Analyzing and Improving Clustering Based Sampling for Microprocessor Simulation

Yue Luo, Ajay Joshi, Aashish Phansalkar, Lizy John, and Joydeep Ghosh
{luo, ajoshi, aashish, ljohn, ghosh}@ece.utexas.edu
Department of Electrical and Computer Engineering
University of Texas at Austin

Abstract

The time required to simulate a complete benchmark program using the cycle-accurate model of a microprocessor can be prohibitively high. One of the proposed methodologies, representative sampling, addresses this problem by simulating only a group of unique phases in a program called simulation points. The methodology selects simulation points by characterizing each fixed chunk of instructions in the program using a feature called Basic Block Vector (BBV), clusters them into groups of similar chunks of instructions, and then selects a representative chunk of instructions from each cluster. The accuracy of this technique is highly dependent on the choice of the feature, clustering technique, and the distance measurement used for clustering. Previous research does not completely address all these aspects.

In this paper, we propose a set of statistical metrics for making a comprehensive and fair evaluation of features, clustering algorithms, and distance measurements in representative sampling. These metrics give more insight into the sampling result. We use them to evaluate two different clustering algorithms, three distance measurements, and a new micro-architecture independent feature that we propose for representative sampling.

We compared a k-medoid clustering algorithm, CLARANS, with the popular k-means algorithm for different distance measurements (projected Euclidean, Euclidean, and cosine). Our results show that for the eight programs we used from the SPEC CPU2000 benchmark suite, CLARANS clustering algorithm results in better quality clusters in the feature space as compared to the k-means algorithm. CLARANS also produces phases that are more homogeneous for CPI. We propose a new micro-architecture independent data locality based feature, Reuse Distance Distribution (RDD), for finding phases in programs. We show that the RDD feature consistently results in more homogeneous phases than Basic Block Vector (BBV) for many SPEC CPU2000 benchmark programs simulated on three different processor configurations.

1. Introduction

Cycle-accurate microarchitecture simulation is one of the most important tools in computer architecture research. However, simulating standard benchmarks in a cycle-accurate simulator is often prohibitively time-consuming. For example, SPEC CPU2000 [1] is a widely used CPU-intensive benchmark suite for evaluating processor designs. Simulating one program from this suite takes days to weeks in a cycle-accurate model in *SimpleScalar*[2], the most popular simulator today among academic researchers. Quite often researchers can only run a few billion instructions, which may not be representative of the actual benchmark program and could result in large errors. Many techniques have been proposed that reduce simulation time, while retaining good accuracy. One class of such techniques takes advantage of the phase behavior in program execution.

It has been well observed that programs show phase behavior. A *phase* can be defined as a portion of dynamic execution of a program most of the performance metrics such as Cycle Per Instruction (CPI), show very little variance. As per this definition, parts of a program that are disjoint in time may belong to the same phase as long as they show similar values for performance metrics. Since the performance metrics remains stable in a phase, simulating only one chunk of instructions in the phase can give fairly accurate estimation of the performance for the entire phase. If one chunk of instructions from every phase is selectively simulated, the simulation time can be greatly reduced with little loss of simulation information in the whole program. Since the instruction chunks are carefully selected to represent the execution of the whole program, we call this type of technique *phase based representative sampling*, or *representative sampling* hereafter. Three recently proposed schemes fall into this category [3][4][5]. These techniques estimate some *target metric* (e.g. CPI, energy per instruction, or cache miss rate) by taking advantage of the phase behavior. To identify phases, they divide the dynamic instruction stream into chunks of instructions, and for each chunk, measure the feature that is distinguishable between phases. We refer to such a feature as a *phase classification feature*, or simply a *feature* as it is often referred to in the data mining community. Then, cluster analysis is performed to group the chunks into clusters. Each cluster corresponds to a phase because the chunks in the same cluster exhibit very similar phase feature. The major characteristics of the three representative sampling techniques are summarized in Table 1. Readers are referred to section 3 for a detailed description. Out of the three techniques, SimPoint [3] is the most popular. SimPoint uses Basic Block Vector (BBV) as the phase classification feature. BBV is a vector built with frequencies of dynamic execution of static basic blocks in a

code. Unlike performance counters in SPEClite [4], BBV is microarchitecture-independent. Thus, the phases identified from BBV are valid across different microarchitecture configurations. In addition, SimPoint uses only a few chunks of a relatively large size (100 million instructions), which makes it very easy to implement and does not need explicit warm-up. Because of its advantage and popularity, we base our study mainly on the approach of SimPoint[1].

Technique	Target Metric	Phase classification feature	Clustering algorithm	Chunk size (million instructions)
SimPoint	IPC	BBV	k-means	100
SPEClite	29 Performance Metrics	Performance Counter Data	k-means	1
Lafage and Seznec [5]	Data Cache Miss-Rate	Data Reuse Distance	Hierarchical	1

Table 1. Recently proposed phase based representative sampling techniques

Although representative sampling is becoming popular for microprocessor simulation, the design space has not been well explored and many questions are still unanswered. This research addresses some of the important questions. In the next section, we describe three such problems. After reviewing the related work in Section 3, we address each of these problems in subsequent sections. Our evaluation methodology is proposed in Section 4. Different clustering algorithms and distance measure are evaluated in Section 5. In Section 6, we propose our new phase feature, RDD, and compare it with BBV. In Section 7, we draw conclusions from this study.

2. Problem Statement

In this section, we describe the problems we address in this paper and specify our contributions.

2.1 Choice of clustering algorithms and distance measures

It is well known in the data mining community that no single clustering algorithm is well suited for all applications. Various algorithms have been proposed for different applications. There has been no study to search for the best clustering algorithm for representative sampling. It

[1] Variance SimPoint and Early SimPoint are extensions to the original SimPoint. These methods use many small chunks of instructions making it impractical and difficult to use. So, these methods are not as popular as the original SimPoint. In this paper we focus on the original SimPoint, but our methodology is applicable to all representative sampling techniques.

is not known whether the choice of algorithm affects the accuracy of clustering. The k-means clustering method [6] used in SimPoint is simple and fast. However, k-means clustering performs well only on clusters that are spherical and have the same variance. Phase classification features such as BBV are characteristics of benchmark programs. Some programs show very regular execution patterns whereas others show unpredictable behavior. It is unlikely that the features from benchmark simulation will satisfy the requirements of k-means algorithm. In addition, k-means algorithm is very sensitive to outliers, which may severely distort the clustering result. Hierarchical clustering, used in [5], does not have a provision for relocating data points that have been incorrectly grouped at an early stage in the clustering process. Another problem with the two clustering algorithms lies in finding a data point to represent a cluster. The representative for a cluster should correspond to a real data point. In k-means, a cluster is represented by the centroid of the cluster, which is very unlikely to coincide with a real data point. In hierarchical clustering, there is no inherent representative data point for a cluster. We therefore feel that other clustering techniques could give better quality clusters and help in selecting a better cluster representative. In this study, we choose a k-medoid method and compare it with k-means. Our reason for choosing this method and the evaluation result is detailed in Section 5.

All three proposed representative sampling techniques used Euclidean distance to measure the dissimilarity of chunks of instructions in their clustering analysis. Just like the choice of the best clustering algorithm, it is well known that no single distance measure is the best for all types of data. Other distance measures such as the Manhattan distance and cosine distance have been applied in different applications. However, no previous research has aimed at finding the best distance measure for identifying simulation points. In this paper, we study a carefully selected distance measure, the cosine distance and evaluate its effectiveness in representative sampling.

2.2 Evaluation methodology

How to fairly evaluate the effectiveness of new clustering algorithms, new distance measures, and new phase features in representative sampling is another question that has not been studied previously. Of course, the final error in target metric can be used to compare different approaches. However, as we will show in Section 4 this is not a reliable method because chance can often play a big role in the sampling error. We propose a methodology for comprehensive evaluation by including statistically sound metrics such as the normalized standard deviation, and we examine them in both the phase classification and target metric space. Our methodology,

unlike the final sampling error, can not only enable a fairer evaluation but also provide a deeper understanding of representative sampling.

2.3 Choice of phase classification feature

The third question we study is what phase classification feature to use in representative sampling. A microarchitecture independent feature, such as BBV in SimPoint, is preferable because it gives the user more confidence that the clustering result will be correct across different microarchitectures. A good feature should be strongly corrected with the target metric (e.g. CPI). For modern microprocessors, the data access latency is one of the most important factors that determine the performance. Data access latency is a function of data locality. Therefore, we focus on microarchitecture independent phase features based on data locality. In this paper, we propose a new data locality based phase classification feature, the Reuse Distance Distribution (RDD). We show that it results in more homogeneous phases than BBV for many benchmarks. More importantly, we show that for these benchmarks, this advantage holds on different microarchitecture configurations.

The contribution of the paper is three-fold:

1. We propose a systematic method to fairly evaluate new clustering algorithms, new distance measures, and a new phase classification features for representative sampling. Our methodology also helps the user to gain better understanding of the sampling method.
2. We investigate the effectiveness of using different clustering algorithms and different distance metrics. We show that the effectiveness varies between the phase feature space and the target metric (CPI) space.
3. We propose a new microarchitecture-independent data locality based Reuse Distance Distribution for identifying phases in a program. We show that for a set of benchmarks it consistently produces more homogenous phases than BBV.

3. Related Work

In this section, we survey recently proposed research using clustering based sampling of intervals to reduce simulation time of benchmarks.

Sherwood et.al. [15] proposed a methodology called Basic Block Distribution Analysis to find a single simulation point in benchmarks. Basic block is a sequence of instructions in a program with a single entry point, single exit point, and no internal branches. Basic Block Vector (BBV) is a vector of length equal to the number of static basic blocks in the code. Each interval (a chunk of 100 million dynamic instructions in sequence) is characterized by a BBV

with each element of the vector showing the frequency of occurrence of a particular static basic block. A BBV is derived for the whole program, called the target BBV, and each entry in the BBV is normalized to total basic blocks, so that sum of all the entries in a BBV is one. Similarly, BBVs are derived for each interval of 100 million instructions and then compared with the target BBV. The comparison is directly made by subtracting one BBV from the other and adding up the absolute values of the difference of each element. The number lies between 0 and 2. The difference of 0 indicates perfect match and 2 indicates a perfect miss-match. A single simulation point is selected by finding the interval with the lowest difference.

Sherwood et.al [3] proposed the *SimPoint* methodology. In this method, instead of selecting one simulation point to represent the whole program [15], they select multiple simulation points to cover all possible phases in the program. SimPoint also uses BBV for phase classification. BBV is usually high dimensional (thousands to hundreds of thousands), and hence random projection is performed on the data to reduce the dimensionality to 15 before using k-means clustering to form interval clusters with similar BBVs. The clustering algorithm forms clusters for different values of k and picks the best solution, determined by BIC (Bayes Information Criterion) [16][17]. The simulation point that is closest to the centroid of a cluster is selected as the cluster representative. The cluster representatives (intervals) together form the simulation points of the programs. After selecting the simulation points, the CPI of whole program can be calculated as a weighted average of CPI values from each of the representative intervals weighted by the cluster size.

. Perelman et.al [14] proposed a method, Early SimPoint, to find early simulation points to reduce the time required for fast-forwarding where check-pointing is not possible. This method tries to find simulation points early in the program's execution without compromising the accuracy. Perelman et.al [14] also proposed Variance SimPoint to use statistical analysis to guide the choice of number of clusters for a user specified confidence interval and probabilistic error bound for CPI. Although Variance SimPoint improves accuracy, it is more difficult to use. One of the main issues is that it uses a sample size of 1 million instructions and assumes a perfect warm-up, which is impractical.

