

# Improving SimPoint accuracy for small simulation budgets with EDCM clustering

Joshua Johnston  
Department of Computer Science  
Texas A&M University  
TAMU 3112  
College Station, TX 77843-3112  
joshua.johnston@neo.tamu.edu \*

Greg Hamerly  
Computer Science Department  
Baylor University  
Waco, TX 76798  
greg\_hamerly@baylor.edu

## Abstract

Detailed processor simulation is extremely costly on large benchmark suites, where each program may run for billions of instructions and take months of simulation time. We can obtain good approximate answers in less time using limited simulation, but deciding which regions to simulate is a difficult problem. SimPoint is one approach for choosing simulation regions, based on the  $k$ -means clustering algorithm. We propose using an alternative clustering model based on a mixture of exponential Dirichlet compound multinomial (EDCM) models. This method outperforms  $k$ -means in performance prediction accuracy when simulation budgets are limited. The EDCM mixture can cluster high-dimension frequency vector data directly, without dimension reduction, and trains quickly.

## 1 Introduction

When designing computer processors, it's important to tune them for performance, power usage, size, etc. Software simulation is a widely used method for evaluation. However, detailed performance evaluation on a simulator is extremely slow. For example, it can take many months of real CPU time to evaluate a full set of SPEC CPU benchmarks for one processor design. Worse, the number of designs to evaluate grows exponentially with the number of parameters involved, so full, detailed simulation is prohibitively slow.

The architecture community has investigated several ways of reducing simulation time, with some dramatic improvements. While it may take years of CPU time to simulate a full benchmark suite, a good approximation might be obtained in hours by limited simulation. Most of these approaches save time by simulating only some parts of a benchmark and ignoring other parts. SimPoint [25] uses  $k$ -means data clustering to identify patterns of behavior in a benchmark program's execution, and then uses detailed simulation on a sample of each behavior to obtain a representative picture of the whole program's execution.

The  $k$ -means algorithm works well for this application, but it has difficulty operating on high-dimension data, like benchmark traces represented as basic block vectors. Often, such programs will have more than 100,000 basic blocks (dimensions). The difficulties of learning are due partly to the cost of processing large amounts of data, and partly to the so-called 'curse of dimensionality,' which requires the number of examples to grow exponentially with the dimension in order to densely populate the input space.

In SimPoint, this difficulty is overcome by using random projections to reduce the data's dimensionality prior to clustering. However, it would be preferable to use a clustering algorithm which could operate on the original data. Because of this, we turn to alternative clustering algorithms that are more successful in high dimensions because they consider dimensions to be independent (the naïve assumption) and do not try to

---

\*Work done primarily while at Baylor University.

compute distances or joint probabilities. An example is the multinomial probability distribution, a standard model in text clustering. Previous work [22, 15] has used mixtures of multinomials for clustering program traces, but with little success. One reason is that the multinomial model does not do well at modeling ‘bursty’ behaviors [20], or those behaviors which are rare but occur repeatedly once they do occur (as we expect in program execution traces).

In this paper, we use the exponential Dirichlet compound multinomial (EDCM) for automatically identifying program phases, in place of  $k$ -means. The EDCM is new [13] and is a very efficient approximation of the Dirichlet compound multinomial (aka the multivariate Pólya distribution). Both are related to the multinomial distribution. The EDCM has been used in mixtures for clustering text documents, but has not been applied to clustering program executions. An EDCM mixture is efficient to train and can cluster in the original space of inputs (more than 100,000 dimensions in our experiments) without difficulty. To our knowledge, this is the first time that basic-block vector program execution traces have been clustered in their original dimension.

We show that, for limited simulation budgets, EDCM clustering provides improved prediction error rates and lower prediction error variance compared with  $k$ -means. This is extremely important because small simulation budgets are where structured program analysis is most useful, as opposed to techniques based on regular or random sampling, like SMARTS [26]. Ultimately, this means less time spent simulating for the same performance prediction accuracy. We also discuss choosing the appropriate number of clusters for the model and how to choose representatives from each cluster.

## 2 Background and related work

We begin with a discussion of the need for reducing the amount of work needed for simulating large workload benchmarks. We then discuss the data representation and learning problems for clustering benchmarks, and how SimPoint uses the  $k$ -means algorithm for clustering. We discuss prior work that is related to this work, and then discuss how these prior approaches motivate our current work.

### 2.1 Detailed architectural simulation

Computer architecture relies on simulation experiments to determine the performance of new processor designs. These simulations are of detailed *software* implementations of the processor design being considered, occurring well before the fabrication of a real chip. One type of study a designer may perform is a design space exploration, trying many variants of a processor design. There are many variables that must be fine-tuned, such as cache size, branch predictor type, etc. Thus, one general design may lead to hundreds of different variants, all of which need to be compared via simulation. Simulation produces a set of summary statistics over a program’s execution, like the cycles per instruction (CPI) rate and the branch prediction error rate.

Detailed processor simulation is slow. The SPEC CPU2000 suite of benchmarks has 36 program/input pairs, and executes a total of approximately 4.06 trillion instructions. At a rate of 400 million instructions per hour for the detailed mode of the SimpleScalar simulator [5], a detailed execution of the entire set of SPEC CPU2000 benchmarks would require over 1.15 years of CPU time. And this must be done for *each* combination of parameters in a design-space exploration, compounding the problem.

Since many programs have repetitive behaviors, it is excessive to simulate a program’s entire execution. A solution is to choose and simulate small portions of the program that will be representative of the entire execution. Of course, choosing *which* portions to use is an important problem. A simple technique is to simulate a small contiguous region of each program, such as the beginning or starting at some fixed offset from the beginning (e.g. after one billion executed instructions). However, this will only represent the behavior of the simulated interval. By taking samples from throughout the program, with the ability to “fast-forward” quickly through regions that are not simulated in detail, researchers can do considerably better. This has led to approaches such as SimPoint and SMARTS [26], among others. The amount of simulation time now depends on the number of samples and the length of each sample, rather than the size of the program itself.

## 2.2 Representation and learning

Sherwood et al. [24] identified that a key to predicting performance of a program is the code it is executing. They developed a basic block vector (BBV) representation for benchmark traces, in which each program’s execution is broken into non-overlapping time intervals. These intervals can be fixed-length (e.g. a new interval begins after 100 million executed instructions), or variable-length (e.g. a new interval begins on the boundary of every  $n$ th procedure call). Before detailed simulation, each interval is profiled using a fast simulation model to determine which basic blocks execute during that interval. A basic block is an atomic unit of program execution, having one entry and one exit. Each interval has its own BBV, which starts with all zero values. When the interval is profiled, if basic block  $d$  executes, the  $d$ th entry in the BBV is incremented by the number of instructions in that basic block. Each program typically has thousands of basic blocks (e.g. a minimum of 2,039 for `art`, an image recognition program, and a maximum of 103,308 for `gcc`, the GNU compiler collection). In a machine learning context, the number of basic blocks is the dimensionality, and each BBV is an example.

