Performance Evaluation 164 (2024) 102410

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Accelerating multi-tier storage cache simulations using knee detection

Tyler Estro [a,*], Mário Antunes [b], Pranav Bhandari [c], Anshul Gandhi [a], Geoff Kuenning [e], Yifei Liu [a], Carl Waldspurger [f], Avani Wildani [c,d], Erez Zadok [a]

[a] *Stony Brook University, Computer Science Building, Engineering Dr, Stony Brook, 11794, NY, USA*
[b] *Instituto de Telecomunicações, Universidade de Aveiro, Campus Universitário de Santiago, Aveiro, 3810-193, Portugal*
[c] *Emory University, Mathematics & Science Center, Suite W401, 400 Dowman Drive, Atlanta, 30322, GA, USA*
[d] *Cloudflare, 101 Townshend, San Francisco, 94107, CA, USA*
[e] *Harvey Mudd College, Department of Computer Science, 301 Platt Boulevard, Claremont, 91711, CA, USA*
[f] *Carl Waldspurger Consulting, 517 Georgia Avenue, Palo Alto, 94306, CA, USA*

ARTICLE INFO

Keywords:
Multi-tier caching
Miss ratio curve
Knee detection
Cache simulation
Evolutionary algorithms

ABSTRACT

Storage cache hierarchies include diverse topologies, assorted parameters and policies, and devices with varied performance characteristics. Simulation enables efficient exploration of their configuration space while avoiding expensive physical experiments. Miss Ratio Curves (MRCs) efficiently characterize the performance of a cache over a range of cache sizes, revealing "key points" for cache simulation, such as knees in the curve that immediately follow sharp cliffs. Unfortunately, there are no automated techniques for efficiently finding key points in MRCs, and the cross-application of existing knee-detection algorithms yields inaccurate results.

We present a multi-stage framework that identifies key points in *any* MRC, for both stack-based (*e.g.*, LRU) and more sophisticated eviction algorithms (*e.g.*, ARC). Our approach quickly locates candidates using efficient hash-based sampling, curve simplification, knee detection, and novel post-processing filters. We introduce *Z-Method*, a new multi-knee detection algorithm that employs statistical outlier detection to choose promising points robustly and efficiently.

We evaluated our framework against seven other knee-detection algorithms, identifying key points in multi-tier MRCs with both ARC and LRU policies for 106 diverse real-world workloads. Compared to naïve approaches, our framework reduced the total number of points needed to accurately identify the best two-tier cache hierarchies by an average factor of approximately 5.5× for ARC and 7.7× for LRU.

We also show how our framework can be used to seed the initial population for evolutionary algorithms. We ran 32,616 experiments requiring over three million cache simulations, on 151 samples, from three datasets, using a diverse set of population initialization techniques, evolutionary algorithms, knee-detection algorithms, cache replacement algorithms, and stopping criteria. Our results showed an overall acceleration rate of 34% across all configurations.

1. Introduction

A cache's miss ratio is one of the most important predictors of its performance. A miss-ratio curve (MRC) for a given cache and replacement algorithm plots the cumulative miss ratio for all accesses as a function of the cache size, providing a powerful tool

* Corresponding author.
E-mail address: testro@cs.stonybrook.edu (T. Estro).

https://doi.org/10.1016/j.peva.2024.102410

Available online 27 February 2024

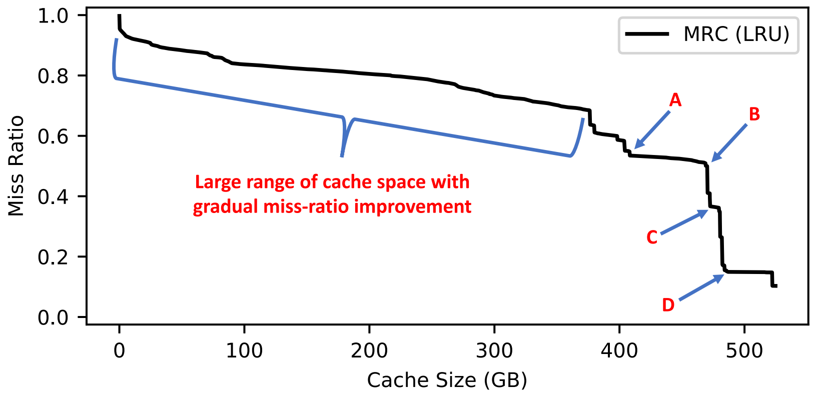


Fig. 1. MRC for trace w10, annotated to illustrate several key points: useful "knees" (points A, C, and D), a useless "cliff" (B), and a large range of cache sizes with relatively gradual miss-ratio improvement.

for analyzing the performance of live systems and dynamically adjusting cache configurations as workload conditions change [1,2]. MRCs can also inform offline evaluations such as comparing caching algorithms or analyzing monetary cost vs. storage-system performance [3].

There are many efficient techniques for generating MRCs [1,4–10]. MRC-reported miss ratios are good indicators of expected performance (*e.g.*, throughput), but real system performance can vary due to additional factors including device characteristics, write policies, and admission policies [3]. Alas, repeatedly reconfiguring and testing a real caching system, with all possible cache sizes, is prohibitively expensive due to the slowness of storage I/O.

Since experimenting with physical devices is costly and time-consuming, simulation offers a more practical way to explore this large search space and evaluate trade-offs such as latency vs. cost. A common first step is to sample a workload: approximation algorithms enable accurate simulation of cache behavior using only a fraction of the original trace data. Small sampled traces can then be used to construct an MRC accurately, enabling quick evaluation of cache performance [1,7]. Many storage-cache simulators have been developed that replay traces while attempting to faithfully reproduce real system behavior [11–13]. However, even simulations can be too expensive to allow exploring a large number of configurations or optimizing live systems in real time. For example, consider a cache with a maximum size of 100 GB. Simulating every 1 GB size step would require 100 experiments. In a multi-tier setup, the number of simulations grows with the number of tiers; a two-tier configuration would require 100^2 experiments, three tiers would need 100^3, and so on. Thus, it is essential to explore this vast configuration space efficiently.

Creating an MRC requires a sequence of cache references. In a multi-tier cache, references to level $n+1$ come from misses in – and write evictions and flushes from – level n; thus the MRC for $n+1$ directly depends on the cache size chosen for level n. A naïve exploration of *multi-tier* configurations would require a separate simulation for each point in level n's MRC to identify misses that become references at level $n+1$, and hence to compute the level $n+1$ MRC. Since an MRC may contain hundreds of points (one for each potential cache size), this approach quickly becomes intractable. Thus, a crucial second step for evaluating multi-tier caches is to limit the number of simulations by intelligently selecting the cache sizes to evaluate at each level.

Intuitively, the most promising candidates are points where a little extra cache space produces a relatively large drop in the miss ratio; such points are often visible as "knees" in MRCs—*e.g.*, points A, C, and D in Fig. 1. (Note that although B has sharp curvature, it is not of interest since C provides a much lower miss rate.) Given enough computational resources, we may be interested in also selecting some points in the large gradually sloping regions that cover a significant range of cache sizes. We refer to both types of points as *key points* from here on.

In this article we describe a multi-stage framework designed to pick an appropriate yet small number of key points in MRCs: **(1)** We first approximate the MRC accurately using a hash-based sampling technique [1,7]; **(2)** Next, we use the Ramer–Douglas–Peucker (RDP) line simplification algorithm [14] to reduce noise by eliminating minor variations in the curve; **(3)** We then run a multi-knee detection algorithm on the remaining points to find cache sizes that provide the greatest miss-ratio improvement for the lowest cost. Our framework currently implements eight different knee-detection algorithms, including our novel Z-Method and modified, multi-knee versions of five widely used single-knee detection algorithms [15–18]; and **(4)** Finally, in post-processing we remove less interesting points, add points in gradually-sloped regions (if desired), and then select the final points based on a ranking that uses hierarchical clustering and relevance metrics.

This article is an extension of our previous work [19]: we additionally provide a more in-depth analysis of our Z-Method algorithm and we demonstrate how our framework can accelerate the optimization of multi-tier caching systems using evolutionary algorithms. This collective work makes several contributions:

1. We establish the novel methodology of using multi-knee detection to efficiently identify optimal multi-tier cache configurations;
2. We present a framework that combines several techniques to find a minimal number of key points in MRCs for both stack and non-stack caching algorithms;

3. We introduce Z-Method, a new multi-knee detection algorithm that uses statistical outlier detection;
4. We demonstrate that, compared to naïve approaches, our framework significantly reduces the number of 2-tier cache evaluations needed to identify good configurations by a factor of 5.5× for ARC and 7.7× for LRU;
5. We evaluate our framework for the additional application of seeding the initial population of evolutionary algorithms. Our results show an overall acceleration rate of 34% across a highly diverse set of configurations and datasets; and
6. We release the code library containing all techniques used in this work [19].

The next section provides some background on MRCs, knee-detection algorithms, and MRC cliff removal techniques. Section 3 presents the point-selection techniques used in our framework, leading to the design of the Z-Method algorithm in Section 4. We evaluate all of our techniques ability to find key points in MRCs in Section 5. We then present an additional evaluation of how our framework can be used to optimize multi-tier caching systems using evolutionary algorithms in Section 6. Finally, we summarize our conclusions and highlights in Section 7.

2. Background

2.1. Miss ratio curves (MRCs)

A key feature of some MRCs is their monotonicity. Cache replacement algorithms such as LRU are stack-based, which means they satisfy the *cache-inclusion property:* the content of a cache of size n is always a subset of a cache of size $n+1$. Ultimately, this property ensures that the miss ratio will never degrade as we increase the size of the cache, producing a monotonically decreasing curve. However, more sophisticated algorithms such as ARC [20] are not stack-based and thus the inclusion property does not hold, causing them to produce MRCs that may contain both convex and concave regions [1]. Thus, non-stack-based MRCs need not be strictly decreasing.

