Evaluating the Relationship Between the Usefulness and Accuracy of Profiles*

Geoff Langdale [1]
[1]Carnegie Mellon University
Pittsburgh, PA 15213
geoffl@cs.cmu.edu

Thomas Gross[1,2]
[2] ETH Zürich
8092 Zürich, Switzerland
trg@inf.ethz.ch

Abstract

The relationship between how accurately a profile predicts future program behavior and how useful it is for profile directed optimization is not straightforward. We gathered extensive data on the results of profile-driven optimization using two different optimization systems (`cc` [1] and `alto` [4]) and selected benchmarks and benchmark runs from the SPEC95 and SPEC2000 suites. Instead of following the traditional SPEC guidelines of training only with the designated "train" profiles and gathering performance statistics with the designated "reference" benchmark runs, we evaluate nearly all possible combinations of training and evaluation runs. We summarize the usefulness of basic block profiles in this wider context, evaluate the reliability of the results that we derived from using a range of evaluation runs, and evaluate the apparently uncontroversial claim that more accurate basic block profiles are connected to better profile-driven optimization performance. We find that while in the `alto` optimization context, there is a significant correlation between more accurate profiles and more useful profiles, no such correlation existed in the `cc` system.

1 Introduction

Profile-directed compiler optimization is a commonly implemented technique and is considered to be an effective one. Profile-directed optimization (PDO) depends on the assumption that having more accurate predictions of future behavior will result in better optimization performance. Both the assumption of effectiveness of profile-directed optimization, and the assumption of a connection between more accurate profiles and better profile-driven optimization performance, require quantification. In this paper, we attempt to evaluate both assumptions.

We present a methodology for evaluating the effectiveness of profile-directed optimization, for determining the significance of variability in profile-directed optimization performance and for measuring the strength of the connection of profile accuracy and profile "usefulness". We use this methodology to analyze profile-driven optimization for two optimization systems, `cc` and `alto`. Both profile-driven optimization systems used profile information to provide significantly better performance in the resulting code. However, we derive somewhat cautionary results concerning the commonly-held assumption that profile accuracy strongly predicts profile-directed optimization performation; there was only a weak connection between profile usefulness and accuracy for `alto` and none whatsoever for `cc`. Those wishing to discover whether these assumptions hold for their own optimizers, programs and profiling methods will need to repeat our analyses.

We evaluate metrics that attempt to measure how accurately a profile predicts a given future program execution. Suppose we have two training profiles p_1 and p_2 and an evaluation run (which when profiled, produces a profile p_E). Suppose also that the binary produced by using p_1 for profile-driven optimization performs better on that evaluation run than the binary produced by using p_2. A

*This research was supported, in part, by a grant from Intel MRL (Microprocessor Research Lab).

"good" profile comparison metric in this case would be one that shows that p_1 is a more accurate prediction of p_E than is p_2; that is, a metric that correlates strongly with profile usefulness.

We must immediately clarify the scope of this paper. We cannot derive results that apply to all profile-gathering techniques, benchmarks and optimizers. In this paper we work within the context of profiles that have been directly gathered by basic block profiling (as opposed to approximate methods such as statistical sampling, or profiles that are entirely synthetic, such as those generated by static estimation). We use two optimizers and a wide range of benchmarks but cannot generalize from these optimizers and benchmark programs to a hypothetical "universe" of benchmarks and optimizers.

Our methodology for analyzing profile-driven optimization performance and its relationship to accuracy is applicable to other optimizers, architectures, benchmark sets, and profiling methods. We feel that applying our methodology to the domain of exact basic-block profiles is the logical starting point for analysis of the relationship between profile usefulness and profile accuracy. The choice of which training profiles to use is more "fundamental" than the choice of profiling methods; regardless of what profiling methods are available, the issue of which training profiles to use will always be present.

2 Experimentation Framework

2.1 Definitions

In the process of profile-driven optimization, a given *run* (deterministic execution of a benchmark program with a certain input) produces a profile that is associated with that run. This profile is then used as input to a profile driven optimizer, and is thus called a *training profile*. The resulting binary can be evaluated with an *evaluation* run. The latter type of run will also have a profile associated with it, the *evaluation profile*, which is the basic block profile that would have results from profiling the binary with the evaluation run.

We draw our benchmarks from the SPEC95 and SPEC2000 benchmarks (if a benchmark exists in both benchmark sets, we use the SPEC2000 version and append "2000" to the benchmark name). The SPEC benchmarks define three standard runs, called `ref`, `test` and `train` (each of which can often be combinations of multiple program runs). The profile-driven optimizations allowed in the context of SPEC benchmarks involve using `train` as the training run and `ref` as the evaluation run (`test` may only be used for a relatively short-running test of the correctness of a given benchmarking setup). In our work, we use all of the available runs as training and evaluation runs, in all combinations. Where the SPEC benchmarks call for aggregating multiple runs into a single evaluation or training run, we consider each run individually. Thus, instead of testing a single training profile and evaluation run for profile-driven optimization, we may gather information on as many as 100 possible combinations of training profiles and evaluation runs (we use 10 different evaluation runs and 10 different training profiles for the SPEC2000 benchmark `perl` resulting in 100 possible combinations). More commonly, we have only the three standard SPEC runs available to us and thus gather information on 9 such combinations.

