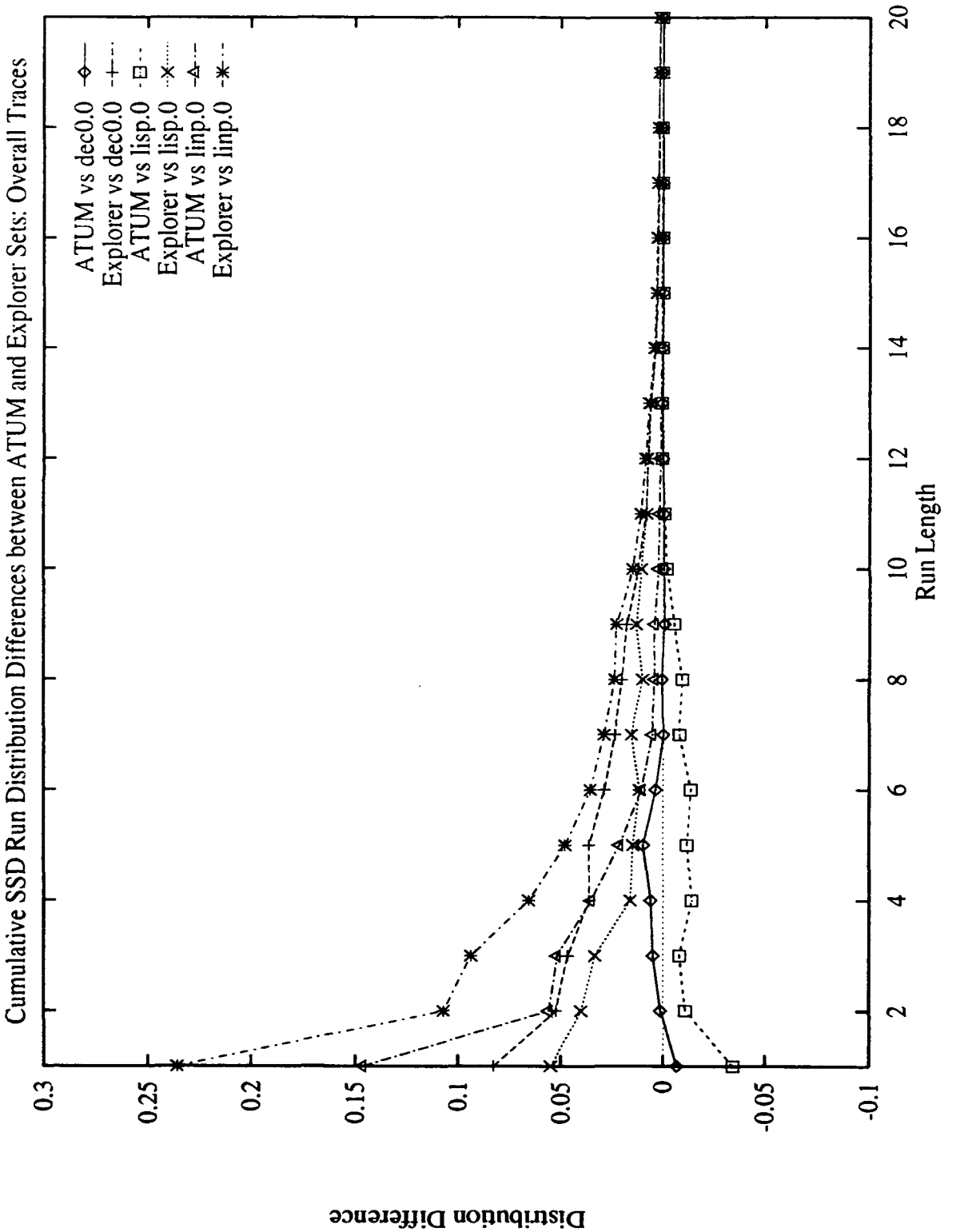


Figure D.20. Differences in SSD Distributions between the Explorer set and some ATUM Overall Traces



Appendix E. *Intra-Trace Results*

E.1 Introduction

This is the supplementary data for chapter 4. These graphs show the intra-trace locality measurements of the Explorer traces.

