Comparing and Combining Profiles

Serap Savari and Cliff Young
Bell Laboratories, a division of Lucent Technologies
{savari,cyoung}@research.bell-labs.com

How much do two profiles of the same program differ? When has a profile changed enough to warrant reexamination of the profiled program? And how should two or more profiles be combined to make a better hybrid profile? To answer these questions, we borrow concepts from information theory that measure the differences between the distributions of random variables. Treating the distribution of execution frequencies in a profile like the distribution of a random variable lets us reuse metrics of difference between distributions and induces a mathematically well-founded way to combine distributions. For concreteness, we report results based on branch profiles, but our algorithms apply to any frequency-based profiling method. Our metric agrees in character with prior metrics while allowing us to adjust the relative importance of coverage and conflict issues. Our hybrid distributions do better than previous profile blending methods.

1 Introduction

Many recent systems use profiles to guide optimization and translation towards a variety of goals. These systems have been striking in their ability to instrument with extremely low performance overhead (e.g., Compaq's Continuous Profiling Infrastructure [1]), their innovative application of profiling to binary translation (e.g., the FX!32 x86-to-Alpha translation system [7]), and their exhibition of speedups in dynamic optimization (e.g., Hewlett-Packard's Dynamo dynamic optimization project [2]). All of these recent systems blur the traditional barrier between static (compiler) and dynamic (hardware) approaches. What used to be done once at compile time can now be done (and redone) at run time. But to do so effectively, we need to be able to compare profiles, to determine when the behavior of a program has changed significantly, and to combine multiple profiles to produce a better hybrid profile. This work uses information theoretic ideas to attack these problems.

Information theory deals with efficient representations of data streams and related issues like coding, compression, and prediction. Information theory is a related field to computer science and statistics, so it is not surprising that ideas from information theory are relevant to problems in profiling. Some information theory has been used recently to analyze dynamic branch prediction schemes [5,9]. To our knowledge, this is the first work that applies information theoretic concepts to profiling.

Profiles are statistics about the execution of a program; they are commonly execution frequencies but they can also include data about system performance at a variety of levels. Our information-theoretic approach applies to any frequency-based profiling method. For concreteness and simplicity of exposition, we constrain our discussion in this paper to branch profiles; much of the relevant scholarly work also relates to branch profiling.

Our first two problems, comparing profiles and determining when program behavior has changed, suggest that we need some sort of metric of the differences between profiles. Prior work in this area includes the concepts of coverage, conflict, and a geometric measure introduced recently by Kistler and Franz [8]. In their extensive cross-validation study of static branch prediction, Fisher and Freudenberger discuss *coverage*, noting that some runs of the program execute different parts of the program from other runs [6]. They experimented with a number of different metrics but did not find any one that satisfactorily explained their results. Calder, Grunwald, and Srivastava later defined coverage as the percentage of branches from a program run that were also executed during a profile training run [3]. We adopt the latter definition of coverage for this paper, and we define both *static coverage* and *dynamic coverage*. Static coverage is the number of branch instructions touched in both the training and testing runs divided by the number of branch instructions touched during the testing run. Dynamic coverage is similar, but it weighs each branch by its execution frequency. The dynamic coverage is the sum of execution frequencies of instructions in the testing run that were also touched during the training run divided by the total branch execution frequency of the testing run.

Conflict is a relative of coverage: it measures the percentage of branches that change majority direction between the training and testing data sets. Similarly to coverage, we also define statically and dynamically weighted forms of conflict. Coverage and conflict are related in the way that control-flow graph (CFG) node profiles and CFG edge profiles are related: both give frequencies for points of the program, but conflict metrics and edge profiles also capture the exit bias of individual branch instructions. We will return to Kistler and Franz' metric later in the paper; it is better to discuss it after we have introduced our own metric.

Fisher and Freudenberger also examined our third problem, combining multiple profiles into a hybrid profile. They investigated different ways of combining profiles that they called *unscaled*, *scaled*, and *polling*. Unscaled combination simply adds the profile counts from different runs; longer runs therefore carry more weight. Scaled combination divides each data point frequency by the total frequency for that profile, then arithmetically averages the biases seen in different profile runs. Polling gives each data set one vote no matter how large the program run. Fisher and Freudenberger report that polling performs poorly, while scaled and unscaled combining both give good results; they report scaled results because scaling seems intuitively better.

The next section introduces our metric, which is based on the information-theoretic concept of relative entropy or Kullback-Liebler distance. Section 3 then gives an algorithm using geometric averaging and relative entropy to blend a pair of profiles. We present empirical results in Section 4. Lastly, we discuss the implications and usefulness of this work.

2 Relative entropy

Our approach treats a frequency profile as a probability distribution. This makes intuitive sense: a profile says where a program is likely to spend its time, while a probability distribution says which values a random variable is likely to assume. In effect, we treat the program counter as a random variable (even though we know it is not—more about this later) then view the frequency profile as a set of samples drawn from the program counter's underlying probability distribution. The statistical concept of a probability distribution is like the profiling ideal of program behavior. We can never exactly know either, but we can approximate them with increasing accuracy as we see more samples or more behavior.

We translate from profile frequencies to probabilities in the obvious way: scale the frequencies by the total frequency over the profile. This is just the first step of the scaled combination method. More formally, let S denote the set of branch instructions in a program and consider the branch trace of the program, which is a sequence of ordered pairs $(s, j) \in S \times \{0, 1\}$. An appearance of $(s, 0)$ indicates that an execution of the program entered branch s and fell through, while $(s, 1)$ indicates that the branch s was taken. Let $N_{s,j}$, $(s, j) \in S \times \{0, 1\}$ denote the number of times ordered pair (s, j) was encountered in the trace. We call our translation the *independent and identically distributed (i.i.d.) branch and decision model*. In this model, we assume that the sequence of ordered pairs is i.i.d. with probability

$$p_{s,j} = \frac{N_{s,j}}{\sum_{t \in S,\, i \in \{0,1\}} N_{t,i}},\ (s, j) \in S \times \{0, 1\}.$$

At the risk of illustrating the obvious, Table 1 lists some sample frequency profiles and their translations into scaled probabilities under the i.i.d. branch and decision model. Each row of the table shows values for one profile. Each column shows the frequencies or probabilities associated with a single event in the profiled program; an event in an edge profile would be an ordered pair $(s, j) \in S \times \{0, 1\}$. Note that *prof1* covers a large portion of the events, that *prof2* and *prof3* are very similar, that *prof5* covers the same events as *prof2* and *prof3* but with different emphasis, while *prof4* overlaps very little with *prof2*, *prof3*, and *prof5*. We will use these profiles to help explain this section and the next section.

	profile	*event1*	*event2*	*event3*	*event4*
frequencies	*prof1*	250	250	250	250
	prof2	1500	500	10	0
	prof3	300	150	10	0
	prof4	0	0	400	450
	prof5	400	500	600	0
probabilities	*prof1*	0.25	0.25	0.25	0.25
	prof2	0.75	0.25	0.00	0.00
	prof3	0.65	0.33	0.02	0.00
	prof4	0.00	0.00	0.47	0.53
	prof5	0.27	0.33	0.40	0.00

Table 1: Sample profile frequencies and their translation into probabilities under the i.i.d branch and decision model.

Before we can discuss the *relative entropy* of two probability mass functions, we need to define the *entropy* of a random variable. Let X be a discrete random variable taking values from an alphabet Ψ with probability mass function $p(x) = \Pr\{X = x\}, x \in \Psi$. Entropy measures the uncertainty of a random variable; it is a lower bound on the average length of the shortest description of X (see Cover [4], section 1.1). Said differently, entropy is the average number of bits required per symbol to optimally encode a stream of symbols chosen from Ψ using the probability distribution p. The entropy $H(X)$ of X is the expected value of $\log(1/p(x))$, where X is drawn according to $p(x)$, i.e.,

$$H(X) = \sum_{x \in \Psi} p(x) \log \frac{1}{p(x)}$$

Because of continuity arguments, we use the convention that $-0\log 0 = 0$. In encoding a stream of values, we use more bits to represent the rarer symbols; the log of the inverse of the probability of a symbol is the right number of bits to use to encode that symbol. To relate this definition of entropy back to dynamic optimization and translation, the entropy of a profile distribution can be seen as a measure of locality in program execution; a program that spends most of its time in a few places (large $p(x)$ at few values of x) will have low entropy, while a program with poor locality (small $p(x)$ at many values of x) will have high entropy. Dynamic optimization and translation systems exploit the principle of locality, so we expect such systems to do better on programs whose profiles have low entropy.

Table 2 shows the entropy of each sample profile and the components that each event contributes to the total. Each component is a term of the form $p(x)\log(1/p(x))$, where x is just one of the profiled events. Low-probability events contribute significant components, so large numbers of low-probability events will raise the entropy of a profile. Profiles with high entropy tend to have large numbers of events that are evenly balanced in probability. Profiles with low entropy concentrate their frequencies and probabilities in a small number of events.

	entropy components				
profile	*event1*	*event2*	*event3*	*event4*	entropy
prof1	0.50	0.50	0.50	0.50	2.00
prof2	0.32	0.50	0.04	0.00	0.85
prof3	0.40	0.53	0.12	0.00	1.05
prof4	0.00	0.00	0.51	0.49	1.00
prof5	0.51	0.53	0.53	0.00	1.57

Table 2: Per-event components of entropy and total entropy for each sample profile.

Entropy tells us something about the locality of a single profile run, but our goal remains to compare two different profiles. The *relative entropy* or *Kullback-Liebler distance* of two probability distributions on Ψ is a measure of their distance. Relative entropy measures the inefficiency of assuming that the distribution is q when it is actually p. This is like the difference between optimizing or translating for some assumed profile when the program run corresponds to a different actual profile. If we are given two probability mass functions $p(x)$ and $q(x)$, then their relative entropy $D(p \| q)$ is defined by

$$D(p \| q) = E_p\left(\log \frac{p(X)}{q(X)}\right) = \sum_{x \in \Psi} p(x) \log \frac{p(x)}{q(x)}.$$

As before, we use the convention that $0\log(0/q) = 0$ and $p\log(p/0) = \infty$; in a moment we will also adjust some of the profile values to eliminate the infinite terms. The description length interpretation of $D(p \| q)$ is that $H(p) + D(p \| q)$ is a lower bound on the average length of the shortest description of X when p is the true distribution of the random variable and q is the distribution for which the description was designed (see Cover [4], section 2.3). In other words, $D(p \| q)$ is the additional number of bits that are transmitted because of the incorrectly assumed distribution. Relative entropy measures a mismatch between probability mass functions that is like the mismatch between training and testing data sets in profile-based optimization: we optimize with a training profile (assumed distribution), but realistic results are measured with a different, testing profile (actual distribution).

Relative entropy is always non-negative (see Cover [4], section 2.6) and is zero only when the two distributions are identical[1]. Although it does provide a sense of the closeness of two distributions, relative entropy is not truly a distance metric because it is not symmetric and it does not satisfy the triangle inequality. However, the asymmetry models a realistic aspect of profiles: one profile can be a good training set for the other without the reverse being true.

Observe that the relative entropy

$$D_{edge}(p \| q) = \sum_{(s,j) \in S \times \{0,1\}} p_{s,j} \log \frac{p_{s,j}}{q_{s,j}}$$

is infinite when there is a term of the form $p\log(p/0)$. Similarly to the continuity arguments above, we modify any probability distribution q so that $q \geq \varepsilon$ for all x while we simultaneously maintain the ratios among the remaining probability mass values. ε should be chosen smaller than the inverse of the total execution fre-

1. One aspect of the formula seems unintuitive: the logarithm allows us to have negative terms when $p(x) < q(x)$. While this is true for any particular value of x, we never achieve a negative total because negative terms are offset by positive terms where the actual distribution is larger than the assumed distribution.

quency; otherwise the modified values can end up larger than the scaled non-zero frequency events. The choice of ε indicates how much we wish to penalize a lack of coverage (smaller values of ε will give larger metric values for points where the training set does not cover the testing set). For the experimental results in this paper, we used a value of $\varepsilon = 1 \times 10^{-30}$.

D_{edge} is a more sophisticated metric than coverage and conflict, because D_{edge} examines the relative frequencies of nodes and edges in the assumed and actual profiles rather than summarizing nodes by presence or absence and edge pairs by majority direction. We would expect that such continuous metrics would be more sensitive than binary ones: e.g., the difference between branch biases of 5% and 95% on different profiles is much larger than biases of 49% and 51%.

Table 3 shows the relative entropies of pairs of our sample profiles. The data for each actual data set appear in rows, while the assumed data sets appear in columns.[2] The diagonal of the matrix is always zero: a profile exactly matches itself. We see very high relative entropies when the assumed profile does not cover the actual profile, e.g., no other profile is a good assumed data set for *prof1*, while *prof4* is a bad assumed data set for all other actual data sets. On the other hand, *prof2* and *prof3* have low relative entropy because they are close in character. *Prof5* covers the same events as *prof2* and *prof3*, but with different emphasis; this distinction is correctly reflected by the slightly higher values for relative entropy when using *prof5* rather than *prof2* or *prof3* for either the assumed or the actual distributions.

assumed / actual	*prof1*	*prof2*	*prof3*	*prof4*	*prof5*
prof1	0.00	25.43	24.85	48.33	24.12
prof2	1.15	0.00	0.04	98.31	0.97
prof3	0.95	0.05	0.00	96.47	0.74
prof4	1.00	55.36	54.36	0.00	52.38
prof5	0.43	2.28	1.35	58.66	0.00

Table 3: Relative entropies of the sample profiles.

One aspect of D_{edge} is unsatisfying: nothing relates the edges that exit a single control-flow graph node to one another. I.e., the counts $N_{s,0}$ and $N_{s,1}$ are related by the logic of the program in a way that $N_{s,0}$ and $N_{t,0}$ are not, but our metric does not capture this difference. We also examined a couple of metrics that strove to capture this difference. Neither alternate metric seemed very different in character from D_{edge}, so we omit them from this paper.

2. These are usually called testing data set (actual) and training data set (assumed). SPEC already uses these terms for their three input data sets: "ref", "train," and "test." So we use *assumed* and *actual* to avoid confusion.

3 Combining profiles

Our algorithm for combining two frequency profiles has a simple implementation and description, but there is a more complex argument for why the algorithm produces useful results. We present the algorithm first; those who are uninterested in the mathematical justification can skim the remainder of this section.

To blend two profiles, we need to find a synthetic profile that is a compromise between the two. Assume our two profiles have been translated into probability distributions using the i.i.d. branch and decision model; call them P_0 and P_1. Then we will search values of the parametric probability mass function P_λ, defined as

$$P_\lambda(x) = \frac{P_0^\lambda(x) P_1^{1-\lambda}(x)}{\sum_{a \in \Psi} P_0^\lambda(a) P_1^{1-\lambda}(a)}$$

P_λ is a sort of geometric weighting of P_0 and P_1; $\lambda = 0$ gives P_0 and $\lambda = 1$ gives P_1, while $\lambda = 1/2$ gives a geometric average of the two probability distributions.

Figure 1 illustrates how P_λ relates to the other probability distributions. The triangle represents the *probability simplex*, the space of probability distributions over Ψ. P_0 and P_1 are points in this space. The space is divided into two parts: points that are closer to

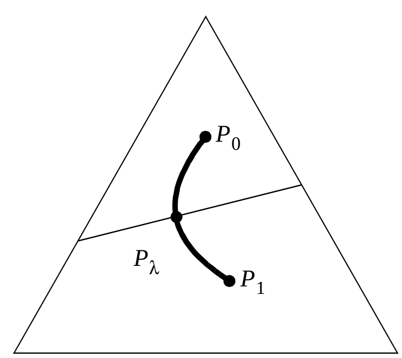

Figure 1: The probability simplex, showing the original distributions P_0 and P_1, and the range of values of the parameterized distribution P_λ. Distributions in the upper part of the simplex are closer to P_0, while those in the lower part of the simplex are closer to P_1. (After Cover [4], Figure 12.9).

P_0, and parts that are closer to P_1. The values of P_λ trace out an arc from P_0 to P_1; we will search along this arc for our hybrid probability distribution.

Next, we use the relative entropy metric from Section 2 to choose the value of λ; we look for a value of λ such that P_λ is equidistant from both P_0 and P_1. We could do this directly, but there is an easier way. To do this, we build a function $c(\lambda)$ that computes

$$c(\lambda) = \sum_{x \in \Psi} P_0^{\lambda}(x) P_1^{1-\lambda}(x)$$

then binary search the interval [0,1] for λ^*, the value that minimizes this function. Then P_{λ^*} is our synthesized profile distribution.

Table 4 tabulates part of the search space when our algorithm combines *prof2* with *prof5*. The first column depicts values of λ in the interval $[0, 1]$; the second through fifth columns show the corresponding probabilities for each event and that value of λ, and the sixth column shows the value of $c(\lambda)$, which is minimized between $\lambda = 0.50$ and $\lambda = 0.63$. Figure 2 charts the values of the relative entropies from the synthesized profile to the original profiles; it captures the trade-off from favoring one profile to the other as we vary λ from 0 to 1.

Our algorithm runs in time proportional to the number of profile events times the number of bits of accuracy desired in λ; in all of our experiments our computations were I/O-bound.

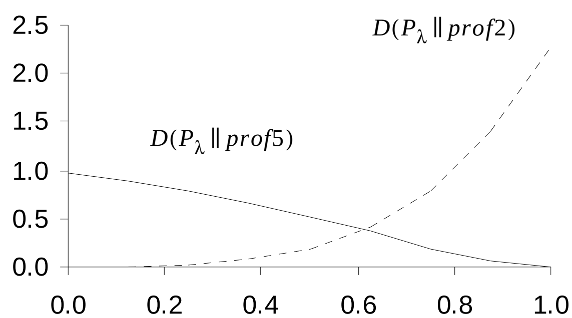

Figure 2: Values of the relative entropies from the hybrid profile P_λ to the original profiles *prof2* and *prof5*. The horizontal axis shows the values of λ. (After Cover [4], Figure 12.10).

We describe how our algorithm performs for combining static branch profiles in Section 4.2; now we continue with the formal explanation of how and why it works.

	Values of P_λ				
λ	*event1*	*event2*	*event3*	*event4*	$c(\lambda)$
0.00 = *prof2*	0.75	0.25	0.00	0.00	1.00
0.13	0.71	0.28	0.01	0.00	0.92
0.25	0.67	0.31	0.02	0.00	0.86
0.38	0.63	0.34	0.03	0.00	0.81
0.50	0.57	0.37	0.06	0.00	0.78
0.63	0.51	0.39	0.10	0.00	0.77
0.75	0.44	0.39	0.17	0.00	0.79
0.88	0.35	0.38	0.27	0.00	0.86
1.00 = *prof5*	0.27	0.33	0.40	0.00	1.00

Table 4: Part of the search space of synthetic profiles when combining *prof2* and *prof5*.

The justification for our algorithm reverses an argument from the theory of hypothesis testing. In hypothesis testing, we have a number (or in our case just a pair) of hypotheses, one of which is true. Based on some amount of evidence, we try to tell which of the hypotheses is the correct one; hypothesis testing analyzes the probability of error and allows us to weigh the hypotheses based on the evidence to minimize errors. The reversal comes from how we think about the evidence we have seen so far. Instead of using the evidence we have seen so far to guess the correct hypothesis, we will construct a set of evidence that is simultaneously close to both original profiles but hard to use to choose which one is correct.

Suppose we are given independent and identically distributed random variables $X_1, X_2, ..., X_n$ that have probability mass function $Q(x)$. We have two hypotheses H_0 and H_1 for the nature of $Q(x)$. Either $H_0{:}Q = P_0$ or $H_1{:}Q = P_1$. Our choice for the solution is based on a decision function $g_n(x_1, ..., x_n)$ which returns the value zero if it chooses hypothesis H_0 and otherwise returns the value one. With this rule there are two error probabilities

$$\alpha_n = \Pr(g_n(x_1, ..., x_n) = 1 | H_0 \text{ true})$$
$$\beta_n = \Pr(g_n(x_1, ..., x_n) = 0 | H_1 \text{ true})$$

We would like to minimize both error probabilities, but there is a trade-off between them.[3] It is possible to show that the best decision rules are likelihood tests of the form

$$\text{Choose } H_0 \text{ if } \frac{P_0(x_1, ..., x_n)}{P_1(x_1, ..., x_n)} > T \text{ and } H_1 \text{ otherwise}$$

for some $T \geq 0$ (see Cover [4], section 12.7). If P_{X^n} is the empirical probability distribution corresponding to $(x_1, ..., x_n)$, then we can further show that the likelihood test

$$\frac{P_0(x_1, ..., x_n)}{P_1(x_1, ..., x_n)} > T$$

is equivalent to

$$D(P_{X^n} \| P_1) - D(P_{X^n} \| P_0) > \frac{1}{n}\log T.$$

Our goal is to find the P_{X^n} that is close to both P_0 and P_1 while still making the decision hard. Our metric tells us the distance from P_{X^n}, the distribution we have seen so far, to the two profiles, P_0 and P_1. There is a space of such empirical distributions P_{X^n} (depicted in Figure 1), and we can use the distance metric to partition them into three pieces: distributions that are closer to P_0, those that are closer to P_1, and those that are equidistant to both P_0 and P_1. Our desired distribution will be in the equidistant set, and it must also be close to the originals.

Suppose that we assign a prior probability $\pi_j, j \in \{0, 1\}$, to hypothesis H_i. In other words, before we see any of the X_i, there is a π_j chance that hypothesis H_j is correct (of course, $\pi_0 + \pi_1 = 1$). In this case, the *a posteriori* probability of error (the odds that we misidentify after having seen *n* samples) is

$$P_e^{(n)} = \pi_0\alpha_n + \pi_1\beta_n$$

Next, define

$$D^* = \lim_{n \to \infty} \min_{g_n} \; -\frac{1}{n}\log P_e^{(n)}$$

As we increase the number of samples that we see, our likelihood of making an error using the best decision rule decreases exponentially. Roughly speaking, D^* is the rate of decay or base of this exponential curve; it describes the rate at which errors decrease as we see more samples.

Then the Chernoff bound (see Cover [4], section 12.9) states that

$$D^* = D(P_{\lambda^*} \| P_0) = D(P_{\lambda^*} \| P_1)$$

and

$$D^* = -\log \sum_{x \in \Psi} P_0^{\lambda^*}(x) P_1^{1-\lambda^*}(x)$$

where

$$\lambda^* = \arg \min_{0 \leq \lambda \leq 1} \sum_{x \in \Psi} P_0^{\lambda^*}(x) P_1^{1-\lambda^*}(x)$$

i.e., λ^* is the value of λ that minimizes the maximum of $D(P_\lambda \| P_0)$ and $D(P_\lambda \| P_1)$, the relative entropy distances from P_λ to P_0 and P_1. Further, the structure of the formula guarantees that the function P_λ is convex in the range [0,1], so λ^* exists and is unique.

4 Experimental results

We collected profiles on a 533MHz Alpha 21164 clone with a 164LX motherboard, 2MB of board cache, and 128MB of main memory. The Alpha runs Digital UNIX version 4.0D (revision 878); we profiled using Atom version 2.47 and custom analysis files. We compiled the SPECint95 benchmarks using base optimization settings. Due to current limitations of our analysis code, we collected statistics for only the last run of each

---

3. To make this example concrete, suppose that we have a coin which might be fair (H_0), or might be biased in favor of heads with a probability of heads equal to 75% (H_1). We get to observe n tosses of this coin, $x_1, ..., x_n$, then based on the values of these tosses, the decision function g_n tells us which hypothesis is more likely. α_n is the probability that the decision function g_n incorrectly tells us that we have the unfair coin when in fact we have a fair coin. β_n is the probability of the reverse failure, where g_n decides incorrectly that we have a fair coin when in fact we have the unfair one. We can trivially minimize α_n or β_n by using the constant decision functions $g_n = 0$ or $g_n = 1$, respectively, but finding a good trade-off between α_n and β_n requires more analysis.

Training Run	all	ref		test		train	
Benchmark	static	static	dynamic	static	dynamic	static	dynamic
099.go	5740	5603	3669393694	5464	1833855003	5046	59750628
124.m88ksim	1418	1408	8135706733	1219	61291271	1164	11682553
126.gcc	19370	14592	26190611	18565	199369904	18644	202317776
129.compress	430	430	4975851051	407	216427	415	3496670
130.li	879	870	8605396422	635	148521036	636	25032723
132.ijpeg	1373	1338	1515921676	1354	42129447	1365	93933314
134.perl	2307	2080	3114254002	1630	1186183	2047	5141087
147.vortex	6501	6482	9334649279	6488	1088823792	6487	302461902

Table 5: Benchmark and profile branch counts

training, testing, or reference input set. Table 5 summarizes the number of static and dynamic branches our profiles recorded on each profiling run. We use static and dynamic in the same sense as in the definitions for our metrics: one static branch instruction in the program text might be executed many times during a run; each of these executions of the static branch is a dynamic branch.

We collected three profiles for each benchmark, one for each of the reference, testing, and training inputs. These are described by the file SPEC95/doc/RUN.txt, which is part of the SPEC95 distribution:

> ...“ref” is the input set for generating publishable numbers, “train” is used for generating feedback directed optimization (if used) and “test” is a short input for helping verify that the benchmark was compiled correctly.

When reporting SPEC results, only the “train” inputs can be used for profiling. Typically the train and test inputs are smaller than the reference inputs.

4.1 Metrics for comparing profiles

Evaluating the effectiveness of a metric is difficult: a “good” metric is one that turns out to predict or measure something useful. We do not have a dynamic profiling/optimization system in which to evaluate our metric, so this section uses static branch prediction accuracy as a stand-in for performance. This is somewhat unsatisfying: we would really like to try our metric in an operating dynamic profiling and optimization system with a concrete goal such as performance, power, cost, etc. Evaluating our metric in other systems with other performance criteria is beyond the scope of this study.

Table 6 lists a plethora of statistics for each benchmark. The columns list assumed-actual profile pairs, i.e., each statistic was collected by training with the assumed data set then testing using the actual data set. There are seven rows for each benchmark. The first four show static and dynamic values for the coverage and conflict metrics defined in Section 1; these have values between 0 and 1. To simplify viewing, we have shaded all cases where dynamic coverage was less than 95% or dynamic conflict exceeded 5%. The next row lists our edge-based relative entropy metric; this ranges from 0 to infinity. The last two rows show the static branch prediction accuracies; these take values from 0 (bad) to 1 (perfect). The “resubstitution” row lists the prediction accuracy from training and testing on the actual data set (it ignores the assumed data set, so we see identical numbers for the same actual data set); this is provided as an upper bound on possible static prediction accuracy. The “cross-validated” row shows the prediction accuracy when training on the assumed data set and running on the actual data set; in all cases the cross-validated value will be less than the resubstitution value. To simplify viewing, we have shaded all cases where prediction accuracy drops by more than 2% from the resubstitution value to the cross-validated value.

Many of the static coverage values seem low: every benchmark except *132.ijpeg* and *147.vortex* have some assumed-actual pair where the static coverage is lower than 95%. In contrast, only a few of the dynamic coverage statistics are poor (*124.m88ksim* test-*, *129.compress* test-ref, four of the 6 cases on *134.perl*). For these profiles, low dynamic coverage seems a much better indicator of poor prediction performance than low static coverage. Unsurprisingly, low dynamic coverage always occurs with a large drop from resubstitution prediction accuracy to cross-validated prediction accuracy (when there is no coverage from the assumed data set, we predict branches will be taken). Coverage is a necessary but not sufficient condition for a good profile.

Static conflict is as unilluminating as static coverage: all benchmarks except *129.compress*, *132.ijpeg*, and *147.vortex* have some assumed-actual pair with greater than 5% static conflict. Dynamic conflict is much better: all values greater than 5% correspond to

Metric (Assumed / Actual)	Benchmark	test-ref	train-ref	ref-test	train-test	ref-train	test-train	Benchmark	test-ref	train-ref	ref-test	train-test	ref-train	test-train
static cover	099.go	0.9522	0.8938	0.9764	0.8840	0.9925	0.9572	130.li	0.7195	0.7310	0.9858	0.9150	1.0000	0.9135
dynamic cover	099.go	1.0000	0.9998	0.9986	0.9984	1.0000	1.0000	130.li	0.9883	0.9827	0.9941	0.9793	1.0000	0.9982
static conflict	099.go	0.1087	0.1624	0.0943	0.1720	0.1140	0.1375	130.li	0.2379	0.2276	0.0882	0.1008	0.0645	0.0912
dynamic conflict	099.go	0.0025	0.0074	0.0030	0.0086	0.0050	0.0059	130.li	0.0583	0.0418	0.0780	0.0374	0.0395	0.0464
KL edge	099.go	0.0316	0.2792	0.1503	0.3771	0.1982	0.1887	130.li	1.4965	2.4849	1.0253	2.7121	0.4329	0.8071
resubstitution	099.go	0.7709	0.7709	0.7723	0.7723	0.7555	0.7555	130.li	0.8512	0.8512	0.8751	0.8751	0.8800	0.8800
cross-validated	099.go	0.7684	0.7635	0.7693	0.7637	0.7505	0.7496	130.li	0.7929	0.8094	0.7971	0.8377	0.8405	0.8336
static cover	124.m88ksim	0.8636	0.8217	0.9975	0.8154	0.9940	0.8540	132.ijpeg	0.9963	0.9948	0.9845	0.9963	0.9751	0.9883
dynamic cover	124.m88ksim	0.9113	0.9969	1.0000	0.9950	1.0000	0.8498	132.ijpeg	1.0000	1.0000	0.9816	1.0000	0.9866	0.9999
static conflict	124.m88ksim	0.0902	0.1321	0.0353	0.1526	0.0395	0.1091	132.ijpeg	0.0120	0.0105	0.0199	0.0126	0.0249	0.0190
dynamic conflict	124.m88ksim	0.0238	0.0207	0.0003	0.0446	0.0180	0.0589	132.ijpeg	0.0001	0.0001	0.0096	0.0008	0.0049	0.0004
KL edge	124.m88ksim	8.9533	3.7468	0.2066	5.4362	2.0927	17.3343	132.ijpeg	0.4944	0.1775	2.6925	0.1655	1.5523	0.1590
resubstitution	124.m88ksim	0.9209	0.9209	0.9291	0.9291	0.9373	0.9373	132.ijpeg	0.8909	0.8909	0.8902	0.8902	0.8954	0.8954
cross-validated	124.m88ksim	0.8971	0.9002	0.9288	0.8845	0.9192	0.8784	132.ijpeg	0.8908	0.8908	0.8806	0.8893	0.8905	0.8950
static cover	126.gcc	0.9885	0.9908	0.7769	0.9624	0.7755	0.9583	134.perl	0.6750	0.9837	0.8613	0.8601	0.9995	0.6849
dynamic cover	126.gcc	0.9999	0.9999	0.9860	0.9997	0.9964	0.9999	134.perl	0.8935	1.0000	0.7293	0.7293	1.0000	0.8814
static conflict	126.gcc	0.0539	0.0527	0.1671	0.0617	0.1658	0.0630	134.perl	0.2375	0.0212	0.1564	0.1571	0.0142	0.2325
dynamic conflict	126.gcc	0.0023	0.0037	0.0060	0.0031	0.0052	0.0043	134.perl	0.0901	0.0013	0.2263	0.2277	0.0015	0.0859
KL edge	126.gcc	0.1553	0.1536	1.3982	0.1939	0.4707	0.1516	134.perl	13.4143	0.0875	34.2514	32.8709	0.1551	14.1100
resubstitution	126.gcc	0.8911	0.8911	0.8892	0.8892	0.8759	0.8759	134.perl	0.9168	0.9168	0.9370	0.9370	0.9126	0.9126
cross-validated	126.gcc	0.8888	0.8874	0.8832	0.8860	0.8708	0.8716	134.perl	0.8266	0.9155	0.7107	0.7093	0.9112	0.8267
static cover	129.compress	0.9465	0.9651	1.0000	1.0000	1.0000	0.9807	147.vortex	0.9981	0.9978	0.9972	0.9997	0.9971	0.9998
dynamic cover	129.compress	0.7824	0.9990	1.0000	1.0000	1.0000	0.9913	147.vortex	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
static conflict	129.compress	0.0465	0.0395	0.0246	0.0147	0.0241	0.0217	147.vortex	0.0068	0.0120	0.0079	0.0076	0.0129	0.0076
dynamic conflict	129.compress	0.2142	0.1376	0.0539	0.0121	0.0511	0.0383	147.vortex	0.0002	0.0007	0.0006	0.0004	0.0006	0.0004
KL edge	129.compress	24.0538	1.5464	6.1607	2.1490	0.6366	2.5058	147.vortex	0.2340	0.7902	0.1591	0.1149	0.3289	0.0726
resubstitution	129.compress	0.8681	0.8681	0.9412	0.9412	0.8484	0.8484	147.vortex	0.9857	0.9857	0.9802	0.9802	0.9782	0.9782
cross-validated	129.compress	0.6539	0.7305	0.8873	0.9292	0.7972	0.8100	147.vortex	0.9855	0.9850	0.9796	0.9798	0.9776	0.9778

Table 6: Metrics comparing pairs of profile inputs

cases where prediction accuracy has dropped by more than 2% from resubstitution to cross-validation. This is also unsurprising, as dynamic conflict is almost the definition of cross-validated misprediction rate.[4] Using high dynamic conflict as an indicator that profiles differ misses a few cases: 3 cases on *124.m88ksim*, 3 cases on *129.compress*, and 4 cases on *130.li*. Tightening the threshold to 2% would catch all of these cases without generating any false positives.

Our metrics based on relative entropy do a reasonable but imperfect job identifying cases where prediction accuracy drops from resubstitution to cross-validation. For most of the bad prediction accuracy cases, the metrics attain values above 1. But they miss some: *129.compress* ref-train, two cases on *130.li*, and there are false positives: *124.m88ksim* ref-train, *126.gcc* ref-test, *129.compress* train-test, and *132.ijpeg* ref-train and ref-test. Using a different threshold than 1 trades off false positives for false negatives. These results are encouraging because they suggest that our metrics are indeed useful for measuring profile similarity; they would be more encouraging if we could beat dynamic conflict as well. But as we see in the next section, our metrics are beneficial when synthesizing a hybrid profile from a pair of original profiles.

4. Most of the numbers appear as if the dynamic coverage is the difference between resubstitution and cross-validation prediction accuracy. This would only be the case if branches that reversed direction always fell through or always jumped. We can infer that most of the execution frequency of branches that reversed direction came from strongly biased branch instructions.

4.2 Combining profiles using relative entropy

benchmark	combiner	ref	test	train
099.go	resubstitution	0.7709	0.7723	0.7555
	unscaled	0.7683	0.7693	0.7503
	scaled	0.7683	0.7683	0.7504
	polling	0.7653	0.7671	0.7503
	KL	0.7669	0.7659	0.7504
124.m88ksim	resubstitution	0.9209	0.9291	0.9373
	unscaled	0.9204	0.9288	0.9192
	scaled	0.9206	0.9204	0.9190
	polling	0.8955	0.9259	0.8787
	KL	0.9200	0.9288	0.9192
126.gcc	resubstitution	0.8911	0.8892	0.8759
	unscaled	0.8886	0.8862	0.8717
	scaled	0.8886	0.8866	0.8720
	polling	0.8876	0.8831	0.8700
	KL	0.8889	0.8867	0.8720
129.compress	resubstitution	0.8681	0.9412	0.8484
	unscaled	0.7305	0.8873	0.7972
	scaled	0.7300	0.9118	0.8013
	polling	0.6496	0.9176	0.8003
	KL	0.7305	0.8873	0.8253
130.li	resubstitution	0.8512	0.8751	0.8800
	unscaled	0.8016	0.7971	0.8403
	scaled	0.8015	0.8282	0.8618
	polling	0.7975	0.7864	0.8252
	KL	0.8136	0.8486	0.8732
132.ijpeg	resubstitution	0.8909	0.8902	0.8954
	unscaled	0.8908	0.8890	0.8949
	scaled	0.8908	0.8894	0.8914
	polling	0.8909	0.8806	0.8905
	KL	0.8908	0.8893	0.8950
134.perl	resubstitution	0.9168	0.9370	0.9126
	unscaled	0.9147	0.7107	0.9112
	scaled	0.9126	0.7052	0.9084
	polling	0.8497	0.7093	0.8485
	KL	0.9058	0.7107	0.9112
147.vortex	resubstitution	0.9857	0.9802	0.9782
	unscaled	0.9855	0.9796	0.9776
	scaled	0.9851	0.9800	0.9778
	polling	0.9851	0.9798	0.9777
	KL	0.9850	0.9798	0.9778

Table 7: Prediction accuracies for combining methods for all benchmarks

Table 7 shows prediction accuracies for a variety of combining strategies for each benchmark. Each column lists the actual (testing) profile; in that column, the other two profiles were combined to build the training profile. As in Table 6, the “resubstitution” row shows the upper bound prediction accuracy from training and testing on the same benchmark. The rows labeled “unscaled,” “scaled,” and “polling” correspond to Fisher and Freudenberger’s unscaled, scaled, and polling methods, respectively. The row labeled, “KL” gives the prediction accuracy from building a hybrid table using the parameterized distribution P_λ then minimizing the relative entropies from P_λ to the two training data sets. For each benchmark and each column, we have shaded the best (largest) non-resubstitution prediction accuracy entries.

Two profiles are usually better than one: the best blending method often does better than the best cross-validated single-profile prediction accuracy (the exceptions are *099.go* test-ref and ref-train, *129.compress* train-test, and *134.perl* train-ref). This matches our intuitions that blending multiple profiles ought to give a better description of program performance than using a single profile will.

In the contest of combining methods, the unscaled method is best 9 times out of 24; the scaled method is best 7 times; the polling method is best twice (matching Fisher and Freudenberger’s observations), and our “KL” method is best 15 times (counting ties for all methods that tie). The “KL” method is also very close when it is not the absolute best; the only bad cases are *129.compress* test and *134.perl* ref.

A slightly more quantitative metric is the difference between each combining method and the best combining method for each benchmark. The average of these absolute differences works out to 1.97% for the unscaled method, 1.23% for the scaled method, 5.70% for the polling method, and 0.57% for our “KL” method. For these benchmarks and inputs, our relative-entropy-based combining method turns out well; it seem worthwhile to look at other performance metrics, benchmarks, and data sets.

Unfortunately, the SPEC rules do not currently allow blending the “train” and “test” input data sets to use on the “ref” input data set. However, for the cases where there are multiple program runs within an input data set, our combining method may do a better job of combining the profiles from those multiple runs.

5 Discussion and Conclusion

Kistler and Franz [8] treated each profile as a vector in the space $\boldsymbol{R}^n$, where n is the number of items in the profile (nodes, edges, or paths). They then built a metric with values in the range [0,1] based on the geometric angle and the vector distance between the two vectors. Their metric is symmetric: the distance from profile *A* to profile *B* is the same as the distance from profile *B* to profile *A*. This does not reflect the way that profiles can be useful: profile *A* may be a good training set for profile *B* without the reverse being true. Kistler and Franz' goal was to determine when program behavior had changed significantly enough to warrant reoptimization; their metric may be very good at this but they unfortunately do not include any experimental examples.

One obvious problem with our metric and our combining method is their heavy reliance on floating point computations. It is future work to find fast, on-line ways to compute them using simpler operators and/or integers. Another obvious practical problem is how to combine more than two profiles at the same time.

An intriguing work by Wang and Rubin suggests that it is possible to combine too many profiles and over generalize beyond the point that is useful to a particular user [10]. In other words, combining a few profiles turned out to be beneficial, but after a certain point, additional profiles led to decreasing and eventually negative marginal benefits from an individual user's perspective. It would be interesting to see if our metrics can shed light on the nature of this problem.

Point profiles make an independence assumption in summarizing frequencies of profiled events without regard to the correlations between the events. Our relative entropy metric, D_{edge}, makes a similar simplifying assumption in stating that the events used to make up the profiles are independent and identically distributed. But events in the program trace are *not* independent or identically distributed: seeing one node or edge strongly constrains the next node or edge that will appear. Path profiles attempt to capture the correlations between some limited number of events in the program; relative entropy similarly generalizes into a Markov-chain based metric called divergence; the size of the Markov chain is similar to the profiling depth in a general path profile [11]. We have not yet applied divergence to path profiles, but we expect to be able to compare and combine them in a similar manner to what we have shown here.

We introduced a new approach to analyzing, comparing, and combining profiles that uses information theoretic ideas. Our comparison metric appears useful but would be more interesting if we could test it in a real system. Our combining method beats all previous approaches to combining profiles from different runs of our programs. There are many concepts in information theory that directly relate to problems in feedback-directed optimization; we expect to find better ways to analyze and explain the behavior of such systems and better and new techniques through the fusion of the two fields.

6 Acknowledgements

We thank the anonymous reviewers and Michael D. Smith for useful suggestions about how to make the paper more readable.

7 References

[1] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S. Leung, R. Sites, M. Vandevoorde, C. Waldspurger, and W. Weihl. "Continuous Profiling: Where Have All the Cycles Gone?" ACM Transactions on Computing Systems, 15(4):357-390, Nov. 1997.

[2] V. Bala, E. Duesterwald, and S. Banerjia. "Transparent Dynamic Optimization: The Design and Implementation of Dynamo," *HP Laboratories Technical Report HPL-1999-78*, June 1999.

[3] B. Calder, D. Grunwald, and A. Srivastava. "The Predictability of Branches in Libraries," *Digital WRL Technical Report 95/6*, Oct. 1995.

[4] T. Cover and J. Thomas, *Elements of Information Theory*, John Wiley & Sons, New York (1991).

[5] E. Federovsky, M. Feder, and S. Weiss. "Branch Prediction Based on Universal Data Compression Algorithms," *Proc. Twenty-Fifth International Symposium on Computer Architecture*. New York: ACM, June-July 1998.

[6] J. Fisher and S. Freudenberger. "Predicting Conditional Branch Directions From Previous Runs of a Program," *Proc. Fifth Intl. Conf. on Architectural Support for Prog. Lang. and Operating Systems*. New York: ACM, Oct. 1992.

[7] R. Hookway and M. Herdeg. "Digital FX!32: Combining Emulation and Binary Translation," *Digital Technical Journal* 9(1):3-12, 1997.

[8] T. Kistler and M. Franz. "Computing the Similarity of Profiling Data," *Workshop on Profile and Feedback-Directed Optimization*. Oct. 1998.

[9] I. Chen, J. Coffey, and T. Mudge. "Analysis of Branch Prediction Via Data Compression," *Proc. Seventh Intl. Conf. on Architectural Support for Prog. Lang. and Operating Systems*. New York: ACM, Oct. 1996.

[10] Z. Wang and N. Rubin. "Evaluating the Importance of User-Specific Profiling," *Proc. 2nd USENIX Windows NT Symposium*. Berkeley: USENIX Association, Aug. 1998.

[11] C. Young. "Path-based compilation." Diss. Harvard University, Dec. 1997.