When presenting the names of non-standard SPEC runs (that is, runs that are not simply the SPEC training, testing or reference runs), we will indicate the source of the run as needed. There are two `perl2000` benchmark runs that involve calculation of "perfect" numbers. We refer to the run that is one of the multiple runs in the SPEC reference benchmark as `ref/perfect` and the run that was part of the SPEC training benchmark as `train/perfect`. Generally the names of these runs are not significant and are included only for reference.

We define *profile usefulness* in terms of an evaluation run. That is, it is meaningless to say that profile p_1 is more useful than p_2; only that p_1 is more useful than p_2 with respect to some evaluation run.

Profile accuracy, as measured by one of our profile comparison metrics, measures how well the behavior associated with a training profile predicts the behavior associated with an evaluation profile, strictly in terms of the contents of the two profiles. Once again, accuracy is defined in terms of an evaluation profile. The accuracy of profile p_1 (given a comparison metric) is calculated strictly by comparing the profile data associated with p_1 with the profile data associated with the evaluation run.

2.2 Profile-Driven Optimization Platform

We have implemented a system for evaluating profile usefulness and accuracy. This system consists of a set of profile gathering tools, a profile manipulation tool, and two optimization platforms (the `alto`[4]) system and the standard Digital Unix C Compiler[1]) using the profiles that we gather. The following steps outline the operation of our system.

First, we produce "base" binaries using the Digital Unix C compiler (DEC C V5.6, subsequently referred to as `cc`).

Second, we use `alto` to gather profile information and build a Control Flow Graph (CFG). The base binaries are instrumented by `alto` and used to gather profile information for the various runs of the benchmark.

Third, these profiles and the benchmark's control-flow graph are passed to the profile manipulation tool, which may apply transformations to real profiles or generate new profiles from scratch. The profile optimization tool can generate profiles in `alto` format or in the standard `pixie` format. At this stage we also gather data on profile characteristics and comparisons between profiles.

Fourth, these new profiles are used as inputs to the profile-driven optimization process. These profiles are used with either `alto` (with full optimizations switched on) or the Digital C compiler (see [1] for details of the optimizations performed) to produce an optimized binary. The profile-driven optimizations that provide the most substantial improvements are similar in both optimzers: code placement optimizations, procedure inlining, and super-block formation (profile-driven optimization steps in super-block formation also affect many subsequent optimizations that are not themselves profile-driven). We also produce binaries with the same set of optimization flags but without profile information, for comparison.

Finally, the optimized binary is run[1]. We can compute cycle counts (using the EV5 performance counters) for all our evaluation runs at this time. We are often measuring only subtly different binaries, with very small variations in run-time. We run our benchmarks on a 333Mhz EV5 21164 machine with 1GB of memory (running Digital UNIX V4.0). The machine, while old, has highly accurate performance counters and mature and well-tuned optimizers.

Our work is not focused on producing peak optimization performance. Our focus is on studying the effects of profile-driven optimization and methods for evaluating its effectiveness, not implementing the fastest possible optimizations. In general, the optimization performance of our system (through either the `alto` path or the `cc` path) is good. Usually, optimization performance is within 5-10% of the DEC C compiler at the highest level of optimization, and sometimes faster, due mainly to the aggressive whole program optimizations implemented in `alto`.

We use the technique of using the evaluation profile as a training profile, a case that we call, after Savari and Young [5], *resubstitution*. While not valid as a practical technique (why run the exact same program execution twice?), resubstitution frequently generates interesting results, allowing us a insight into how much benefit results from having "perfect" information. We do not use resubsitution cases when reporting average benefits from using profile-directed optimization.

Our goal is to investigate the usefulness and accuracy of profiles, not to generate superior SPEC results or to find the ideal "representative" training profile. Our use of non-standard SPEC training profiles and evaluation runs means that our results cannot be considered to be valid SPEC results. This does not render the results invalid in a research sense. As stated above, even the (highly questionable in a benchmarking sense) use of resubstitution can generate interesting data. We carry out analyses to determine whether our observed performance effects from shorter-running evaluation runs than the SPEC "ref" benchmarks represent real effects or whether the effects are simply due to experimental error; the former is true for nearly all combinations of optimizer, benchmark and evaluation run.

[1]Currently, we have some missing data points (including the SPEC2000 version of gcc) due to bugs in one or the other of the optimizers, including a number of the baseline "non-profile-directed optimization case" results. We are also missing some entire benchmarks in the `cc` optimization context. Our results are not significantly altered by restricting the benchmark sets to only those benchmarks that worked across both optimization environments, so we have opted to present more information (the benchmarks that worked only under the `alto` environment) rather than less.