The learning problem is then to identify program behaviors, sample a subset of program intervals which are representative of these behaviors, and compute a set of weights indicating the proportion of execution each behavior represents. Simulating only the sample and using the weights to combine the results gives an estimate of the performance of the whole program. Researchers can execute a disjoint sample of intervals by either fast-forwarding (using faster, non-detailed simulation) between detailed simulation intervals, or by checkpointing (saving the state of the system) right before the detailed simulation regions.

The BBV is not the only possible method for representing a program trace. In general, any statistic that is regularly collected about an executing process can be used, and is called a frequency vector. For example, frequency vectors could count branch edges, loops, procedures, registers, opcodes, data use, or program working sets [12, 19]. So long as the chosen representation is able to capture the unique behaviors of a program, it is a viable alternative. We choose to use BBVs as they provide a fine-grained view of program behavior that is as general as other instrumentation methods [18], and are the standard representation used for SimPoint.

## 2.3 SimPoint and $k$ -means

The SimPoint project [25] proposed performing data clustering on the basic block vectors (BBVs) of a program to identify its key behaviors. After dividing a program into  $k$  clusters, each cluster contains BBVs with similar code execution patterns, regardless of when they execute (temporal information is not used in clustering). SimPoint then chooses  $k$  intervals (‘simulation points’), one from each cluster, to be simulated in detail. Associated with each simulation point is a weight which corresponds to the size of its cluster. After performance statistics have been gathered on these simulation points with detailed simulation, a weighted combination of their statistics gives an estimate of the performance of the entire program.

SimPoint employs the  $k$ -means algorithm for clustering. The advantages of this algorithm are that it is easily implemented and efficient. Various optimizations are also available because of the hard-assignment property of the algorithm (each example belongs completely to exactly one cluster). However,  $k$ -means makes several limiting assumptions, such as the restriction that each cluster has the same, spherical covariance. Also,  $k$ -means suffers from the curse of dimensionality since it uses distances for comparing examples, and, in high dimension, all examples appear to be far apart.

To combat the difficulties of high dimension, SimPoint uses random linear projections [7] prior to clustering. This technique chooses a small number of randomly-chosen vectors onto which the original BBVs are linearly projected. Previous studies have shown that just 15 dimensions are sufficient to obtain good results with SimPoint [16]. Random projection also dramatically reduces the run time of  $k$ -means, as the projected BBVs are much smaller than the original vectors by several orders of magnitude.

## 2.4 Related work

Other researchers have used various profiling techniques to categorize program behavior or stability. Early work by Denning and Schwartz identified that programs often have repetitive usage patterns for memory areas, called working sets [8]. Balasubramonian et al. [4] used hardware counters to collect statistics about performance over time, such as branch miss rates, CPI, etc. Looking at these statistics allowed them to identify changes in behavior so caches might be dynamically reconfigured. Dhodapkar and Smith [9, 10, 11] pointed out a relationship between program behaviors and instruction working sets, and that changes in the former can be correlated to changes in the latter.

Other attempts at using models other than  $k$ -means for SimPoint did not fare as well as what we present here. A mixture of multinomial models, proposed by Sanghai et al. [22], did not significantly improve on the  $k$ -means algorithm, as shown by Hamerly et al. [15]. The multinomial mixture did not fare well in high dimension, did not improve on the predicted error rates, and random projection is not as straightforward to use for multinomials as for  $k$ -means. Other experiments we have performed, which we don't report here, showed that general Gaussian mixture models (which are similar to  $k$ -means models) also do not improve performance prediction significantly over  $k$ -means.

## 2.5 Motivation for our approach

We wish to perform a SimPoint-style clustering analysis on basic block vectors and sample cluster representatives for the purposes of simulation. However, we wish to do this without the use of random projections and with a statistical model that is better at representing bursty program behavior [20]—those behaviors which are rare but occur repeatedly once they do occur.

While SimPoint operates well using  $k$ -means for clustering and random linear projections to make the problem tractable, we want to learn a clustering of the original data without having to use dimension reduction. There are three reasons for this:

- A pair of true, well-separated clusters could collapse together in the projection. If this happens, the two unrelated true clusters cannot be separated by a clustering algorithm. This problem is unlikely to occur if the projected space is sufficiently large [7, 14], but it is still a difficulty.
- If we want to analyze the original unprojected data, we must map back to the original space after clustering in the projected space. While not difficult, it requires extra computation and effort.
- Random projection adds nondeterminism to simulation prediction results, which makes it more difficult to perform repeatable research. Further, different projections may lead to different clusterings without a clear method to choose among them.

Existing approaches (e.g.  $k$ -means or multinomial mixtures) also do not account for bursty data. Many programs have this property, as they often spend time in loops over the same code regions. If a piece of code executes once within a time interval, we observe that it will be likely to execute again (perhaps many times) in that interval. Figure 1 shows how program execution data is bursty, for rare basic blocks as well as common basic blocks. Compare this figure to Figure 1 in [20], as evidence that program execution exhibits bursty behavior, like word occurrences in text documents. The EDCM is a probability distribution that is good at modeling such bursty behavior.

## 3 Methodology

In this section, we describe using the EDCM mixture model to automatically classify workload execution behaviors. The EDCM mixture clusters the same BBV data that SimPoint analyzes, but in the original dimension rather than having to rely on random linear projections to reduce the data's dimension. We discuss basic clustering questions that arise from this application, such as how we train EDCM mixtures, how we choose cluster representatives, and how one might select the number of clusters that should be used.