Lau et.al [7] explored various other microarchitecture-independent features (structures) that can be used for phase classification with SimPoint. They used a weighted average of Coefficient of Variation (CoV) of all clusters in each phase to compare these structures. This is similar to our NS metric. They experiment with various features that profile memory strides for phase classification. They conclude that BBV is more accurate than the memory stride profiles.

All these metrics are based on the memory access signatures, but, unlike our RDD feature, are weakly correlated to data locality.

Lafage and Seznec [5] proposed an approach to select representative slices of program execution based on a microarchitecture-independent feature, reuse distance expressed in terms of instructions executed between two accesses to the same address. They used hierarchical clustering to classify program slices of 1 million instructions. Their results show an average relative error of 1.52% in data cache miss-rate for the SPEC95 suite.

Todi [4] proposes a method for selecting and executing representative intervals, for SPEC2000 benchmarks, which reduces the simulation time and maintains the accuracy of simulated result. The approach in this methodology is based on collecting performance metrics using the performance monitoring counters for every interval of 1 million instructions and then using clustering to find representative intervals for phases. The main drawback of this technique is that since the measured phase classification features are for a particular machine, the clusters may not be valid for other microarchitecture configurations.

4. Evaluation Methodology

We need a good methodology to compare various representative sampling techniques. The error in the sampled simulation is the final result that one probably cares most. However, it is the result of several factors lumped together, and provides little insight into the relative advantages and disadvantages of each technique. Moreover, because the error is the result of several factors that may exhibit different trends, the error curves cross each other and make it difficult to compare different techniques. Therefore, we only use final error as one of the metrics in our evaluation methodology. We will discuss this in detail when we examine the sampling error in this section.

In order to design a comprehensive and fair evaluation methodology, we need to understand the rationale of representative sampling, and discern the different factors that affect the quality of the result. Representative sampling starts with selecting and measuring the phase classification feature and performing cluster analysis based on it. This is done entirely in the phase feature space. We expect the instruction chunks inside each cluster to be similar to each other. Since the distance between two phase feature values reflects the dissimilarity between two chunks of instructions, it is desirable that the data points within the same cluster be very close to each other. A better clustering algorithm will result in more cohesive clusters in the phase classification feature space. Nevertheless, we also want homogeneous phases in the target metric space. If the phase feature is strongly correlated with the target metric, then a cohesive cluster

will map to a homogeneous phase. On the other hand, if the correlation is weak, then the phases will not be homogenous despite the better cohesiveness in the phase classification feature space (from using a better clustering algorithm). The accuracy of the sampling result is also impacted by the choice of the data point to represent a phase. Usually, if the homogeneity in a phase is improved, then randomly selecting any data point to represent this phase should result in smaller error. However, as in SimPoint, the representative data point is not selected randomly. Instead, the one closest to the center of the cluster in the phase classification feature space is selected. It is possible that, even though the target metric in the phase is not homogeneous, the data point picked happens to exhibit a target metric value equal to the mean target metric value of the phase, resulting in zero error.

Therefore, our evaluation methodology consists of three components in the feature and target metric space. First, we examine the cohesiveness of clusters in the phase classification feature space. Then, we measure the homogeneity of target metric in each phase. Lastly, we look at the final sampling error. The last two metrics are in the target metric space. Another difficulty in representative sampling is how to determine the optimal number of clusters/phases - an open problem in data mining research. In representative sampling the optimal number of clusters usually depends on an optimization criterion e.g. BIC score threshold used in SimPoint. In this study, we present results for all cluster numbers between 4 and 10, which covers the range of the number of clusters used in SimPoint.

In the phase feature space, the cohesiveness of clusters can be measured by the Average Distance (AD) from each data point to the representative data point of the cluster it belongs to.

$$AD = \sum_{x} distance(x, c_i) / n ,$$

where $\boldsymbol{x}$ is a data point and $\boldsymbol{c_i}$ is the representative data point for the cluster that $\boldsymbol{x}$ belongs to and $\boldsymbol{n}$ is total number of data points. This metric can be used to compare different clustering algorithms with the same phase classification feature and the same distance measure. We want the program behavior in each phase to be as homogeneous as possible. A better algorithm will give tighter clusters and thus a smaller average distance.

We use the Normalized Standard Deviation (NS) metric, defined as follows, to evaluate the homogeneity of phases in the target metric space.

$$NS = \sqrt{(\sum_{i=1}^{k} \frac{n_i}{n} S_i^2)} / S ,$$

where $\boldsymbol{n_i}$ is the size of cluster $\boldsymbol{i}$. $\boldsymbol{S_i}$ is the standard deviation of the target metric in cluster $\boldsymbol{i}$, n is the total number of data points, and $\boldsymbol{S}$ is the standard deviation of the target metric for all data points.

NS reflects the tightness of the cluster in the target metric space. The lower the normalized standard deviation, the more homogeneous the phases are. NS is determined by the quality of clustering in the phase classification feature space as well as by the correlation between the feature and the target metric. Since the calculation of NS only involves target metric, it can be used to compare different clustering algorithms, different distance measures, and different phase features.

Lau et. al. used weighted average of Coefficient of Variation (COV) to evaluate different phase classification features [7]. The average COV is similar to NS. One major difference is that in NS the denominator is the total standard deviation while in COV it is the mean target metric value. Average COV can also be used to measure the homogeneity of phases, but NS possess one advantage: it also reflects the benefit of stratified sampling. *Stratified sampling* is a sampling method well studied in statistical sampling theory. It has the same rationale as the representative sampling discussed above. In stratified sampling, the population is divided into homogeneous groups called *strata*, which corresponds to our term phases or clusters. Unlike in the representative sampling where the data point closest to the center is selected, multiple data points are randomly sampled from each stratum[2]. In *proportional stratified sampling*, the number of data points sampled in each stratum is proportional to the total number of data points in the stratum. The accuracy of a sample design is measured by the variance of its sample mean. A more complex sampling design is often compared to the simple random sampling, where data points are just randomly picked from the whole population. The *design effect* (deff) of a sampling plan is the ratio of the variance of the estimate obtained from the (more complex) sample to the variance of the estimate obtained from a simple random sample of the same number of data points. It measures the benefit of using more complex sample designs. From [8] it can be shown that

$$NS^2 = V_{stratified} / V_{random} = deff_{stratified},$$

where, $\boldsymbol{V_{stratified}}$ is the variance of sample means in proportional stratified sampling, and $\boldsymbol{V_{random}}$ is the variance in simple random sampling. Therefore, NS shows us how much we have gained compared to simple random sampling, or how much variance we have reduced by taking the extra effort to cluster. If NS is close 1, then the target metric in each cluster is as varied as in the whole benchmark. It is similar to randomly grouping data points into clusters and not achieving any reduction in variance within a phase. We will get similar accuracy just by random sampling. On

[2] Stratified sampling has the advantage that it gives a confidence interval after the sampling to quantify the accuracy of the result. It is not used in representative sampling mainly because it requires far more chunks of instructions to be simulated.

the other hand, if NS is close to 0, then the target metric remains almost constant in each phase and our result will be much more accurate than simple random sampling.

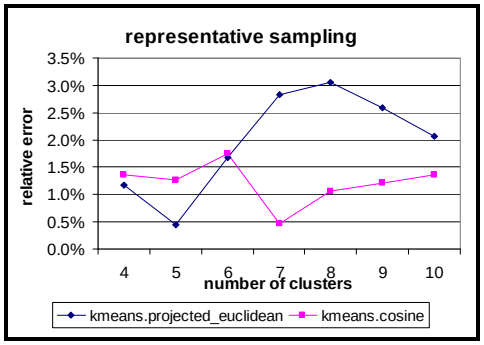
We also examine the final Relative Error (RE) in target metric compared to full cycle-accurate simulation, which is defined as,

$$RE = \left| \sum_{i=1}^{k} \frac{n_i}{n} y_i - \bar{y} \right| / \bar{y} ,$$

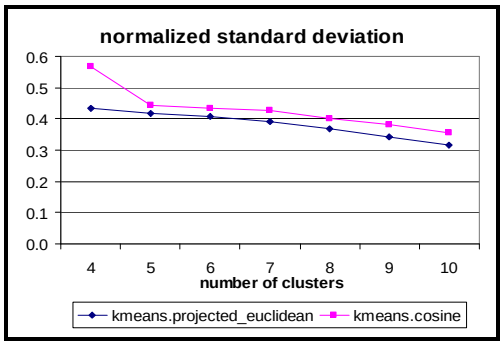
where y_i is the target metric of the representative data point in cluster i. $\bar{y}$ is the true target metric of the benchmark. The relative error is determined not only by the quality of the clustering and the correlation between the feature and target metric, but also by how close the target metric for the representative data point is to the mean target metric of a cluster. It is the final metric that the user cares about. However, unlike NS, which is calculated from all *n* data points, the relative error is affected by only *k* representative data points. Since $k<<n$, it is far less stable than NS. Consider, for example, *bzip2-source* from SPEC CPU2000 benchmark suite. Figure 1b shows NS in CPI for two distance measures, Euclidean distance with random projection and cosine distance. As expected, when we divide the data points into more clusters, the overall homogeneity improves. In contrast, Figure 1a shows the relative error for CPI of the simulation using representative sampling. The error curve does not have a clear trend. Sometimes the error even goes up significantly with more clusters, which is clearly not expected, and this indicates the unreliability of the error metric. In addition, projected Euclidean distance gives more homogeneous phases for all cluster numbers between 4 and 10. However, result for the relative error is much more "messy". The curves cross each other so neither of the two distance measures consistently gives smaller error, which makes comparison very difficult. To show the usefulness of the NS metric, the relative error of using stratified sampling is shown in Figure 1c. The result is the average of 2000 repetition of the experiment. In each experiment, a total of 200 chunks are selected. It is clear that the error in stratified sampling follows the NS and the projected Euclidean distance shows smaller errors; just as we would have expected from its better NS result. Based on the above analysis, we use NS as our main evaluation metric in the space of the target metric.

All three metrics (AD, NS, and RE) are needed to evaluate a representative sampling technique, and every metric provides a different insight. Improving one metric may not automatically make other metrics better. Suppose, we use a better clustering algorithm and get smaller average distances, but NS does not improve, then we know it is because the correlation between the phase classification feature and the target metric is not strong enough. We need to search for a better phase classification feature. If we get more homogeneous CPI in each phase

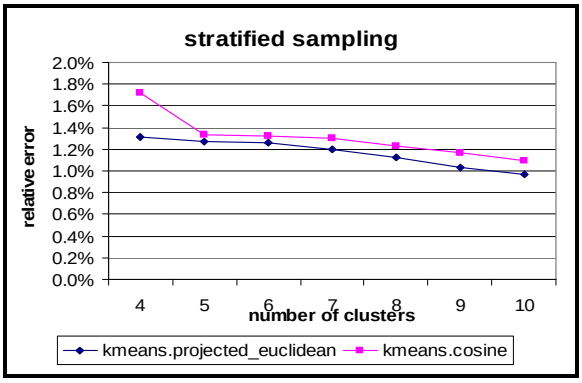
(i.e. smaller NS) but the final error remains large, then it indicates that the error introduced by picking the central data point dwarfs our improvement in homogeneity. If longer simulation time is affordable, then one can also resort to classic stratified sampling.



(a) Normalized standard deviation for CPI



(b) Relative error in CPI for representative sampling



(c) Relative error in CPI for proportional stratified sampling

Figure 1. Comparing projected Euclidean distance with cosine distance for *bzip2-source*.