Figure E.1. Chi-Square Probability versus Bin Size : BIASLisp

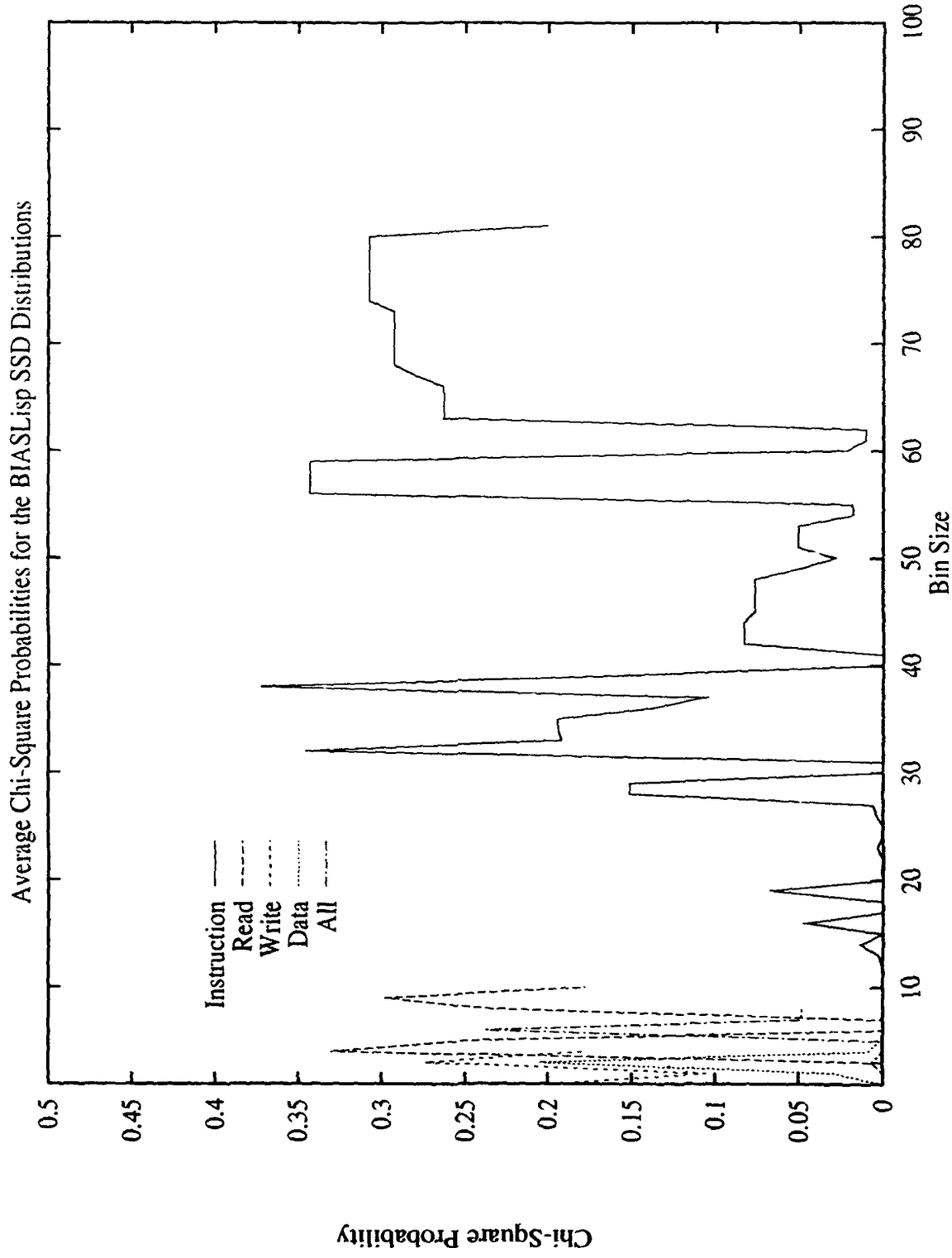


Figure E.2. Chi-Square Probability versus Bin Size : Boyer

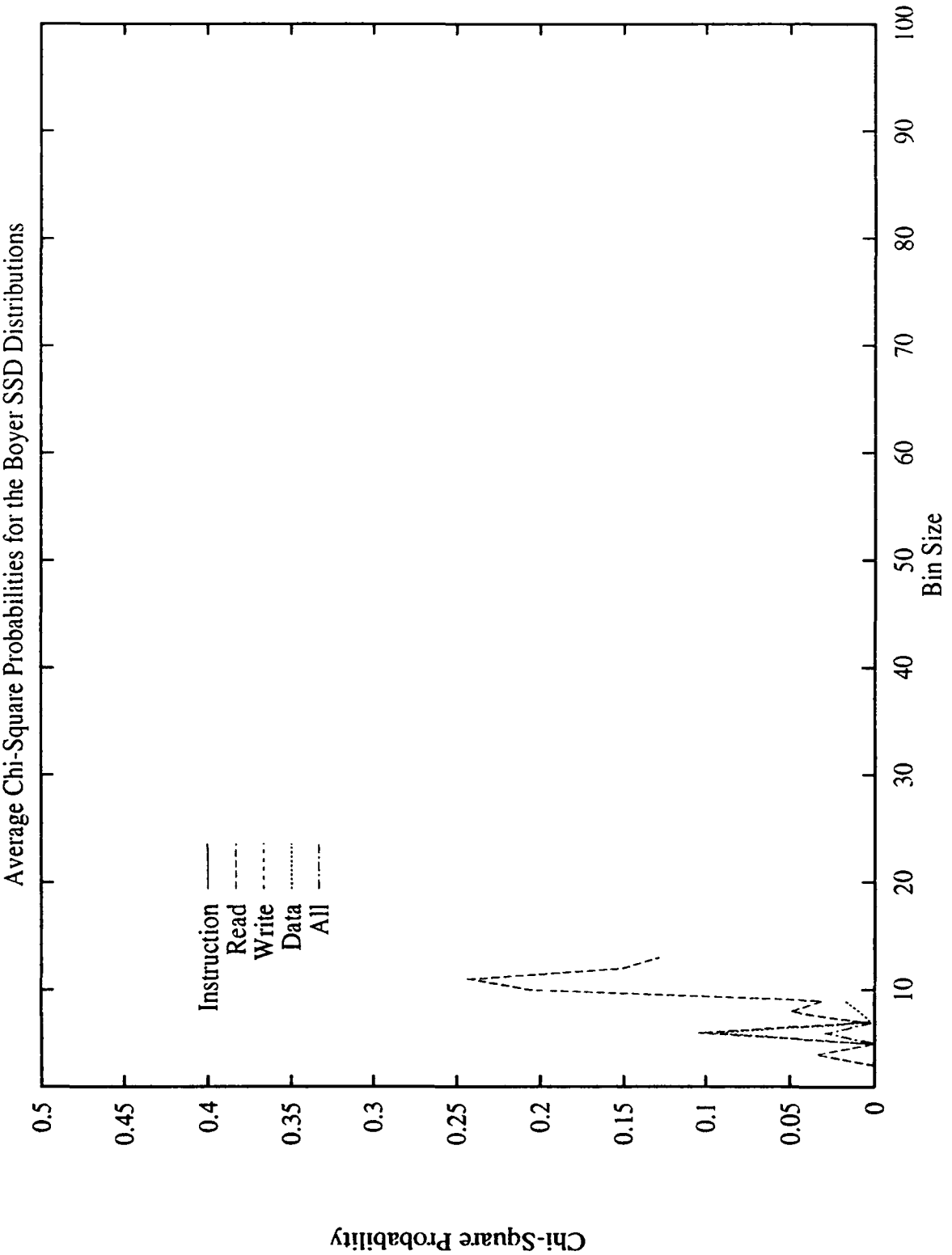


Figure E.3. Chi-Square Probability versus Bin Size : Compile

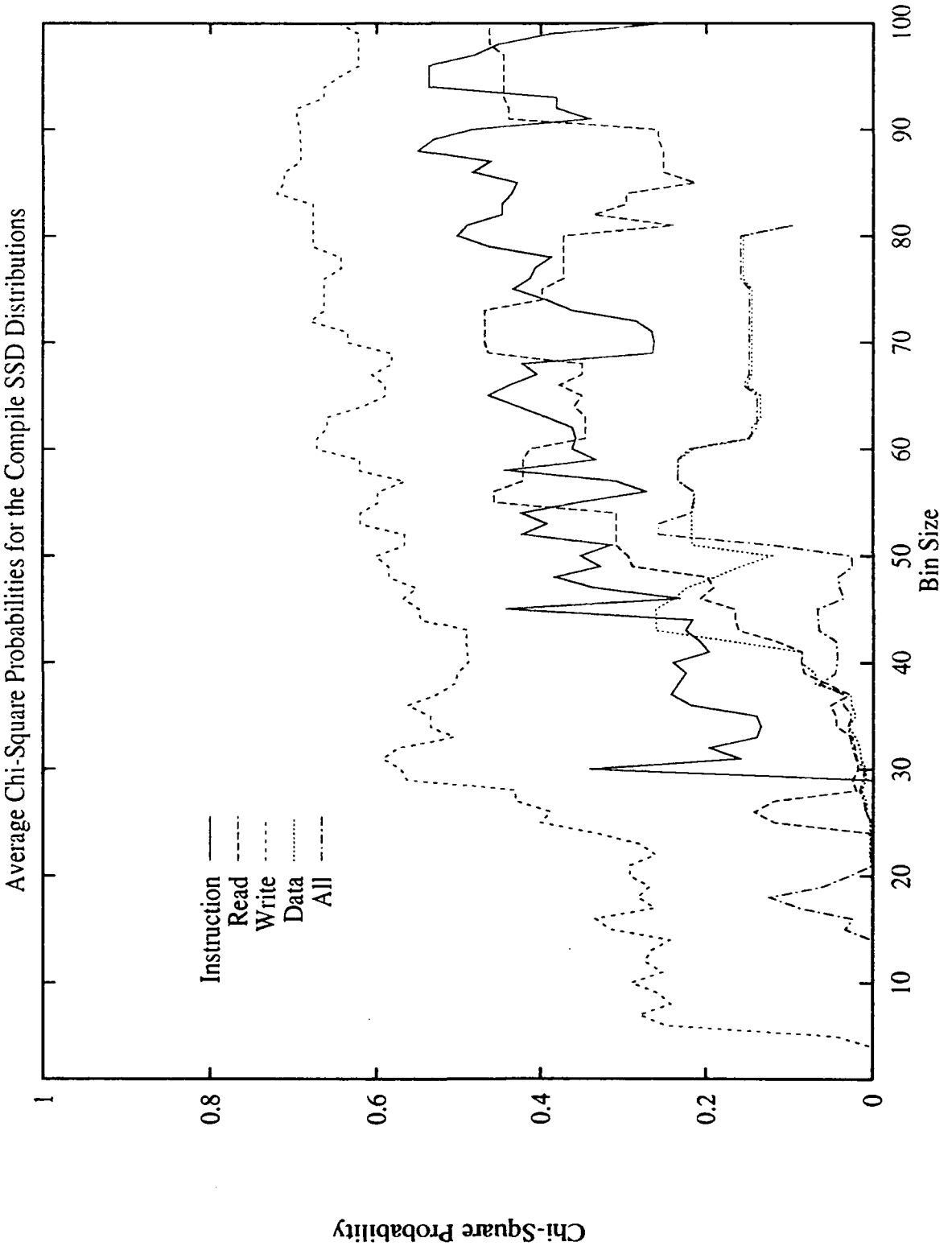


Figure E.1. Chi-Square Probability versus Bin Size : FFT

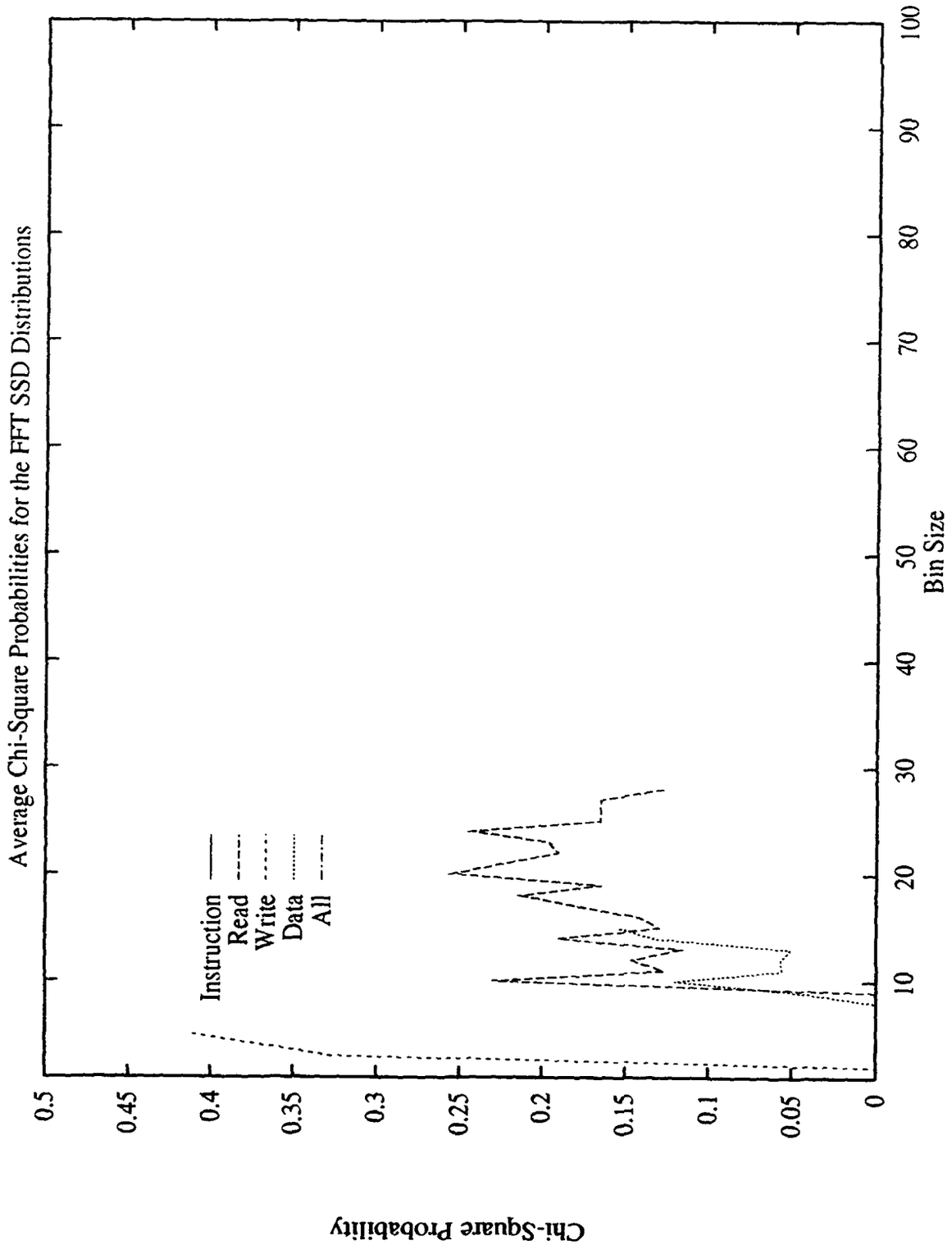


Figure E.5. Chi-Square Probability versus Bin Size : GLISP-Comp

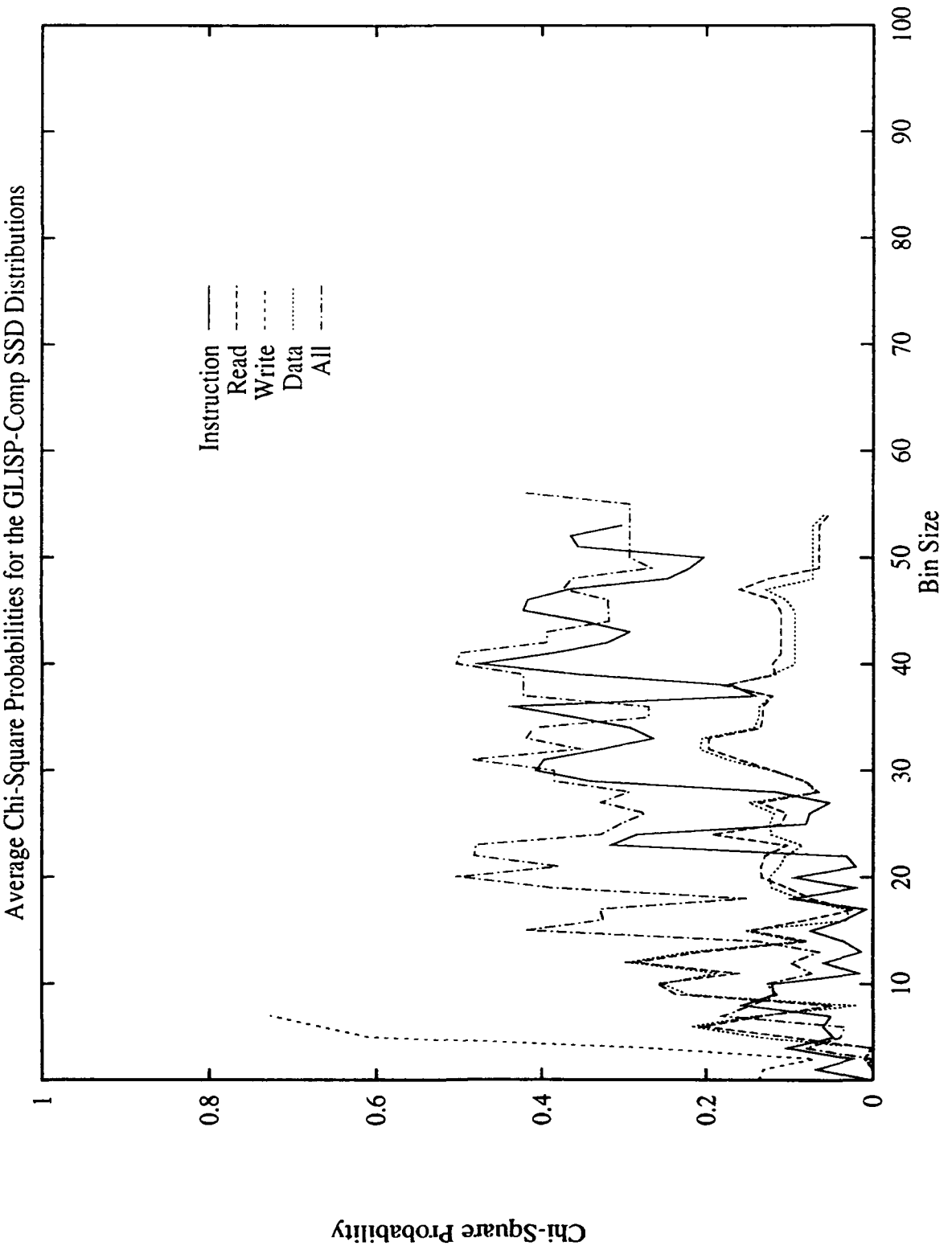


Figure E.6. Chi-Square Probability versus Bin Size : GLISP-Pay

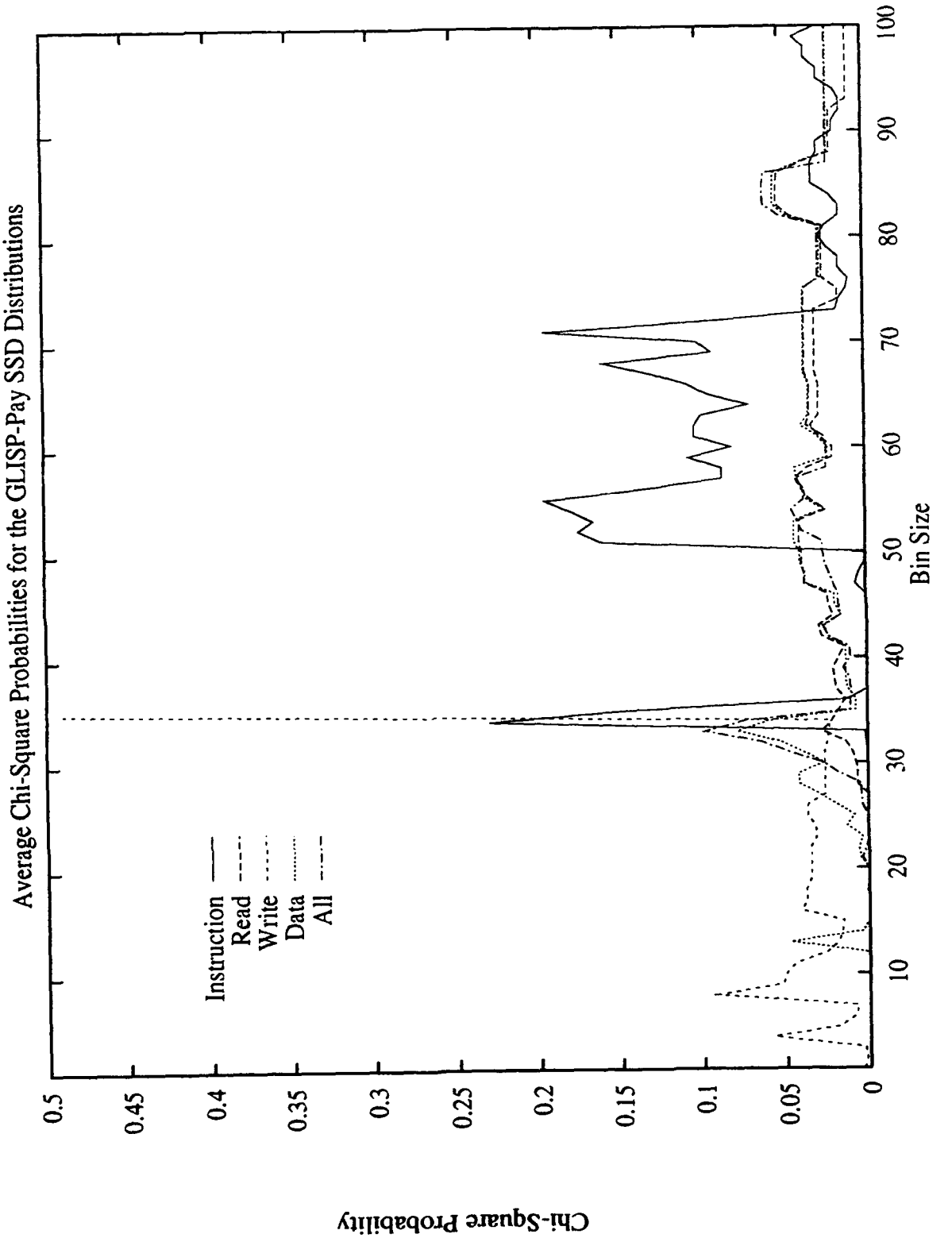


Figure E.7. Chi-Square Probability versus Bin Size : QSIM

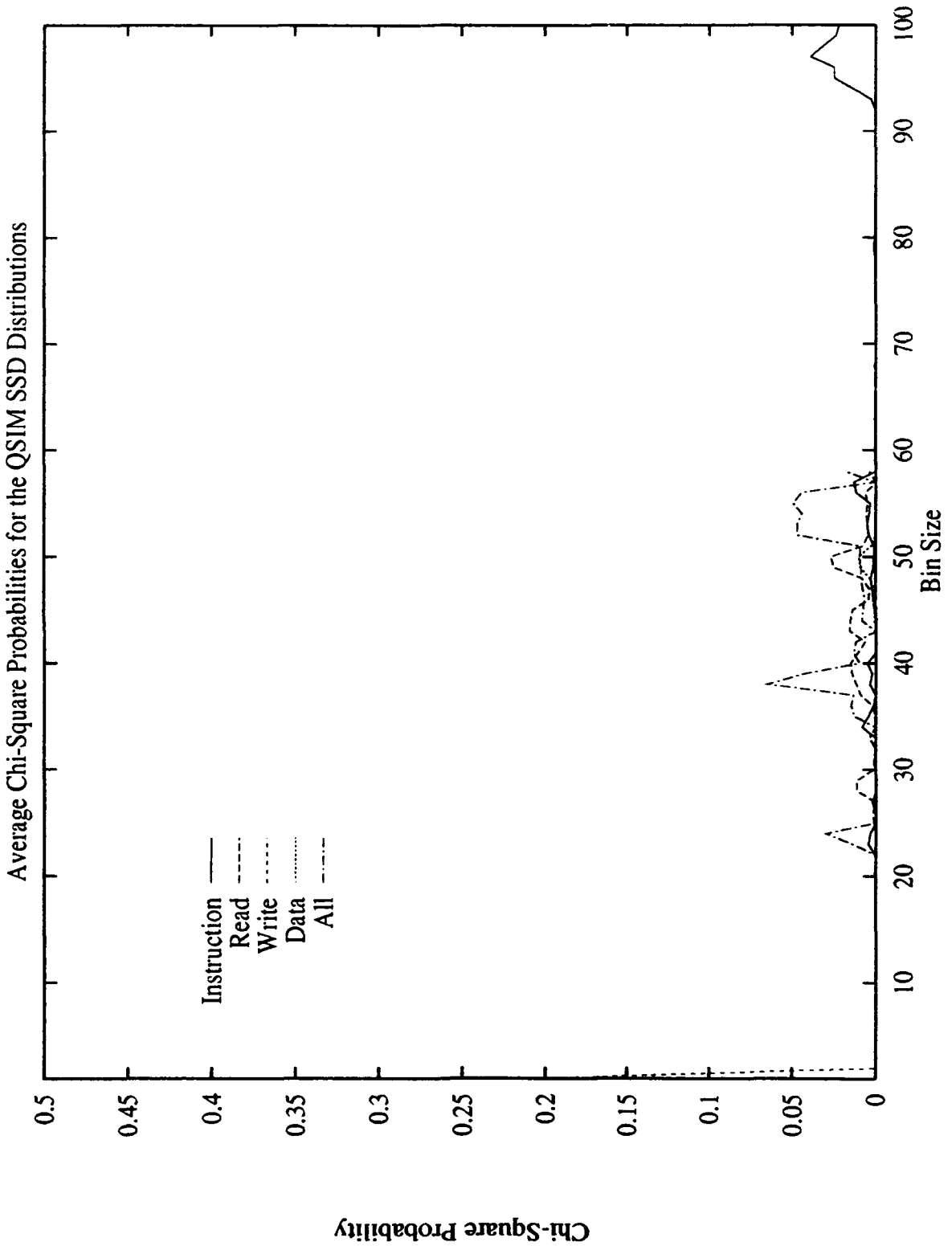


Figure E.8. Chi-Square Probability versus Bin Size : Reducer

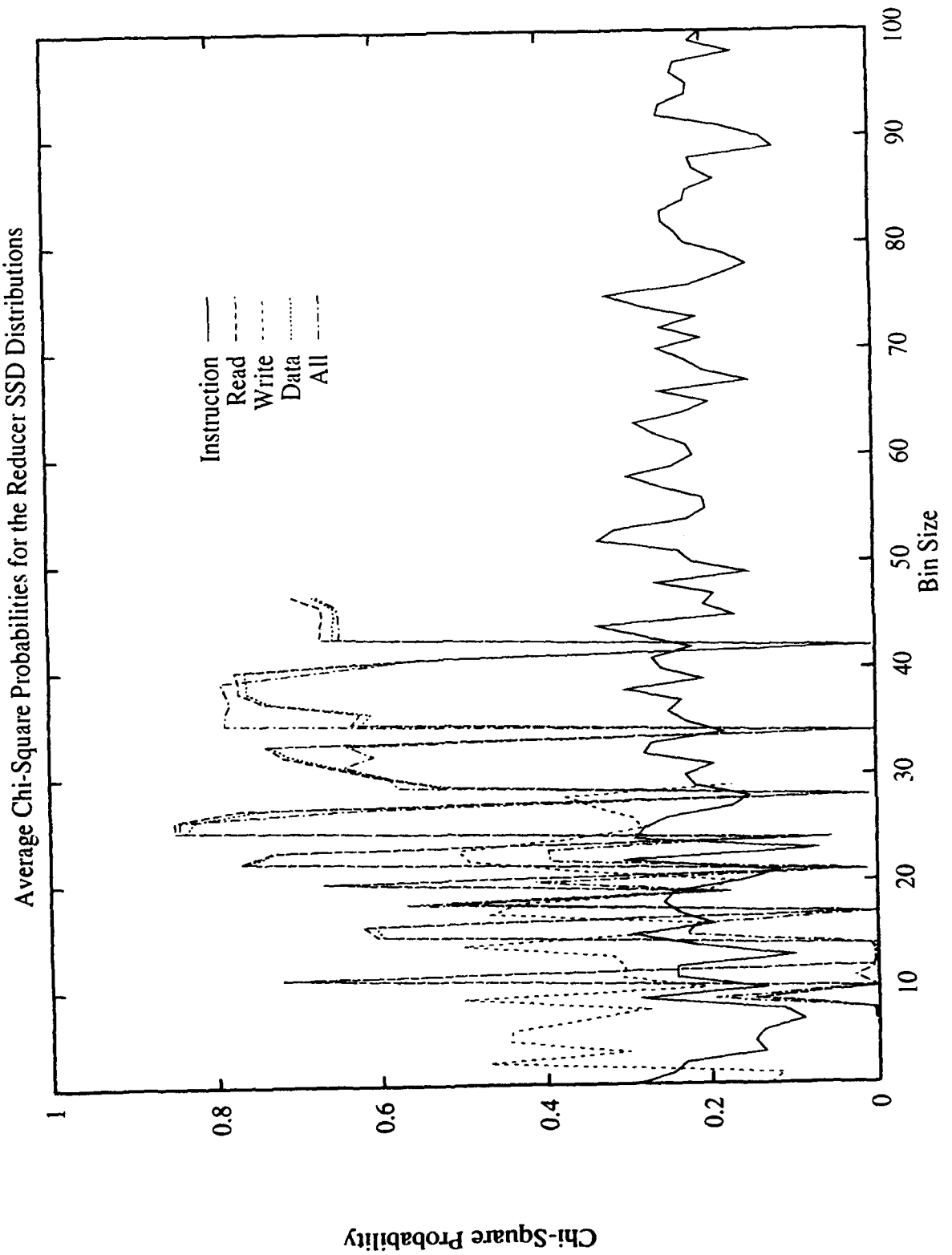


Figure E.9. Chi-Square Probability versus Bin Size : TMYCIN

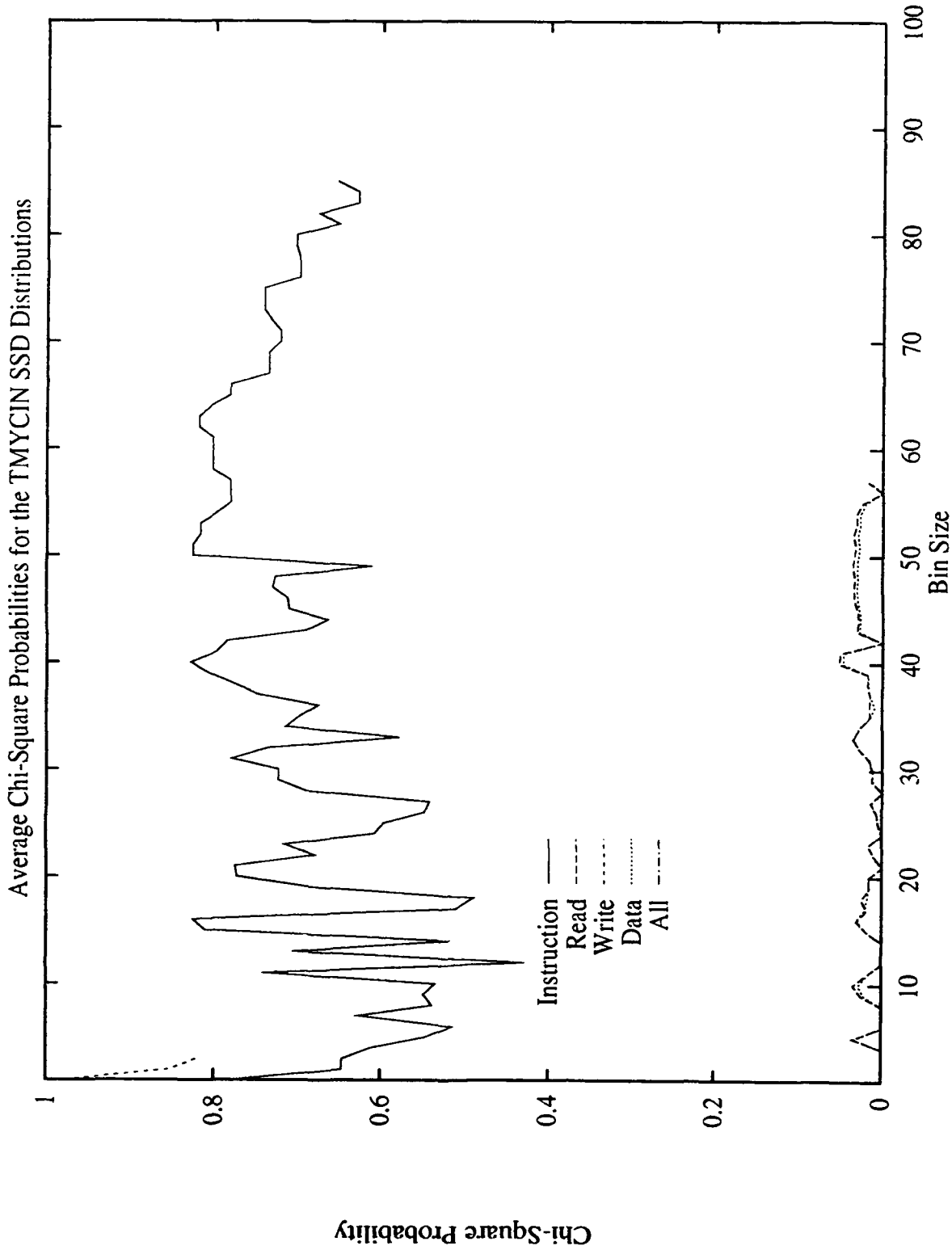


Figure E.10. 4th Order Structural Entropy versus Time Overall Trace : BIASLisp

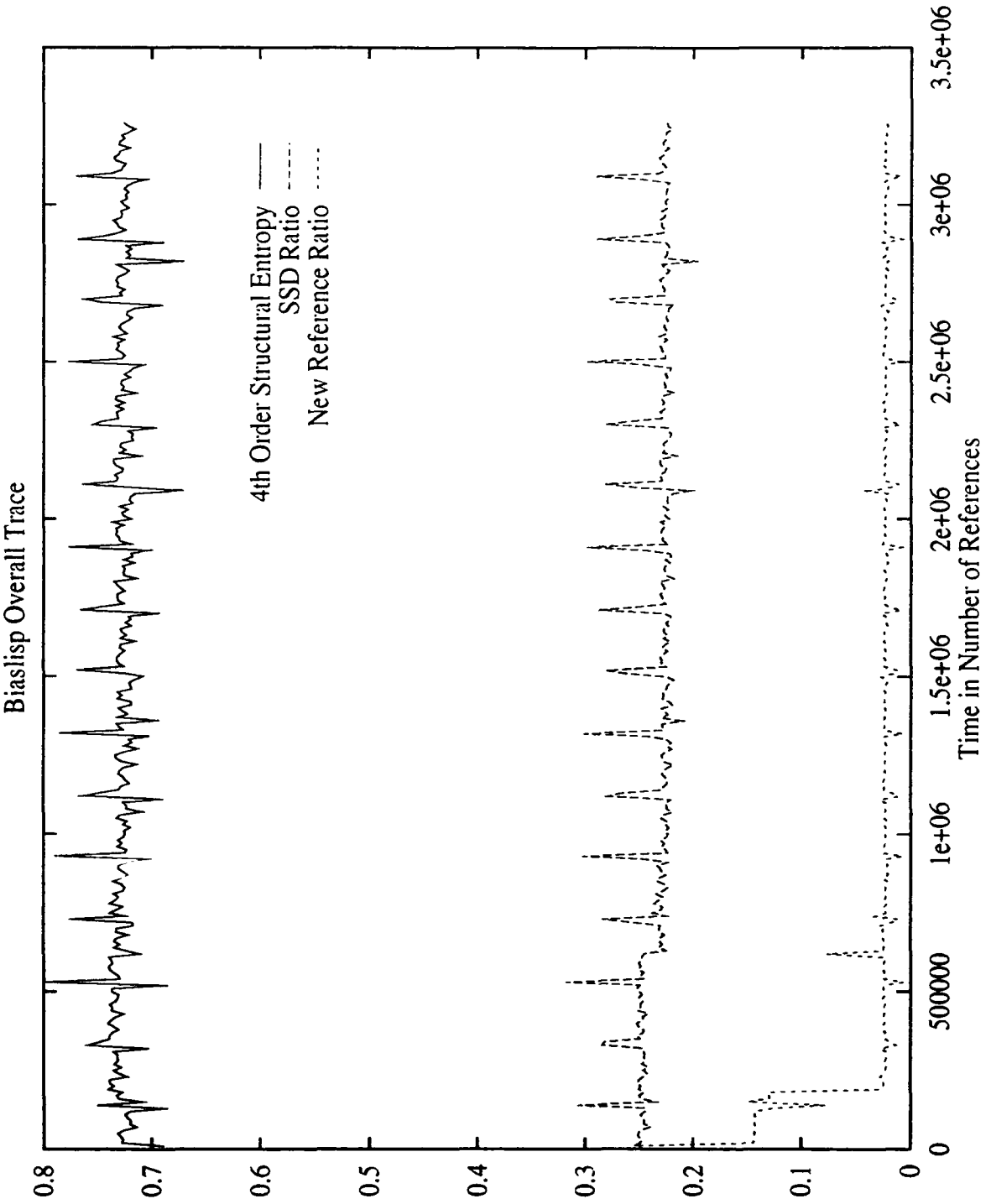