3 Results

3.1 Usefulness of Profile-Directed Optimization

We gathered cycle counts for each combination of optimizer, benchmark, training profile and evaluation run. We repeated each evaluation run 11 times, discarding the first cycle count score due to significant differences in the first run (almost certainly due to page faults as the program binary is brought into memory from disk). We calculated average cycle counts from the other 10 evaluation runs. We present these average cycle counts normalized by the average cycle counts of the comparison binaries; that is, the optimized binaries that did not used profile-directed optimization. Thus, for a given evaluation run, a binary produced by profile-directed optimization that runs 5% faster than the binary produced by non-profile directed optimization is assigned a score of 0.95 in Table 1.

In Table 1, we present results showing the relative performance of profile-directed optimization for our different benchmarks as compared to the same benchmarks optimized without profile directed optimization. As each benchmark has multiple evaluation and training runs, we present the average profile-driven optimization performance for all of the combinations of evaluation and training runs, excluding the "resubstitution" case.

Overall, profile-directed optimization is an effective technique (an average improvement of 3%), but the results are sharply variable: there are several benchmarks where all training profiles make the program slower for each evaluation run. A majority of benchmarks for both optimizers have at least one combination of training profile and evaluation run where profile-directed optimization performs badly.

Examining the individual benchmark runs, we observe a wide range of performance variability. Table 2 presents the top and bottom benchmark runs by profile-driven optimization variability. There is a huge range of variability among evaluation runs.

Given that cycle counts have a degree of variability due to experimental error, we used a simple technique (one-way ANOVA[2] [8]) to determine whether, for each evaluation run, the differences between cycle counts from binaries trained on different training profiles were significant. That is, were we observing real differences between training runs or were the differences that we observed entirely due to experimental error? This issue is somewhat more pressing for this work than it is for more conventional profile-driven optimization research, as some of the runs which we were using as evaluation runs were comparatively brief (as compared to the standard SPEC "`ref`" runs). The one-way ANOVA procedure ("one-way" because we vary only a single variable; "ANOVA" is short for "ANalysis Of VAriance") attempts to determine, given a set of experimental results gathered at different 'levels' (in this case, using different training profiles), whether there are statistically significant differences among the results for different levels. That is, we attempt to disprove the null hypothesis that the average cycle counts for a given evaluation run are the same regardless of which training profile was used. If the probability that this could be the case is sufficiently low, we can reject this null hypothesis and conclude that in fact there are statistically significant differences between the profile-driven optimization effects of different training profiles.

We were able to reject the null hypothesis of "no significant difference exists between the effect of training profiles" at a significance level of 0.05 (that is, we found that it is no more than 5% likely that, given no effect at all from training profiles, we would have seen the pattern of variability that we did) for all but 5 evaluation runs (4 under `alto` - two runs in the `art` SPEC2000 benchmark and one run each for `compress` and `parser`, 1 under `cc` - one run under `compress`). For the vast majority of our benchmark runs, the probability that we would have observed the variability that we did due to to factors other than the training profile is negligible (under 0.001).

[2] The results for one-way ANOVA are far too verbose to present here, and many of the details are beyond the scope of this paper. One-way ANOVA merely detects that there exists some significant difference between at least one training profile and the rest - it does not in itself yield results analyzing how many of the training profiles differ significantly from the others. Thus, the results of a one-way ANOVA should be treated with a degree of caution - when we say that a significant difference exists for some `perl2000` evaluation run with 10 different training profiles, we are only allowed to say that "some difference exists among the usefulness of those 10 profiles", as opposed to the stronger statement "each and every one of these profiles is significantly different from every other one" or any of the intermediate possibilities. We carried out post-hoc analyses to distinguish between this set of posibilities, but the details are again beyond the scope of the paper.

Optimizer	Benchmark	Number of runs	Normalized execution time		
			Minimum	Maximum	Mean
alto	ammp	3	0.97	0.98	0.98
	bzip2	5	0.87	1.01	0.93
	compress	3	0.94	1.06	0.99
	crafty	3	0.89	0.93	0.91
	gap	3	0.95	0.97	0.95
	go	5	0.96	1.06	0.99
	gzip	7	1.00	1.14	1.06
	ijpeg	3	0.96	0.98	0.97
	li	3	0.97	0.99	0.98
	m88ksim	3	0.83	1.00	0.89
	mcf	3	1.00	1.02	1.01
	parser	3	1.00	1.02	1.01
	perl2000	10	0.83	1.08	0.96
	twolf	3	0.93	1.01	0.97
	vortex2000	5	0.86	0.91	0.89
	ALL CASES		0.83	1.14	0.97
cc	ammp	3	0.99	1.04	1.02
	bzip2	5	0.91	1.06	0.96
	compress	3	0.92	1.02	0.99
	crafty	3	0.94	0.98	0.96
	equake	3	0.95	1.01	0.99
	gap	3	0.92	0.99	0.95
	go	5	0.99	1.14	1.06
	ijpeg	3	0.94	0.98	0.96
	li	3	0.84	0.92	0.87
	m88ksim	3	0.88	1.07	0.96
	mcf	3	0.99	1.00	1.00
	perl2000	10	0.86	1.13	1.00
	twolf	3	0.93	0.98	0.95
	vortex2000	5	0.90	0.99	0.94
	ALL CASES		0.84	1.14	0.97

Table 1: Execution time of PDO binaries over all evaluation runs and training profiles (each set of evaluation run results normalized such that the non-profile-directed optimization case is equal to 1.0 for each evaluation run).