5. Evaluation and analysis of clustered sampling using BBV

5.1 Comparing clustering algorithms and distance measures

Although k-means clustering algorithm is popular in representative sampling, it has serious limitations, as discussed in Section 2.1. There has not been any study on how different clustering algorithms affect the quality of representative sampling. Therefore, we would like to evaluate a different algorithm against the k-means algorithm. We choose k-medoid method [4] because it overcomes the limitations of k-means. Firstly, k-medoid methods are less strict about

the distribution of the data points and are robust to the existence of outliers. Secondly, in k-medoid methods, the medoid, which is a real data point, naturally represents the cluster.

There are several k-medoid methods such as Partitioning Around Medoid (PAM) and Clustering LARge Applications (CLARA) [9]. The problem with these methods is that they have a high computational requirement. The time complexity for PAM is $O(k(n-k)^2)$, where k is the number of clusters and n is the number of data points. CLARA algorithm exhibits a time complexity of $O(k(40+k)^2+k(n-k))$. We choose to use CLARANS (Clustering Large Applications based on RANdomized Search) algorithm, proposed by Ng and Han [10][11], because of its lower computation cost, which is basically linearly proportional to the number of data points.

CLARANS algorithm can be viewed as a search through a graph $G_{n,k}$, where n is the number of data points and k is the number of clusters. In this graph, a node is represented by a set of k data points $\{O_{m1}, O_{m2}, \ldots O_{mk}\}$. Each data point is a selected medoid. Once k medoids have been determined the remaining nodes can be assigned to the medoid that is the closest. Thus a node in the graph (i.e. a set of medoids) corresponds to a clustering solution. Two nodes are neighbors (i.e. connected by an arc) if their sets differ by only one data point. CLARANS searches the graph neighbor by neighbor for a node with minimum cost. Each node has $k(n-k)$ neighbors. Examining all $k(n-k)$ neighbors of a node is time consuming when n and k are large. To limit the cost of searching, CLARANS limits the maximum number of neighbors examined to a user specified parameter, *maxneighbor*. CLARANS algorithm proceeds as follows.

1. Input parameters *numlocal* and *maxneighbor*. Initialize *i* to 1, and *mincost* to a large number.
2. Set *current* to an arbitrary node in $G_{n,k}$.
3. Set *j* to 1.
4. Consider a random neighbor *S* of current, and calculate the cost differential of the two nodes.
5. If *S* has a lower cost, set *current* to *S*, and go to Step (3).
6. Otherwise, increment *j* by 1. If *j* <= *maxneighbor*, go to Step (4).
7. Otherwise, when *j* > *maxneighbor*, compare the cost of *current* with *mincost*. If the former is less than *mincost*, set *mincost* to the cost of current, and set *bestnode* to *current*.
8. Increment *i* by 1. If *i* > *numlocal*, output *bestnode* and halt. Otherwise, go to Step (2).

In all three representative sampling techniques we reviewed, Euclidean distance between the phase classification features is used to measure the dissimilarity between the chunks of instructions. It is well known that no single distance measure is the best for all clustering

applications. Other distance measurements such as the Manhattan distance and cosine distance have been applied to different applications.

In this study the cosine distance is of special interest. It has been successfully used in automatically clustering documents into different topics. Documents are often represented as vectors, where each element is the frequency with which a particular term occurs in the document. If we compare BBV with a text document vector, we can see their similarity. An element in BBV is the number of times a specific static basic block is executed, which is similar to the number of times a specific word occurs in a document. In addition, both are very high dimension vectors (thousands to more than a hundred thousand dimensions). Because of the similarity between BBV and the document vector, and the success of cosine distance in document clustering, it is very interesting to see whether cosine distance can be applied to representative sampling. If p and q are two vectors, then cosine distance is defined as

$$cosine\_distance(p, q) = 1 - \frac{p \bullet q}{\|p\| . \|q\|},$$

where $\bullet$ indicates vector dot product, and $\|p\|$ is the length of vector p. Because the result is divided by the norm of the vectors, the cosine distance is really a measure of the angle between p and q. If the angle is 0°, then the two vectors are the same except for the magnitude. The cosine distance will be 0, which is the minimum value. If the angle is 90°, then the two vectors do not share any elements. In other words, the code in the two chunks of instructions are complete different because they do not share any basic blocks. In this case, the cosine distance reaches the maximum value of 1.

5.2 Experiment Setup

We use 8 programs with the reference data set from the SPEC CPU2000 benchmark suite. The programs and the number of instructions are listed in Table 2. Following SimPoint, we divide the instruction stream of each program into intervals of 100 million instructions. In all our experiments we use CPI as the target metric because it is the most commonly used performance metric. To evaluate the result in the target space, we simulate all 8 benchmarks in *sim-outorder* [2] to collect CPI for each chunk.

Benchmark-Input Pair	Number of instructions (million)
art-110	41,798
bzip2-source	108,878
equake	131,518
gcc-166	46,917
lucas	142,398
mcf	61,867
vortex-1	118,976
vpr-route	84,068

Table 2. Number of instructions and simulation time of selected SPEC CPU 2000 benchmarks with reference data set. The data set name is appended to the benchmark name.

Pipeline	
Issue Width	8 instructions/cycle
Decode Width	8 instructions/cycle
Register Update Unit	128 entries
Load-Store Queue	32 entries
Commit Width	8 instructions/cycle
Cache Hierarchy	
L1 Data	16KB; 4-way assoc., 32B lines, 2-cycle hit
L1 Instruction	8KB; 2-way assoc., 32B lines, 2-cycle hit
L2 Unified	1MB; 4-way assoc., 64B lines, 20-cycle hit
Memory Access Latency	151 cycles
Combined Branch Predictor	
Bimodal	8192 entries
PAg	8192 entries
Return Address Stack	64 entries
Branch Target Buffer	2048 entries; 4-way assoc.
Misprediction Latency	14 cycles

Table 3. Processor Configuration

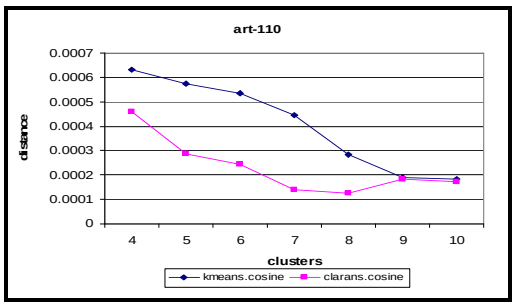
The processor configuration used in the simulation is shown in Table 3. The same configuration has been used in study on cache warm-up [12] and in validation of *SimPoint* [13]. K-means and CLARANS clustering algorithms were each evaluated using projected Euclidean and cosine distance measures. Because clustering algorithms are less effective at high dimensional Euclidean space, the dimensionality of BBV is reduced to 15 through random projection just as in SimPoint. Thus it is given the name “projected Euclidean distance”.

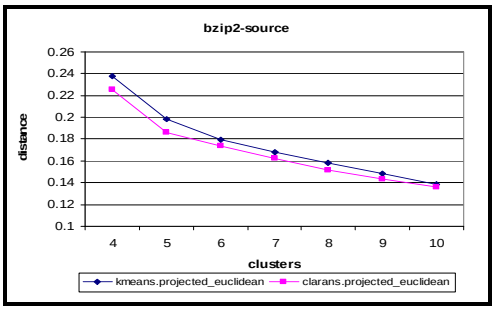
The number of clusters evaluated is from 4 to 10, which covers most of the range of number of clusters used in SimPoint. Both k-means and CLARANS may give different result with different random seeds. In k-means, a random seed is used to choose the initial centroids

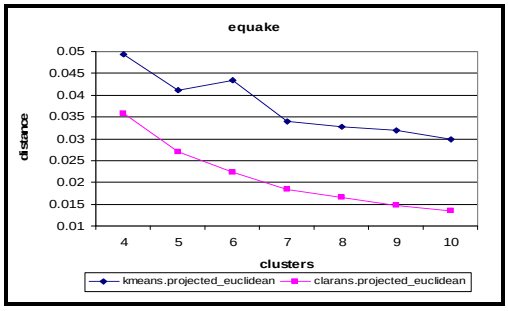
while in CLARANS a random seed controls the random search. Therefore, we run each experiment 5 times with different random seeds and the data shown below is the average result.

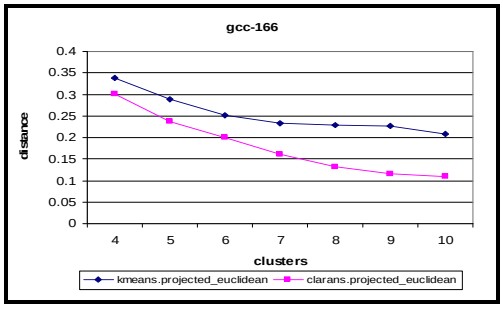
5.3 Experiment result

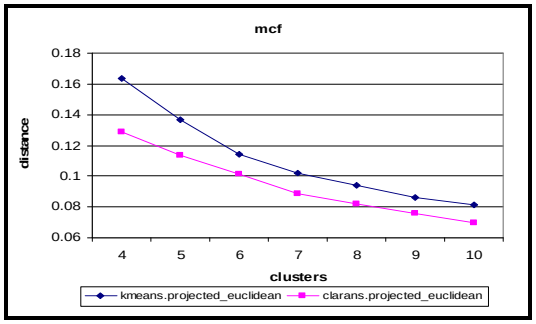
Following the evaluation methodology proposed in Section 4, we first evaluate the two clustering algorithms in the BBV space. Figure 2 compares the average distance of k-means and CLARANS algorithms for project Euclidean distance. The graphs for cosine distance are in *Appendix A.* Since average distance metric cannot be compared between different distances, the result for projected Euclidean distance and cosine distance are drawn separately. CLARANS clearly produces tighter clusters than k-means in most cases. For some benchmarks, such as *equake* and *gcc-166* the reduction in average distance is significant. At 10 clusters, for projected Euclidean distance, CLARANS reduces the distance by over 50%, while for cosine distance the reduction is almost 90%.

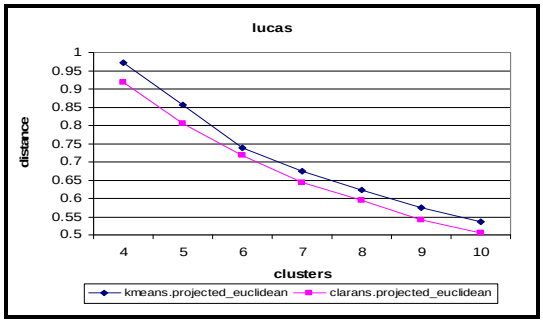












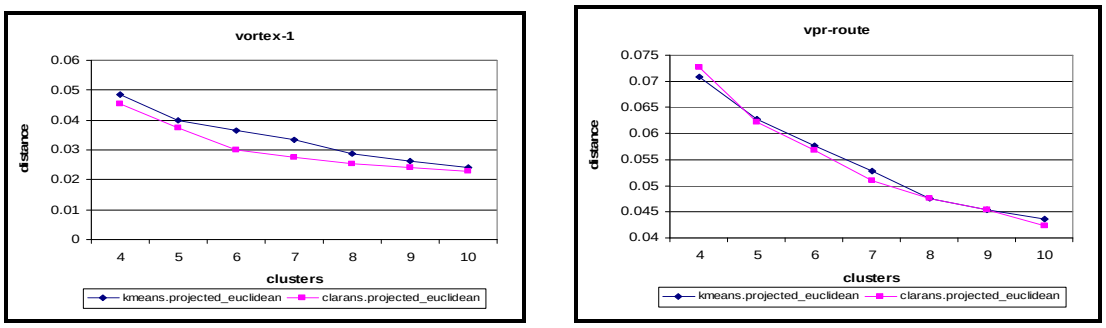
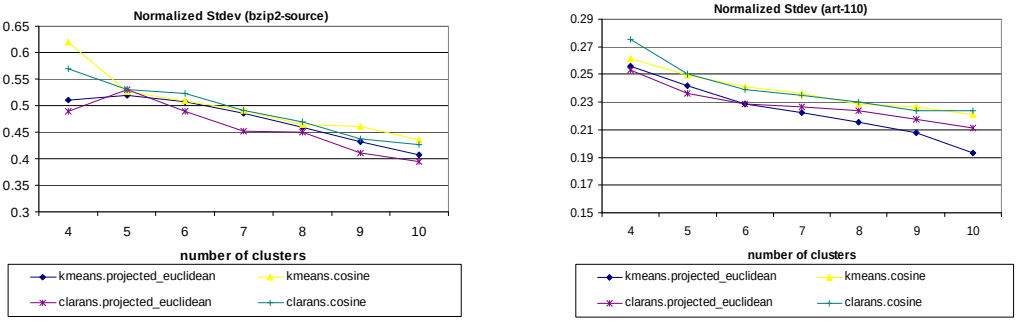


Figure 2. Average distance for different clustering algorithms and distance measures.