Figure E.11. 4th Order Structural Entropy versus Time Overall Trace : Boyer

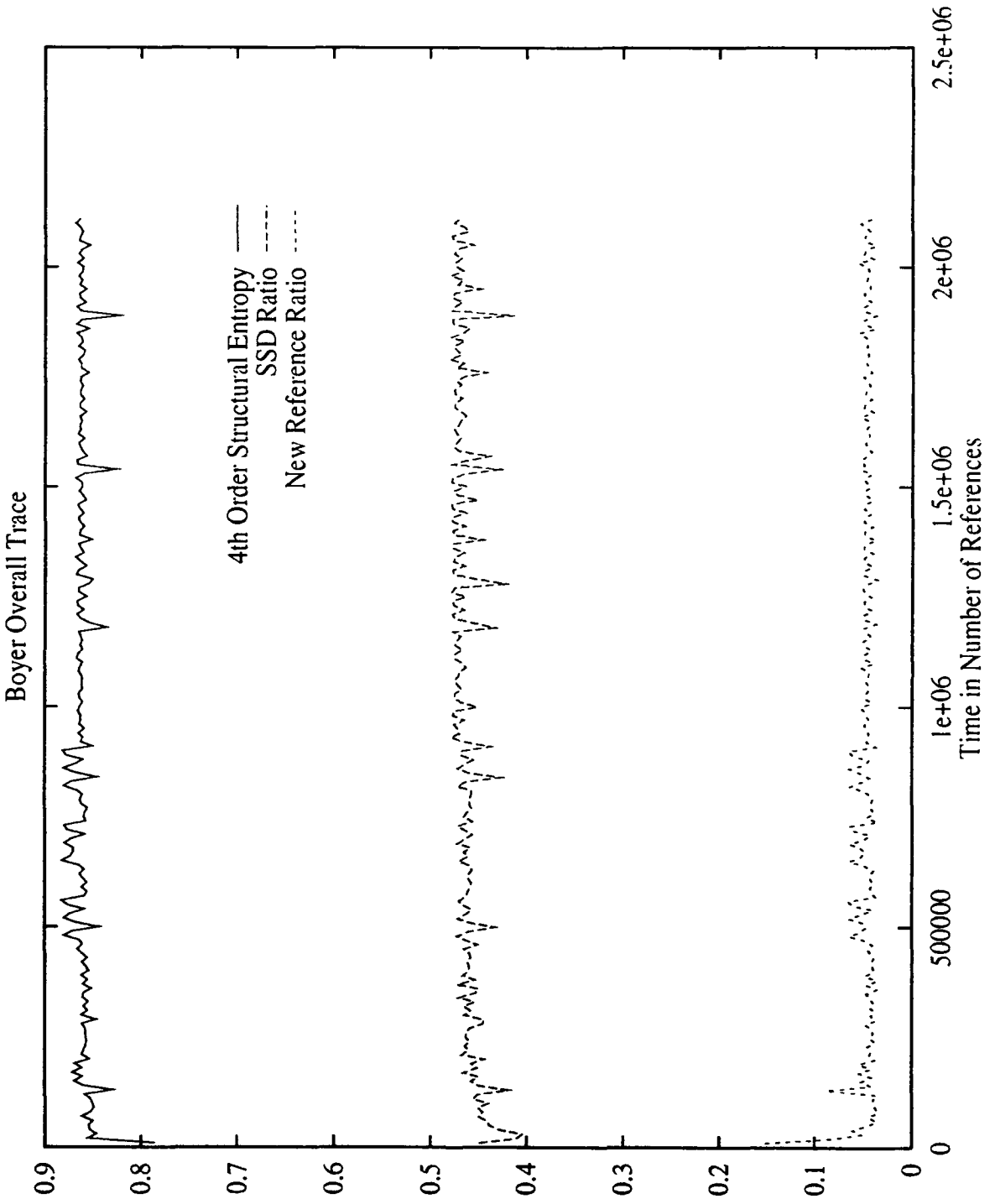


Figure E.12. 4th Order Structural Entropy versus Time Overall Trace : Compile

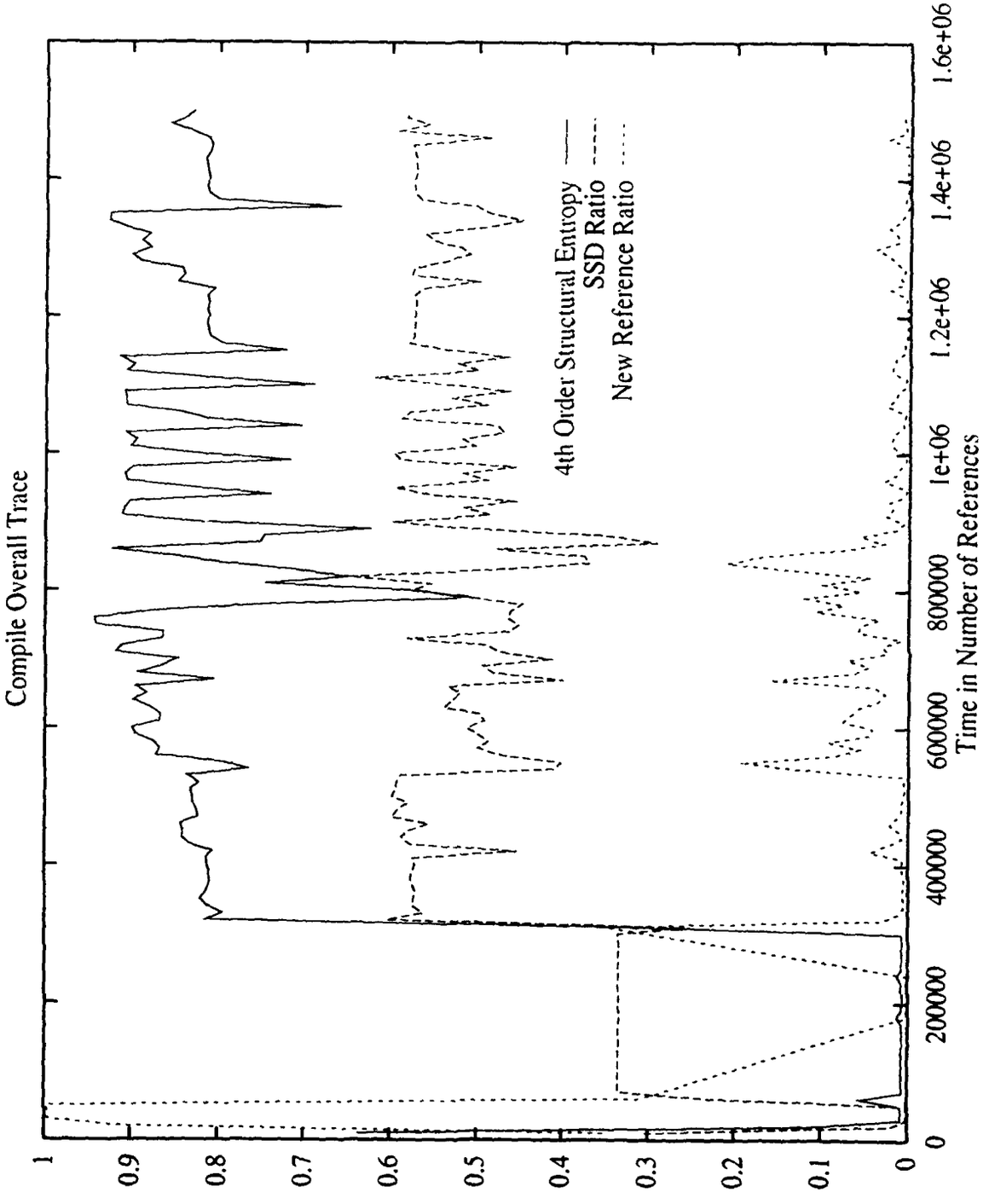


Figure E.13. 4th Order Structural Entropy versus Time Overall Trace : FFT

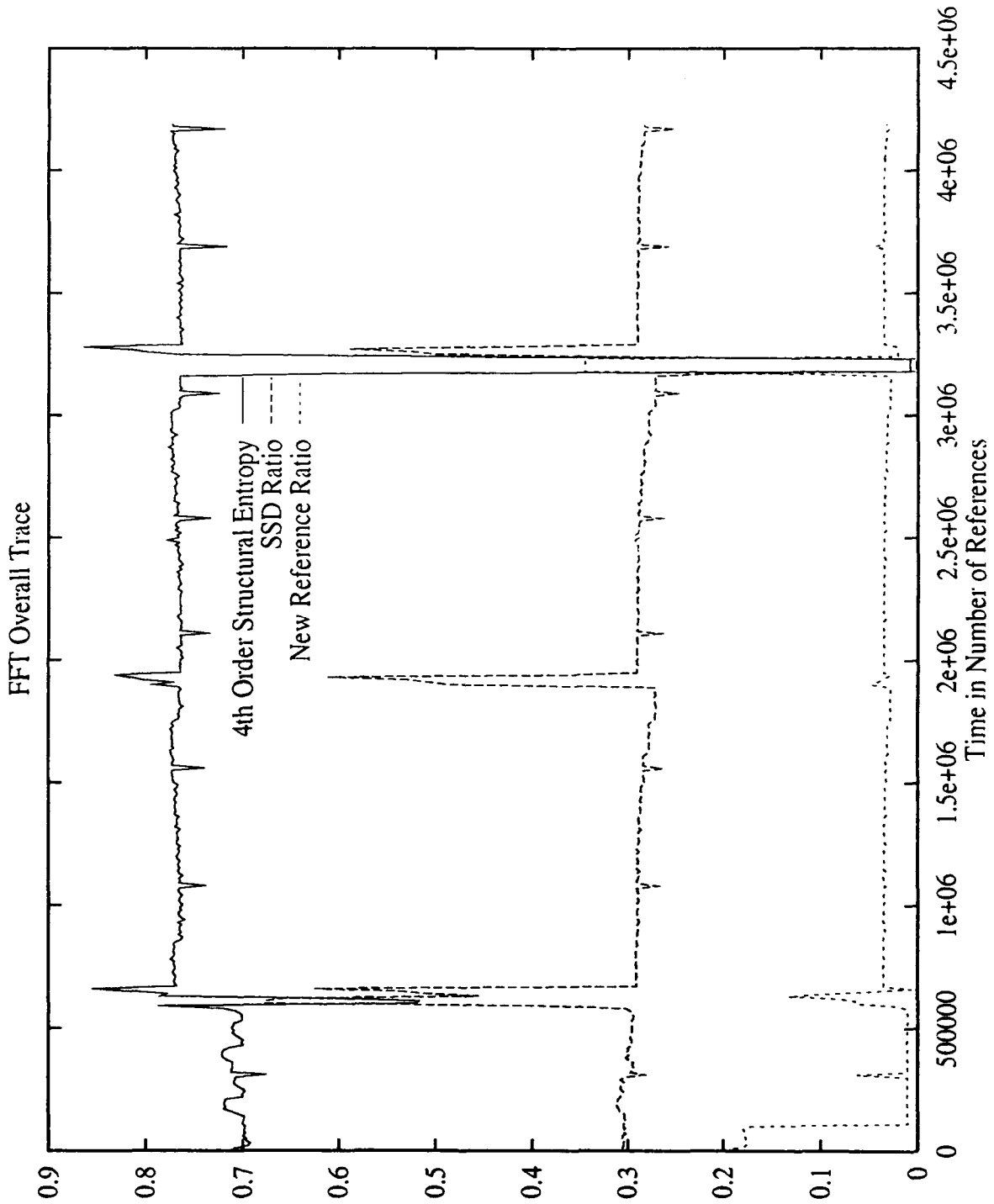


Figure E.14. 4th Order Structural Entropy versus Time Overall Trace : GLISP-Comp

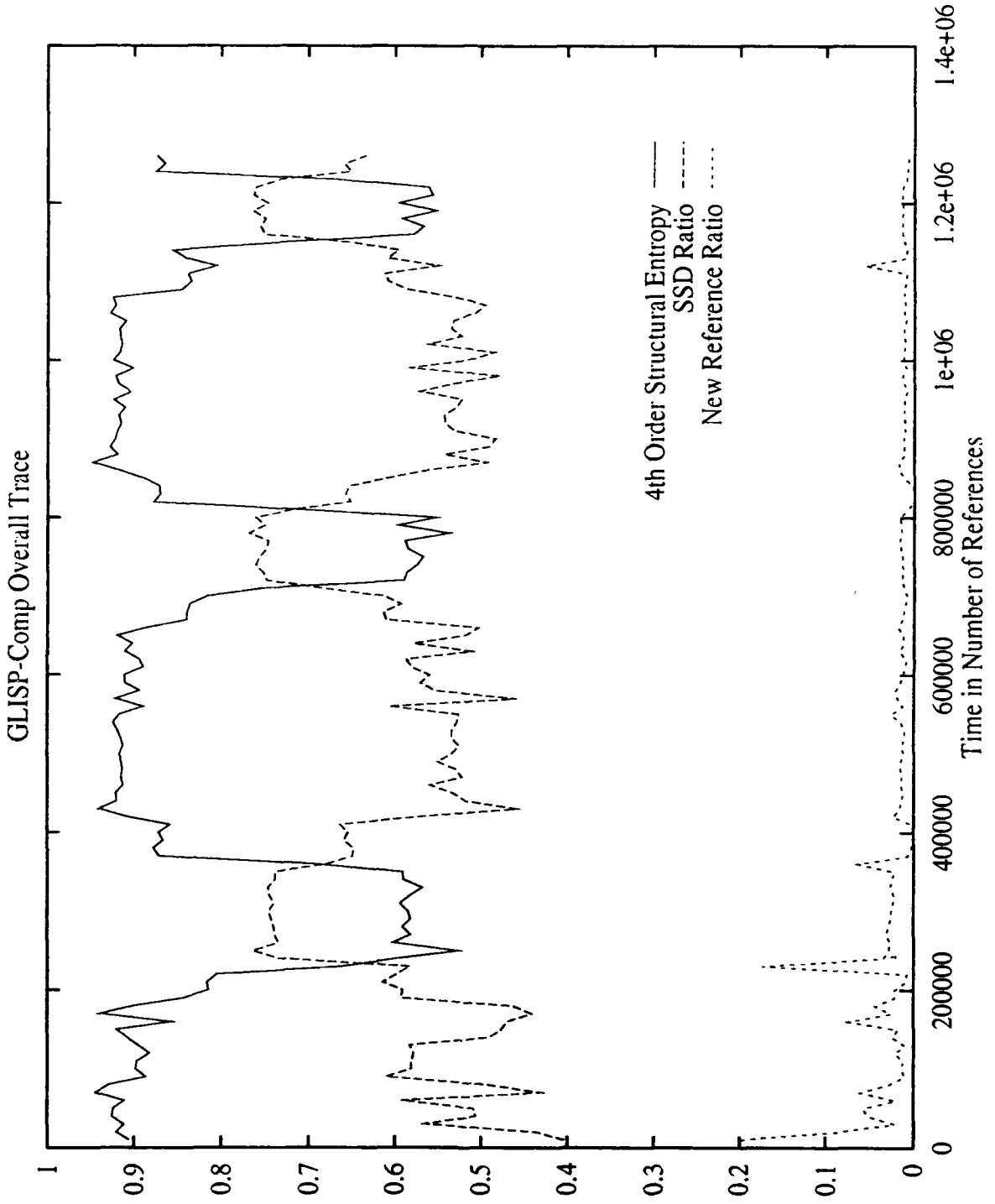


Figure E.15. 4th Order Structural Entropy versus Time Overall Trace : GLISP-Pay

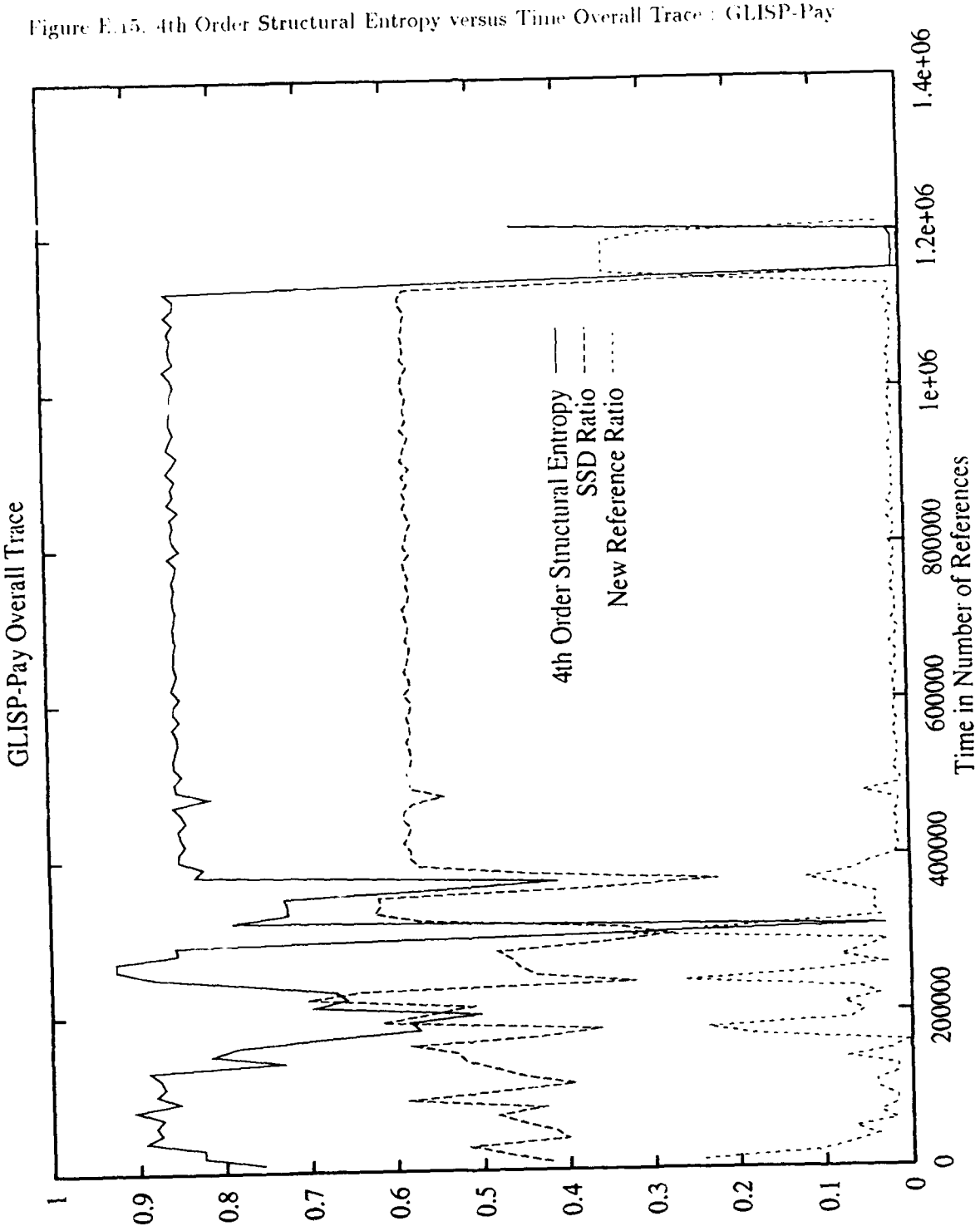


Figure E.16. 4th Order Structural Entropy versus Time Overall Trace : QSIM

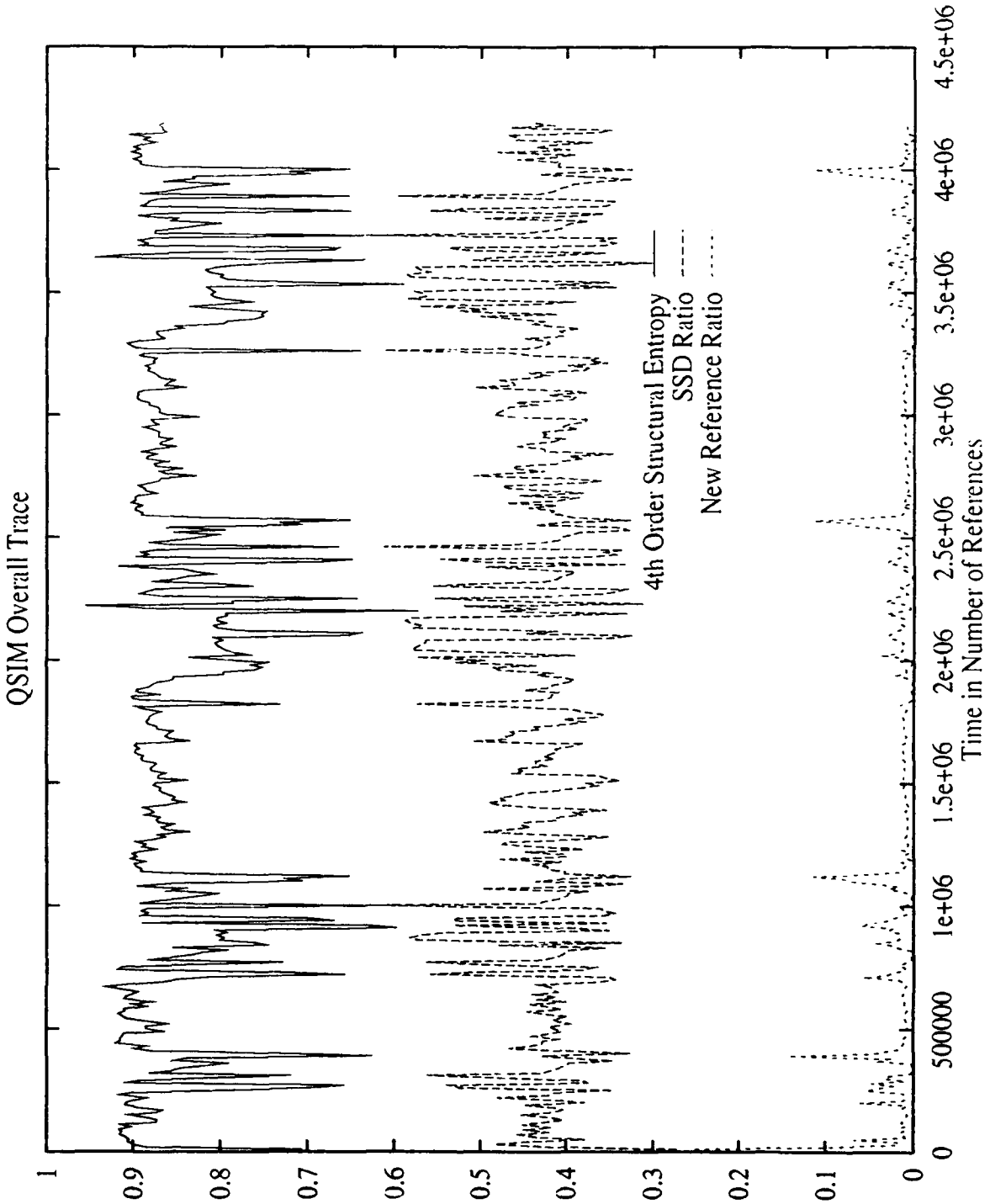


Figure E.17. 4th Order Structural Entropy versus Time Overall Trace : Reducer

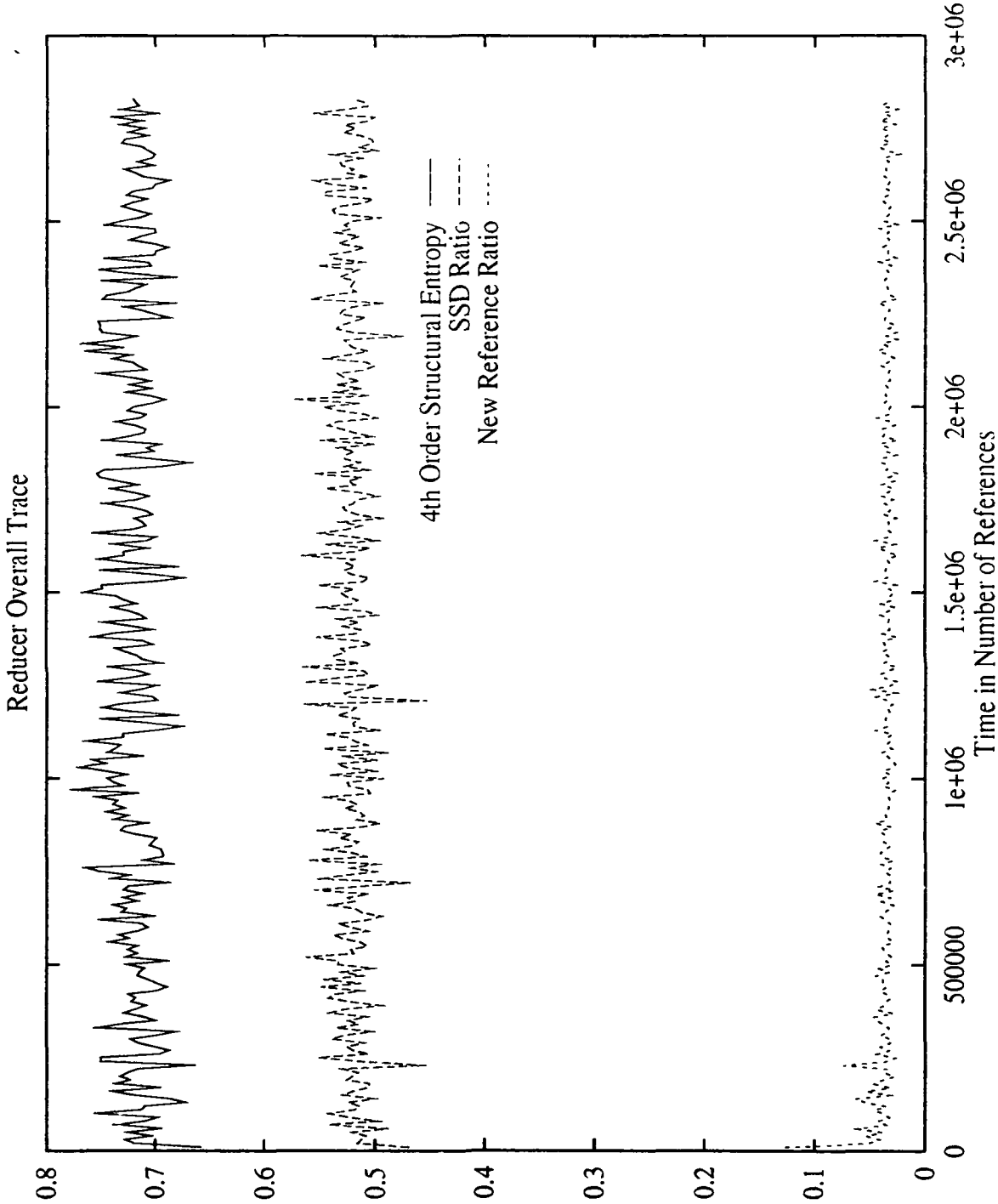
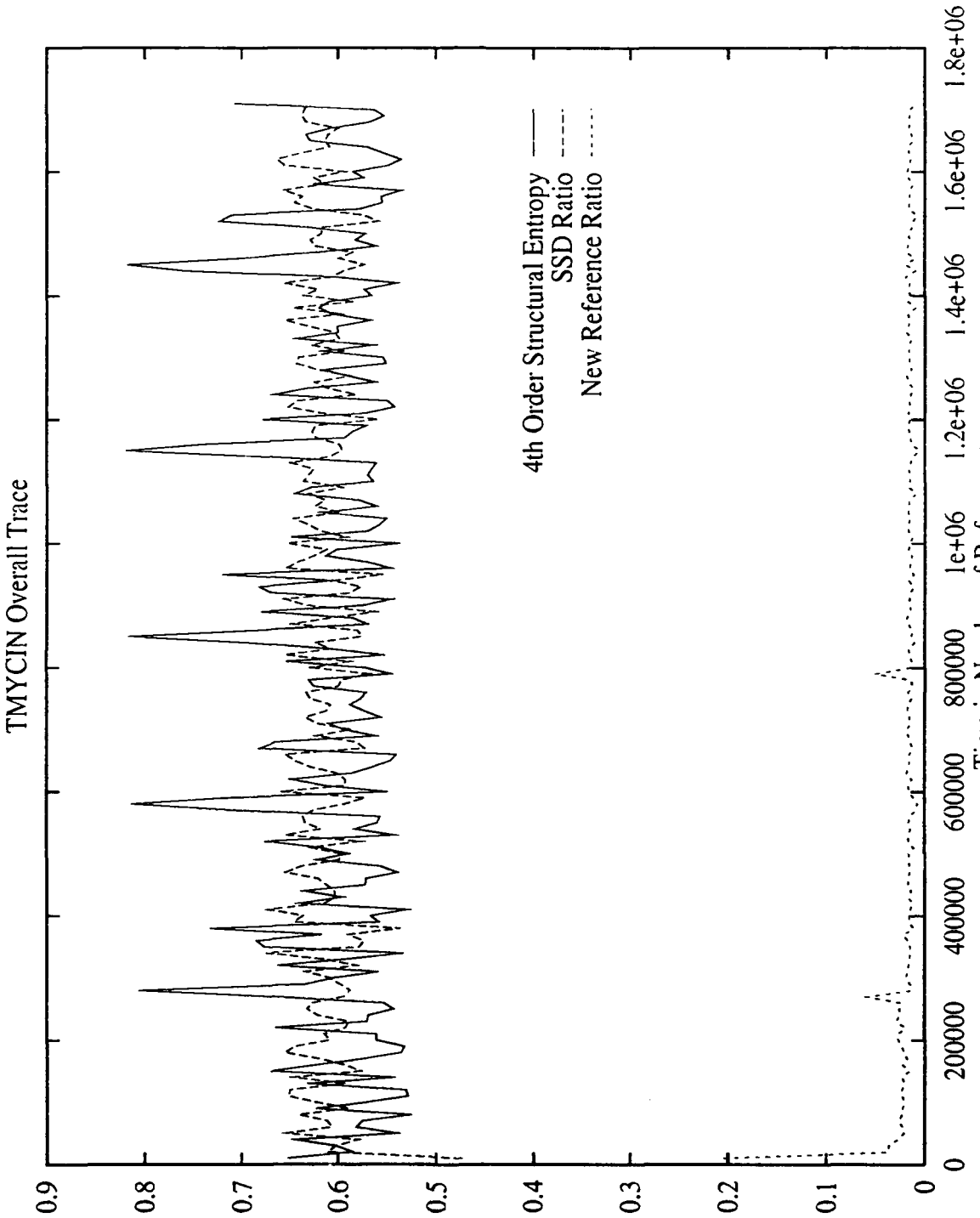


Figure E.18. 4th Order Structural Entropy versus Time Overall Trace : TMYCIN



Bibliography

AHH89. A. Agarwal, M. Horowitz, and J. Hennessy. An analytical cache model. *ACM Transactions on Computer Systems*, 7(2):184–215, May 1989.