Optimizer	Benchmark	Evaluation run	Fastest Case	Slowest case	Mean	Standard Deviation
alto	perl2000	train/diffmail	0.90	1.05	0.96	0.0457
alto	perl2000	ref/diffmail	0.90	1.06	0.96	0.0457
alto	perl2000	ref/perfect	0.80	0.96	0.89	0.0446
cc	perl2000	train/scrabble	0.82	1.00	0.93	0.0433
cc	go	ref2	1.00	1.12	1.08	0.0427
cc	go	train	1.01	1.14	1.08	0.0426
cc	go	test	1.00	1.12	1.08	0.0412
alto	perl2000	ref/makerand	0.78	0.93	0.87	0.0408
...	...	...	...	...	...	...
alto	gzip	program	1.13	1.14	1.13	0.0021
alto	parser	ref	1.00	1.00	1.00	0.0020
alto	ijpeg	train	0.98	0.98	0.98	0.0013
alto	ammp	train	0.98	0.98	0.98	0.0010
cc	mcf	ref	1.00	1.00	1.00	0.0009
alto	parser	train	1.01	1.01	1.01	0.0008
alto	ammp	ref	0.98	0.98	0.98	0.0006

Table 2: Evaluation runs with highest and lowest variability due to profile-directed optimization profile choice; units are normalized as for Table 1.

3.2 Connection of Usefulness and Accuracy

3.2.1 Profile Accuracy Metrics

All of our comparison metrics compare a list of basic block counts in a training profile with a list of basic block counts in an evaluation profile. They return a single number, a score that indicates how well the basic block counts in the training profile predict the basic block counts in the evaluation profile. Thus, a more accurate training profile better predicts the CFG-level behavior of the evaluation run. Most of these metrics are asymmetric.

A profile comparison metric consists of a comparison type and a way of applying it over the program. The comparison types we use in this paper are key-matching, static coverage and relative entropy.

Key-matching is introduced in [7]. It uses a parameter that determines how many blocks are selected for key-matching. For example, if a function has 50 blocks, and the matching level is 40% (or 0.4), then we perform key-matching on the top 20 blocks as follows: the key-match score is the number of blocks in the top 20 blocks in the training profile that are also in the top 20 of the evaluation profile. Key-matching metrics are denoted by KM(level) - "level" is always 0.1 in this paper.

Static coverage (denoted "STCOV") measures what proportion of the blocks executed ("covered") in the evaluation profile are also executed in the training profile.

Relative entropy (denoted "ENT") as a method of comparing profiles was introduced by Savari and Young [5] and is fully described there. Relative entropy treats the profiles being compared as distributions of random variables and uses an information-theoretic approach to measure the difference between the two distributions.

We use two methods for applying these comparisons to our programs. Firstly, we can apply the comparisions to the whole program's set of basic block counts directly. This is the default method. Secondly, we can apply them only to the entry counts of functions, ignoring all other basic block data (denoted by prefixing "FE-" to the comparison name in our results).

3.2.2 Evaluating the Connection Between Comparison Metrics and Usefulness

To measure the association between profile usefulness and a given profile comparison metric, we use the Spearman Rank Correlation Coefficient [8], r_s. r_s can be calculated by assigning ranks to the values being compared (scoring

ties as the average rank values - so if there is a tie between the top two values, they both are assigned the rank of 1.5) and calculating the more familiar Pearson correlation coefficient [8] over those ranks. Thus, calculations of r_s discard the magnitude of the differences between data points. This makes r_s weaker (more likely to miss a real effect) than Pearson's correlation coefficient but much more robust in the presence of non-linear relationships, outliers and (more generally) data that does not hold to a bi-variate normal distribution.

When analyzing the correlation between profile accuracy and usefulness, we must be aware that there is no "natural" population of profiles for a given benchmark. For most benchmarks, we have a limited number of runs available to us, and they have been chosen artificially. If we include other profile types besides profiles derived directly from real runs, we are introducing further artificial biases into our population. Admittedly, the choice of benchmark runs from the SPEC benchmark sets are artificial also, but they are not the artificial choices of the authors of this paper - that is, they are not hand-picked to advance our favored hypotheses.

We will proceed to show an example of how we evaluate the connection between profile usefulness and accuracy. Firstly, we present the average cycle count scores and usefulness scores for the benchmark `perl2000` and the `ref/perfect` benchmark evaluation run. For each training profile, we have an average cycle count (reflecting how many cycles the binary that was produced by profile-driven optimization using that profile took to run the evaluation run) and an accuracy score (reflecting how close the training profile was to the profile produced by the evaluation run). For this example, we will use the accuracy scores provided by relative entropy[3].