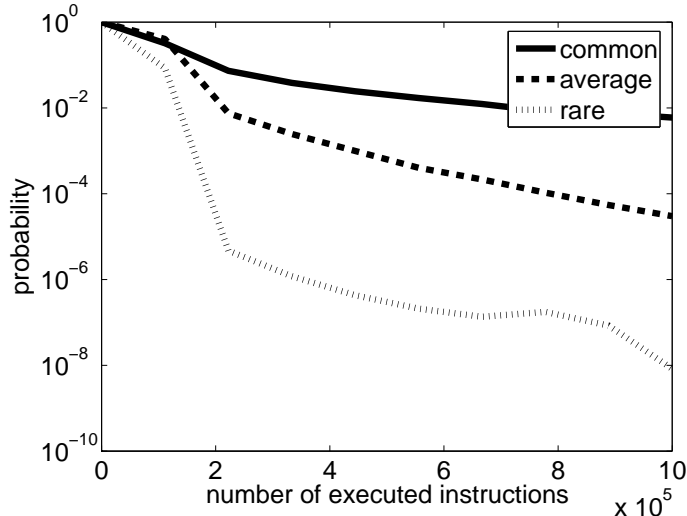


Figure 1: The probability of seeing the given number of executed instructions within one interval for common (top 5% of basic blocks, in terms of occurrence frequency), average (next 10%), and rare (remaining 85%) basic blocks. This graph shows an average over the SPEC CPU2000 suite of benchmarks. Burstiness is indicated by the probability curves which do not decrease exponentially, as would be expected with a non-bursty source, but instead tend to level off, forming a long tail.

### 3.1 The EDCM and its relatives

The EDCM is an approximation of the DCM (Dirichlet compound multinomial), which is a relative of the multinomial probability distribution. The EDCM possesses several useful properties which allow it to perform well in our domain with high dimension and bursty code execution behaviors. We begin by discussing the multinomial, DCM, and EDCM probability distributions by themselves. For what follows, we discuss single instances of each type of distribution, which is essentially one cluster in a clustering model. In the end, we will use a mixture of multiple EDCM distributions for clustering.

#### 3.1.1 Multinomial

We begin our discussion with the multinomial, as it is a more familiar distribution which is linked to the EDCM. A multinomial distribution defines a probability for a vector  $x$ , which contains the counts of occurrences of  $D$  independent events. For a fixed multinomial distribution, each feature  $x_d$  ( $1 \leq d \leq D$ ) has probability  $\theta_d$  of occurring ( $\sum_d \theta_d = 1$ ). We observe  $n$  events ( $n = \sum_d x_d$ ) and compute the joint probability of those events using

$$Pr(x|\theta) = \frac{n!}{\prod_{d=1}^D x_d!} \prod_{d=1}^D \theta_d^{x_d}. \quad (1)$$

Note that as  $x_d$  increases, the probability of  $x$  decreases (because generally  $\theta_d < 1$ ). However, in bursty phenomena, such as program execution traces, we expect that if a single event (a basic block) is executed once, then it is likely to be executed again soon, due to temporal locality. Thus, we do not want to penalize repeated events by reducing probabilities as severely as the multinomial distribution does.

#### 3.1.2 Dirichlet compound multinomial

The Dirichlet compound multinomial (DCM), also called the multivariate Pólya distribution, is a variant of the multinomial model [21]. Its construction allows it to better model bursty behavior (which we discuss

below).

The DCM can be considered a distribution over multinomial distributions. In our application, a multinomial can represent a ‘bag of basic blocks,’ where each basic block  $x_d$  has some probability  $\theta_d$  of occurring within an interval of time, without regard to order of execution within the interval. The DCM can be viewed as a ‘bag of bags of basic blocks.’ From the DCM we draw (i.e. generate) the parameters for a multinomial (a bag of basic blocks), from which we draw individual basic blocks to generate an interval/BBV. After integrating out the intermediate multinomial model, the probability of generating a vector  $x$  (e.g. a basic block vector) using the DCM with parameter vector  $\alpha$  is

$$Pr(x|\alpha) = \frac{n!}{\prod_{d=1}^D x_d!} \frac{\Gamma(s)}{\Gamma(s+n)} \prod_{d=1}^D \frac{\Gamma(x_d + \alpha_d)}{\Gamma(\alpha_d)}, \quad (2)$$

with  $n = \sum_d x_d$  and  $s = \sum_d \alpha_d$ . Note that unlike the multinomial parameters  $\theta_d$ , the DCM parameters ( $\alpha_d$ ) and EDCM parameters ( $\beta_d$ , discussed next) are not required to sum to one.

### 3.1.3 Exponential Dirichlet compound multinomial

The EDCM distribution is an approximation of the DCM that brings it into the useful family of exponential distributions. Exponential distributions are notable because their probabilities are fast to evaluate and their parameters are simple to estimate, when compared with non-exponential distributions like the DCM. Elkan [13] proposed the EDCM, noting that for text document modeling, the vector representation of a document is likely to be sparse. That is, if  $x$  represents a document with each dimension  $x_d$  representing the count of vocabulary word  $d$ , most entries in the vector will be zero. This is because most documents contain only a small subset of the entire vocabulary. This is also true in the program trace data we consider, where, on average, over 86% of entries are zero. Additionally, an approximation can be made if  $\alpha \ll 1$ , which Elkan shows is true for the majority of words in text documents, and we find to be true for the majority of basic blocks in BBV data. Elkan uses these facts to simplify and approximate the DCM, formulating the EDCM’s estimation for the probability of  $x$  as

$$Pr(x|\beta) = \frac{n!}{\prod_{d:x_d \geq 1}^D x_d} \frac{\Gamma(s)}{\Gamma(s+n)} \prod_{d:x_d \geq 1} \beta_d. \quad (3)$$

Here,  $\beta$  is used to distinguish the parameters of the EDCM from  $\alpha$ , the parameters of the DCM. Similarly to Equation 2,  $n = \sum_d x_d$  and  $s = \sum_d \beta_d$ .

### 3.1.4 Burstiness

An event is considered bursty if, when it occurs once, it is likely to occur many times within a short time [6, 17]. Madsen et al. [20] performed analysis on a corpus of text documents and defined three different groups of words—common, average, and rare. The rare words comprised 89% of the vocabulary but only 8% of all the word occurrences in the text documents. These occurrences are very important, though, because they serve as markers to help identify document content. In Figure 1, we showed a similar analysis for basic blocks that illustrate that they too have bursty behavior, due to temporal locality. For example, if a benchmark contains a tight loop, the basic blocks within the loop will occur many times within the interval that contains that loop.

## 3.2 EM learning for a mixture of EDCMs

Before this section, we used  $x_i$  to denote an element of the vector  $x$ . From here on we use  $x_i$  to denote the  $i$ th vector in a collection  $X$ , and  $x_{id}$  to denote the  $d$ th element of that vector.

To cluster basic block vectors for phase analysis, we use a mixture of several EDCM probability distributions. The mixture is trained on the basic block vectors, and in the end each EDCM represents a probability

density for one cluster, or phase. Each cluster has its own set of  $\beta$  parameters for the EDCM as well as a prior probability for the cluster (where the sum of the priors is one). The learning goal is to estimate a suitable set of parameters for the  $\beta$  vector of each cluster and the prior probabilities for the clusters.