2.2. Knee-detection algorithms

Many heuristic algorithms have been developed that find a single knee in a curve, although the precise definition of a "knee" varies. One can define a knee point as the point with the maximum curvature in a function. For continuous functions, curvature [21] is mathematically defined as follows:

$$K_{f(x)} = \frac{f''(x)}{(1+f'(x)^2)^{\frac{3}{2}}} \tag{1}$$

However, knee-detection algorithms are applied to discrete sets of points, instead of a well-defined continuous function. As such, there are several methods to measure the curvature of the discrete sequence. Menger curvature [16,18] defines the curvature for a sequence of three points as the curvature of the circle circumscribed by those points. This method relies only on a local criterion, using only three points to estimate the knee point without considering any others. As such, noisy data can lead to poor accuracy when estimating the knee point. We use this method as a baseline reference to compare with other methods.

The L-method [17] fits two straight lines from the head of a curve to a candidate point, and from the candidate point to the curve's tail. The candidate that minimizes the Root Mean Squared Error (RMSE) between the straight lines and the points of the curve is returned as the knee point; this represents the sharpest angle in the curve.

Similar to the L-method, Dynamic First Derivative Thresholding (DFDT) [15,22] tries to identify the point where the function has a sharp angle. Instead of fitting two straight lines, this method relies on the first derivative of the curve. After computing that derivative, a thresholding algorithm is used to identify the value that separates the derivative values as "high" or "low". The knee is then the point with a derivative value that is closest to the previously computed threshold.

Kneedle [16] uses the point on the curve that is furthest away from a line defined by the head and tail points of the curve. Both axes of the original curve are normalized to $[0,1]$ to easily find the point with maximum curvature. Kneedle was designed for single or multi-knee detection in a streaming scenario where new data is arriving continuously. The authors used this technique to detect relevant points for network congestion control and latency.

There are several algorithms that can find multiple knee points in a curve, but they have limitations that make them unsuitable for MRCs. The Kneedle algorithm's primary use case is anomaly detection, where it serves as an initial filter to reduce the number of candidates needing further analysis. As such, for Kneedle, recall is more important than precision: it aggressively captures all anomalies, producing many false positives. In some cases it is possible to reduce the number of false positives, but doing so requires extensive tuning of its sensitivity parameter.

A few multi-knee detection algorithms have been developed for use in multi-objective optimization problems, where the notion of a knee guides the exploration of meaningful candidate solutions [23]. However, these problems use a stricter definition of a knee that assumes a set of well-behaved, Pareto-optimal points. Several other knee-detection methods [15,17,18,22,24] are only effective at finding a *single* knee in a small and relatively smooth set of points. In contrast, MRCs can consist of a relatively large number of points, can be noisy or non-monotonic, and commonly contain more than one significant knee. In this work, we had to develop techniques to overcome these limitations (see Section 3) by enabling these algorithms to find multiple knee points.

2.3. Cliff removal techniques

An alternative to detecting knees in an MRC is to modify the underlying cache-replacement policy so that it does not have any cliffs, yielding a *convex* MRC. Talus [25] removes cache performance cliffs by dividing the cache into two *shadow partitions*, each receiving a fraction of the input load. Varying the sizes and input loads of each partition emulates the behavior of smaller or larger caches. Given an MRC as input, Talus computes the partition sizes and their respective input fractions to ensure that their combined aggregate miss ratio lies on the convex hull of the original MRC. Originally proposed for processor caches, Talus inspired the SLIDE [1] technique for removing performance cliffs from software caches that employ sophisticated non-LRU replacement policies. CliffHanger [26] applied a similar idea to key–value web caches, but instead estimated the MRC gradient without explicitly constructing one.

The recent eMRC [27] technique generalized Talus's cliff removal to multi-dimensional *miss ratio functions*, such as the three-dimensional miss-ratio surface for a two-tier cache. The eMRC convex-hull approximation technique leverages the absence of cliffs to efficiently generate the miss ratio function for a multi-tier cache. However, eMRC *requires* convexity, which limits its applicability to modeling multi-tier cache systems that employ cliff removal. As real-world multi-tier cache systems do not yet perform cliff removal, eMRC is unable to approximate their non-convex MRCs. In contrast, our approach does not require convexity to accelerate multi-tier cache evaluations, making it broadly applicable to production deployments of existing caches.

2.4. Evolutionary algorithms: Population initialization

The initial population of an evolutionary algorithm functions as the first guess at a set of good solutions to an optimization problem. The quality of this first set can significantly influence the quality of the final solution and the speed at which an algorithm converges [28,29]. Studies have shown that some evolutionary algorithms, such as Genetic Algorithms, are more sensitive to the initial population, while other algorithms like Particle Swarm Optimization are less dependent on the initial population [30]. This sensitivity has also been shown to be problem-dependent, such that an algorithm may be influenced by the initial population for certain functions, numbers of dimensions, or population sizes [28].

There are several categories of population-initialization techniques. Common stochastic variants, such as random initialization, are favored for their simplicity, generic nature, and applicability to a wide range of problems [31]. They are also popular because they produce a relatively uniform distribution as the population size increases. But statistical methods such as Latin Hypercube sampling [32] and quasi-random sequences (*e.g.*, Halton sequence [33]) have been shown to outperform random initialization for most problems [34].

There are also application-specific techniques aimed at specific, narrow problems. They often exploit domain knowledge or use a problem's characteristics with specific evolutionary algorithms. These methods have been applied to improve convergence speed in problems such as grammar-guided genetic programming [35] and flexible job-shop scheduling [36]. Initialization techniques in this class typically perform better than more generic variants, but they are usually limited to specific problems.

Lastly, there are domain-agnostic, general heuristics applicable to problems that meet some set of conditions. Examples include approaches developed to optimize any two-stage stochastic mixed-integer problem [37]. Such techniques can be applied to any problem where some of the variables are constrained to integer values.

The techniques described in this article focus on a domain-agnostic, generic approach that can be applied to any problem where a curve that is correlated to the solution can be derived from features of the input data. We show how such techniques can be used on miss ratio curves to optimize a cache with evolutionary algorithms.

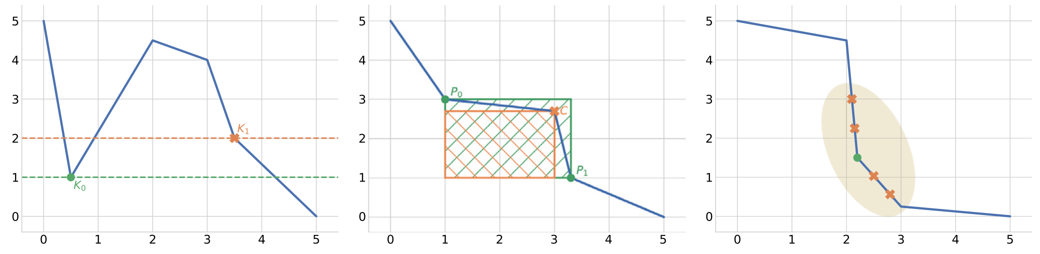
3. Point selection techniques

In this section, we introduce our framework for finding multiple key points in MRCs. We first designed a pre-processing stage to deal with the large volume of data (Section 3.1). Next, we made substantial modifications to each point-selection technique, enabling them to output a set of multiple knees instead of just one (Section 3.2). Finally, we added a post-processing stage (Section 3.3) that filters and ranks knees based on an appropriate definition.

3.1. Pre-processing

A curve can contain an arbitrary number of data points. The largest MRC that we evaluated contains 276 K points even after sampling-based size reduction [1]; the original MRC is 10,000× larger. However, the knee-detection algorithms evaluated in this work were originally designed to work with small or partial data, such as for clustering optimizations. Our main idea is to reduce the number of points while preserving those that follow the shape of the curve; this greatly reduces the computational costs of subsequent steps while also improving knee-detection accuracy by minimizing irrelevant fine-grained variation.

The Ramer-Douglas-Peucker (RDP) algorithm modifies a curve by finding a similar one with fewer points [14]. RDP fits a line between the curve's endpoints and then finds the point in between that is farthest from this line. If the distance between that point and the line is over a given threshold, the curve is split there and the algorithm is reapplied recursively on the two new segments. Once the distance is smaller than the threshold, *all* intermediate points are removed. The main drawback of RDP is the need to define a threshold, which can be understood as the maximum allowed reconstruction error. The choice of threshold is difficult because it depends on the curve's complexity.



(a) Undesirable knee filter. (b) Cliff-point removal filter. (c) Knee clustering and ranking.

Fig. 2. Graphical representation of the post-processing methods. (a) Representation of the filter that removes unwanted knees, knee K_1 is removed since knee K_0 achieved better performance. (b) Representation of the overlapping rectangles used by the corner detection algorithm. Point C is identified as a cliff point and then removed. (c) Representation of the clustering and ranking elements. The orange ellipse represents the cluster of knee candidates. The orange candidates are filtered out, while the green knee is selected as the best representative based on Eq. (2).

We modified the original RDP algorithm to address this difficulty. Instead of defining a threshold for the maximum allowed perpendicular distance between a point and the fitted straight line, we use a relevance-based cost metric that computes the difference between the fitted straight line and the data points in the current segment.

We evaluated four different metrics that assess how far our linear reduction is from the original data: Root Mean Squared Log Error (RMSLE), Root Mean Squared Percentage Error (RMSPE), Relative Percent Difference (RPD), and symmetric mean absolute percentage error (SMAPE). Of these four, the best performance came from SMAPE: it found the smallest set of points that minimized the reconstruction error.

3.2. Methods

Except for Kneedle, the algorithms we evaluate in this work (see Section 2.2) were not designed to detect multiple knees. Thus, we developed a recursive algorithm that can be used to adapt any single-knee detection technique to handle multiple knees. The basic idea is to use a single-knee technique to select the best knee in a segment. We then split the current segment at that knee, and for each new segment check whether it is sufficiently linear (computed using the SMAPE metric). If not, we repeat the process recursively. Apart from applying this recursive generalization, we do not alter the core knee-detection technique, using it as a black box. All of the methods we evaluated, even Kneedle, require our pre- and post-processing methods to work properly on MRCs.