Table 3 shows the cycle counts and relative entropy scores for a list of training runs (the names refer to the different benchmark runs available for `perl2000` and are not of any interest aside from the fact that they label cases). To calculate a score for how closely relative entropy predicts scaled cycle counts, we take the r_s value of two variables (cycle count and relative entropy) over the list of cases (training profiles), which turns out to be $r_s = 0.87$. This value is statistically significant at the 0.01 level; that is, if there was no association whatsoever between two variables, we'd expect to see a r_s value this high less than 1 in 100 times. In fact, the chance that we would see such a strong association between two unconnected variables in such a list of cases is less than 1 in 1700. The proportion of scaled cycle count variation explained by relative entropy is $r_s^2 = 0.75$ - that is, 75% of the variation in average profile-driven optimization performance in this particular case can be explained in terms of relative entropy.

Training run	Cycle count (GCycles)	Relative entropy
`ref/diffmail`	45.819	8.05
`ref/makerand`	47.581	22.57
`ref/perfect`	40.774	0
`splitmail1`	46.495	8.52
`splitmail2`	45.640	8.20
`splitmail3`	47.176	8.35
`splitmail4`	45.281	8.29
`train/diffmail`	45.615	8.06
`train/perfect`	42.515	2.45
`train/scrabble`	48.923	20.44

Table 3: Example 1: `perl2000` scaled cycle counts and accuracy metrics for a single evaluation run (`ref/perfect`)

Note that this benchmark has a quite large number of possible training profiles (10). Many of our benchmarks have only 3 or 4 runs available, so we are often in the situation of calculating correlations over a tiny set of cases. In this circumstance, it is possible to have apparently strong correlations that are in fact statistically meaningless *on their own* . Only when they occur as a pattern across multiple evaluation runs and/or benchmarks can we attach any weight to these results.

Table 4 shows this analysis repeated for all of our evaluation runs in `perl2000`. We will see a larger set of results - now, we have a table with r_s numbers for each evaluation run. Not all of the correlations are significant at a 0.01 level (those that are are marked with a "**") or even at a 0.05 level (marked with a "*"). For example, the value $r_s = 0.382$, seen for the evaluation run `ref/diffmail` is fairly low: there is a 14% chance that two unconnected variables might show a rank correlation equal to or greater than this value (3 evaluation runs fall into the category of

[3]More accurate profiles produce lower relative entropy scores, zero represents a perfect match

Evaluation run name	r_s score
ref/diffmail	.382
ref/makerand	.778**
ref/perfect	.867**
splitmail1	.697*
splitmail2	.612*
splitmail3	.685*
splitmail4	.612*
train/diffmail	.394
train/scrabble	.285

Table 4: Example 2: All perl2000 evaluation runs with the rank-correlation values of cycle counts and relative entropy calculated over each training run

not being significant at the 0.05 level). However, even considering only these three values in isolation, it is unlikely that we would see three such correlations (that is, positive and in the range $0.285 < r_s < 0.394$) between relative entropy and average cycle count if overall, there was no connection between relative entropy and average cycle count for any of these runs. In fact, the chance that such three correlations this strong or stronger would have arisen by chance given no connection between relative entropy and cycle count is about 1%.

Note that it is quite possible to have negative r_s scores; in this case, more accurate profiles actually result in worse profile-driven optimization performance.

We can compute a summary value for the overall connection of usefulness and accuracy over a benchmark by averaging the r_s values for each evaluation run, yielding an aggregate correlation of $mean(r_s) = 0.59$ for the perl2000 benchmark.[4]

Using such a procedure to gather aggregate numbers for each benchmark, this time over a range of comparison metrics, we derive Table 5. This table shows the aggregate r_s scores for each comparison, benchmark and optimizer, as well as overall mean scores for r_s comparison and optimizer. It is clear that the perl2000 benchmark, presented above, and particularly the perfect evaluation run, represent a quite favorable case - note the large number of benchmarks in this table for which the aggregate r_s scores are either very low (i.e. no correlation) or actually negative (i.e. more accurate profiles have worse profile-driven optimization performance). Particularly, the results for the cc optimizer show no overall pattern of a connection between profile usefulness and profile accuracy.

In the alto case, all of the profile comparison metrics yielded small but significant correlations between profile accuracy scores and profile usefulness scores. Key-matching performed slightly worse than the other two profile accuracy metrics, entropy and static coverage. The "function-entry" versions of these latter accuracy metrics performed slightly better than the versions that considered all of the basic blocks in the program, although such a small difference is not likely to be significant.