ASH86. A. Agarwal, R. L. Sites, and M. Horowitz. Atum: a new technique for capturing address traces using microcode. In *Proc. of the 13th Annual Symposium on Computer Architecture*, pages 119–127, New York NY (USA), June 1986. IEEE.

BMM84. Richard B. Bunt, Jennifer M. Murphy, and Shikharesh Majunmdar. A measure of program locality and its application. In *Proceedings of the 1984 Conference on Measurement and Modeling of Computer Systems*, pages 28–39, 1984.

Den68. Peter J. Denning. The working-set model for program behavior. *Communications of the ACM*, 11(5):323–333, May 1968.

FR91. Christine Fricker and Philippe Robert. An analytical cache model. *INRIA. domaine de Voluccau*, 1991.

HD77. D. W. Hammerstrom and E.S. Davidson. Information content of cpu memory referencing behavior. In *Fourth Annual Symposium in Computer Architecture*, pages 184–192, 1977.

Hob89. W.C. Hobart, Jr. *An Investigation of the Locality of Memory accesses during Symbolic Program Execution*. Ph.D. dissertation, University of Texas at Austin, August 1989.

Kle75. Leonard Kleinrock. *Queueing Systems Volume I: Theory*. John Wiley and Sons, Los Angeles, California, 1975.

Kur86. S. Kurtén. The phase structure of program behavior. Technical report, October 1986.

LS73. P.A.W. Lewis and G.S. Shedler. Empirically derived micromodels for sequences of page exceptions. *IBM Journal of Research and Development*, pages 86–100, March 1973.

Mas83. Takashi Masuda. A method for the detection of program locality. In A.K. Agrawala and S.K. Tripathi, editors, *Performance '83*, pages 173–187. North-Holland Publishing Company, 1983.

MB76. A. Wayne Madison and Alan P. Batson. Characteristics of program localities. *Communications of the ACM*, 19(5):285–294, May 1976.

MB88. Jennifer M. Murphy and Richard B. Bunt. Characterising program behaviour with phases and transitions. In *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 226–223, 1988.

Sha48. C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