3.3. Post-processing

Given the differences between single- and multi-knee detection and the large number of points produced by using our recursive strategy on some of the knee-detection algorithms, we developed three different filters to further reduce and select the most relevant knees.

The first filter, shown in Fig. 2(a), removes useless knees. When dealing with non-monotonic curves, a knee-detection algorithm can incorrectly choose a knee that is *above* a previously detected one. We remove such knees since they are sub-optimal and do not add useful information.

The second filter, shown in Fig. 2(b), removes cliff points located after a smooth, near-horizontal area that precedes a sharp descent. These points are found using a corner-detection algorithm that computes the overlapping area of two rectangles. The first rectangle, shown in green, is drawn from the neighbor points P_0 and P_1 (assuming that RDP pre-processing was used, these are the previous and following points) that are adjacent to the knee candidate point C. The second rectangle, drawn in orange, has its corners placed at C and the lower left of the green rectangle. The filter computes the percentage overlap between these two rectangles, and a knee candidate is removed if the overlap exceeds a threshold.

The third and final filter, shown in Fig. 2(c), uses a hierarchical clustering algorithm to group knees by their distance along the x-axis, using a percentage of the x range as a threshold. After grouping the knees into clusters, the knees within each cluster are ranked based on their relevance score, computed from two metrics: (i) the improvement given by each knee (*i.e.*, how much it decreases on the y-axis from the highest knee in the cluster) and (ii) the smoothness of the improvement, computed using the coefficient of determination (R^2). Specifically, the relevance score S is given by Eq. (2):

$$S(K_i, \vec{L}) = |K_h - K_i| \cdot R^2(\vec{L}), \tag{2}$$

where K_i is the ith knee, and K_h is the knee with the highest value on the y-axis (in a single cluster). $\vec{L}$ is a vector containing all the knees in the cluster up to and including the ith one: $\vec{L} = [K_0, \dots, K_i]$. The highest-ranked knee in each cluster is selected as its representative knee.

4. Z-method

4.1. Design concepts

Our design for Z-Method was inspired by the DFDT [15] and DSDT [22] knee-detection algorithms, which use first and second derivatives, respectively. In statistics, a *z-score* (also known as a *standard score*) is a transformation that normalizes a data value by quantifying how many standard deviations away it is from the mean; typically, a point whose z-score has an absolute value greater than three is considered an outlier [38]. For the purpose of detecting knees, such outliers in the second derivative indicate a significant change in the y-axis. The foundation of our Z-Method technique is in detecting such outliers and intelligently selecting knees among them.

Although the second derivative is useful, we found that large and small knees often tend to cluster, causing several points in close proximity to be selected, rather than the single most optimal knee in the vicinity. To remedy this, we introduced two hyper-parameters, dx and dy, that specify the minimum x and y distances, respectively, between all selected points. These parameters limit the total number of knees selected and give users control over the algorithm. For example, users interested only in large knees can give relatively high values for dx and dy to minimize the number of points.

Z-Method was designed to function independently of the techniques described in Sections 3.1 and 3.3. As such, it works for curves that are non-monotonic, with both convex and concave regions (see Section 2.1).

4.2. Algorithm description

Algorithm 1: Z-Method Multi-Knee Detection

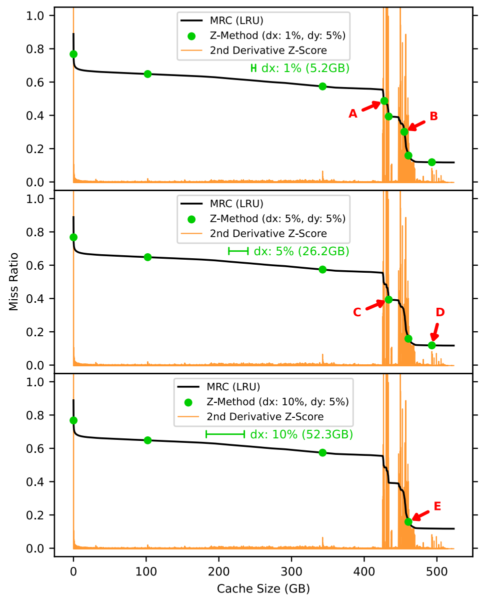
```
Input: Data $D$ with $(x, y)$ points, $dx, dy, dz$
Output: List of $(x, y)$ points corresponding to knees
1  $\Delta x \leftarrow$ length($D$) $\cdot$ ($dx/100$)
2  $\Delta y \leftarrow$ (max($y$) $-$ min($y$)) $\cdot$ ($dy$ / 100)
3  $D'' \leftarrow$ calculate second derivative of $D$
4  $Z \leftarrow$ calculate z-scores for $D''$
5  $K \leftarrow$ empty list
6  $zLimit \leftarrow 3$        # standard outlier threshold [38]
7  while TRUE do
8      $C \leftarrow$ points in $Z$ with z-score $\geq zLimit$
9              and at least $\Delta x$ and $\Delta y$ apart from all points in $K$
10     $Z \leftarrow Z$ - $C$
11     if $zLimit \leq 0$ and length($C$) == 0 then
12         Remove points from $K$ to ensure that $y$ always decreases as $x$ increases
13         return $K$
14     $G \leftarrow$ group all points in $C$ such that all adjacent points in each group are $< \Delta x$ apart
15     sort $G$ in descending order by max z-score of each group
16     foreach group in $G$ do
17         $p \leftarrow$ point in group with the lowest $y$ value
18         if $p$ is at least $\Delta y$ from all points in $K$ then
19             $K$.append($p$)
20     $zLimit \leftarrow zLimit - dz$
```

As shown in Algorithm 1, Z-Method takes as input a discrete curve D consisting of an ordered list of (x, y) points, along with parameters dx, dy, and dz. The parameters dx and dy both influence the size and number of selected knees, while dz controls the maximum number of iterations in the main selection loop (lines 7–20).

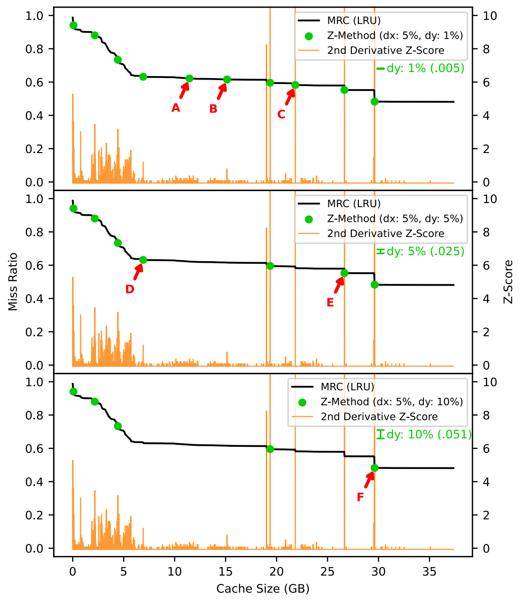
We first convert dx and dy, specified as percentages, into absolute values Δx and Δy for the input curve (lines 1–2). This normalization ensures that these parameters function similarly for different curves. We then approximate a list of second derivatives of the curve, D'', using second-order polynomial fitting [39]; next we calculate the z-scores of all points in D'' as Z, both of which are found in linear time (lines 3–4). We initialize a list K to contain all selected knees, and set our starting value of $zLimit$ to 3, since a z-score ≥ 3 is a widely accepted value for outliers [38] (lines 5–6).

We then enter the main selection loop (lines 7–20), which selects points and progressively decrements the $zLimit$ value. First, we create a new list C that contains candidate points: points in Z that have a z-score greater than the current $zLimit$ and are at least a minimum Δx and Δy distance from all other already-selected points (lines 8–9). The complexity of this step is $O(|C| \times |K|)$. All candidate points C are removed from Z so that we will not consider them again in future iterations (line 10). The termination clause is then checked (lines 11–13) to ensure that we have candidate points to operate on.

We next group the candidate points C into G, such that the adjacent points in each group are less than Δx apart, based on the dx parameter constraint (line 14). This takes $O(|C|)$ time, effectively forming groups of points such that there is at least Δx distance



(a) Effects of Z-Method parameter *dx*.



(b) Effects of Z-Method parameter *dy*.

Fig. 3. (a) Effects of $dx = 1\%$ (top panel), 5%, and 10% on trace w09. Red arrows denote points in a panel that were not selected when the dx value increased (next panel down). (b) Effects of $dy = 1\%$ (top panel), 5%, and 10% on trace w62. Red arrows denote points in a panel that were not selected when the dy value increased (next panel down). The minimum distance that Z-Method enforces between points is shown below the legends in green.

between every group. We then sort the groups in G in descending order by the maximum z-score of each group (line 15). Here, we are sorting the location of each group in the list of groups G rather than the points within each group. From each group, we select the point with the lowest y value (line 17); we then check that the selected point is not within a minimum Δy distance from other points that have already been selected, enforcing the dy parameter (line 18). The complexity of this loop is $O(|G| \times |K|)$. A point is added to the list of knees K if it satisfies this constraint (line 19). We then decrement the $zLimit$ by dz and continue with the next loop iteration (line 20).

This loop terminates only after we have reached a $zLimit \leq 0$ and there are no remaining points that can be selected given the dx and dy parameters (line 11). At $zLimit = 0$, we consider all points in D that have not already been considered in previous iterations. By starting at z-score ≥ 3 and iteratively approaching 0, we select the largest knees first and gradually lower our threshold for how big a knee should be.

Finally, we eliminate any points that may have been poorly selected due to non-monotonicity in the curve. A final pass removes points where increasing the x value makes the y value worse (line 12); in our MRC application, such points are clearly undesirable. This simple pass requires time linear in the size of K. The overall complexity of this algorithm is therefore $\mathcal{O}(|D| \times |K|)$, where D is the size of the input curve and K is often a trivially small value. For example, with dx set to 5%, enforcing at least 5% distance on the x-axis between each selected knee point, the maximum size of K would be 20.