We use the standard expectation-maximization algorithm for learning an EDCM mixture for a fixed number of clusters,  $k$ . We start with a random set of initial parameters (a randomized M-step). The initial parameters are generated by first fitting one EDCM distribution to the entire set of BBVs, giving a set of parameters  $\beta_0$  that accounts for the overall distribution of basic blocks within the entirety of a program’s execution. When considering using  $k$  EDCM distributions within a mixture model, we create  $k$  copies of  $\beta_0$ :  $\beta_1 \dots \beta_k$ , that introduce slight random perturbations into the elements of parameter vectors. More formally,

$$\beta_{id} = \beta_{0d} r_{id},$$

with each  $r_{id}$  being a random value sampled uniformly from the range  $[0.95, 1.05]$ .

We define  $c_j$  as the  $j$ th cluster. Each E-step calculates the membership probability  $P(c_j|x_i)$  for each cluster  $c_j$  and example  $x_i$ , according to Bayes’ rule. Each M-step chooses the cluster priors and  $\beta$  values that maximize the likelihood of the model based on the current expected memberships. The interested reader is referred to [13] for details of the algorithm.

The training proceeds with alternating E- and M-steps until the models converge. We test for convergence by looking at the values of the parameter vectors  $\beta_i$  ( $1 \leq i \leq k$ ) from one iteration to the next. When the total percent change across all  $\beta_i$  falls below a certain threshold  $\epsilon$  (a simple parameter, set to  $10^{-7}$  in our case), we say the model has converged to a stable set of parameters and cease iteration. Other common methods include looking at a model’s log-likelihood and terminating iteration when it shows little change across iterations. Examining the parameters  $\beta$  for convergence is analogous to stopping  $k$ -means iteration when the cluster means have stabilized.

Since the EM algorithm can get stuck at local optima which are far from the global optimum, it is customary to use several randomized restarts of the algorithm. Each restart uses a different random initialization, and is allowed to run to convergence. Among the different models learned, we choose the one having the best likelihood.

### 3.3 Choosing simulation points

After performing clustering, we must select a representative (simulation point) from each cluster to simulate in detail. We examine several different methods of choosing simulation points from our final clusterings. Let the notation  $s_j$  denote the interval chosen as the simulation point for cluster  $c_j$ .

**Mode (highest probability)** Because the EDCM is a probability distribution, each example  $x_i$  is assigned a probability  $Pr(x_i|c_j)$  by cluster  $c_j$ . Choose a representative for cluster  $c_j$  by finding the example with the highest probability in that cluster:  $s_j = \arg \max_{x_i \in c_j} Pr(x_i|\beta_j)$ .

**Smallest Euclidean ( $L_2$ ) distance** Compute the centroid,  $\bar{x}_j$ , of each cluster  $c_j$  as the mean of the examples within it:  $\bar{x}_j = \frac{1}{|c_j|} \sum_{x_i \in c_j} x_i$ . Choose the example that has the smallest Euclidean distance to the centroid:  $s_j = \arg \min_{x_i \in c_j} \sqrt{\sum_{d=1}^D (x_{id} - \bar{x}_{jd})^2}$ .

**Smallest Manhattan ( $L_1$ ) distance** As in the  $L_2$  distance, compute the centroid  $\bar{x}_j$  for cluster  $c_j$ . Choose the example that has the smallest Manhattan distance to the centroid:  $s_j = \arg \min_{x_i \in c_j} \sum_{d=1}^D |x_{id} - \bar{x}_{jd}|$ .

### 3.4 Choosing the number of clusters

Before training an EDCM mixture model, we must specify a value for the number of clusters,  $k$ . We do not have *a priori* knowledge of the  $k$  that is appropriate. Intuitively, we want a  $k$  that is small, so the clustering is a concise description of program phase behavior, and because  $k$  is directly related to the amount of simulation

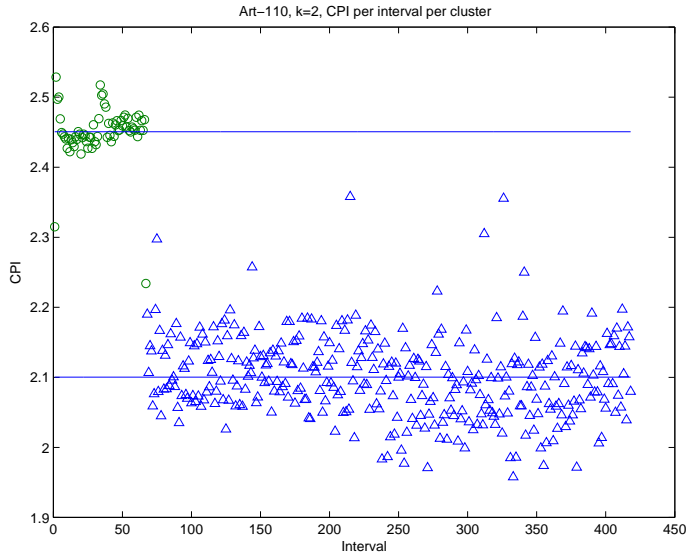


Figure 2: A  $k = 2$  clustering by an EDCM mixture of the benchmark `art-110`. The horizontal axis indicates execution time, the vertical axis corresponds to CPI, and each dot represents a 100-million instruction interval. Each dot is labeled with one of the two clusters (indicated by shape), and the average CPI is plotted for the two clusters on horizontal lines.

we must perform (since each cluster has a simulated representative). However, we also want a sufficiently large  $k$  to capture the unique patterns of behavior within a program. We tried the Bayesian information criterion (BIC) [23] and Akaike information criterion (AIC) [2] to select a value for  $k$ . Both the BIC and the AIC penalize the log-likelihood of a model (given observations) with a penalty that is proportional to the number of estimated parameters.

The BIC (aka the Schwartz criterion), is  $\mathcal{L} - p \log(n)/2$ , where  $\mathcal{L}$  is the log likelihood,  $p$  is the number of estimated parameters, and  $n$  is the number of examples. Since the BIC subtracts a term from  $\mathcal{L}$ , we seek the model which maximizes the BIC. The number of free parameters for an EDCM mixture is  $p = k(1 + D) - 1$ :  $k$  priors (which are constrained to sum to one, leaving  $k - 1$  free parameters) and  $k$  parameter vectors  $\beta$  of  $D$  dimensions each. SimPoint’s choice of  $k$  is based on the BIC. Since  $k$ -means in low dimension is a simpler model than the EDCM mixture in high dimension ( $k$ -means has fewer free parameters), the  $k$ -means complexity penalty tends to be much lower than the complexity penalty for the EDCM in this application.