We then examine the normalized standard deviation, which is shown in Figure 3. The NS shows a downward trend. Therefore, as we increase the number of phases, the CPI in each phase shows lesser variance. Because CLARANS has improved the quality of the clusters in the BBV space, it is expected that it will produce more homogeneous phases than k-means. However, this is not always true. The curves for CLARANS and k-means sometimes cross each other. For example, for benchmark *lucas*, CLARANS shows an advantage over k-means in the projected Euclidean space for all number of clusters. But, for normalized standard deviation, the picture is mixed. The two curves cross each other twice. CLARANS is better for 4 and 5 clusters but k-means wins for 6, 7 and 8 clusters. CLARANS overcomes k-means again at 9 clusters and they finally tie at 10 clusters. This type of behavior indicates that the correlation between BBV and CPI is not strong enough for the reduction in distance in the BBV space to translate into better homogeneity in CPI for each phase. Nevertheless, CLARANS algorithm produces more homogeneous CPI phases in most cases. Table 5 shows the clustering algorithm and distance measure with the overall lowest normalized standard deviation. In 5 (or 6) out of 8 benchmarks, CLARANS is better than k-means. The cosine distance we experimented with, on the other hand, does not seem to perform better than the projected Euclidean distance. Cosine distance is the best for only *vortex-1*.



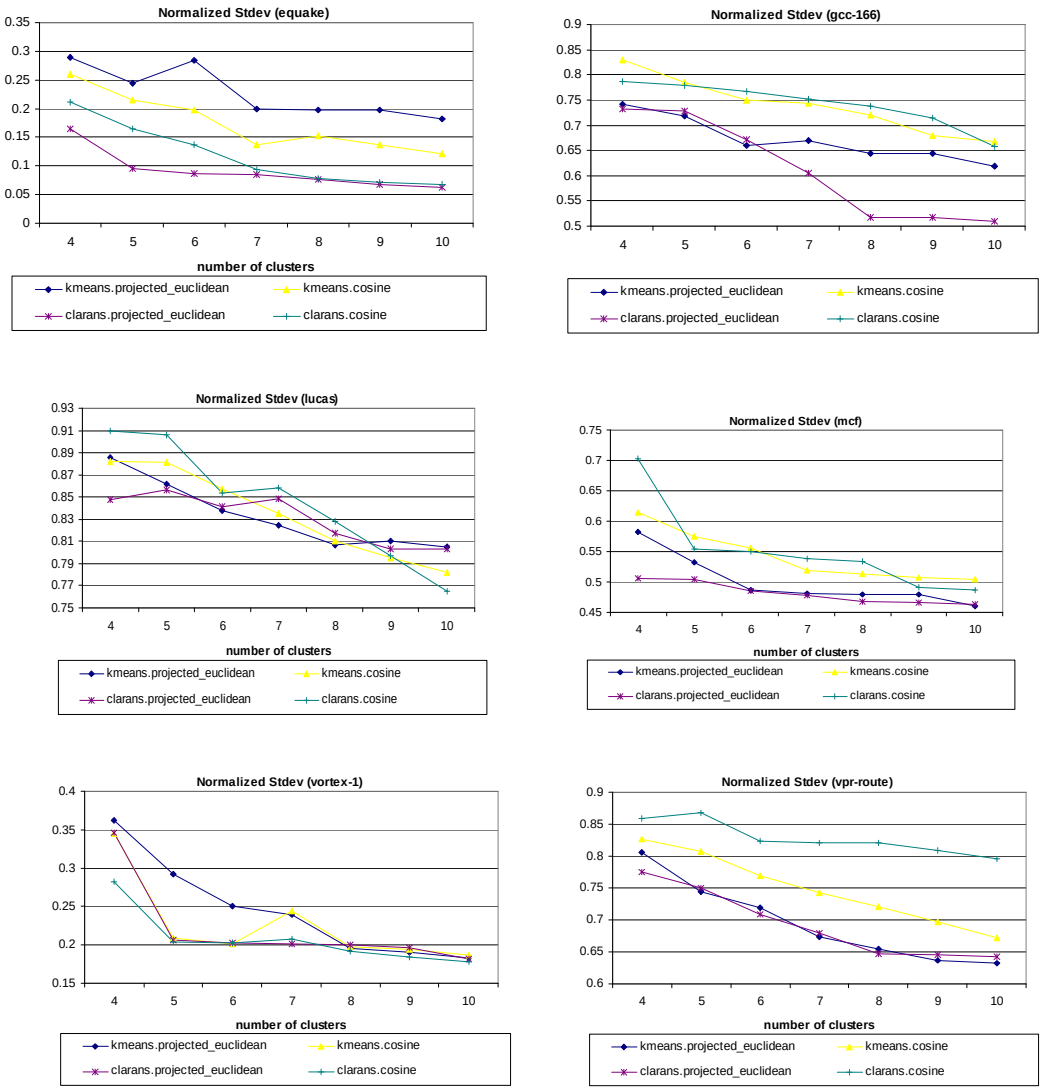


Figure 3. Normalized standard deviation for different clustering algorithms and distance measures

Benchmark	Best clustering algorithm and distance measure
art-110	Kmeans with projected Euclidean distance
bizp2-source	CLARANS with projected Euclidean distance
equake	CLARANS with projected Euclidean distance
gcc-166	CLARANS with projected Euclidean distance
lucas	No clear winner
mcf	CLARANS with projected Euclidean distance
vortex-1	CLARANS with cosine distance
vpr-route	Tie between k-means and CLARANS with projected Euclidean distance

Table 4. Best clustering algorithm and distance measure for different benchmarks

Figure 4 shows the relative error of representative sampling using different clustering algorithms and distance metrics. Since the normalized standard deviation decreases as more phases are

Figure 4. Relative error in the CPI from representative sampling for different clustering algorithms and distance measures.

identified, the errors are expected to follow suite. However, they do not show a general trend and vary a lot, crossing each other multiple times. If it is sometimes difficult to pinpoint the algorithm with the best NS, then it is almost impossible to identify the one with consistently lowest error. If we only focus on projected Euclidean distance (used in SimPoint), then CLARANS consistently results in smaller error for *equake* and *vortex-1*. For the remaining 6 benchmarks, they are comparable.

To conclude, better clustering algorithms such as CLARANS can definitely produce tighter clusters in the BBV space. However, as we are moving further away from BBV space and toward the final error, the benefit of better algorithms diminishes because more factors come to play a role. In occasional cases k-means produces less variation in CPI in the phases, but CLARANS generally result in more homogeneous CPI phases. The final simulation error depends not only on the homogeneity in the phases, but also on whether the data point closest to the cluster center in BBV space exhibits CPI value equal to the mean CPI of the phase. This adds more uncertainty to the result and makes it difficult to evaluate different algorithms with error. Nevertheless, CLARANS shows smaller error for two benchmarks and gives comparable result for the rest.

7. Reuse Distance Distribution - A new feature for phase classification

Due to the gap between processor and memory performance, data access latency is one of the most important factors that determine program performance in modern day microprocessors. Data access latency is a function of the inherent locality in the data address stream of a program. Therefore, we feel that a feature based on the data locality of a program will be able to find phases in a program that have similar data locality, and hence show similar performance. It is important that the locality feature should be microarchitecture independent. This increases the confidence that the phases identified by the feature will be valid across different microarchitectures.

BBV feature, used in *SimPoint*, is based on the intuition that the performance of a program at a given time is directly related to the code it is executing at that time. The BBV feature does not capture the properties of the data locality of the program. However, it is possible that a static section of code in a program has different memory reference patterns at different points of time in its execution. A feature based on the temporal and spatial data locality in a program will be able to capture this behavior, and may therefore perform better than BBV in finding phases that are more homogeneous.

There has been some previous research work to explore the design space for finding such a feature based on the data locality of the program. Lau et. al. [7] extensively examined different phase features that are based on data accesses, including local stride (with and without PC hash), global stride (with and without PC hash), loops with local stride, memory working set size, working set bit vector, and memory access frequency vectors. They conclude that the BBV feature is better than the data accesses based features that they studied. The features used in their study are related to the access pattern of the data address stream, but do not directly measure the data locality. Since data locality impacts program performance, these phase features may not be a strong enough indicator of the program performance.

Lafage et. al. [5] used average memory reuse distance (RDI) feature for finding phases in a program that are homogeneous with reference to data cache miss-rate. We implemented a feature similar to RDI to understand whether the phases identified by RDI are also homogeneous in CPI. For the 8 program-input sets we used from SPEC CPU2000 benchmark suite, we found that BBV performs significantly better than RDI for finding phases in a program that are homogeneous with reference to the CPI.

7.1 Reuse Distance Distribution (RDD) Definition

A reuse pair of memory accesses is a pair of addresses in the data memory stream of a program that map to the same memory line, without any intermediate accesses to the same memory line. A memory line is analogous to a cache block. Reuse distance is the number of memory addresses that are accessed between the accesses to the two addresses in the reuse pair. We define the RDD feature as the relative frequencies of the different reuse distances of all the reuse pairs in the data address stream of a program. The reuse distances can have a large number of unique values (theoretically ranging from 0 to the total number of data memory references made by a program minus two). Therefore, in order to make the distribution more easily manageable, we use a grouped frequency approach to represent the relative frequency of reuse distances. The width of each interval in the histogram is exponentially distributed – i.e. a reuse distance of r is classified into interval $\lceil \log_e r \rceil$ (ceiling of natural log of the reuse distance). In general, interval n consists of reuse distances from e^{n-1} to e^n. By definition, a reuse distance of zero is classified into the first interval. We can represent the RDD feature as a vector with n elements, where element i ($i \leq n$) is the relative frequency of the number of reuse distances for interval i.

For a given memory line size, the RDD feature characterizes the temporal locality of the data memory address stream. Information about the spatial locality of a memory address stream can also be characterized by measuring the RDD feature for a range of different memory line

sizes. The RDD feature is able to discriminate the execution slices from each other based on the temporal and spatial locality of their data access stream.

The following example is a simple illustration of how the RDD feature is calculated for an address stream.