4.3. Parameters

We present qualitative evaluations of the algorithm's parameters dx and dy, as well as its overall success at finding *key points*. Furthermore, we demonstrate that Z-Method is effective for both stack and non-stack algorithms, by evaluating with LRU and ARC cache replacement policies.

Parameter: dx. The dx parameter has several functions within Z-Method. It is provided as a percentage of the maximum cache size in the given MRC. The most transparent effect of dx is that it constrains the minimum x distance, or cache size, between selected points. Since no two points can have an x distance less than dx between them, this provides an upper bound on the total number of selected points, and also influences the number of points that are actually selected. Because it affects the "grouping" stage of the algorithm, dx also effectively defines the width of the knees.

Fig. 3(a) shows the effects of dx on workload w09 with LRU cache replacement, with dy fixed and dx set to 1%, 5%, and 10%. The black line in each plot represents the MRC for LRU cache replacement. The green dots are the points selected by Z-Method when using the dx and dy parameters indicated in the legend. The vertical orange lines show the second derivative z-score of the MRC at each cache size. Because the z-score values have a large, workload-dependent dynamic range, we truncate them at 10 in this plot and for the remainder of this article. A z-score range up to 10 is sufficient to identify all points considered as outliers (*e.g.*, z-score $\geq$ 3).

For the MRC plotted in Fig. 3(a), we will focus on the knee(s) in the region of cache sizes between approximately 425 GB and 475 GB. In the top plot with $dx = 1\%$, Z-Method considers this region to contain four separate knees, since their distances from each other are at least 1% of the maximum cache size. When we move from 1% to 5% in the middle plot, we can see that points A and B from the top plot have been removed. Those points are no longer within dx of each other, so they are grouped together; we are now left with two points at wider, more prominent, knees.

A similar effect is seen when we increase dx from 5% to 10% in the bottom plot. The two knees at points C and E are grouped together and C is removed. Point D is also removed, since its cache size is less than 10% away from point E. Significantly, the knee point E was favored rather than the less interesting point D.

Parameter: dy. The dy parameter is also specified as a relative percentage, which is then converted into an absolute value for the given MRC. It functions similarly to dx, except that it constrains the y distance, or delta miss ratio, between any two selected points. This effectively influences the height of knees and how many points are selected, while providing an upper bound on the total number of points that can be selected.

Fig. 3(b) shows the effects of dy using workload w62 with LRU cache replacement by fixing dx and varying dy between 1%, 5%, and 10%. The format is otherwise the same as in Fig. 3(a). In the top and middle plots, the most interesting change occurs at point C. With $dy = 1\%$ in the top plot, this very small knee is considered significant and is selected. However, when we increase dy from 1% to 5% in the middle plot, points A, B, and C are removed, as the y distance between these points and adjacent points is no longer less than dy. Similarly, point E is removed when we move from 5% to 10% in the bottom plot, while the taller knee point F is retained. We can also see that point D is removed as well, as increasing dy reduces the number of selected points.

It is important to note that for both of these parameters, we are not guaranteed to *always* have a point that is dx or dy apart from every other point. Enforcing a hard separation rule would add a great deal of complexity and would provide little benefit, since we already select points by their order of importance.

Parameter: dz. The dz parameter controls the amount that the $zLimit$ variable is decremented in each iteration of Z-Method. This affects the overall running time by influencing the total number of iterations. It can also affect the size of candidate point groups. For example, with a starting $zLimit$ of 3, $dz = 0.1$ would only consider points with a z-score ≥ 2.9 on the second iteration, but $dz = 0.5$ would consider a potentially larger set of points that have a z-score ≥ 2.5. While this may seem significant, the dx and dy parameters are still the predominant influence on how groups are formed, so we did not observe any trends or significant changes when modifying dz.

Finding key points. In Fig. 4, we show the points selected by Z-Method with dx and dy set to 5%, for multiple workloads using both LRU and ARC cache replacement policies. We evaluated these plots based on whether or not they selected all of the points that we consider *key points*. To reiterate, Z-Method should first select the largest knee points and then eventually select those within any regions that cover at least 5% of the x and y axes. The first row of plots (LRU1-3) shows examples where Z-Method performed well for LRU. All prominent knees were selected and large ranges of cache space with gradual decreases in miss ratio also contained an adequate number of points. The second row of plots (LRU4-6) shows examples where Z-Method missed key points. For example, in plot LRU4, points A and B missed the knee points directly to their left. There were similar issues in LRU5 and LRU6.

We show how to improve the lower-quality points A and B in LRU4 by modifying Z-Method parameters. The green points were selected using the default dx of 5%, while the blue points were selected using a dx of 3.2%. These blue points more accurately capture these two knees, and are more optimal than A and B.

There were also cases where Z-Method could pick slightly less optimal points due to extreme shapes in a curve and the nature of the z-score metric. This can be observed in plot LRU5, which exhibits a nearly flat region followed by a massive, steep knee, then another nearly flat region. Point C is not quite at the bottom of the knee because the z-score at the very bottom was slightly below the standard threshold for an outlier of 3. This could be remedied by lowering the threshold (and modifying Z-Method's parameters if needed). It should be noted that Z-Method will still pick a point that is relatively close to the knee in these edge cases.

The third row of plots (ARC1-3) shows where Z-Method performed well for ARC. In addition to always selecting prominent knees and points in gradually sloped regions, we also see that points were never selected in concave regions where the miss ratio increased due to the non-monotonicity of ARC. A key feature of Z-Method is that it will never select points with a higher miss ratio that any other previously selected points with a lower cache size.

The fourth row of plots (ARC4-6) depicts a situation where Z-Method missed key points. In cases such as ARC5, modifying the parameters was sufficient for identifying higher quality points, but plots ARC4 and ARC6 exhibit a problem that is unique to non-stack-based cache eviction algorithms (*i.e.*, ARC). Unlike stack-based algorithms (*e.g.*, LRU), we cannot easily generate a fine-grained MRC that includes every potential cache size. Instead, best practice is to sample the workload [1] and then generate the MRC using a subset of points that still preserves the shape of the curve. This is typically done using 100 points. We used 100 points to generate all ARC MRCs during the point selection process throughput this work, but plotted the z-score and points selected against MRCs generated using 1000 points to better show how Z-Method selects points in MRCs that more closely represent the "true" curves. This value worked well for the majority of our workloads, but there were edge cases (*e.g.*, ARC4 and ARC6) where the z-score did not accurately capture important knees present at the very end of the curve. For example, point D in ARC4 was selected because there was a very small knee with a positive z-score at the top of the cliff, but there was a much larger knee to its immediate right. This could be remedied by increasing the number of points used to generate the MRC. 220 points were enough to remedy this value for both ARC4 and ARC6 (not shown in the plots due to complexity). The task of generating an MRC is separate from Z-Method, so we did not include this value as a parameter.

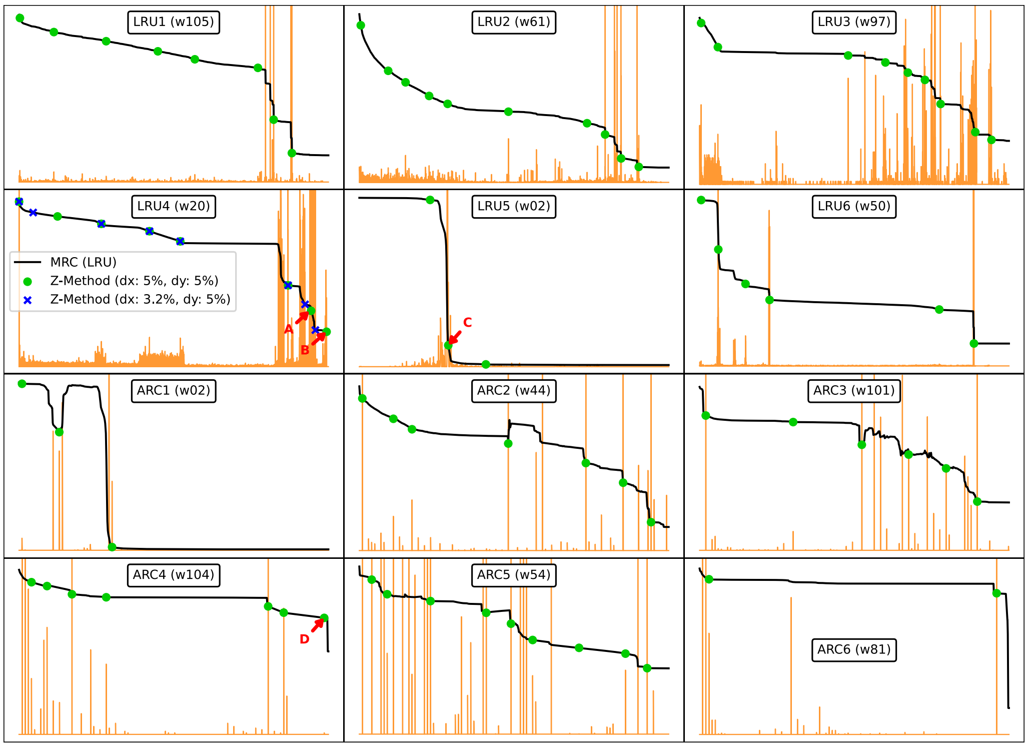


Fig. 4. From the top: the first row shows LRU plots where Z-Method picked fairly good points; the second row has LRU plots where Z-method missed a few, better points. The third and fourth rows are the same but for ARC (third row good points selected; fourth row missed some points). The plot labeled LRU4 shows points selected by Z-Method using a dx of both 5% (green) and 3.2% (blue). Sub-optimal points A and B in LRU4 were selected using dx of 5%, missing nearby knee points. The knees were appropriately selected when dx was lowered to 3.2%. Sub-optimal point C in LRU5 was selected due to the extreme shape of the curve and the standard z-score threshold of 3. The knee was appropriately selected when lowering the threshold. Sub-optimal point D in ARC4 was selected due to the ARC MRC being generated with too few points. The knee point to the immediate right was selected when the number of points in the MRC was increased from 100 to 220.