The AIC does not penalize the complexity of the model (number of parameters  $p$ ) as much as the BIC does,  $AIC = -2\mathcal{L} + 2p$ , with  $p$  calculated as above for the EDCM mixture. Note that with the AIC, the likelihood term and penalty have different signs than in the BIC. Therefore, a lower AIC is better, and we choose the model with the lowest AIC.

We emphasize that in this paper, our purpose is primarily to demonstrate that EDCM mixture models are preferable to  $k$ -means for small simulation budgets (i.e. small  $k$ ). We have found that the AIC performs better than the BIC for choosing  $k$  for an EDCM mixture in this application. However, as our focus is directly on prediction performance for a fixed, small value of  $k$ , we leave to future work the issue of exploring AIC versus BIC for the EDCM.

## 4 Experimental evaluation

We use EDCM mixtures to cluster the SPEC CPU2000 benchmark suite. Each benchmark’s execution has been split into fixed-length intervals of 100 million instructions each. We rely on the work of others—

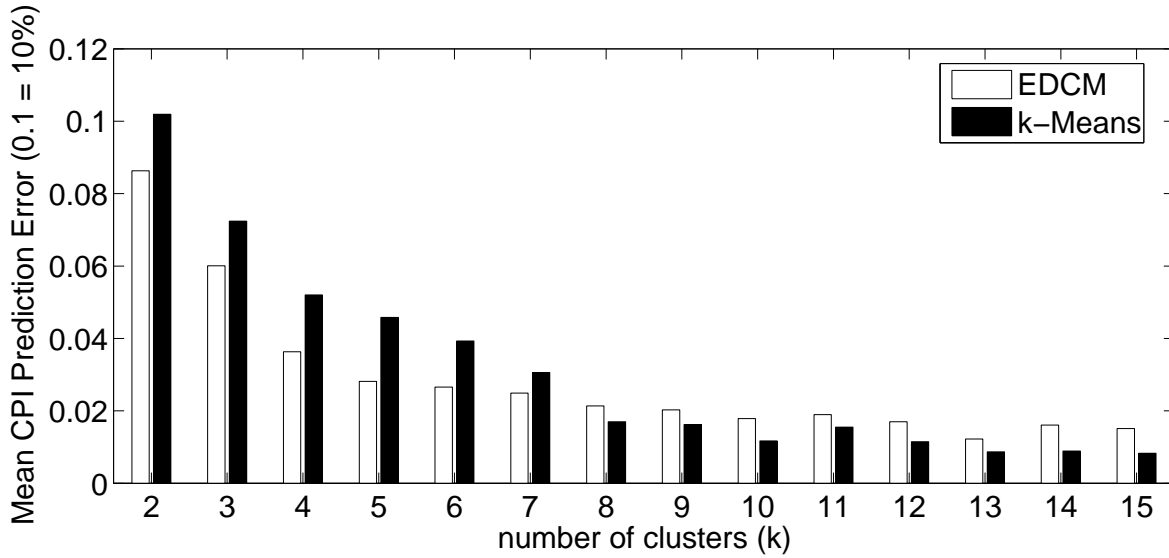


Figure 3: This plot shows the *mean* CPI prediction error of EDCM and  $k$ -means clustering for  $k=2$  to 15 clusters. Each bar represents the average error across all 36 SPEC CPU2000 benchmarks. It's clear that for small simulation budgets (i.e. small  $k$ ), EDCM clustering gives better prediction performance (lower prediction error).

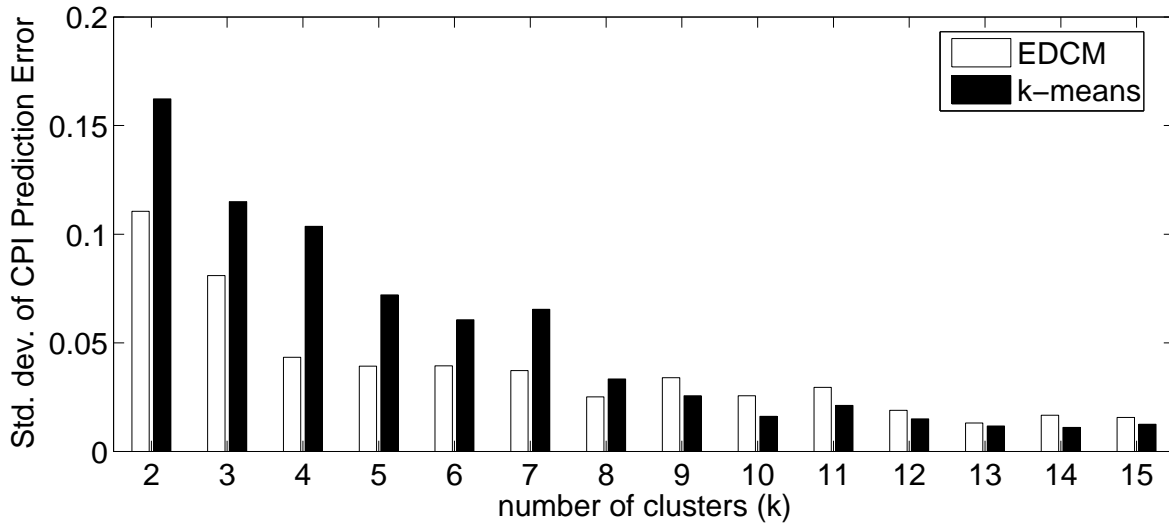


Figure 4: This plot shows the *standard deviation* of CPI prediction error of EDCM and  $k$ -means clustering for  $k=2$  to 15 clusters. Each bar represents the standard deviation of the CPI prediction error across all 36 SPEC CPU2000 benchmarks. We can see that, again, for small  $k$ , EDCM gives a lower standard deviation of CPI prediction error, meaning that it is more consistent at accurate predictions.