Example:

Consider the following data memory address stream (address, access #):

0x2004 (#1), 0x2022 (#2), 0x300c (#3), 0x2108 (#4), 0x3204(#5), 0x200a (#6), 0x2048 (#7), 0x3108(#8), 0x3002(#9), 0x320c (#10), 0x2040(#11), 0x202f (#12)

For a memory line of 16 bytes, the memory lines to which these addresses maps is calculated by masking the least significant 4 bits in the address i.e. addresses 0x0000 to 0x000f will map to memory line 0, 0x0010 to 0x001f will map to memory line 1 etc. Therefore, the address in the data stream, 0x2004, will map to memory line 0x200, address 0x2022 to 0x202, etc. The sequence of memory lines accessed by this address stream is:

0x200 (#1), 0x202 (#2), 0x300 (#3), 0x210 (#4), 0x320(#5), 0x200(#6), 0x204(#7), 0x310(#8), 0x300(#9), 0x320(#10), 0x204(#11), 0x202(#12)

Addresses for reference #1 and #6 are different, but they map to the same memory line, 0x200, and therefore form a reuse pair (#1, #6). Similarly, reference #3 and #9 map to the same memory line, 0x300. The list of all the reuse pairs in the example address stream is (#1, #6), (#2, #12), (#3, #9), (#5, #10), (#7, #11), For reuse pair (#1, #6), the reuse distance is the number of memory lines accessed between the reference #1 and #6, which is 4. According to the logarithmic scheme used for classifying reuse distance into intervals, a reuse distance of 4 is classified into interval 2 ($\lceil \log_e 4 \rceil$). Following is a list of the reuse distances, and the intervals they are classified into for all the reuse pairs in the example address stream, (reference #, reference #, reuse distance, interval):

(#1, #6, 4, 2), (#2, #12, 9, 3), (#3, #9, 5, 2), (#5, #10, 4, 2), (#7, #11, 3, 2),

We observe that out of the 5 reuse pairs of memory accesses, 4 reuse pairs are classified into the 2nd interval, and 1 reuse pair is classified into the 3rd. The RDD feature (for n=5 intervals) for the example memory address stream can be represented by the following vector: RDD_{16} = <0, 0.8, 0.2, 0, 0> i.e. 80% of the reuse distances are in the 2nd interval and 20% of the reuse distances are in the 3rd interval.

7.2 Comparing RDD and BBV features for phase classification

As described in section 4, normalized standard deviation is a more reliable and insightful metric as compared to the final error in CPI for comparing two features used for phase

classification. We therefore used normalized standard deviation metric for comparing the RDD and BBV features. We used the 8 programs listed in Table 1. Similar to *SimPoint*, we divide the instruction stream of each program into intervals of 100 million instructions. RDD feature (for memory line sizes of 16, 64, 256, and 4096 bytes) and the BBV features are measured for every interval of 100 million instructions. The different clustering algorithms and distance measures that were used for clustering points in the BBV feature space are listed in Table 4. For clustering the points in the RDD feature space, we used the CLARANS and k-means clustering algorithm each with Euclidean and cosine distance measures. To evaluate the result in the target metric space (CPI), we gathered CPI data for the processor configuration described in Table 2. The same configuration was used for validating *SimPoint* [3].

In order to make a meaningful comparison between the two features, for every program, we selected the best (smallest normalized standard deviation in CPI) algorithm-distance pair for the RDD feature, and compared it with the best algorithm-distance pair for the BBV feature. Figure 5 shows a plot of the best algorithm-distance pair for RDD and BBV features for the 8 program-input pairs used in this study.

From these graphs we observe that, irrespective of the number of clusters, the RDD feature gives lower normalized standard deviation in CPI than the BBV feature for *gcc-166, lucas, mcf, vpr-route,* and *art-110* programs. For *gcc-166,* the normalized standard deviation for BBV and RDD features are almost the same if 4 clusters are selected. However, for 5 to 7 clusters, the RDD feature shows a smaller normalized standard deviation as compared to the BBV feature (0.45 for RDD compared to 0.61 for BBV for 5 clusters). For cluster sizes of 9 and 10, the normalized standard deviation for RDD feature is smaller than BBV feature by 11%. For *lucas,* the normalized standard deviation for 4 clusters formed using the RDD feature is smaller than the normalized standard deviation from forming 10 clusters using the BBV feature. This shows that *lucas* significantly benefits from the RDD feature as compared to the BBV feature. Forming more than 4 clusters using the BBV feature does not significantly benefit *mcf.* The normalized standard deviation given by the BBV feature for 4 to 10 clusters is not very different, showing that the benefit from increasing the number of clusters is not very significant. However, the normalized standard deviation for *mcf* using RDD feature is not only smaller than that of BBV, but also reduces from 0.45 to 0.33 when the number of cluster are respectively increased from 4 to 10. For *vpr-route,* the normalized standard deviation for RDD is significantly smaller than BBV for 4 clusters (0.56 for RDD feature compared to 0.8 for BBV). For all other clusters sizes, the normalized standard deviation for RDD feature is smaller by at least 24% than that for the BBV feature. The normalized standard deviation for *art-110* is very small (0.25 for BBV

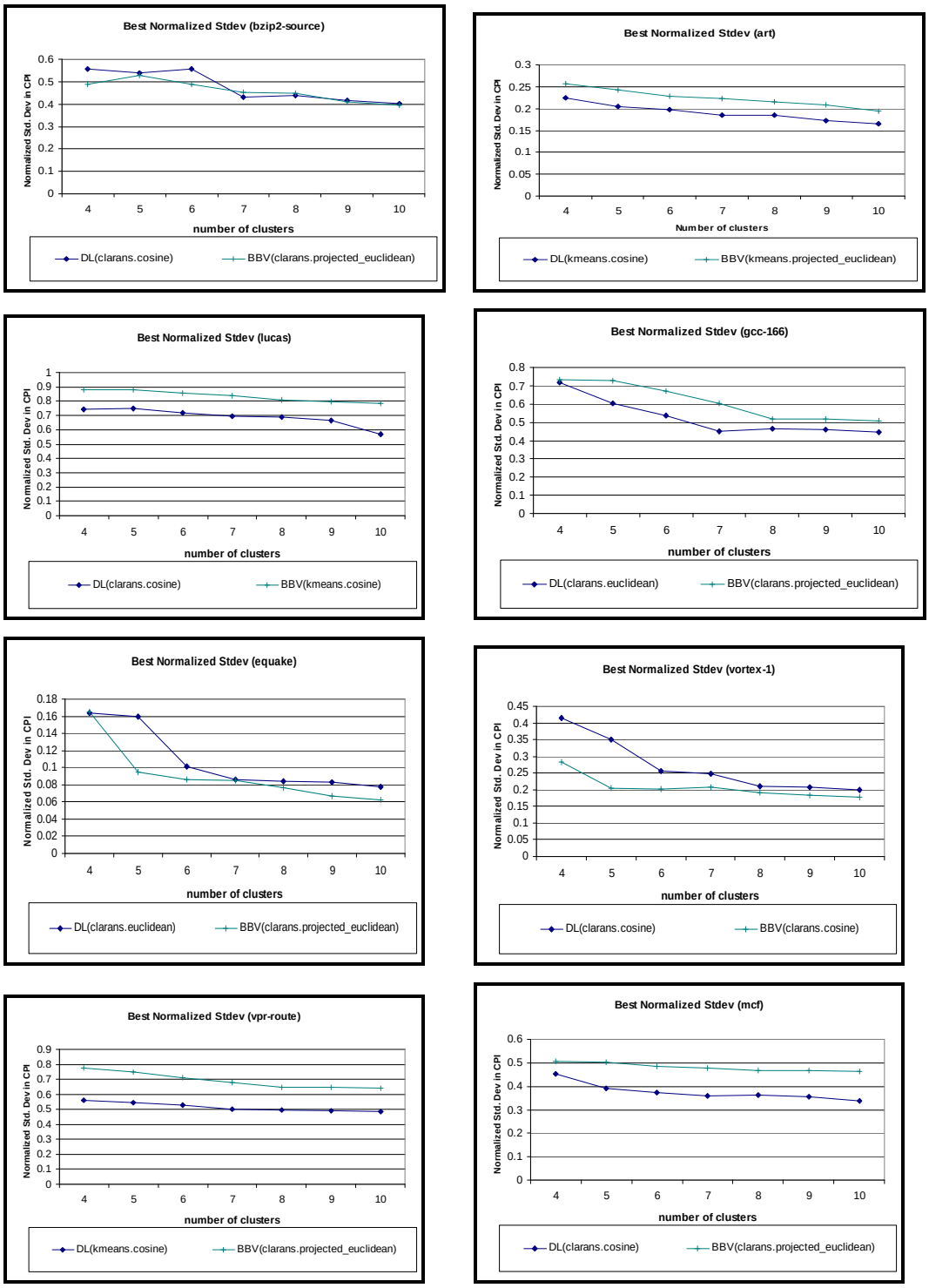


Figure 5. Normalized standard deviation in CPI for BBV and RDD feature for 8 program-input pairs from SPEC CPU2000 integer benchmark suite.

feature and 0.22 for RDD feature) even when just 4 clusters are selected. This shows that *art-110* substantially benefits from clustering using the BBV or RDD feature. Although the normalized standard deviation for *art-110* is small for the BBV feature, the RDD feature improves it by approximately 14% irrespective of the number of clusters that are selected.

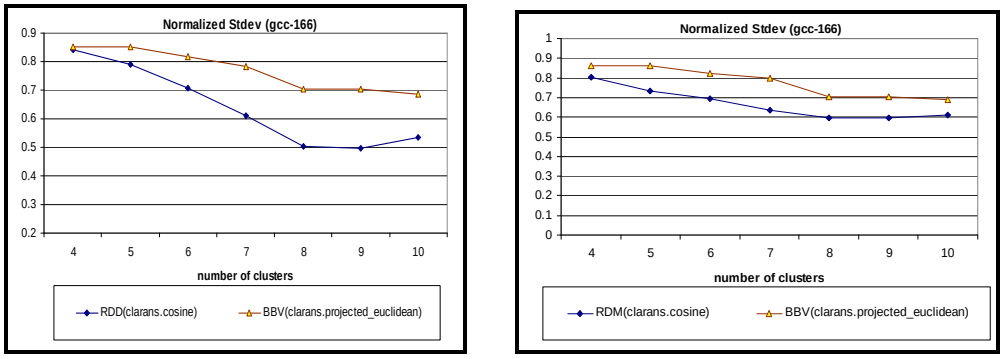
For *bzip2-source*, the graphs for both the features cross each other and hence we conclude that none of the features clearly outperforms the other. However, for *vortex-1* and *equake*, the BBV feature always gives a lower normalized standard deviation in CPI as compared to the BBV feature.