In all cases where Z-Method missed key points, there were slight modifications that enabled those points to be selected. We fixed dx and dy to 5%, the z-score outlier threshold to 3, and generated ARC MRCs using 100 points for this evaluation, since we do not yet have a way of automatically selecting the ideal values. Even so, the overwhelming majority of MRCs we looked at still found all key points with these values.

5. Evaluation: Miss ratio curves

In this section, we evaluate our framework's ability to find key points in miss ratio curves. We first compared the accuracy of 8 different knee-detection algorithms, including Z-Method, for identifying knees in single-tier MRCs, and then evaluated our framework's ability to quickly find optimal multi-tier configurations.

5.1. Experimental setup

We evaluated our techniques on 106 real-world block traces collected by CloudPhysics [7], each representing a week of virtual disk activity from production VMware environments. We used hash-based spatial sampling [1,7], with a size-based sampling rate ranging from 0.1 to 0.0001, to reduce these workloads and thus the running time while maintaining an accurate representation of the originals. We dynamically varied the rate by powers of 10, such that each sampled trace was guaranteed to contain between 100K and 1M requests. The traces contain heterogeneous request sizes, so we also transformed all requests into 4KB block-aligned operations to facilitate accurate sampling, consistent with previous work [27].

To evaluate multi-tier systems, we extended PyMimircache [13], a cache simulator with an easily modifiable Python front end and an efficient C back end. Our extension generates two-tier MRCs by simulating an L1 cache with the original sampled trace, then simulating L2 with the requests that missed in L1. L1 cache sizes were selected using the MRC of the original trace, while L2 sizes were chosen using the MRCs of each intermediate trace. The total miss ratio of the two-tier configuration was calculated as the product of the miss ratios of L1 and L2. We modeled a simple write-back policy by treating both reads and writes as cache references, as done in previous work [27]. The cache eviction policy was configured as either LRU or ARC and was the same in both tiers. We generated two-tier MRCs for each trace, for both LRU and ARC replacement policies, resulting in a total of 212 MRCs.

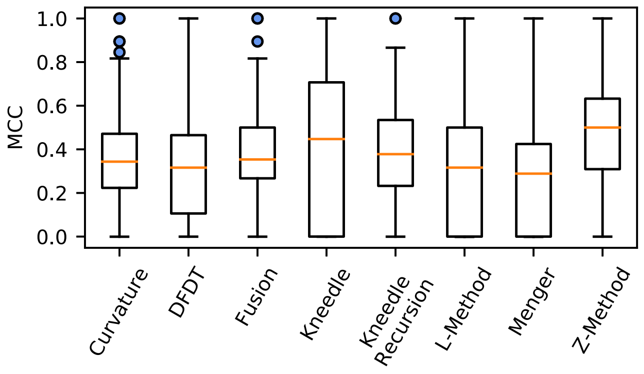


Fig. 5. An MCC evaluation of 8 knee detection algorithms using our optimized hyper-parameters for accurately identifying knees that were manually selected by experts. Higher MCC values and lower standard deviations are better. Kneedle and Z-Method have the highest median MCC values of 0.45 and 0.5, respectively, with Z-Method achieving much tighter bounds.

5.2. Knee-detection algorithms

We evaluated the accuracy of our framework using Z-Method and several other knee-detection algorithms: Curvature, DFDT, Kneedle, L-Method, and Menger. We also included the *Fusion* method, which considers all points retained by RDP and relies on our post-processing filters to select relevant knees.

Kneedle finds knees using peak-detection methods, and can be used for single-knee detection by selecting only the highest possible peak; we call that approach *Kneedle Recursion*. We analyzed each method's ability to find knee points that had been manually curated in the 212 single-tier MRCs by four domain experts.

Most of our techniques have one or more hyper-parameters that can influence which points are selected. To achieve the best performance for each MRC, it is necessary to tune the hyper-parameters of each algorithm appropriately. While the default parameters offered acceptable performance, a more complete evaluation requires optimized parameters [40]. Therefore, we developed a cost function and ran an optimization algorithm for each knee-detection method using all 212 MRCs.

When designing the cost function, we carefully considered the target use case of our framework. We want to find the ideal parameters that have the lowest error globally across all MRCs, while keeping the number of knees as close as possible to the number of knees identified manually. Constraining the number of knees is necessary since our framework is designed to pick only the most relevant points. A technique that finds parameters that correctly identify all knee points but also selects many non-relevant points, while having a high precision score, would be inefficient for our use case.

We use the Matthews correlation coefficient (MCC) [41] as the basis of our cost function. MCC measures classification quality by considering the balance ratios of the four confusion matrix categories: true positives and negatives, and false positives and negatives. Although the knee-detection problem is better modeled as a regression, we based our evaluation on binary classification, since we wanted to control the impact of false positives and negatives (*i.e.*, non-relevant points being classified as knees and vice-versa). Prior work [41] has shown that the MCC is more informative than an F1-score for evaluating accuracy in binary classification problems. As such, the cost function we used was the following:

$$Cost(E,K) = \frac{1}{n}\sum_{i=0}^{n} MCC(E_i,K_i) + \max\left(\left(\frac{1}{n}\sum_{i=0}^{n}|K_i|\right) - K_t, 0\right), \tag{3}$$

where E represents the expected (manually selected) knees for all MRCs, and K represents knees picked for all MRCs (n defines the number of MRCs) by our framework using some configuration of hyper-parameters and a knee-detection algorithm. $MCC(E_i,K_i)$ represents the MCC computed from the expected and detected knees of the ith MRC. Finally, $|K_i|$ represents the number of knees detected in the ith MRC, and K_t represents a threshold for the acceptable number of knees.

Fig. 5 shows our evaluation. Three techniques stand out: Fusion, Kneedle, and Z-Method. Fusion achieves tighter margins than all other techniques, spanning only 0.23 MCC between the upper and lower quartiles, suggesting that the expected performance in unseen traces would be well-bounded. Kneedle and Z-Method achieve the highest median MCC values of 0.45 and 0.50, respectively, with Z-Method having a smaller standard deviation when compared with Kneedle. The much tighter bounds of Z-Method are more significant than the improvement in median, making Z-Method the ideal candidate for our multi-tier evaluation.

We also experimentally measured the time and memory usage of our framework using each knee-detection algorithm for all MRCs. We used the default hyper-parameters for each algorithm, as they do not significantly impact the overheads. The results of the running-time benchmarks are shown in Fig. 6.

All of the algorithms have comparable execution times across all MRCs with the exception of L-method. This is especially true for the upper quartile of L-Method, which is 1307.79 ms. The second highest upper quartile is Kneedle Recursion, which is 74.4% lower than L-Method at 334.62 ms. This is expected since L-method uses straight-line fitting ($O(n^2)$) for each point to detect the ideal knee, leading to a time complexity of $O(n^3)$. Combined with our recursive algorithm that enables it to detect multiple knees,

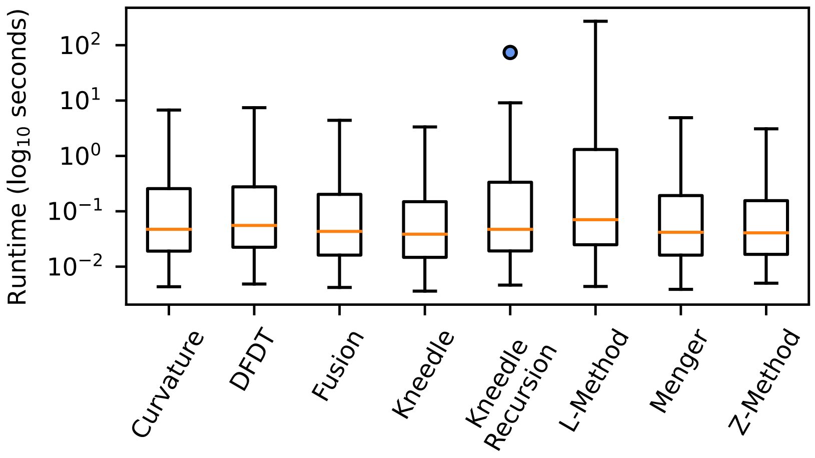


Fig. 6. Running times for 8 knee-detection algorithms. The y-axis scale is logarithmic. L-Method displays a considerably higher running time than other methods, due to its high complexity. Other methods are comparable, with Kneedle and Z-Method having the lowest median running times.

the expected time complexity for L-method is $O(n^3 \log n)$. Note that Kneedle (non-recursive version) and Z-Method have the lowest median running times of 38.59 ms and 40.81 ms, respectively.

The memory overhead is nearly linear in the file size for each MRC, ranging from approximately 53 KB to 71 MB after sampling. There were no significant differences across any of the techniques, so we do not present any further memory overhead analysis.

5.3. Multi-tier MRCs

Miss-ratio curves are typically used to find configurations that minimize both miss ratio and cache size(s). We seek multiple configurations that are optimal in two or more objectives.

Consider designing a multi-tier cache, with many device options, for a large workload. With an unlimited budget, one could simply purchase enough DRAM to hold the entire data set, but that is rarely economical. Instead, most system administrators will want to trade cost off against performance, meaning that they will be interested in *Pareto-optimal* solutions, *i.e.*, those where a given objective cannot be improved without making one or more others worse.

Only a subset of all possible cache configurations are Pareto-optimal. When a set contains every Pareto-optimal configuration for a given workload and no others, it is called the *true Pareto-optimal front*. Any point in this front minimizes the cache size(s) and the miss ratio; the front as a whole can be considered to mark the "best" points.