There was a substantial amount of variability among the aggregate r_s scores for each benchmark. Some of this variability is simply random; the aggregate r_s scores for the benchmarks with a small number of runs are subject to a great deal of randomness as they involve comparisons among only 9 or 16 values. However, some benchmarks clearly have far stronger associations between usefulness and accuracy than others. Recall that the correlation coefficients in this table rank how well profile usefulness correlates with profile accuracy; they have nothing to say about how well profile-directed optimization works overall.

A major weakness of the above approach to evaluating the connection of profile-directed optimization performance and profile accuracy is that, due to the use of non-parametric methods and averaging across different benchmarks, small variations in one benchmark are weighted as heavily as huge variations in another. There is no simple way to avoid this problem without recourse to parametric correlation methods. However, we can derive results that are more useful by restricting our above analyses to only those evaluation runs with greater variability due to profile-directed optimization. The overall (per-optimizer) results from restricting our analysis to the top half of evaluation runs with the highest level of profile-directed optimization variability are shown in Table 6.

The failure of our profile accuracy metrics to explain

[4]This is not generally good practice; more statistically rigorous is to transform each r_s value to a z-score (normal score), take the average over these z-scores and transform back into the range of r_s. However, this procedure is complex and results in average r_s scores little different from those that we derive from simple averaging. Similarly, we will not present significance results for aggregate r_s scores here; the statistical justification for these results in beyond the scope of this paper.

Optimizer	Benchmark	$mean(r_s)$					
		ENT	STC	KM(0.1)	FE-ENT	FE-STC	FE-KM(0.1)
alto	ammp	-0.67	-0.79	-0.50	-0.50	-0.58	-0.67
	art	0.20	0.23	0.35	0.25	0.07	0.23
	bzip2	-0.12	-0.06	0.09	0.00	0.14	0.16
	compress	1.00	0.91	0.83	1.00	0.29	0.58
	crafty	0.83	0.67	0.67	0.67	0.17	0.50
	equake	0.17	0.17	0.00	0.17	0.00	0.58
	gap	0.83	0.83	0.50	0.83	0.79	0.50
	go	0.26	0.13	0.20	0.34	0.30	0.23
	gzip	-0.16	-0.24	-0.28	-0.26	0.12	-0.24
	ijpeg	-0.83	-0.67	-0.67	-0.83	0.00	-0.79
	li	0.83	0.96	0.50	0.83	0.96	0.50
	m88ksim	0.67	0.67	0.67	0.67	0.79	0.67
	mcf	0.50	0.62	0.67	0.50	0.58	0.87
	parser	-0.50	-0.83	-0.83	-0.50	-0.29	-0.67
	perl2000	0.59	0.60	0.45	0.52	0.60	0.50
	twolf	0.33	0.33	0.17	0.33	0.58	0.29
	vortex2000	0.38	0.36	0.38	0.64	0.17	0.48
	vpr	0.52	0.59	0.57	0.55	0.51	0.54
	alto MEAN	0.27	0.25	0.21	0.29	0.29	0.24
cc	ammp	0.00	-0.04	0.33	0.33	-0.58	0.00
	bzip2	0.16	0.04	0.15	0.06	-0.07	-0.09
	compress	0.33	0.17	0.17	0.33	0.29	0.29
	crafty	0.67	0.33	0.00	0.33	-0.46	0.00
	equake	-0.83	-0.83	-0.50	-0.50	0.00	0.00
	gap	-0.33	-0.33	-0.33	-0.33	-0.46	-0.17
	go	-0.72	-0.76	-0.74	-0.72	-0.60	-0.65
	ijpeg	0.33	0.00	0.00	-0.33	0.00	-0.33
	li	-0.17	-0.46	0.33	-0.17	-0.46	0.33
	m88ksim	-0.17	-0.17	-0.17	-0.33	-0.12	-0.17
	mcf	0.33	0.33	0.17	0.33	0.58	0.29
	perl2000	-0.18	-0.23	-0.17	-0.19	-0.21	-0.14
	twolf	0.33	0.33	0.17	0.33	0.58	0.33
	vortex2000	0.04	-0.01	-0.02	0.06	0.02	0.06
	cc MEAN	-0.06	-0.12	-0.04	-0.06	-0.11	-0.02

Table 5: The connection of usefulness and accuracy: aggregated r_s scores over optimizers, benchmarks and different comparison metrics

Optimizer	ENT	STC	KM(0.1)	FE-ENT	FE-STC	FE-KM(0.1)
alto mean	0.57	0.52	0.48	0.58	0.45	0.45
cc mean	-0.15	-0.22	-0.14	-0.17	-0.03	-0.03

Table 6: Aggregated r_s scores over optimizers, considering only the top half of evaluation runs by PDO variability

`cc` profile-directed optimization performance turns out to be unconnected to profile-directed optimization variability. Even considering only benchmarks and benchmark runs that had large variations in profile-directed optimization performance did not improve the connection between profile accuracy and profile usefulness when using `cc`. However, our `alto` results become substantially stronger when we eliminate benchmark runs with small variations in profile usefulness. Entropy-based methods, in particular, improve markedly. The "FE-ENT" accuracy metric predicts 34% of the variation in our profile-directed optimization results under `alto` - a modest result, but the strongest one so far.