Pipeline		
Issue Width	8 instructions/cycle	16 instructions/cycle
Decode Width	8 instructions/cycle	16 instructions/cycle
Register Update Unit	128 entries	256 entries
Load-Store Queue	64 entries	128 entries
Commit Width	8 instructions/cycle	8 instructions/cycle
Cache Hierarchy		
L1 Data	16KB; 2-way assoc., 32B lines, l-cycle hit	32KB; 2-way assoc., 32B lines, 1-cycle hit
L1 Instruction	16KB; 2-way assoc., 32B lines, 1-cycle hit	32KB; 2-way assoc., 32B lines, 1-cycle hit
L2 Unified	256KB; 4-way assoc., 64B lines, 12-cycle	256KB; 8-way assoc., 64B lines, 16-cycle hit
Memory Access	hit	100 cycles
Latency	100 cycles	
Combined Branch Predictor		
Bimodal	2048 entries	8192 entries
PAg	2048 entries	8192 entries
Return Address Stack	32 entries	64 entries
Branch Target Buffer	512 entries; 4-way assoc.	2048 entries; 4-way assoc.
Misprediction	7 cycles	10 cycles
Latency		

Table 5: Processor configurations for the additional two microarchitectures used to compare RDD and BBV features

For this processor configuration, the RDD feature is consistently better than the BBV feature for 5 out of the 8 benchmark programs. However, it is possible that the RDD feature is better than BBV just for the microarchitecture configuration that was selected for this experiment. Therefore, we feel that it is important to evaluate RDD and BBV features on microarchitectures

that are very different from the one chosen for this experiment. We therefore selected two configurations, same as those used for evaluation of SMARTS sampling methodology [18], as additional points to compare BBV and RDD features. We repeated our experiment with the 8 benchmark programs for the two microarchitecture configurations. Table 5 shows the microarchitecture details of the two configurations. Figure 6 shows a plot of the best algorithm-distance pairs (for normalized standard deviation) for RDD and BBV features for the *gcc-166* program on the two microarchitectures. The graphs for the other 7 programs are in *Appendix B* for the 8-way configuration and in *Appendix C* for the 16-way configuration.



(a) 8-way processor configuration (b) 16-way processor configuration

Figure 6. Normalized standard deviation in CPI for BBV and RDD features (for *gcc-166* program)

We again observe that the RDD feature gives lower normalized standard deviation in CPI than the BBV feature for the same five programs, *gcc-166*, *art*, *lucas*, *mcf*, and *vpr-route* programs. For *bzip2-source*, none of the features clearly outperformed the other. For *equake* and *vortex-1* programs, the BBV feature gives a lower normalized standard deviation in CPI as compared to the RDD feature.

In this study, we found that the RDD feature is consistently better than BBV for phase classification in 5 out of 8 programs that we used. The validation using three different microarchitecture configurations has increased our confidence that the results are independent of the microarchitecture and are generally applicable. This suggests that the best feature for finding phases is program dependent. However, the best feature for every program often holds true on different microarchitecture configurations. Based on these results, we feel that one can select the best feature for a program using one microarchitecture and also use it on similar microarchitectures.

7.3 Comparing clustering algorithm-distance measure for RDD feature

In this section, we compare the quality of CLARANS and K-means clustering algorithms for clustering using the RDD feature. We use the Average Distance (AD) metric to compare the quality of clusters in the RDD feature space. Figure 7 shows the graphs for the AD for different number of clusters for *gcc-166*. The graphs for the other 7 programs are in Appendix D. We observe from the graphs that for all the programs, irrespective of the distance measured used and the number of clusters formed, CLARANS clustering algorithm shows a lower AD as compared to the k-means algorithm. On the three different microarchitectures, for the RDD feature, CLARANS algorithm produces more homogeneous phases in CPI in most cases than k-means algorithms. This observation with RDD agrees with our previous conclusion with BBV, that, in general, CLARANS produces more homogeneous phases in CPI.

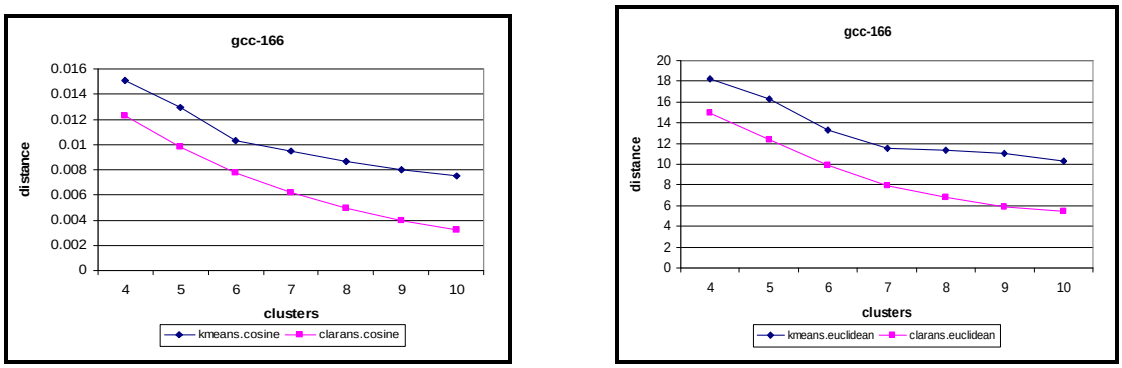


Figure 7. Average Distance for different number of clusters in the RDD feature space (for *gcc-166* program)

8. Conclusion

In this paper, we proposed to use a set of statistical metrics, and showed that, unlike error in target metric space, these metrics are very reliable, insightful, and provide a deeper understanding of the quality of clustering in representative sampling. We used these metrics to evaluate the benefit from using CLARANS clustering algorithm and cosine distance measure. We proposed and evaluated a new data locality based microarchitecture independent feature, RDD, for phase classification in a program.

Our experiments showed that for BBV and RDD features, for all benchmarks, CLARANS produces more cohesive clusters in the feature space as compared to the k-means clustering algorithm. CLARANS algorithm also results in more homogeneous phases in CPI for many, but not all, benchmarks. From this we can conclude that a better clustering algorithm can

improve the quality of clustering in the feature space, but the benefit obtained in the target metric space also depends on the correlation between the feature and the target metric.

The new feature that we propose, RDD, is consistently better than BBV for phase classification in 5 out of 8 programs on three different microarchitectures. Therefore, the best feature for finding phases is program dependent, but often holds true on different microarchitecture configurations. This helps the user to choose the best feature for more efficient microprocessor simulation. The user can select the best feature for a program using one microarchitecture, and be confident that the results can be used on different microarchitectures.

References

[1] Standard Performance Evaluation Corporation. SPEC CPU2000 V1.2. http://www.spec.org/cpu2000/

[2] D. Burger, and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin-Madson, June 1997.

[3] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (October 2002), 45-57.

[4] R. Todi. SPEClite: Using Representative Samples to Reduce SPEC CPU2000 Workload. IEEE 4th Annual Workshop on Workload Characterization. 2001.

[5] T. Lafage and A. Seznec, "Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream", Kluwer Academic Publishers, pp. 145-163, September 2000.

[6] Hartigan, J.A. and Wong, M.A. A K-means clustering algorithm. Applied Statistics 28, 100–108. 1979

[7] Jeremy Lau, Stefan Schoenmackers, and Brad Calder, Structures for Phase Classification, 2004 IEEE International Symposium on Performance Analysis of Systems and Software, March 2004.

[8] W. G. Cochran. Sampling Techniques, 3rd ed. John Wiley & Sons, 1977.

[9] L. Kaufman and P. J. Rousseeuw. Finding Groups in Data: an Introduction to Cluster Analysis. John Wiley & Sons. 1990.

[10] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. Proceedings of the 20th VLDB Conference. Santiago, Chile. 1994.

[11] Raymond T. Ng, Jiawei Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. IEEE Transactions on Knowledge and Data Engineering archive Volume 14 , Issue 5 , 2002. pp. 1003 – 1016.

[12] J. W. Haskins, Jr. and K. Skadron. "Memory Reference Reuse Latency: Accelerated Warm-up for Sampled Microarchitecture Simulation." In Proceedings of the International Symposium on Performance Analysis of Systems and Software, Mar. 2003.

[13] Example Error Rates for SimPoint, http://www.cs.ucsd.edu/~calder/simpoint/error-rates.htm

[14] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (September 2003), 244-255.

[15] T. Sherwood, E. Perelman and B. Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In the proceedings of International Conference on Parallel Architectures and Compilation Techniques (September 2001)

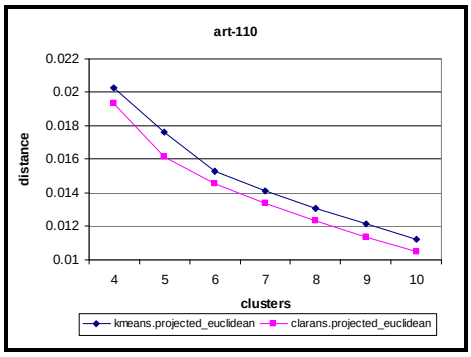
[16] R.E. Kass and L. Wasserman. A reference Bayesian test for nested hypotheses and its relationship to Schwarz criterion. Journal American Statistical Association 90(431):928-934, 1995

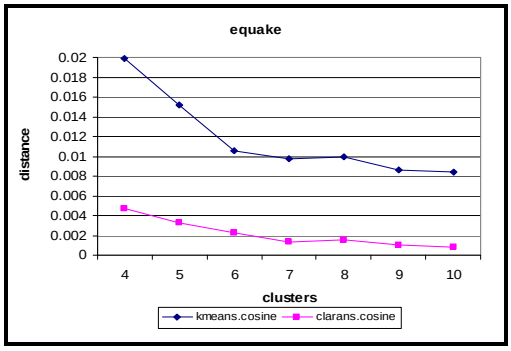
[17] D. Pelleg and A. Moore. X-Means: Extending K-means with efficient estimation of the number of clusters. In Proceedings of the 17th International Conference on Machine Learning, pages 727-734. Morgan Kaufmann, San Francisco, CA. 2000.

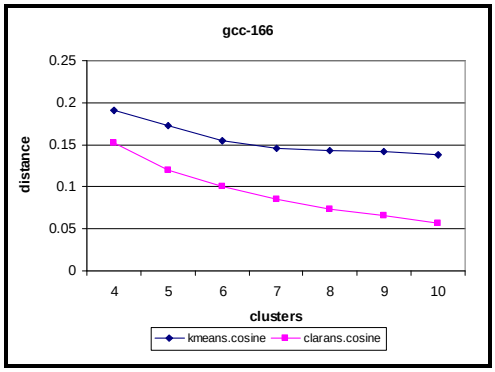
[18] R. Wunderlich, T. Wenisch, B. Falsafi, and J.Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In 30th Annual International Symposium on Computer Architecture, June 2003.

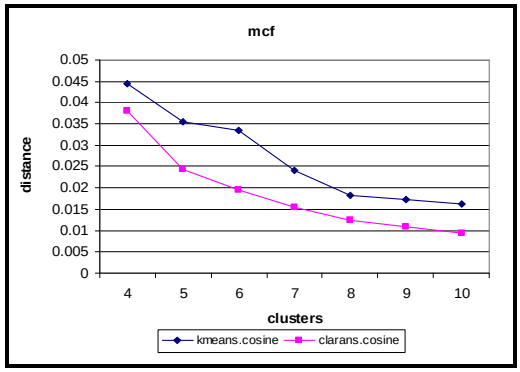
Appendix A

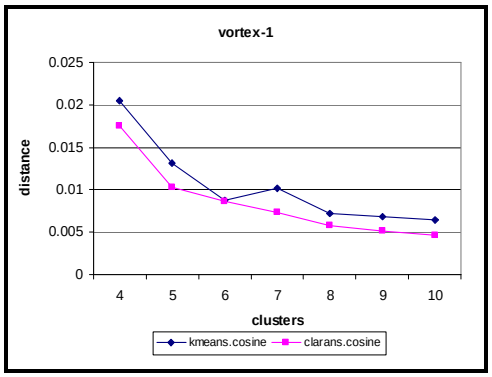
Average Distance (AD) Metric for BBV feature using cosine distance measure