However, it is often not feasible to find the true Pareto front for a large configuration space. Instead, the space can be sampled in an attempt to find optimal points, creating a *Pareto approximation*. Our work aims to find a minimal number of key points in MRCs. Thus, we are trying to find the most significant Pareto-optimal points by efficiently generating accurate Pareto approximations of multi-tier MRCs.

There are multiple metrics for evaluating the quality of Pareto approximations [42]; the most commonly used is the *HVI* [43], which measures the size of the space between the points in a front and a user-defined reference point; a larger space is better.

Fig. 7 shows an example of how HVI is measured in a 3-dimensional space. The blue shape represents a simple linear series descending from (0,0,10) to (10,10,0). If this were a two-tier MRC, the x-axis would be the L1 size, y-axis the L2 size, and the z-axis the miss ratio. The hypervolume is the volume between points on the Pareto front (here, the blue shape) and a user-defined *reference point*, here the *nadir point*[1] at (10,10,10), where all objectives are maximized. To find the hypervolume of the point at (5,5,5), we draw a rectangular prism from it to the reference point. The resulting $5 \times 5 \times 5$ cube has a hypervolume of 125. If we were to instead find the hypervolume of the point (4,4,4), we would have a $6 \times 6 \times 6$ cube with a hypervolume of 216. Thus, configurations with lower cache sizes and miss ratios result in larger hypervolumes. The total hypervolume of a dataset is the non-overlapping hypervolume of all points on its Pareto front, making HVI a useful metric for our multi-knee detection framework.

Another metric highly relevant to our problem is the Ratio of Non-Dominated Individuals (RNI) [44], which is the fraction of dataset points that are on the Pareto front. As discussed earlier, points not on the front represent sub-optimal configurations, so a higher ratio is better. RNI does not measure the magnitude of quality; instead, it informs us of a point selection technique's efficiency. Therefore, evaluating HVI and RNI together is a comprehensive approach to analyzing techniques that find the minimal number of key points in MRCs.

[1] Although the reference point is placed at the largest coordinates, prior literature on hypervolume indicators uses the term "nadir" rather than "zenith" because it represents the worst performance; we follow that convention.

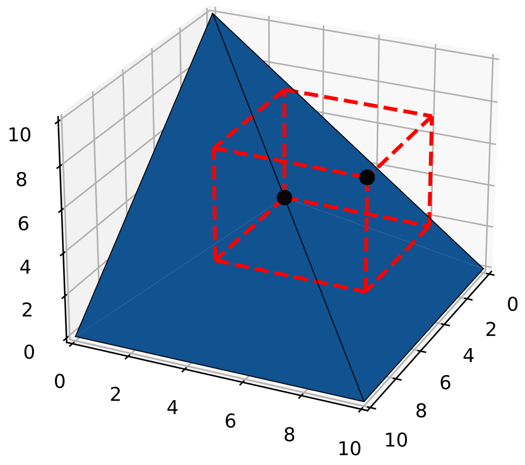


Fig. 7. An example of how the *HyperVolume Indicator (HVI)* is calculated for a single point of a 3-dimensional data series. A cube is drawn from the reference point (10, 10, 10) to the data point (5, 5, 5) creating a 5 × 5 × 5 cube with a hypervolume of 125.

Table 1
Evaluation results of our framework using Z-Method across 2-tier ARC and LRU MRCs, derived from 106 real-world block traces collected from CloudPhysics. The averages of 3 metrics are presented for each algorithm: number of points (lower is better), HyperVolume (HVI) as a percentage of Even50's HyperVolume (higher is better), and Ratio of Non-Dominated Individuals (RNI) (higher is better).

Method	Average points		Average HVI %		Average RNI	
	ARC	LRU	ARC	LRU	ARC	LRU
Even4	20	20	59.63	61.00	0.30	0.28
Even10	110	110	86.43	83.31	0.39	0.40
Even13	182	182	90.41	91.07	0.41	0.34
Even50	2550	2550	100.00	100.00	0.33	0.34
Z-Method	23.33	20.33	86.99	90.75	0.94	0.97

We evaluated our framework across all 212 two-tier MRCs using Z-Method, compared to a naïve approach of selecting evenly-spaced points. We also tried geometrically-spaced points, but this yielded worse results than even spacing so we omit them from this analysis. It was not practical to evaluate every point in MRCs containing thousands of points, so we used 50 evenly-spaced points (Even50) as a reasonable approximation of the full configuration space and the true Pareto front. We varied the number of evenly-spaced points to most closely match Z-Method's average HVI or number of points, resulting in Even4, 10, and 13.

In Table 1, we show the averages across all 212 MRCs of the number of points selected, HVI as a percentage of Even50's HVI, and RNI. When measuring the efficiency of a method, a lower number of points and a higher RNI are better; when measuring the accuracy of a method, higher HVI is better. The number of points for even spacing is always constant, calculated as $X + X^2$ for two-tier MRCs using Even$X$. Z-Method has HVI similar to that of Even10 for ARC and to Even13 for LRU, but Z-Method evaluates 5.5× fewer points for ARC and 7.7× for LRU to get those results. This efficiency is also reflected in Z-Method's RNI of 0.94 for ARC and 0.97 for LRU. Conversely, the RNI of the evenly-spaced methods ranges from 0.28 to 0.41, meaning that the majority of points they select are sub-optimal and uninteresting to explore.

In Fig. 8, we show box plots for all 212 MRCs. Fig. 8a displays the number of points selected by each technique. We can see that Z-Method and Even4 selected approximately the same median number of points. A significant result is that in the worst case, Z-Method still picked fewer points than Even10. We can also see cases where Z-Method picked very few points. There are times when such a low number of points is appropriate, but this can also represent cases where the default parameters were too conservative, resulting in too few points and a low HVI.

Fig. 8b displays the HVI as a percentage of Even50's HVI. This figure reveals several outliers where Z-Method performed poorly, but also many cases where it had a higher HVI than Even50. These results inform us about Z-Method's sensitivity to its hyper-parameters. The default parameters worked well for the majority of our workloads, but needed to be tuned better for others. With the right parameters, Z-Method performed better than naïve approaches while selecting a minimal number of points. Lastly, Fig. 8c displays the RNI. Z-Method consistently had a greater RNI than all of the evenly spaced methods, indicating that it properly identified key points. We can also see that there were diminishing returns when increasing the number of evenly spaced points. The median RNI decreased from Even13 to Even50, meaning that Even50 selected many points that did not contribute to the Pareto front.

In Fig. 9, we show visualizations of the points chosen by each method for a few selected two-tier MRCs with fairly different characteristics.[2] Fig. 9a (top row) displays the MRCs for workload *w04* using LRU replacement, where several knees of various

[2] A similar figure that appeared as Fig. 4 in an earlier version of this paper [19] inadvertently showed visualizations for Even5 instead of Even4.

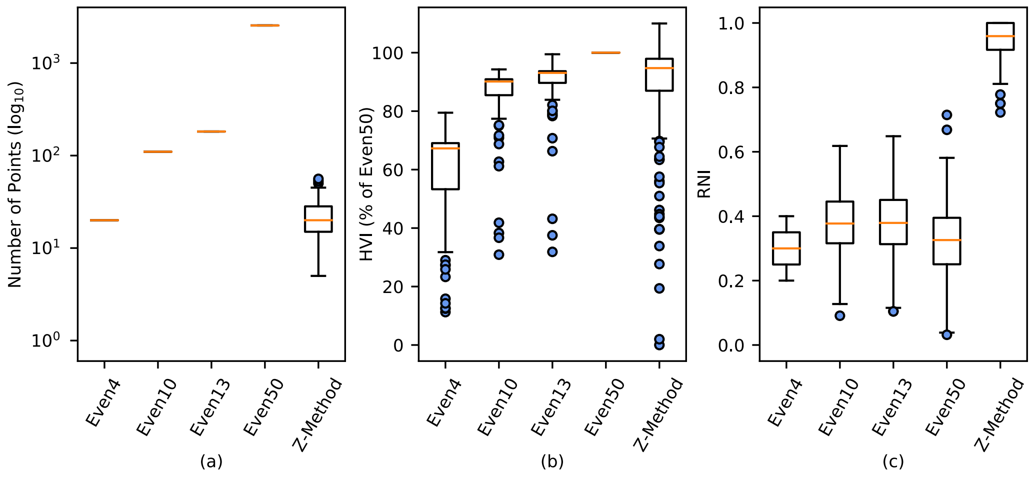


Fig. 8. Evaluation results of our framework using Z-Method across 2-tier ARC and LRU MRCs, derived from 106 real-world block traces collected from CloudPhysics. Box plots of 3 metrics are presented: (a) number of points (lower is better), (b) HyperVolume (HVI) as a percentage of Even50's HyperVolume (higher is better), and (c) Ratio of Non-Dominated Individuals (RNI) (higher is better).

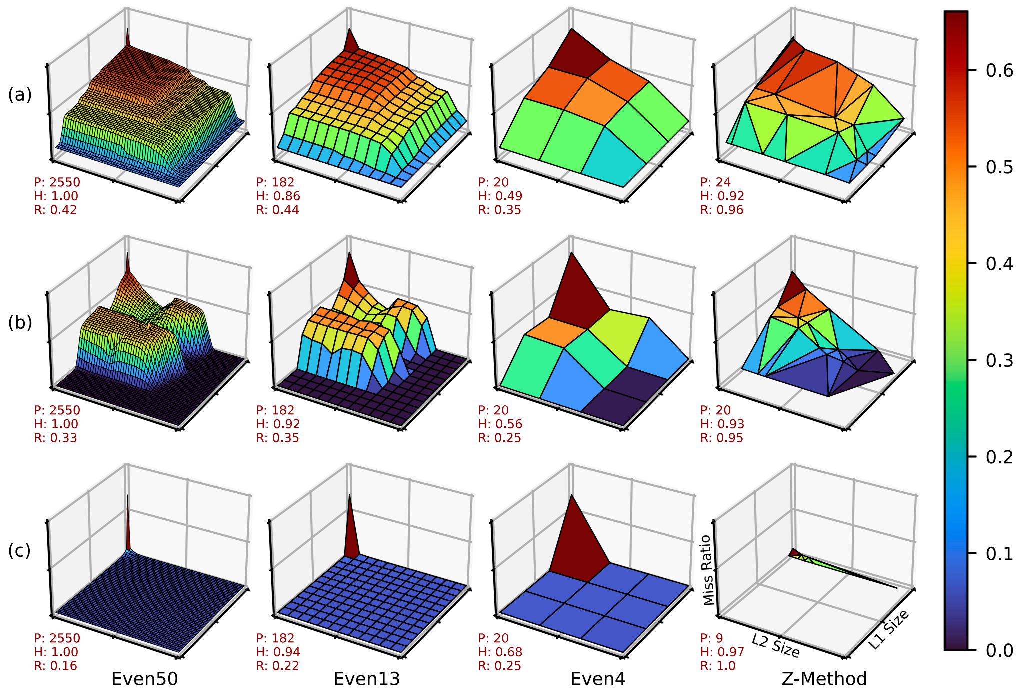


Fig. 9. Examples of point selection on two-tier MRCs that highlight three different commonly observed scenarios. Each point represents the total miss ratio of a configuration of some L1 and L2 sizes. The x and y axes are the normalized L1 and L2 sizes, respectively, while the z-axis is the miss ratio. Axis labels are omitted to reduce clutter. Each row contains MRCs of a single workload using 4 different point-selection methods, listed at the bottom of each column. The P value indicates the total of number of points (lower is better), H is the HyperVolume as a percentage of Even50 (higher is better), and R is the Ratio of Non-Dominated Individuals (higher is better).

sizes are followed by gradually-sloped regions. We can see that Z-Method accurately selects each knee, achieving 92% of Even50's HVI while evaluating over 100× *fewer* points. Conversely, Even13 and Even4 perform poorly, selecting points at the tops of the cliffs before the knees, resulting in lower HVI's of 86% and 49%, respectively. When several knees are present, Z-Method has more opportunities to exploit these significant improvements in miss ratio, performing much better than evenly-spaced points.

Fig. 9b (middle row) displays the MRCs of workload *w66* using ARC replacement, which exhibits large amounts of non-monotonicity, creating several hilly regions. Z-Method finds the interesting knee points at the hill bottoms, while the post-processing filter prevents selecting any points at the hilltops. Z-Method is even more efficient here than in the previous figure while still being highly accurate, selecting only 20 points and achieving 93% of Even50's HVI. Even13 gets close to Z-Method's HVI, but requires 9.1× more points.

Finally, Fig. 9c (bottom row) displays the MRCs of workload *w06* using ARC replacement, which contains only a couple of interesting points at the very beginning of the plot. Z-Method finds 3 points in this tiny space that are more optimal than those found by Even4 or Even13; it also does not waste time exploring the large, flat MRC region that offers almost no improvement in miss ratio. With only 9 points, Z-Method achieves 97% of Even50's HVI, while Even13 evaluates 20.2× *more* points but achieves only 94% of Even50's HVI. MRCs that contain only a handful of good points are fairly common, even in multi-tier settings, and our framework dramatically reduces the time spent exploring them.

6. Evaluation: Population initialization

In this section, we show how our multi-tier knee detection framework can also be applied to population initialization for evolutionary algorithms, to search large configuration spaces more efficiently [45,46]. In many cases, evaluating the fitness of a configuration is an expensive operation, making the speed of convergence particularly important. The initial population of an evolutionary algorithm functions as the first guess at a set of good solutions, so the population's quality can significantly influence the quality of the final solution and the speed at which an algorithm converges [28,29]. As such, heuristics to intelligently select a population have been developed for a variety of scenarios and optimization problems [33,34]. Evaluating multi-tier caching systems fits this scenario well; replaying a workload repeatedly on numerous cache configurations can be costly in both time and money. We demonstrate how the key points found by our multi-knee detection framework can be used to seed the initial population of evolutionary algorithms.

6.1. Experimental setup

We configured each experiment with choices for an input I/O trace, a population-initialization technique, an evolutionary algorithm, a knee-detection algorithm, a cache-replacement algorithm, and two parameters controlling the stopping criteria for the optimization. For each configuration, we analyzed the convergence of an evolutionary algorithm with each of the population-initialization techniques and with our multi-knee detection framework.

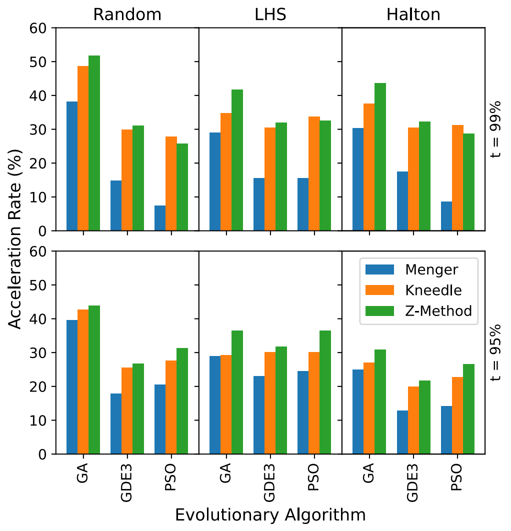
We used our PyMimircache [13] cache simulator extension (see Section 5.1) for all experiments. Simulation enabled us to study a wide variety of configurations, as trace replay on real hardware would be far too slow and would limit the configuration space we could explore. We optimized a single variable, cache size, for the performance metric of I/O operations per second (IOPS) per dollar ($), or IOPS/$. We calculated theoretical values for the IOPS using the same methodology as eMRC [27], and computed dollar costs from a given configuration's cache size and current market values for that type of device [47,48]. We normalized both the IOPS and dollar cost and then combined them to determine IOPS/$.

We evaluated this use case on three different sets of real-world block traces obtained from CloudPhysics [7] and the publicly available FIU [49] and MSR [50] traces, for a total of 151 individual traces. We used uniform randomized spatial sampling [1,7] with a size-based sampling rate R (ranging from 0.1 to 0.0001) on the larger traces to reduce the running time while maintaining an accurate representation of the original trace. Our sampling produced a fairly diverse set of MRC sizes, with a mean of $70,446 \pm 110,014$ blocks, ranging from 263 to 829,424 blocks.

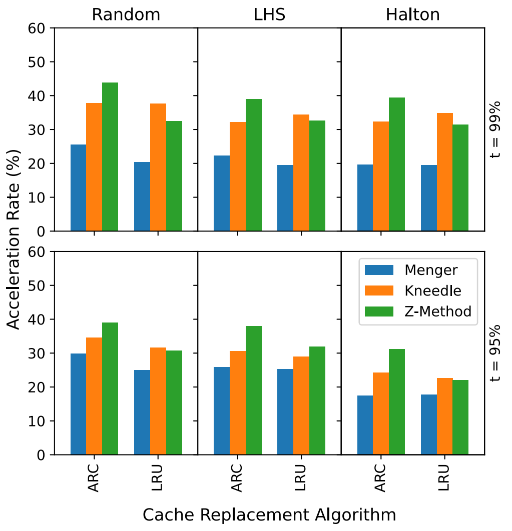
We evaluated the speed of convergence of evolutionary algorithms using four population-initialization techniques: our multi-knee detection framework, random initialization, Latin Hypercube sampling (LHS) [32], and Halton sequences [33]. For techniques that include randomization (all but multi-knee), we ran them three times with different random seeds to obtain stable results. Three seeds is generally considered the minimum acceptable number for this type of analysis. However, given the size of our dataset and configuration space, even three random seeds resulted in experiments that required significant running time while still remaining viable. We ran a total of over **3M** experiments, which sufficiently covers the search space, allowing us to evaluate our proposed solution with statistical confidence. We experimented with three types of evolutionary algorithms: Generalized Differential Evolution 3 (GDE3) [51], a Genetic Algorithm (GA) [52], and Particle Swarm Optimization (PSO) [53]. We focused on a subset of the knee-detection algorithms available in our framework: Menger, Kneedle, and Z-Method. We selected these three because Menger represents a baseline knee-detection method that uses a local feature, Kneedle is a well-known algorithm that greatly benefits from our framework, and Z-Method is our novel algorithm designed for this specific application.

We used both Adaptive Cache Replacement (ARC) and Least-Recently Used (LRU) cache replacement policies. These two popular policies present interesting scenarios for multi-knee detection since the MRCs produced by LRU are guaranteed to be monotonically decreasing, while ARC's MRCs can contain both convex and concave regions (see Section 2.1). Lastly, we enforced two types of stopping criteria for the optimization: (1) the number of evaluations and (2) an objective value that was some percentage of the "best" value for that configuration. The number-of-evaluations stopping criterion was fixed at 300. We found this value sufficient to allow approximately 99% of our experiments to converge. To handle the objective-based stopping criterion, for each trace we simulated 1000 cache sizes evenly spaced from the minimum to the maximum and calculated their IOPS and dollar costs. We obtained the maximum IOPS/$ from these simulations, and treated it as the "best" value when calculating the objective stopping criterion for all optimizations involving that trace.

A rule of thumb for the population size is to use ten times the number of parameters in the solution [54]. Since we are only trying to optimize a single parameter (cache size), this implies a minimum population of size 10. If our framework picks fewer points, we iteratively selected points in the center of the largest gap in the curve until we reached 10. It is also possible to optimize the hyper-parameters of the techniques in our framework to select a desired number of points, but we did not explore hyper-parameter optimization in this work.



(a) Acceleration rates using GA, GDE3, and PSO algorithms.



(b) Acceleration rates using ARC and LRU cache replacement.