We found no similar improvements from restricting our analysis to smaller (e.g. top quarter by PDO variability) subgroups of our evaluation runs. Not suprisingly, the bottom half of evaluation runs by PDO variability showed no significant correlation (under `alto` or `cc`) between profile accuracy and profile usefulness.

3.2.3 Discussion

There was no reason to suppose that any reliable connection between accuracy and usefulness existed in the `cc` optimization context whatsoever. We conjecture that the much more extensive and high-level optimizations present in `cc` sufficiently transform the control-flow-graph to the point where the relatively subtle differences between training profiles are irrelevant. This does not mean that profile-driven optimization does not work in `cc`, nor does it mean that arbitarily inaccurate profiles will produce profile-driven optimization performance indistinguishable from good ones. What it does mean is that, within the fairly narrow range of profiles and benchmarks we tested, accuracy could not be shown to have any connection to usefulness. We evaluated many other profile comparison metrics than (carrying out key- and weight-matching at multiple levels, using dynamic coverage) presented here and found that none of them performed any better than the comparison metrics presented.

Our results for the `alto` optimization context were more encouraging, but still relatively weak. Even when restricting our analysis to benchmarks with large profile-directed optimization variability, we could explain no more than a third of the variation in average cycle counts by some accuracy metric.

One of the most startling results was the fact that the accuracy metric "FE-STC" performed as well as it did despite the fact that it ignores away nearly all of the information in the block profile. This extremely simple metric can be calculated by determining the number of functions entered in the training profile and the evaluation run divided by the total number of functions entered in the evaluation run.

The effectiveness of this metric (and similarly restricted metrics) could result from there being little variation in within-function behavior from run to run (that is, when profiles produced from benchmark runs differ, it is because they cover a different set of functions, not because they have radically different behavior within those functions). An alternate possibility is that the optimizations in `alto` really only effectively worked at a per-function level and thus made little use of the within-block information (code placement optimizations that work at a whole-function level and procedure inlining are both examples of optimizations that work very well with only per-function information, although both benefit from knowledge of call site counts - or, nearly equivalently, call graph edge counts). These possibilities are not easily separated, although the fact that our "function-entry only" comparison metrics are strongly correlated ($r_s > 0.9$) with their whole-program counterparts for nearly all benchmarks is suggestive that the former possibility is true (across both optimizers, `bzip2` and `gzip` were the only exceptions).

4 Related Work

Wall [7] makes the first systematic attempt to evaluate profile accuracy. Wall compares real profiles and static estimates for accuracy using key- and weight-matching to compare profiles. His comparisons use key- and weight-matching at both fixed levels (top k) and, similar to our work, at levels proportional to the total number of blocks (top $N\%$). He shows strong improvements in accuracy from using real profiles over static estimates. He briefly analyzes some theoretical optimization algorithms, showing weaker results, and warns against unrealistic expectations concerning profile driven optimization. Wu and Larus discuss static estimation in [9], using Dempster-Shafer theory to combine branch prediction heuristics. Key- and weight-matching are used to evaluate the accu-

racy of the static profiling methods. Wagner et. al do a similar analysis to Wall's in [6].

These works do not attempt to establish any connection between profile accuracy and profile-driven optimization performance. Our work diverges from all of these works by connecting accuracy metrics to actual profile-driven optimization performance in two mature optimizers.

Fisher and Freudenberger report that profile data gathered from previous runs yields good branch predictions [3]. They mention the possibility that the differences in real benchmark runs might be related to the benchmark's coverage of the program as opposed to differences in behavior in code that is covered by both runs. This is an interesting observation, which unfortunately they were not able to quantify. Our results suggest that this intuition was correct (at least in terms of what information `alto` was able to use effectively); the comparatively strong predictive value of the accuracy metric "FE-STC" (function entry static coverage) supports this.

An extensive treatment of information-theoretic methods for comparing and combining profiles, including the relative entropy comparison used in this work, appears in Savari and Young [5]. Our work validates the use of relative entropy as a profile comparison metric.

Cohn and Lowney compare the differences in usefulness between profile-driven optimization and static estimation on the Compaq Alpha in [1]. They report a substantial speedup (17%) on the SPEC 95 integer benchmarks from using feedback directed optimization. Their results show a larger effect from profile-driven optimization than this paper; they use more aggressive optimizations on a more recent iteration of the Alpha architecture. Another difference between their work and ours is that we use a wider variety of benchmarks (including SPEC2000 and floating point benchmarks) and benchmark runs than they do; this may also contribute to the performance gap between this paper and their work.

Eeckhout et al. [2] use statistical data analysis techniques to cluster similar "program-input pairs" (in our terms, pairs consisting of a benchmark and an evaluation run). They concentrate on overall benchmark characteristics as opposed to profile accuracy and/or profile usefulness.For our analyses in this paper, we have little need to reduce the number of "program-input pairs" to cover a hopefully representative set of benchmarks, training profiles, and evaluation runs, as our analyses benefit from more data points rather than fewer. This is true even if some of the training profiles and evaluation runs produce very similar effects.

5 Conclusion

Profile-directed optimization is a worthwhile technique, on average, in both of the optimizers evaluated. On average, we saw an improvement over non-profile-directed optimizations of about 3.5% on `alto` and 5% on cc; these aggregate numbers concealed substantial variations (the best case for either optimizer was approximately 17% better than non-profile-directed optimization and the worst case for either was approximately 14% worse).

Nearly all of the benchmark runs showed significant variation in profile-directed optimization performance. In only 1% of our evaluation runs were we unable to detect significant variation among profile-directed optimization performance (that is, no variation due to profile-directed optimization existed or it was so small that we were unable to separate this variation from experimental error). Again, large differences existed between the evaluation runs with the largest amount of profile-directed optimization variability and those with the smallest - the standard deviations in speed-up over the non-profile-directed-optimization case ranged from effectively zero to nearly 5%.

Profile accuracy is only weakly associated with profile usefulness in one of our optimizers (`alto`) and not connected at all with profile usefulness in another (`cc`), for our set of benchmarks and benchmark runs. While considering only benchmarks or runs with higher variability in profile-driven optimization performance improved the connection on `alto`, the connection between usefulness and accuracy still accounted for only 34% of the observed variation in profile-driven optimization performance. While the comparatively weak (non-parametric) correlation methods that we had to use may have caused us to be overly conservative, it seems unlikely that any accuracy metric whatsoever would explain in excess of 50% of the variation. Of the variation in profile usefulness explainable by profile accuracy metrics, much of it was explainable by fairly simple profile accuracy metrics, most notably static coverage of function entries ("FE-STC"). We find some quantitative support for Fisher and Freuden-

berger's claim [3] that differences in exact profiles are mainly due to a different set of functions being covered in different runs, as opposed to different behavior within the functions from run to run.

That the overall results are negative for `cc` and weak for `alto` is not entirely suprising. Much of the variation in our training profiles does not necessarily cause different optimization outcomes. That which does does not necessarily help. Not every optimization "decision" produces better performance, regardless of whether it is based on good information - few compiler optimizations are truly "optimizations", particularly when interacting with many other optimizations. We see substantial and significant variations due to profile choice in profile-driven optimization, and for most benchmarks, much of this variation is not explainable in terms of profile accuracy. This suggests that there is a large component of randomness in the outcome of the profile-driven optimization process.

Our major contributions are twofold. Firstly, we have developed a methodology for evaluation of profile-driven optimization performance and its connection to profile accuracy that can be applied to any combination of processor architecture, optimizer, and set of benchmarks. Secondly, our results show that there exists at least one optimizer for which usefulness and accuracy are not correlated (in our experimental context) and one in which this correlation exists but fails to explain the bulk of profile-directed optimization performance.

Therefore, any claims about profile-directed optimization techniques or more accurate profiling techniques (or the necessity of obtaining more accurate precise basic block profiles - dynamically or otherwise) should be evaluated *experimentally*, not in terms of profile accuracy. We have shown that there are a range of cases where little or no connection between profile accuracy and profile usefulness exists. Thus, it is incumbent on designers of profile-directed optimization systems to demonstrate that the profile-directed optimizations in their systems are actually effective over a wide range of benchmarks, rather than merely showing that the profiles gathered are of high accuracy.

References

[1] R. Cohn and P. Lowney. Feedback directed optimization in Compaq's compilation tools for Alpha. In *In Proc. 2nd Workshop on Feedback Directed Optimization, 1999*, 1999.

[2] L. Eeckhout, H. Vandierendonck, and K. D. Bosschere. Workload design: Selecting representative program-input pairs. In *The Eleventh International Conference on Parallel Architectures and Compilation Techniques (PACT-2002)*, 2002.

[3] J. Fisher and S. Freudenberger. Predicting conditional branches from previous runs of a program. *Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 85–95, 1992.

[4] R. Muth, S. Debray, S. Watterson, and K. de Bosschere. alto: A link-time optimizer for the DEC Alpha. Technical Report TR98-14, Department of Computer Science, The University of Arizona, 1998.

[5] S. Savari and C. Young. Comparing and combining profiles. In *Proc. Second Workshop on Feedback-Directed Optimization (FDO)*, 1999.

[6] T. A. Wagner, V. Maverick, S. L. Graham, and M. A. Harrison. Accurate static estimators for program optimization. *ACM SIGPLAN Notices*, 29(6):85–96, 1994.

[7] D. W. Wall. Predicting program behavior using real or estimated profiles. 26(6):59–70, June 1991. *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*.

[8] R. Walpole, R. Myers, and S. Myers. *Probability and Statistics for Engineers and Scientists*. Prentice Hall, 1998.

[9] Y. Wu and J. Larus. Static branch frequency and program profile analysis. In *In 27th International Symposium on Microarchitecture*, pages 1–11, 1994.