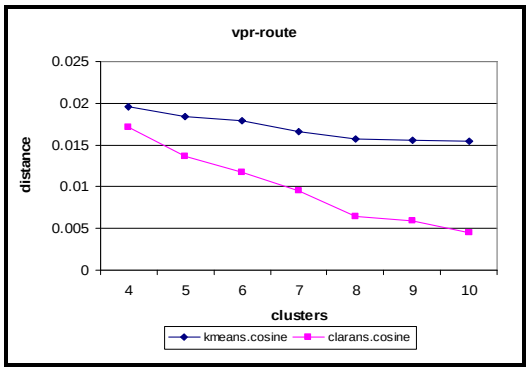


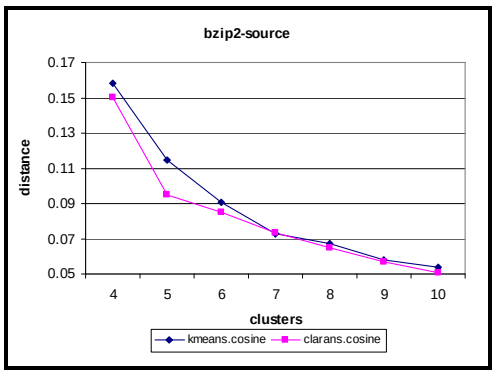


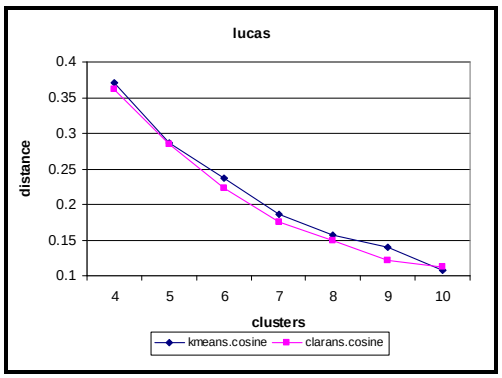






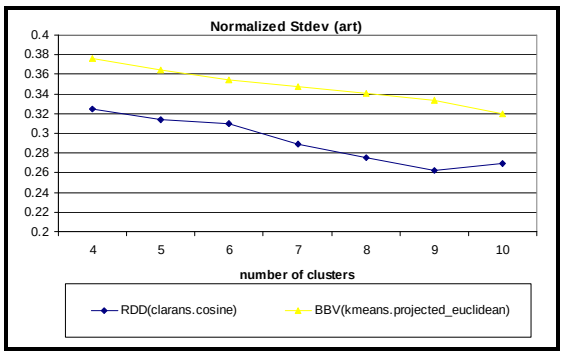


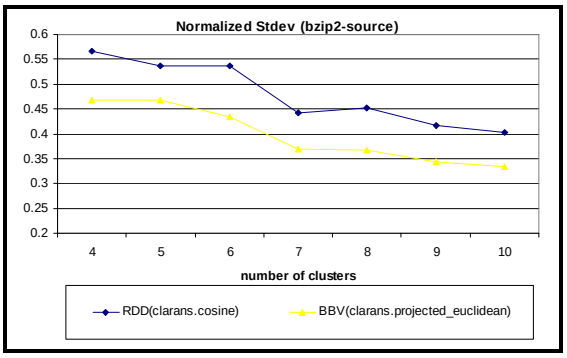


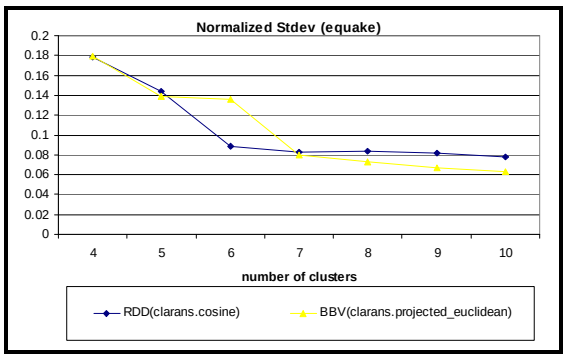


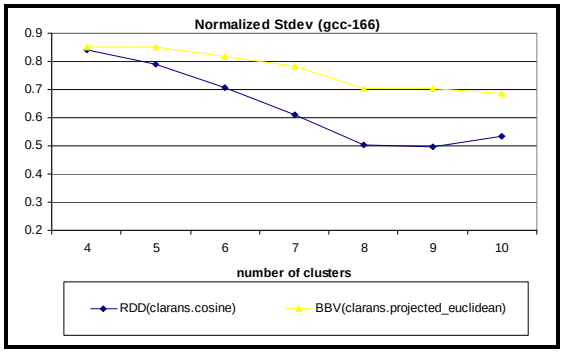
Appendix B

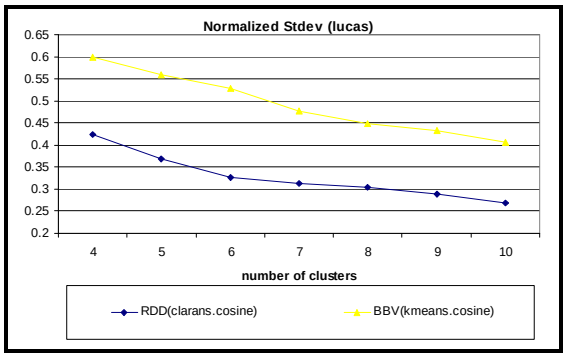
Normalized Standard Deviation in CPI for 8-way configuration (RDD feature)

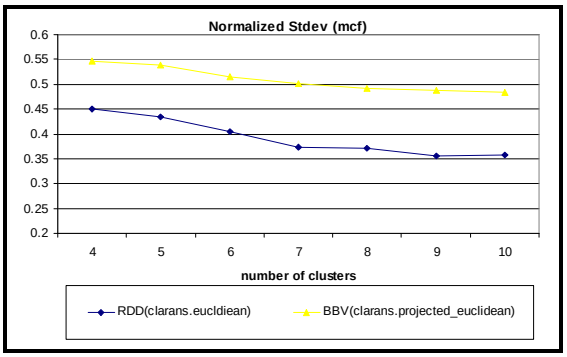


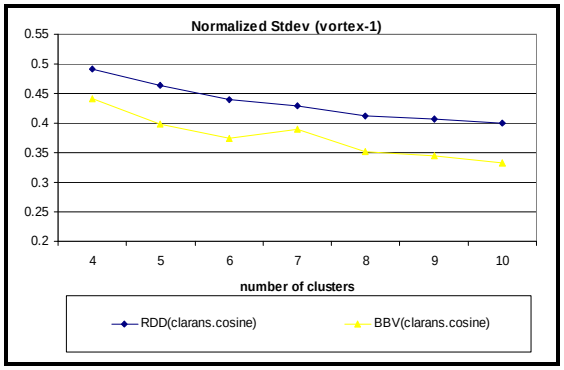


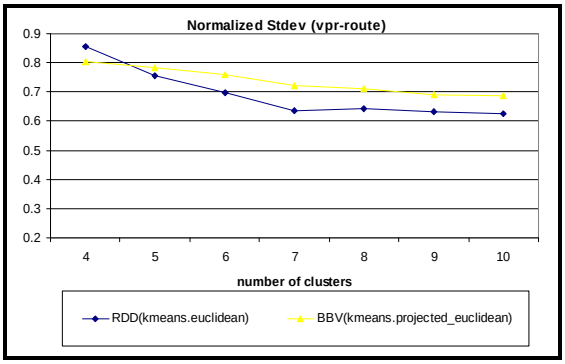






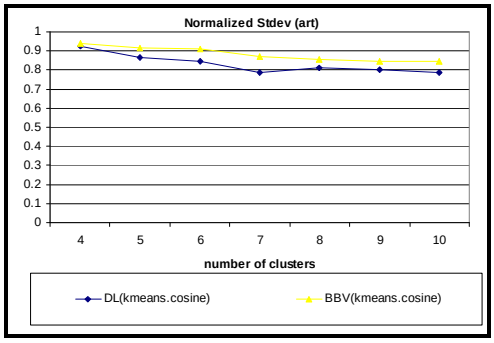


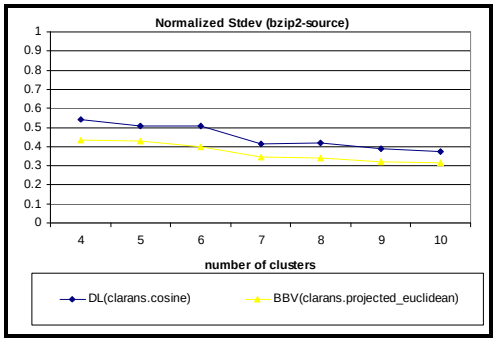


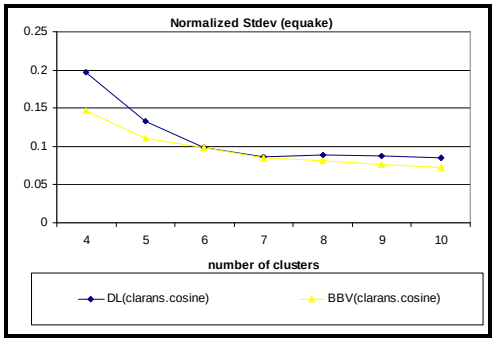


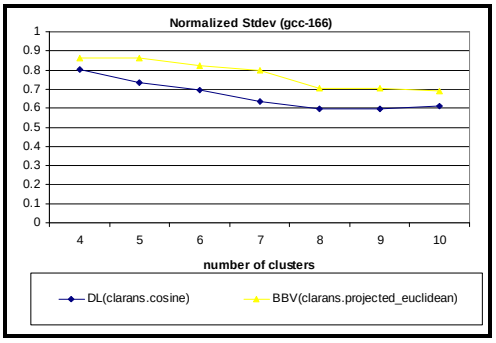
Appendix C

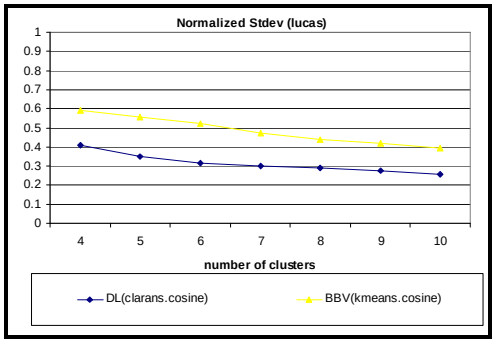
Normalized Standard Deviation in CPI for 16-way configuration (RDD feature)

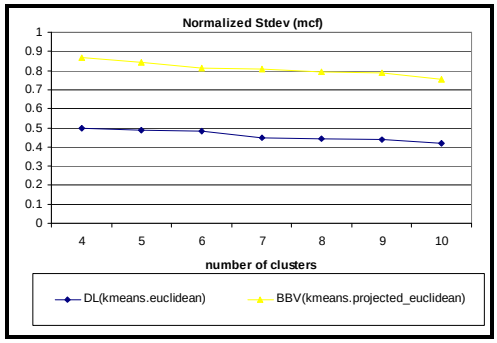


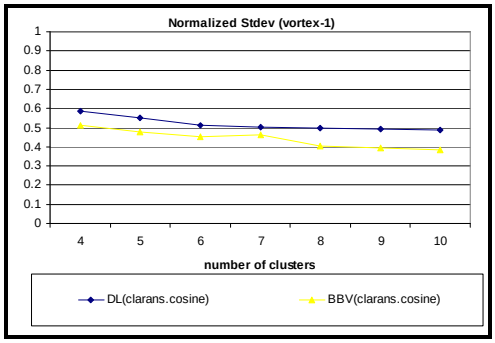


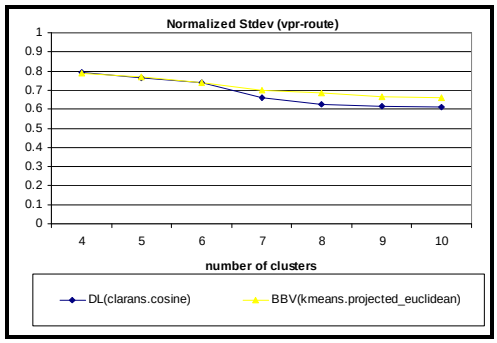






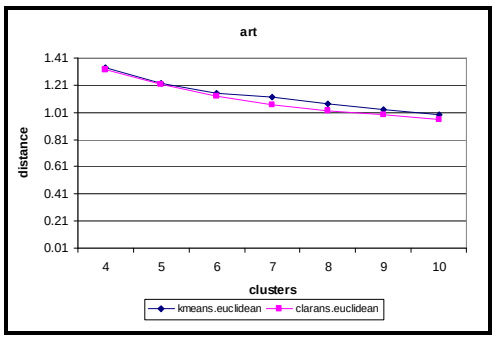


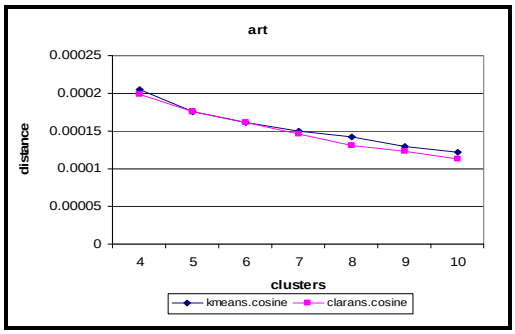


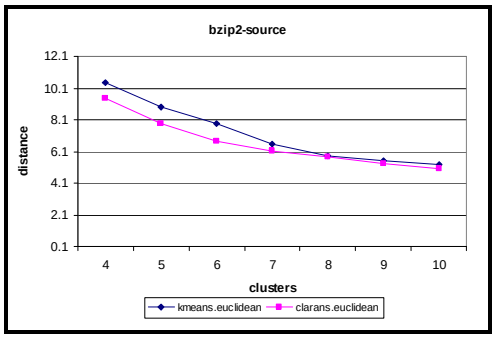


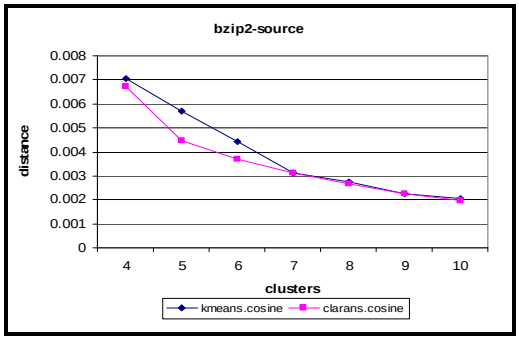
Appendix D

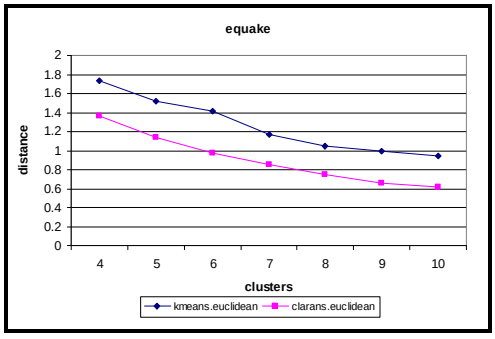
Average Distance (AD) Metric for RDD feature

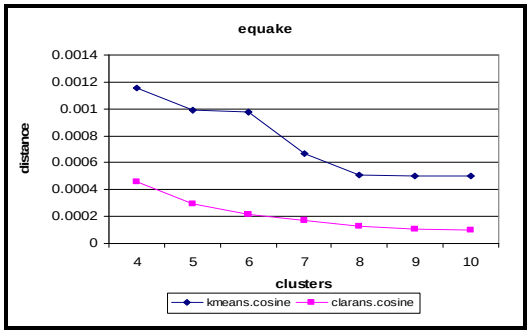


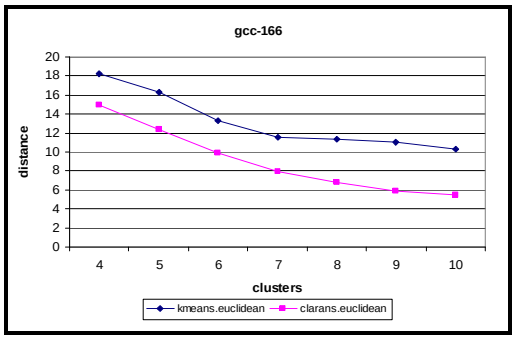


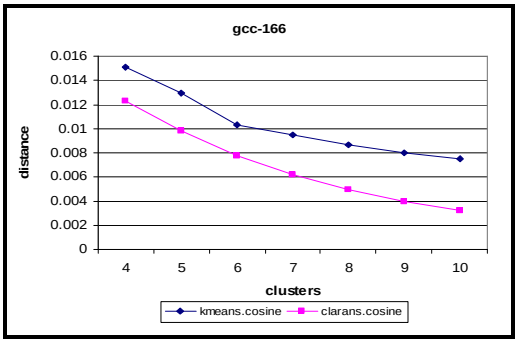


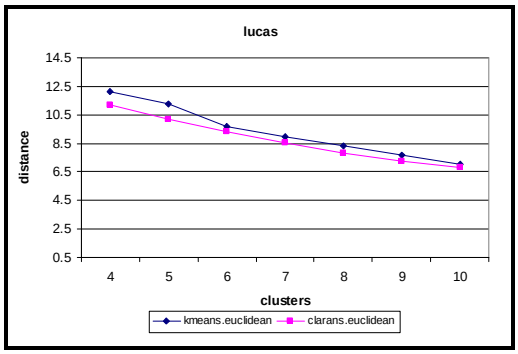


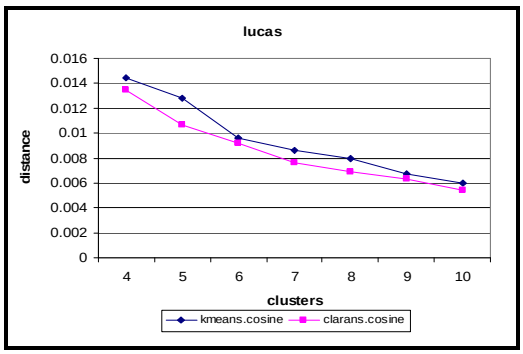


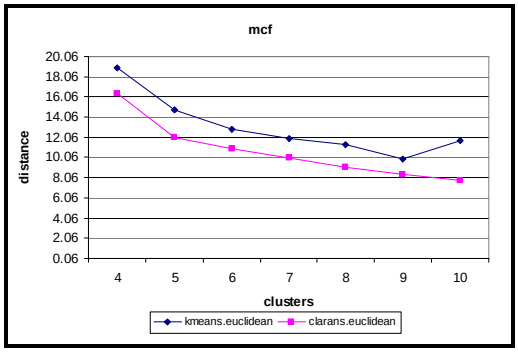


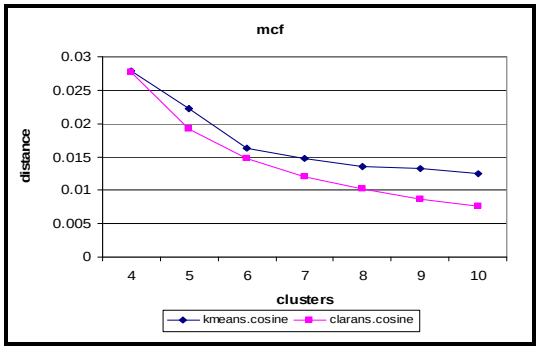


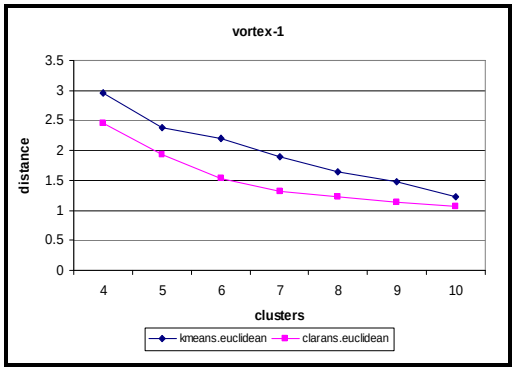


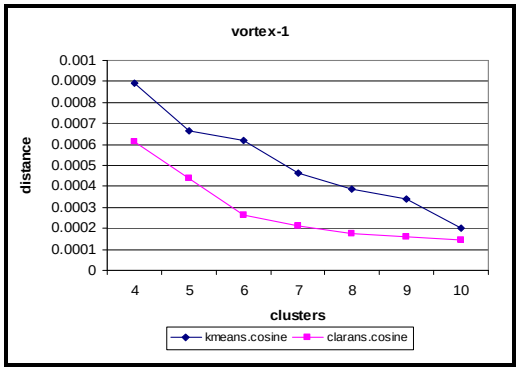

lucas
distance
14.5
12.5
10.5
8.5
6.5
4.5
2.5
0.5
4 5 6 7 8 9 10
clusters
kmeans.euclidean
clarans.euclidean

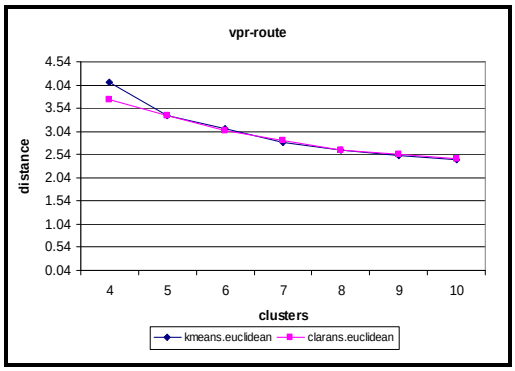

lucas
distance
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
4 5 6 7 8 9 10
clusters
kmeans.cosine
clarans.cosine

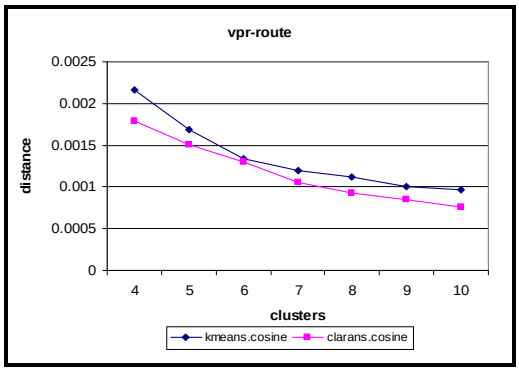

mcf
distance
20.06
18.06
16.06
14.06
12.06
10.06
8.06
6.06
4.06
2.06
0.06
4 5 6 7 8 9 10
clusters
kmeans.euclidean
clarans.euclidean


mcf
distance
0.03
0.025
0.02
0.015
0.01
0.005
0
4 5 6 7 8 9 10
clusters
kmeans.cosine
clarans.cosine


vortex-1
distance
3.5
3
2.5
2
1.5
1
0.5
0
4 5 6 7 8 9 10
clusters
kmeans.euclidean
clarans.euclidean


vortex-1
distance
0.001
0.0009
0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
4 5 6 7 8 9 10
clusters
kmeans.cosine
clarans.cosine


vpr-route
distance
4.54
4.04
3.54
3.04
2.54
2.04
1.54
1.04
0.54
0.04
4 5 6 7 8 9 10
clusters
kmeans.euclidean
clarans.euclidean


vpr-route
distance
0.0025
0.002
0.0015
0.001
0.0005
0
4 5 6 7 8 9 10
clusters
kmeans.cosine
clarans.cosine