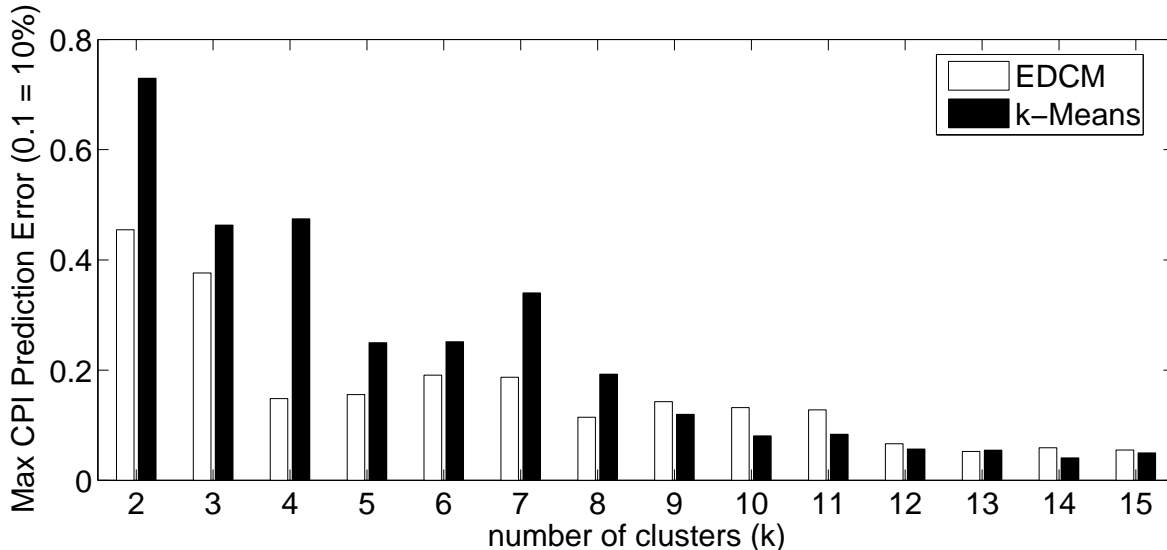


Figure 5: This plot shows the *maximum* CPI prediction error of EDCM and  $k$ -means clustering for  $k=2$  to 15 clusters. Each bar represents the maximum CPI prediction error across all 36 SPEC CPU2000 benchmarks. For small  $k$ , EDCM gives lower maximum errors, making its results more useful than  $k$ -means for evaluating performance across all benchmarks.

each of these benchmarks has been simulated in its entirety [25] on the SimpleScalar simulator [5]—for the true metrics of performance on a per-interval basis. The simulated CPU uses the Alpha Instruction Set Architecture, and each binary is compiled with full optimizations. For full details on the simulated CPU, see [16]. To evaluate the accuracy of our approach, we compare our method’s predicted performance in terms of weighted average CPI (cycles per instruction) with the true average CPI as obtained from full simulation. We compare the prediction performance of the EDCM, implemented in MATLAB, with the current version of  $k$ -means implemented in SimPoint. We choose simulation points for EDCM using the  $L_1$  distance to cluster means. SimPoint selects simulation points using Euclidean ( $L_2$ ) distance to the cluster means.

The underlying assumption behind SimPoint is that basic block vectors with similar behavior will accordingly exhibit similar performance metrics (e.g. CPI). An important point is that we do not use the CPI metrics for clustering; we only use the basic block vectors. As an illustration, Figure 2 shows the `art-110` benchmark that was clustered with an EDCM mixture, using  $k = 2$  clusters. The graph shows the value of the CPI for each interval in the program trace, grouped by cluster. Note that the CPI values group tightly within each cluster.

#### 4.1 Prediction accuracy and model complexity

Our first question is whether or not EDCM mixture models find clusterings that are suitable for predicting simulation metrics. We compare EDCM mixtures to SimPoint (using  $k$ -means) for different fixed simulation budgets. We run each method with a number of different numbers of clusters (values of  $k$ ), and each value of  $k$  represents a simulation budget for which we compare performance prediction of the two methods.

A primary research concern for processor design is with the amount of time spent in detailed simulation, which is directly proportional to the number of clusters (and therefore number of selected simulation points) a program’s execution is broken into, and the size of each simulation point. The motivation behind using clustering in SimPoint is to reduce the amount of simulation time while retaining high prediction accuracy. Many researchers that use simulation will want to fix their simulation budget to spend as little time as possible performing simulations. This motivates not only the general SimPoint approach, but also leads the

practitioner to prefer fewer simulation points. Therefore, we examine the results of clustering and prediction using EDCM mixture models and  $k$ -means for values of  $k$  in the range of 2 to 15. For each value of  $k$ , we cluster all 36 program/input pairs in the SPEC CPU2000 suite of benchmarks, generate  $k$  simulation points, and calculate the predicted average CPI of the program/input pair using these selected intervals of program execution. We compare the true average CPI with the average CPI predicted by both EDCM mixture models and  $k$ -means. This gives a percent prediction error which tells us, for a specific value of  $k$ , the accuracy of each method at both clustering and choosing simulation points that represent overall program behavior.

We make several points of comparison between the prediction errors of EDCM mixture models and  $k$ -means. The first comparison is the average amount of prediction error made per value of  $k$ , where the average is taken across the 36 different program/input pairs in the benchmark suite. Figure 3 shows that for smaller values of  $k$ , the EDCM provides better accuracy at predicting average CPI values. Above a certain value of  $k$  (about  $k \geq 8$ ), the two methods both provide less than 2% prediction error, on average.

Figure 4 shows the standard deviation of the predicted CPI values across all program/input pairs per value of  $k$ . Again, we see that for smaller values of  $k$ , EDCM mixture models provide predictions that have smaller standard deviations than  $k$ -means. This is significant because it means that the EDCM is doing a more consistent job at making accurate predictions.

Figure 5 compares the maximum prediction error per value of  $k$  for both clustering methods. This graph shows that EDCM mixture models, when they do perform poorly (on particularly difficult program/input pairs where small values of  $k$  may not be enough to capture the range of program behaviors), still outperform  $k$ -means clustering. In fact, we see that the EDCM’s maximum CPI prediction error also drops much more quickly as  $k$  increases than does the same measure for  $k$ -means. Once both methods have enough clusters ( $k$  is large enough) to capture subtle differences in program behavior, the maximum error for both is consistently low.

Thus we see that the EDCM mixture is consistently better at clustering program phase behavior and predicting CPI performance for small simulation budgets (small values of  $k$ ). For large simulation budgets, most methods of selecting simulation regions will work well, because a larger sample affords more opportunities for the sample to represent the program as a whole. However, when simulation budgets are limited and small, we still want to obtain the most accurate predictions possible, and this is where the EDCM mixture is the clear winner.

There are several reasons to expect that the EDCM mixture is a better model for program behavior for small samples.

- The SimPoint method with  $k$ -means operates on a reduced-dimension version of the original frequency vector data. Dimension reduction can cause some clusters which were naturally far apart to overlap when projected. While this is unlikely to happen, the EDCM can guarantee this won’t happen, since it operates on the original data.
- The EDCM mixture incorporates a prior probability for each cluster, meaning that some clusters can be larger and others smaller (with respect to the percentage of the number of frequency vectors in each cluster). The  $k$ -means algorithm uses uniform prior probabilities (which are implicit in the algorithm), meaning that  $k$ -means clusters tend to have the same size. This difference allows EDCM to be more flexible than  $k$ -means at fitting the naturally occurring clusters when those clusters have varying sizes.

Figure 6 shows how the cluster sizes vary for both clustering methods. We report this as the standard deviation of cluster sizes averaged over all benchmarks for  $k$  from 2 to 15. Cluster sizes are measured as percentages of the total execution, so if three clusters contain 500, 300, and 200 frequency vectors, then their sizes are reported as 0.5, 0.3, and 0.2, respectively. We can see that EDCM has consistently higher standard deviations, indicating that it is finding clusters whose sizes vary more than the clusters found by  $k$ -means, whose clusters tend to be more uniformly sized.

- The basic model of a cluster in the  $k$ -means algorithm is a spherical Gaussian, where the variance of the Gaussian is the same across all clusters (these attributes are also implicit in the algorithm). While the spherical Gaussian is a reasonable model that is widely used for clustering, it is not as flexible or appropriate as the EDCM which explicitly models the counts in frequency vectors.

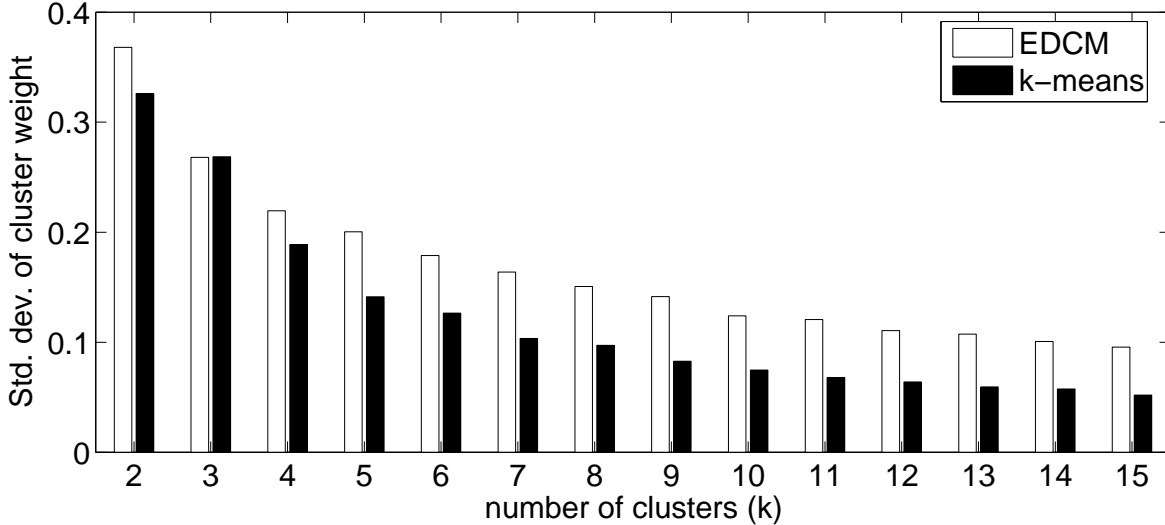


Figure 6: This plot shows the standard deviation of the size of each cluster for EDCM and  $k$ -means. Higher bars indicate more variation in cluster size, while lower bars indicate more uniform cluster sizes. From this we can see that an EDCM mixture, which explicitly incorporates a prior probability for each cluster, allows it to recognize clusters of more varying sizes. Each bar represents the standard deviation over all benchmarks for each  $k$ . Each raw cluster size is measured as the ratio of the number of frequency vectors in that cluster to the total number of frequency vectors in a benchmark.

## 4.2 Choosing representatives

Once we have a clustering of basic block vectors (i.e. program execution intervals) with an EDCM mixture, we need to then select a single interval as the representative, or simulation point, for that cluster. SimPoint uses the interval whose basic block vector is closest (in Euclidean, or  $L_2$  distance) to the  $k$ -means center, which is the geometric mean of the vectors in the cluster.

We investigate several different methods for choosing cluster representatives from the clusterings induced by the EDCM mixture model. Using the methods outlined in Section 3.3, we found that using the  $L_1$  (Manhattan) distance to choose simulation points resulted in prediction with the lowest percent error.

It's not surprising that  $L_1$  distance outperforms  $L_2$  distance. BBVs have a large number of dimensions, and distance metrics tend to report large pairwise distances between any randomly sampled pair of points in high dimension spaces. This makes it more difficult for a distance metric to reliably discern between noisy points which are truly far apart and those which just appear to be far apart because of noise. Others have shown that in high dimension, for the distance metric  $L_p(a, b) = \sum_{d=1}^D (|a_d - b_d|^p)^{1/p}$ , lower values for  $p$  (such as the Manhattan ( $L_1$ ) distance) give more meaningful results than higher  $p$  (e.g. the Euclidean ( $L_2$ ) distance) [1].

We had initially suspected that using probabilities for selecting simulation points would yield good results, but it did not. We discovered that certain basic blocks exhibit very bursty behavior and have very high frequency in the few intervals they appear in, such as a single basic block occurring up to  $10^7$  times in one interval. The EDCM does not penalize bursty behavior; instead, it gives higher probabilities to such vectors. Compare this characteristic to the multinomial, which rewards intervals that have a more uniform distribution across the basic blocks. So while it is good that the EDCM can represent bursty data, it causes problems when asking for the most probable vector from the distribution (i.e. the vector closest to a mode).

### 4.3 Time and space complexity

The time and space complexity of the EM algorithm for training an EDCM mixture is linear in the number of clusters, the number of basic block vectors, and the dimension of the data. All these are comparable to the linearity of  $k$ -means for the same attributes. Both algorithms will also depend on the number of required iterations (but a reasonable average-case bound on this is an open question in machine learning; see for instance [3]).

A direct comparison of the running times of EDCM and  $k$ -means on this application should consider the amount of raw data that each is processing. Since it does not reduce dimension, the EDCM mixture is operating on approximately 1,500 times more data than does  $k$ -means with its projected datasets. This figure is adjusting for the fact that some basic blocks never execute and are pruned by our EDCM implementation.

The EDCM mixture converges quickly with EM training. In our experiments, running the EDCM to cluster the 36 benchmarks in the SPEC CPU2000 suite, with ten random restarts per  $k$  ( $2 \leq k \leq 15$ ) takes about 104 hours of CPU time on a dual-processor, hyper-threaded 2 GHz Intel Pentium 4 computer with 2 GiB of RAM (without parallelized code). Running  $k$ -means (also not parallelized) on all 36 benchmarks for each  $k$  from 2 to 15 and 10 random restarts, takes 812 seconds of CPU time.

While this seems like a large difference in running times, there are several reasons for the gap. First, SimPoint is highly optimized for the application at hand, and is written in C++, while our EDCM mixture implementation is written in MATLAB and is currently not optimized. We plan to provide an optimized version in future work. Second, as we mentioned, EDCM is actually processing far more data than is  $k$ -means. When viewed in this light, the EDCM mixture is actually quite fast. Finally, we want to emphasize that architecture researchers are quite willing to spend a fair amount of time clustering benchmarks prior to simulation, since the simulations themselves will take a much larger amount of time in a large design space exploration over many processor configurations.

## 5 Conclusions and future work

The purpose of SimPoint is to cluster program traces in order to choose representative intervals (simulation points). Detailed simulations are expensive, so researchers need methods that reduce the amount of a benchmark they have to simulate in order to get accurate ideas of overall performance statistics. SimPoint does a good job of clustering benchmark data with the  $k$ -means algorithm, and choosing simulation points. These simulation points are run through new hardware designs simulated in software, offering accurate predictions about performance without requiring the entire benchmark to be fully simulated.

We proposed using EDCM mixture models to cluster the same data as  $k$ -means, and we found that EDCM mixtures provide very accurate prediction performance. For fixed simulation budgets, a researcher will want to spend as little time as possible in simulation, translating into a desire for fewer simulation points and smaller values of  $k$ . For small values of  $k$  ( $2 \leq k \leq 8$ , which equates to 200 to 800 million detailed instructions simulated), we find that the EDCM mixture model outperforms  $k$ -means. We see that the average error, variance in the errors of the predictions, and the maximum prediction error using the EDCM are all lower on the SPEC CPU2000 suite of benchmarks than using  $k$ -means. For larger values of  $k$  ( $k > 10$ ), the two methods both predict performance with around 1% error.

The EDCM mixture offers more than empirical reasons why it is a favorable clustering model. First, it is able to use the entire dimension of the data. In practice,  $k$ -means requires dimension reduction before it can begin clustering, which may collapse true clusters of behavior, causing information about the data to be lost to  $k$ -means. Second, the EDCM mixture is able to handle bursty data well. We have seen that the BBV data is indeed bursty. Third, the EDCM mixture is a probability density function that assigns probabilities to each example, giving nice statistical interpretations to our data that  $k$ -means does not offer.

Future work includes implementing an optimized version of the EDCM mixture algorithm implemented in the SimPoint software, so that others may use it. We also want to further analyze the phenomenon of burstiness in program traces, examine the differences between the clusterings generated by SimPoint and the EDCM mixture, and leverage the probabilities assigned to each example by the EDCM mixture.

## References

- [1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *Proc. Eighth International Conference of Database Theory*, pages 420–434, London, 2001.
- [2] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *International Symposium on Information Theory*, pages 267–281, 1973.
- [3] D. Arthur and S. Vassilvitskii. How slow is the k-means method? In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153, New York, NY, USA, 2006. ACM.
- [4] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *33rd International Symposium on Microarchitecture*, pages 245–257, 2000.
- [5] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report CS-TR-97-1342, U. of Wisconsin, Madison, June 1997.
- [6] K. W. Church and W. A. Gale. Poisson mixtures. *Natural Language Engineering*, (1):163–190, 1995.
- [7] S. Dasgupta and A. Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. Technical Report TR-99-006, Berkeley, CA, 1999.
- [8] P. Denning and S. C. Schwartz. Properties of the working-set model. *Communications of the ACM*, 15(3):191–198, Mar. 1972.
- [9] A. Dhodapkar and J. Smith. Comparing program phase detection techniques. In *36th International Symposium on Microarchitecture*, pages 217–227, Dec. 2003.
- [10] A. Dhodapkar and J. E. Smith. Dynamic microarchitecture adaptation via co-designed virtual machines. In *International Solid State Circuits Conference*, pages 198–199, Feb. 2002.
- [11] A. Dhodapkar and J. E. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *29th Annual International Symposium on Computer Architecture*, pages 233–244, May 2002.
- [12] L. Eeckhout, J. Sampson, and B. Calder. Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation. In *Proceedings of the 2005 IEEE International Symposium on Workload Characterization*, pages 2–12, Austin, TX, USA, 10 2005. IEEE.
- [13] C. Elkan. Clustering documents with an exponential-family approximation of the Dirichlet compound multinomial distribution. In *International Conference on Machine Learning*, pages 545–552, 2006.
- [14] Y. Feng and G. Hamerly. PG-means: learning the number of clusters in data. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 393–400, Cambridge, MA, 2007. MIT Press.
- [15] G. Hamerly, E. Perelman, and B. Calder. Comparing multinomial and k-means clustering for SimPoint. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 131–142, 2006.
- [16] G. Hamerly, E. Perelman, J. Lau, B. Calder, and T. Sherwood. Using machine learning to guide architecture simulation. *Journal of Machine Learning Research*, 7:343–378, February 2006.
- [17] S. M. Katz. Distribution of content words and phrases in text and language modelling. *Natural Language Engineering*, (2):15–59, 1996.
- [18] J. Lau, J. Sampson, E. Perelman, G. Hamerly, and B. Calder. The strong correlation between code signatures and performance. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 236–247, March 2005.
- [19] J. Lau, S. Schoenmackers, and B. Calder. Transition phase classification and prediction. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, pages 278–289, Jan. 2005.
- [20] R. Madsen, D. Kauchak, and C. Elkan. Modeling word burstiness using the Dirichlet distribution. In *International Conference on Machine Learning*, pages 289–296, 2005.
- [21] T. Minka. Estimating a Dirichlet distribution. <http://research.microsoft.com/~minka/papers/dirichlet/>, 2003.
- [22] K. Sanghai, T. Su, J. G. Dy, and D. R. Kaeli. A multinomial clustering model for fast simulation of computer architecture designs. In *Knowledge Discovery and Data Mining (KDD)*, pages 808–813, 2005.
- [23] G. Schwartz. Estimating the dimension of a model. *Annals of Statistics*, (6):461–64, 1979.
- [24] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.
- [25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, Oct. 2002.

- [26] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA)*, pages 84–95, June 2003.