Sha50. C.E. Shannon. Predication and entropy of printed english. *Bell System Technical Journal*, 30:50–64, September 1950.

Smi78. A. J. Smith. A comparative study of set associative mapping algorithms and their use for cach and main memory. *IEEE Transactions on Software Engineering*, 4(2):121–130, March 1978.

Spi77. Jeffrey R. Spirn. *Program Behavior: Models and Measurements*. Elsevier Scientific Publishing Company, 335 Jan Van Galenstraat, P.O. Box 2311, Amsterdam, The Netherlands, 1977.

SW88. Press W.H. Flannery B.P. Teukolsky S.A. and Vetterling W.T. *Numerical Recipes in C. The Art of Scientific Programming.* Cambridge University Press, 40 West 20th Street, New York, NY 10011 USA, 1988.

TP87. Matthew J. Thazhuthaveetil and Andrew R. Pleszkun. On the structural locality of reference in lisp list access streams. *Information Processing Letters*, 26(2):105–110, October 1987.

WM88. Wing Shing Wong and Robert J.T. Morris. Benchmark synthesis using lru cache hit function. *IEEE Transactions on Computers*, 26(2):637–645, June 1988.

Vita

Michael Erwin Bletzinger was born in Springfield, Ohio on June 1, 1960, the son of Irmgard and Peter Bletzinger. He graduated from Baker High School, Fairborn, Ohio and in 1983 recieved a Bachelor of Science in Electrical Engineering at the University of Dayton in Dayton Ohio. He became an employee of the Department of Defense in September 1984 at Wright Patterson Air Force Base, Dayton Ohio. He is currently employed at the Naval Surface Warfare Center, Crane Division in Crane Indiana.

Permanent address: 821 Park Square Drive
Bloomington, IN 47403

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1992	Master's Thesis

4. TITLE AND SUBTITLE
An Investigation of Structural Locality in the Memory Referencing Behavior of Computer Programs

5. FUNDING NUMBERS

6. AUTHOR(S)
Michael E. Bletzinger

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology,
Wright Patterson AFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/92D-04

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The nature of structural locality as defined by same stack distance access is investigated in this thesis. The question is whether structural locality can be characterized as an inherent type of behavior. The results revealed that structural locality is strongly influenced by a program's design and phase of execution. Entropy measurements revealed that the predictiveness of structural locality is also influenced by program design. A Markov model was refined to capture the characteristics of structural locality that were measured. Trace synthesis demonstrated some success in reproducing same stack distance run distributions when the model had enough states to encompass the entire distribution. Entropy measurements on the synthesized traces showed that the predictiveness of structural locality was not solely due to the same stack distance behavior. A similar investigation on first-time memory referencing behavior revealed the same types of results.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Locality, Structural Locality, Analytical Model, Memory Referencing Behavior, Cache, Markov Model	171
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL