Empirical Measurements of Six Allocation-intensive C Programs

Benjamin Zorn Dirk Grunwald
Department of Computer Science
Campus Box #430
University of Colorado, Boulder 80309–0430

CU-CS-604-92 July 1992

University of Colorado at Boulder

Technical Report CU-CS-604-92
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

Benjamin Zorn Dirk Grunwald
Department of Computer Science
Campus Box #430
University of Colorado, Boulder 80309–0430

Empirical Measurements of Six Allocation-intensive C Programs*

Benjamin Zorn Dirk Grunwald
Department of Computer Science
Campus Box #430
University of Colorado, Boulder 80309–0430

July 1992

Abstract

Dynamic memory management is an important part of a large class of computer programs and high-performance algorithms for dynamic memory management have been, and will continue to be, of considerable interest. This paper presents empirical data from a collection of six allocation-intensive C programs. Extensive statistics about the allocation behavior of the programs measured, including the distributions of object sizes, lifetimes, and interarrival times, are presented. This data is valuable for the following reasons: first, the data from these programs can be used to design high-performance algorithms for dynamic memory management. Second, these programs can be used as a benchmark test suite for evaluating and comparing the performance of different dynamic memory management algorithms. Finally, the data presented gives readers greater insight into the storage allocation patterns of a broad range of programs. The data presented in this paper is an abbreviated version of more extensive statistics that are publically available on the internet.

1 Introduction

This paper presents empirical data about the allocation behavior of six allocation-intensive C programs. The data presented describes the distributions of objects sizes, holding times, and interarrival times in each of the programs measured. This data is valuable to designers of dynamic memory management (DMM) algorithms for the following reasons:

- It has been long observed that tailoring a DMM algorithm to the observed empirical behavior of programs results in a more efficient algorithm [9]. The data from these programs provides designers with specific information about the allocation behavior of a broad class of C/Unix programs.
- Such data, specifically about the allocation behavior of C programs, is rarely published.
- The programs measured in this study provide DMM algorithm designers with a test suite of allocation-intensive programs with which they can fairly compare alternative DMM algorithms. We have already used these programs for this purpose [6].
- Such data provides readers with insight into the allocation behavior of a broad class of programs. This insight may help the reader understand the trade-offs in DMM implementations, including the relative advantages and disadvantages of using automatic memory management techniques such as garbage collection.

*This material is based upon work supported by the National Science Foundation under Grants No. CCR-9010624, CCR-9121269, and CDA-8922510

CFRAC	Cfrac is a program to factor large integers using the continued fraction method. The input is a 22-digit number that is the product of two primes.
ESPRESSO	Espresso, version 2.3, is a logic optimization program. The input file was one of the larger example inputs provided with the release code (cps).
GHOSTSCRIPT	GhostScript, version 2.1, is a publicly available interpreter for the PostScript page-description language. The input used is the Users Guide to the GNU C++ Libraries (126 pages). This execution of GhostScript did not run as an interactive application as it is often used, but instead was executed with the NODISPLAY option that simply forces the interpretation of the Postscript without displaying the results.
GAWK	Gnu Awk, version 2.11, is a publicly available interpreter for the AWK report and extraction language. The input script processes a large file containing numeric data, computing statistics from that file.
PERL	Perl 4.10, is a publicly available report extraction and printing language, commonly used on UNIX systems. The input used was a perl script that reorganizes internet domain names located in the file /etc/hosts.
CHAM	Chameleon is an N-level channel router for multi-level printed circuit boards. The input file was one of the example inputs provided with the release code (ex4). We also measured another channel router (YACR), but the results obtained were not significantly different that those from CHAM.

Table 1: General Information about the Test Programs

Dynamic memory management has always been an important part of a large class of computer programs. Recently, interest in this field has increased, as evidenced by the number of workshops devoted entirely to the subject (garbage collection workshops at recent Object-oriented Programming Languages and Systems (OOPSLA) Conferences and the 1992 International Workshop on Memory Management to name a few). One reason for this increased interest is that object-oriented design encourages programming with large interconnected dynamic structures and broadens the class of programs that use dynamic memory allocation. The increasing use of dynamic memory management brings with it the need to reevaluate the performance of old algorithms for memory management and consider new ones.

Many studies of the relative performance of DMM algorithms have been published through the years. In addition to comparing the performance of DMM algorithms, a number of these papers also present empirical measurements of the allocation behavior of particular programs or systems. In particular, in 1971 Margolin *et al* presented empirical measurements of the dynamic allocation patterns they observed in a time-sharing operating system [9]. This empirical data has been used in subsequent DMM measurement studies, some as recent as 1985 [10]. Batson et al present empirical data concerning the distribution of program segment sizes in the B5500 operating system [2]. In his book *Data Structure Techniques*, Standish [11] presents data from Charles Weinstock's thesis [12] showing the distribution of size requests in the BLISS/11 compiler. Standish also mentions that Weinstock uses Batson's empirical data in comparing the performance of different DMM algorithms. In a later paper, Batson and Brundage present empirical data about segment sizes and holding times collected from Algol-60 programs [1].

More recently, in 1984 Bozman et al [3] compared the performance of a large number of DMM algorithms based on empirical data gathered from several days of execution on several different multiuser operating systems. In the paper, they present the empirical data gathered, including the average inter-arrival time and holding time for each of the block sizes allocated. This data represents some of the most complete information published to date, and has been used in a 1989 performance evaluation of DMM algorithms [4]. Most recently, DeTreville has published the results of extensive empirical measurements of heap usage in the Topaz computing environment [5].

From this discussion, we can conclude two things. First, empirical measurements of actual programs are valuable in both designing and evaluating DMM algorithms. For as long as these algorithms have been proposed and evaluated, empirical data has been used in the measurement process. Second, there is a relative lack of empirical data, as evinced by the use of the Margolin data 14 years after it was

Program	CFRAC	ESPRESSO	GHOSTSCRIPT	GAWK	PERL	CHAM
Lines of Code	6,000	15,500	29,500	8,500	34,500	7,500
Execution Time (instructions $\times 10^6$)	66.9	611.1	159.2	17.6	33.5	87.1
Objects Allocated	227,091	186,636	108,550	32,165	26,390	103,548
Max Objects Allocated	1,231	2,959	6,195	2,447	483	103,413
Bytes Allocated	3,339,166	14,641,338	18,767,795	722,970	790,801	2,927,254
Max Bytes Allocated	17,395	136,966	467,739	63,834	24,452	2,711,158
Size Classes (SC)	22	328	177	48	79	22
Mean Size	14.7	78.4	172.9	22.5	30.0	28.2
Median Size	14	28	116	24	32	24
Mode Size	14	24	116	24	32	24
Interarrival Time Classes (ITC)	911	13,425	3,502	856	587	4,316
Mean Interarrival Time	295	3,273	1,465	548	1,271	825
Median Interarrival Time	241	96	1,121	451	207	482
Mode Interarrival Time	37	15	69	122	44	66
Holding Time Classes (HTC)	12,748	76,299	15,339	5,638	5,053	13,169
Mean Holding Time	172,000	467,000	5,670,000	1,287,000	574,000	565,000
Median Holding Time	600	25,593	794	89	6,137	479
Mode Holding Time	600	37	236	86	1570	99

Table 2: Test Program Performance Information. The SC, ITC, and HTC values indicate the number of distinct size, interarrival times, and holding times respectively in each of the sample programs. All times are presented in instructions.

published. In general, because empirical measurements of dynamic storage allocation in systems are relatively scarce, algorithm evaluators must and do take whatever is available. The intent of this paper is to make additional empirical measurements of allocation-intensive programs widely available. In a companion paper we investigate how different models of allocation behavior, based on empirical data as presented here, can be used to accurately evaluate DMM algorithms [14].

The remainder of this paper has the following organization: Section 2 describes the programs we have measured and Section 3 presents empirical measures of the programs, including distribution of object sizes, object holding times, and object interarrival times. In Section 4, we conclude by summarizing our data and indicating how it can be obtained on the internet.

2 Programs

To gather the data for this paper, we instrumented six allocation-intensive C programs, described in Table 1. The programs represent a wide range of memory intensive tasks, including number factoring, interpreters, logic optimizers, and CAD/VLSI tools. In each case, we actually collected data from at least two input data sets for each program. In order to simplify the presentation here, we show data from only one of these input data sets. Data from all input sets is publically available via the internet.

The data was gathered by tracing the execution of each program using AE [8] on a Sun SPARC processor. AE is an efficient program tracing tool that captures all instruction and data references, as well as special indicators for calls to the `malloc` and `free` procedures used for memory allocation. These large, complex traces were distilled to a time-ordered memory allocation trace including only calls to `malloc` and `free`. Each memory allocation trace event was time-stamped using the number of instructions since the beginning of the program.

The version of CHAM that we measured does not release much of its allocated memory by calling `free`. For this program, we monitored the data references of the traced program, and artificially deallocated memory when it was no longer referenced. The artificial `free` events were inserted in the memory allocation trace, essentially modeling perfect memory deallocation.

After gathering the "allocation trace" of allocate and free events, we measured three data distributions from each trace: object size, object holding time, and object interarrival time. These three distributions capture the allocation behavior of the programs, allowing their empirical behavior to be characterized.

Table 2 summarizes the vital statistics of each program and presents some basic measures of each of these distributions, including the total number of objects and bytes allocated by each program, as well as the maximum number of objects and bytes allocated at any one time by each program. The programs are allocation-intensive, allocating an object every 300–3000 instructions. The fraction of time spent doing memory allocation for these programs depends on the DMM implementation used, but ranges from approximately 5–30%.

The table shows the mean, median, and mode for each of the three distributions mentioned. In most cases, the median is much smaller than the mean, indicating that the distribution is greatly skewed by a very large range of values. As the table shows, the size distribution is less skewed than the others, while the holding time distribution is greatly skewed. Even the size distribution is heavily skewed to smaller objects, the table indicating that in all but one program the most common size class is 32-bytes or smaller.

The table also shows the number of distinct sizes, interarrival times, and holding times (SC, ITC, HTC, respectively). These values indicate that the number of distinct values observed in each distribution is small compared to the total number of instructions executed by each program. For the interarrival times and holding time distribution, this result suggests that program behavior is relatively regular, resulting in a smaller number of distinct classes.

3 Data

This section presents more complete information about the observed data distributions in the test programs. In previous work, the distribution of object sizes has been the most widely reported distribution, partly because it is the easiest to measure of the three. Here, we describe the distributions of object size, interarrival time (IAT), and holding time (HT) equally.

The first set of tables (Tables 3, 4, and 5) show how many distinct classes of size, IAT, and HT are required to cover from 50% to 100% of the observed data. This coverage data indicates how skewed the distributions are and how many classes really represent the important aspects of program behavior.

The tables show that most of the data in each distribution is accounted for by a small number of classes. In particular, Table 3 shows that at most two size classes are required to cover 50% of the observed data in all of the test programs. Furthermore, 95% of the observed data is covered by at most 34 classes. Algorithms such as Oldehoeft's adaptive exact-fit allocator [10] and our own CUSTOMALLOC [6] exploit this empirical behavior by adapting allocation policies to the most commonly observed object sizes.

Interestingly, we observe that a small number IAT classes and HT classes also account for a large percentage of the allocation in many of the programs measured. This empirical result, which indicates a large degree of regularity, has yet to be exploited by proposed DMM algorithms.

One important statistical characterization of a distribution is a listing of the quantiles. We present the 5% quantiles of the three distributions in Tables 6, 7, and 8.

Program	Number of Size Classes to Cover					
	50%	75%	90%	95%	99%	100%
CFRAC	2	4	5	6	6	22
ESPRESSO	2	3	14	34	88	328
GHOSTSCRIPT	2	3	7	11	19	177
GAWK	1	1	2	4	13	48
PERL	1	4	15	25	43	79
CHAM	1	2	3	4	5	22

Table 3: Size Classes Required to Cover Percentages of All Objects in the Test Programs.

Program	Number of IAT Classes to Cover					
	50%	75%	90%	95%	99%	100%
CFRAC	7	25	53	85	185	911
ESPRESSO	12	222	1955	5002	11560	13425
GHOSTSCRIPT	31	119	413	849	2418	3502
GAWK	4	34	140	241	547	856
PERL	6	15	43	79	325	587
CHAM	44	320	1018	1573	3282	4316

Table 4: IAT Classes Required to Cover Percentages of All Objects in the Test Programs.

Program	Number of HT Classes to Cover					
	50%	75%	90%	95%	99%	100%
CFRAC	24	187	2823	5230	10478	12748
ESPRESSO	9730	29640	57636	66968	74433	76299
GHOSTSCRIPT	13	194	4484	9912	14254	15339
GAWK	2	115	2422	4030	5317	5638
PERL	41	838	2416	3734	4790	5053
CHAM	93	641	3993	7992	12134	13169

Table 5: HT Classes Required to Cover Percentages of All Objects in the Test Programs.

Quantile	Program					
	CFRAC	ESPRESSO	GHOSTSCRIPT	GAWK	PERL	CHAM
0 (min)	4	0	9	0	1	8
5	10	24	29	4	7	12
10	10	24	36	10	11	12
15	10	24	40	24	16	20
20	10	24	64	24	20	20
25	12	24	116	24	24	20
30	12	24	116	24	32	20
35	14	24	116	24	32	24
40	14	28	116	24	32	24
45	14	28	116	24	32	24
50 (median)	14	28	116	24	32	24
55	14	28	116	24	32	24
60	14	28	116	24	32	24
65	16	28	260	24	32	24
70	16	32	260	24	32	24
75	18	32	260	24	32	24
80	18	32	260	24	32	24
85	20	40	260	24	32	24
90	20	64	260	24	32	24
95	20	200	260	24	38	24
100 (max)	266	19680	20016	8192	5632	36788

Table 6: Quantiles of Object Sizes in the Test Programs. All sizes are in bytes.

Quantile	Program					
	CFRAC	ESPRESSO	GHOSTSCRIPT	GAWK	PERL	CHAM
0 (min)	8	5	67	9	5	19
5	37	15	69	14	44	66
10	37	15	113	73	44	66
15	37	15	475	122	44	66
20	46	41	737	122	89	66
25	46	47	812	122	99	186
30	118	50	932	122	99	250
35	152	61	958	224	127	343
40	184	65	994	299	127	416
45	228	87	1051	446	155	446
50 (median)	241	96	1121	451	207	482
55	252	128	1295	451	207	516
60	271	226	1639	451	279	577
65	293	290	1773	475	650	633
70	306	332	1821	562	912	722
75	350	475	1980	764	995	853
80	395	776	2138	1046	2403	1000
85	560	2062	2310	1046	2491	1119
90	654	5510	2578	1175	4050	1444
95	803	8852	3068	1608	5206	1919
100 (max)	20987	7349826	417089	23914	18216	1182249

Table 7: Quantiles of Object Interarrival Times in the Test Programs. All times are measured in SPARC machine instructions.

Quantile	Program					
	CFRAC	ESPRESSO	GHOSTSCRIPT	GAWK	PERL	CHAM
0 (min)	127	35	104	66	73	0
5	158	247	236	86	966	46
10	172	484	236	86	1157	99
15	316	715	267	86	1375	115
20	388	1330	298	86	1570	160
25	391	2443	329	86	1705	261
30	439	5763	391	86	2665	296
35	541	7113	512	86	3413	328
40	542	11021	594	86	3880	362
45	581	16533	635	89	6029	401
(median)	600	25593	794	89	6137	479
55	713	37571	953	89	6256	579
60	822	54098	1124	89	22603	797
65	848	73330	1225	197	24524	1082
70	1038	100872	1307	1082	24979	1201
75	3518	135314	1772	18147	25362	1559
80	4986	171381	4065	36353	25814	2458
85	7678	222732	15820	50289	26408	3830
90	9034	305620	24951	852702	26939	11399
95	11087	446868	33143577	17016325	30080	58356
100 (max)	66937238	611086817	159174866	17617998	33547661	87184842

Table 8: Quantiles of Object Holding Times in the Test Programs. All times are measured in SPARC machine instructions.

The quantile tables show the three distributions are greatly skewed. In particular, the median is a small fraction of the maximum in every case. As Jain points out [7], such a skew suggests that the median value should used as a characterization of the central tendency of these distributions. As we have already seen, because a few small size classes dominate the size distributions of these programs, the quantile plot of object sizes is relatively uninteresting. The IAT quantiles show that the interarrival time between object allocations in the test programs is very frequent, the median time ranging from 100–1000 instructions. Likewise, most of the allocated objects are very short-lived, as the holding time quantiles show. The median object lifespan in the programs ranges from 90 to 26,000 cycles, with the 90th percentile less than 1 million cycles in all cases.

The final set of tables (Tables 9, 10, and 11) in this section present the ten most common classes observed in each of the three distributions. These tables illustrate that a small number of classes cover a large fraction of total observations in each of the distributions. From the percentages presented in Table 9, we see that the top ten sizes account for almost 90% of all object allocation in the six test programs. GHOSTSCRIPT has the most interesting size distribution with common objects of size 116 and 260 bytes. The other programs exhibit a very predictable distribution of object sizes with large fractions of the total objects less than or equal to 64 bytes in size.

The interarrival times indicated in Table 10 show that some programs are dominated by a single small interarrival time, on the order of 40–80 cycles. This regularity probably arises from an allocation occurring in a small, frequent loop. As the table shows, approximately 15–20% of all allocations are separated by this single, small IAT. Beyond the most frequent IAT, four of the six programs also captured almost 50% of the total distinct IAT's in the top ten. We see that the GHOSTSCRIPT program showed the least regularity of the programs measured.

Table 11 shows that the top ten holding times in each program represent a significant percentage of the total holding time distribution. GAWK, in particular, shows a great deal of regularity in object holding time: 62% of all objects lived either 86 or 89 cycles. Surprisingly, GHOSTSCRIPT, which shows

CFRAC				
Rank	Size	Frequency	Pct.	Cum. Pct.
1	14	71617	31.54	31.54
2	10	49107	21.62	53.16
3	20	37979	16.72	69.89
4	18	26250	11.56	81.44
5	12	20763	9.14	90.59
6	16	19901	8.76	99.35
7	28	733	0.32	99.67
8	26	453	0.20	99.87
9	4	250	0.11	99.98
10	22	15	0.01	99.99

GAWK				
Rank	Size	Frequency	Pct.	Cum. Pct.
1	24	28403	88.30	88.30
2	2	1413	4.39	92.70
3	7	457	1.42	94.12
4	4	352	1.09	95.21
5	5	301	0.94	96.15
6	6	241	0.75	96.90
7	3	160	0.50	97.39
8	8	147	0.46	97.85
9	11	97	0.30	98.15
10	14	87	0.27	98.42

ESPRESSO				
Rank	Size	Frequency	Pct.	Cum. Pct.
1	24	67828	36.34	36.34
2	28	49132	26.33	62.67
3	32	33372	17.88	80.55
4	44	2717	1.46	82.00
5	8	2623	1.41	83.41
6	40	2430	1.30	84.71
7	36	2364	1.27	85.98
8	48	1668	0.89	86.87
9	52	1217	0.65	87.52
10	56	999	0.54	88.06

PERL				
Rank	Size	Frequency	Pct.	Cum. Pct.
1	32	18261	69.20	69.20
2	7	731	2.77	71.97
3	20	728	2.76	74.73
4	15	575	2.18	76.90
5	6	488	1.85	78.75
6	19	462	1.75	80.50
7	16	350	1.33	81.83
8	14	344	1.30	83.13
9	8	335	1.27	84.40
10	5	333	1.26	85.67

GHOSTSCRIPT				
Rank	Size	Frequency	Pct.	Cum. Pct.
1	116	40796	37.58	37.58
2	260	40315	37.14	74.72
3	40	7708	7.10	81.82
4	32	2854	2.63	84.45
5	92	2714	2.50	86.95
6	36	2415	2.22	89.18
7	26	1999	1.84	91.02
8	24	1551	1.43	92.45
9	60	1362	1.25	93.70
10	252	1357	1.25	94.95

CHAM				
Rank	Size	Frequency	Pct.	Cum. Pct.
1	24	66983	64.69	64.69
2	20	20888	20.17	84.86
3	12	9967	9.63	94.49
4	8	4068	3.93	98.41
5	16	843	0.81	99.23
6	422	350	0.34	99.57
7	924	189	0.18	99.75
8	1692	142	0.14	99.89
9	1688	72	0.07	99.96
10	88	13	0.01	99.97

Table 9: Frequency of the 10 Most Common Size Classes in the Test Programs. All sizes are in bytes.

CFRAC				
Rank	IAT	Frequency	Pct.	Cum. Pct.
1	37	43530	19.17	19.17
2	46	20028	8.82	27.99
3	152	12758	5.62	33.61
4	241	12050	5.31	38.91
5	228	11914	5.25	44.16
6	293	9965	4.39	48.55
7	118	5991	2.64	51.18
8	561	4583	2.02	53.20
9	361	4084	1.80	55.00
10	627	3576	1.57	56.58

GAWK				
Rank	IAT	Frequency	Pct.	Cum. Pct.
1	122	5767	17.93	17.93
2	451	5203	16.18	34.11
3	14	2774	8.62	42.73
4	1046	2510	7.80	50.53
5	224	888	2.76	53.30
6	454	576	1.79	55.09
7	203	376	1.17	56.26
8	181	374	1.16	57.42
9	119	373	1.16	58.58
10	1056	350	1.09	59.67

ESPRESSO				
Rank	IAT	Frequency	Pct.	Cum. Pct.
1	15	28301	15.16	15.16
2	47	15455	8.28	23.44
3	61	11067	5.93	29.37
4	84	5816	3.12	32.49
5	96	5585	2.99	35.48
6	41	5238	2.81	38.29
7	89	4289	2.30	40.59
8	50	4052	2.17	42.76
9	87	3977	2.13	44.89
10	332	3684	1.97	46.86

PERL				
Rank	IAT	Frequency	Pct.	Cum. Pct.
1	44	3919	14.85	14.85
2	127	2563	9.71	24.56
3	207	2530	9.59	34.15
4	99	2518	9.54	43.69
5	89	1315	4.98	48.68
6	155	1300	4.93	53.60
7	279	1253	4.75	58.35
8	2403	1186	4.49	62.84
9	4350	1096	4.15	67.00
10	2445	512	1.94	68.94

GHOSTSCRIPT				
Rank	IAT	Frequency	Pct.	Cum. Pct.
1	69	8192	7.55	7.55
2	958	4078	3.76	11.30
3	1780	3649	3.36	14.67
4	989	2726	2.51	17.18
5	932	2635	2.43	19.60
6	1651	2457	2.26	21.87
7	1020	2208	2.03	23.90
8	2151	2035	1.87	25.78
9	1992	1709	1.57	27.35
10	748	1578	1.45	28.80

CHAM				
Rank	IAT	Frequency	Pct.	Cum. Pct.
1	66	16805	16.23	16.23
2	67	2009	1.94	18.17
3	39	1756	1.70	19.87
4	224	1626	1.57	21.44
5	494	1589	1.53	22.97
6	577	1516	1.46	24.43
7	433	1492	1.44	25.88
8	516	1402	1.35	27.23
9	455	1217	1.18	28.40
10	967	1211	1.17	29.57

Table 10: Frequency of the 10 Most Common IAT Classes in the Test Programs. All times are measured in SPARC machine instructions.

CFRAC				
Rank	HT	Frequency	Pct.	Cum. Pct.
1	600	11915	5.25	5.25
2	391	11803	5.20	10.44
3	541	9122	4.02	14.46
4	713	8722	3.84	18.30
5	847	8534	3.76	22.06
6	168	6712	2.96	25.02
7	148	6472	2.85	27.87
8	542	3611	1.59	29.46
9	714	3607	1.59	31.04
10	538	3582	1.58	32.62

GAWK				
Rank	HT	Frequency	Pct.	Cum. Pct.
1	86	12852	39.96	39.96
2	89	7178	22.32	62.27
3	197	887	2.76	65.03
4	158	374	1.16	66.19
5	411	340	1.06	67.25
6	464	102	0.32	67.57
7	485	102	0.32	67.88
8	408	102	0.32	68.20
9	429	102	0.32	68.52
10	540	95	0.30	68.81

ESPRESSO				
Rank	HT	Frequency	Pct.	Cum. Pct.
1	37	3989	2.14	2.14
2	246	1329	0.71	2.85
3	375	1037	0.56	3.41
4	247	812	0.44	3.84
5	42	685	0.37	4.21
6	304	682	0.37	4.57
7	368	643	0.34	4.92
8	361	553	0.30	5.21
9	358	532	0.29	5.50
10	418	508	0.27	5.77

PERL				
Rank	HT	Frequency	Pct.	Cum. Pct.
1	1570	1226	4.65	4.65
2	1375	1226	4.65	9.29
3	1705	1099	4.16	13.46
4	3880	1096	4.15	17.61
5	1318	1096	4.15	21.76
6	936	1095	4.15	25.91
7	1127	1095	4.15	30.06
8	6068	336	1.27	31.33
9	6263	336	1.27	32.61
10	2705	303	1.15	33.76

GHOSTSCRIPT				
Rank	HT	Frequency	Pct.	Cum. Pct.
1	236	11239	10.35	10.35
2	267	7167	6.60	16.96
3	1225	6262	5.77	22.73
4	298	5870	5.41	28.13
5	594	3731	3.44	31.57
6	329	3550	3.27	34.84
7	965	2780	2.56	37.40
8	635	2581	2.38	39.78
9	794	2509	2.31	42.09
10	360	2501	2.30	44.39

CHAM				
Rank	HT	Frequency	Pct.	Cum. Pct.
1	99	3349	3.23	3.23
2	9	2332	2.25	5.49
3	46	1998	1.93	7.42
4	291	1685	1.63	9.04
5	115	1525	1.47	10.52
6	123	1510	1.46	11.97
7	1186	1251	1.21	13.18
8	352	1226	1.18	14.37
9	1125	1214	1.17	15.54
10	117	1086	1.05	16.59

Table 11: Frequency of the 10 Most Common HT Classes in the Test Programs. All times are measured in SPARC machine instructions.

less regularity in its IAT distribution, shows substantial regularity in its holding time distribution. It is also interesting to note that all of the holding times in this table are less than 10,000, again indicating that objects in all of these applications are most likely very short-lived. This result has implications for generation-based garbage collection algorithms that might be used to collect these objects [13].

4 Summary

We have presented data from one input data set for each of the six programs measured. Using the trace extraction and reduction techniques described, we have collected similar data from at least two inputs to each of these programs. We have also collected other statistical characterizations of the test programs and compared the accuracy of different synthetic models of program allocation in a companion paper [14]. All of the collected data has been formated as a C program input file with declarations of the distributions of object size, holding time, etc. These C files can easily be compiled and linked with a program intended to manipulate the data. C files representing all inputs to all the programs described in this paper are publically available via anonymous FTP from ftp.cs.colorado.edu in the directory pub/cs/misc/MallocStudy. The compressed C files are labeled cfrac-1,2,3.c.Z, etc. where each number represents a different input. In addition, a header file "DataHeader.h" is included that declares all the data structures defined in the .c files. Please feel free to use the data provided in this directory but we would appreciate your sending us e-mail ({zorn,grunwald}@cs.colorado.edu) indicating that you intend to use the data and how you intend to use it.

References

[1] A. P. Batson and R. E. Brundage. Segment sizes and lifetimes in Algol-60 programs. *Communications of the ACM*, 20(1):36–44, January 1977.

[2] A. P. Batson, S. M. Ju, and D. C. Wood. Measurements of segment size. *Communications of the ACM*, 13(3):155–159, March 1970.

[3] G. Bozman, W. Buco, T. P. Daly, and W. H. Tetzlaff. Analysis of free-storage algorithms—revisited. *IBM Systems Journal*, 23(1):44–64, 1984.

[4] R. P. Brent. Efficient implementation of a first-fit strategy for dynamic storage allocation. *ACM Transactions on Programming Languages and Systems*, 11(3):388–403, July 1989.

[5] John DeTreville. Heap usage in the Topaz environment. Technical Report 63, Digital Equipment Corporation System Research Center, Palo Alto, CA, August 1990.

[6] Dirk Grunwald and Benjamin Zorn. CustoMalloc: Efficient synthesized memory allocators. Technical Report CS-CS-602-92, Department of Computer Science, University of Colorado, Boulder, Boulder, CO, July 1992.

[7] Raj Jain. *The Art of Computer Systems Performance Evaluation.* Wiley Professional Computing. John Wiley and Sons, Inc., New York, 1991.

[8] James R. Larus. Abstract execution: A technique for efficiently tracing programs. *Software—Practice and Experience*, 20(12):1241–1258, December 1990.

[9] B. H. Margolin, R. P. Parmelee, and M. Schatzoff. Analysis of free-storage algorithms. *IBM Systems Journal*, 10(4):283–304, 1971.

[10] Rodney R. Oldehoeft and Stephen J. Allan. Adaptive exact-fit storage management. *Communications of the ACM*, 28(5):506–511, May 1985.

[11] Thomas Standish. *Data Structures Techniques.* Addison-Wesley Publishing Company, 1980.

[12] Charles B. Weinstock. *Dynamic Storage Allocation Techniques.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1976.

[13] Benjamin Zorn. The measured cost of conservative garbage collection. Technical Report CU-CS-573-92, Department of Computer Science, University of Colorado, Boulder, Boulder, CO, February 1992.

[14] Benjamin Zorn and Dirk Grunwald. Evaluating models of memory allocation. Technical Report CS-CS-603-92, Department of Computer Science, University of Colorado, Boulder, Boulder, CO, July 1992.