Fig. 10. The acceleration rate ($\overline{AR}$) achieved using our multi-knee detection framework for population initialization vs. other techniques. The height of each bar represents the acceleration rate (y-axis), where higher is better. The bottom row represents experiments using the objective threshold $t = 95\%$; the top row is those using $t = 99\%$ (the most challenging threshold to meet). Each group of bars, from left to right, shows the $\overline{AR}$ using our framework for population initialization with the baseline (Menger, in blue), with Kneedle (orange), and with our Z-Method (green), each evaluated against three different initialization methods on the x2-axis (Random, LHS, and Halton, at the top of the figure). (a) shows the $\overline{AR}$ for three different Evolutionary Algorithms on the x-axis (GA, GDE3, and PSO), with results included from both LRU and ARC cache replacement algorithms. (b) shows the same results as in (a) but separated by LRU vs. ARC, with results included from all three Evolutionary Algorithms.

6.2. Acceleration rate

We evaluated our experiments using the overall *acceleration rate* ($\overline{AR}$) [55] to quantify the increased convergence speed when using our framework to select an initial population for evolutionary algorithms. This metric compares the number of function calls (NFCs) made by two separate sets of optimization problems. For our purpose, the NFCs will correspond to the number of epochs (iterations) an evolutionary algorithm takes to converge. Each optimization problem uses an evolutionary algorithm to optimize the IOPS/$ of a cache and has several parameters: an input trace, an evolutionary algorithm, a knee-detection algorithm, a population-initialization technique, a threshold for the value-based stopping criterion, and a cache-replacement algorithm. The $\overline{AR}$ compares two sets of problems and reports the percent difference in convergence speed. For all of our evaluations, we compared a set of problems PM that use our multi-knee detection framework for population initialization, against a set PO that uses some other technique for initialization. The $\overline{AR}$ is calculated as follows:

$$\overline{AR} = \left(1 - \frac{\sum_{i=1}^{|PM|} NFC(PM_i)}{\sum_{i=1}^{|PO|} NFC(PO_i)}\right) \times 100\% \tag{4}$$

i.e., an x% $\overline{AR}$ implies an x% reduction in running time.

We evaluated 151 traces, four population initialization techniques, two cache-replacement algorithms, and three variations of all other parameters, resulting in 32,616 total problems and over three million cache simulations. The overall $\overline{AR}$ achieved using our multi-knee detection framework for population initialization was 34%. In the remainder of this section, we show the effects of each configuration parameter by creating subsets of the total problems via parameter constraints.

Fig. 10 presents bar plots showing the $\overline{AR}$ achieved using our multi-knee detection framework for population initialization. Each bar represents a comparison between two sets of optimization problems, PM and PO, which are different sets for each bar, depending on the constraints of the axes and legend. PM uses our multi-knee detection framework for population initialization with a knee-detection algorithm identified by the color of the bar (Menger, Kneedle, or Z-Method). PO uses the initialization technique shown on the x2-axes (top of figure): a pseudo-random number generator (Random), Latin Hypercube sampling (LHS), or Halton sequences. The value of the objective threshold t stopping criterion for both problem sets is shown on the y2-axes.

In each plot, we show three knee-detection algorithms: Menger, Kneedle, and Z-Method. The bars labeled "Menger" represent a baseline knee-detection method; it was the least accurate of those depicted. "Kneedle" is the robust version that we optimized for this scenario; we employed Global RDP (gRDP) before using Menger or Kneedle, reducing noise in the MRCs and improving knee detection. After the knee-detection phase, we applied the post-processing filters described in Section 3.1 to refine the selected knees.

We varied the objective-threshold stopping criterion t between 99%, 98% (omitted in Fig. 10 for brevity), and 95% for all configurations. We observed that a lower threshold usually yielded a lower $\overline{AR}$. A higher threshold makes optimization more difficult by requiring a solution closer to the most optimal value. Thus, a lower threshold increases the number of acceptable solutions, giving less-informed population initialization techniques a better chance at finding a solution.

Fig. 10(a) shows the effects of using different evolutionary algorithms. The optimization problems analyzed in each group of bars are shown on the x-axis: Genetic Algorithm (GA), Generalized Differential Evolution 3 (GDE3), and Particle Swarm Optimization (PSO). Results in this figure include problem sets with configurations using both ARC and LRU cache replacement. Therefore, in the upper leftmost subfigure of Fig. 10(a), the first blue bar on the left represents, for the 151 traces, the $\overline{AR}$ for all optimization problems using: (1) our framework with Menger for population initialization vs. Random, (2) using a Genetic Algorithm (GA), (3) with either ARC or LRU cache replacement, and (4) an objective threshold $t = 99\%$.

Z-Method is our novel method designed for this task; it outperformed Kneedle in most cases. Kneedle was still competitive, usually only a few percent behind Z-Method and even winning by a small margin in three out of the 18 configurations displayed. (It should be noted that Kneedle benefited greatly from our framework's pre- and post-processing filters, and that Kneedle on its own yielded much poorer results; see Section 2.2). We also saw a wide range of $\overline{AR}$ values in these configurations, from under 10% $\overline{AR}$ (Menger with PSO) to over 50% $\overline{AR}$ (Z-Method with GA). Lastly, ranking the $\overline{AR}$ from highest to lowest yields GA (best), GDE3, and then PSO. We attribute this ranking to the difference in efficiency of the evolutionary algorithms. The more efficient algorithms have a lower $\overline{AR}$ because they are less sensitive to the initial population.

Fig. 10(b) shows the effects of using either ARC or LRU cache replacement. Results in this figure include problem sets with configurations using any of the three evolutionary algorithms. The most relevant difference between ARC and LRU is that LRU MRCs always decrease monotonically, while ARC can have both convex and concave regions (*i.e.*, can go up and down). This poses a unique challenge for knee detection, since a curve can have a knee that is less optimal than a previous point. Said differently, in ARC and similar non-stack cache algorithms, counterintuitively, adding more cache can actually *hurt* performance. This non-monotonic property can also distort many metrics that knee-detection algorithms use to detect or rank points. Z-Method significantly outperformed Kneedle for all ARC cases in Fig. 10(b), while Kneedle beat Z-Method somewhat for all but one of the LRU configurations. This is to be expected, as Z-Method was originally designed with cache optimization for non-monotonic MRCs in mind, enforcing constraints in a grid-like pattern to prevent proximal knees. Conversely, Kneedle was designed under the assumption of monotonicity. Again, we note that our pre- and post-processing filters were essential to Kneedle's good results.

The population-initialization techniques did not alter any of the previous trends we observed in Fig. 10, but we did see an approximately 5% difference in $\overline{AR}$ between the worst and best case: Random performed the worst, followed by Halton, and then LHS. These results are consistent with those seen in previous studies [32–34].

7. Conclusion

The many configurations of multi-tier caching systems produce a wide range of performance and costs. As the configuration space continues to grow due to advancements in caching and storage technology, exploring the space through physical experiments or traditional simulation becomes infeasible.

We introduced the novel concept of applying multi-knee detection to MRCs using a framework for selecting key points, reducing the cost of exploration significantly. We present Z-Method, an algorithm that robustly and efficiently identifies multiple key points in MRCs with minimal overhead. We also designed a recursive algorithm that enables any single-knee-detection algorithm to find multiple knees. We demonstrated that our framework using Z-Method can be applied to reduce the total number of points required to identify optimal two-tier cache configurations by an average factor of approximately 5.5× for ARC and 7.7× for LRU compared to naïve approaches. Finally, we evaluated our framework across a highly diverse set of configurations and datasets for the additional application of seeding the initial population of evolutionary algorithms, showing an overall acceleration rate of 34% compared to commonly used population-initialization techniques.

CRediT authorship contribution statement

Tyler Estro: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Mário Antunes:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Pranav Bhandari:** Writing – review & editing, Methodology, Investigation, Conceptualization. **Anshul Gandhi:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Geoff Kuenning:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Yifei Liu:** Writing – review & editing, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Carl Waldspurger:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization,. **Avani Wildani:** Writing – review & editing, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization. **Erez Zadok:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Erez Zadok reports financial support was provided by Dell EMC Co Ltd. Erez Zadok reports financial support was provided by NetApp. Erez Zadok reports financial support was provided by Meta Platforms Inc. Erez Zadok reports financial support was provided by IBM. Erez Zadok reports financial support was provided by National Science Foundation. Mario Antunes reports financial support was provided by Government of Portuguese Republic Ministry of Science Technology and Higher Education. Mario Antunes reports financial support was provided by Foundation for Science and Technology. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was made possible in part thanks to Dell-EMC, NetApp, Facebook, and IBM support; a SUNY/IBM Alliance award; and National Science Foundation awards CCF-1918225, CNS-1750109, CNS-1900589, CNS-1900706, CNS-1951880, CNS-2106263, CNS-2106434, and CNS-2214980. This work is also partially funded by FCT/MCTES through national funds and when applicable co-funded EU funds under the project UIDB/50008/2020-UIDP/50008/2020.

References

[1] C.A. Waldspurger, T. Saemundson, I. Ahmad, N. Park, Cache modeling and optimization using miniature simulations, in: Proceedings of the 2017 USENIX Annual Technical Conference, ATC '17, USENIX Association, Berkeley, CA, USA, 2017, pp. 487–498, URL http://dl.acm.org/citation.cfm?id=3154690.3154737.

[2] D.S. Berger, B. Berg, T. Zhu, S. Sen, M. Harchol-Balter, RobinHood: Tail latency aware caching — Dynamic reallocation from Cache-rich to Cache-poor, in: USENIX Symposium on Operating Systems Design and Implementation, OSDI, 2018.

[3] T. Estro, P. Bhandari, A. Wildani, E. Zadok, Desperately seeking ... Optimal multi-tier Cache configurations, in: Proceedings of the 12th USENIX Workshop on Hot Topics in Storage, HotStorage '20, USENIX, Boston, MA, 2020.

[4] R.L. Mattson, J. Gecsei, D.R. Slutz, I.L. Traiger, Evaluation techniques for storage hierarchies, IBM Syst. J. 9 (2) (1970) 78–117, http://dx.doi.org/10.1147/sj.92.0078.

[5] D.K. Tam, R. Azimi, L.B. Soares, M. Stumm, RapidMRC: Approximating L2 miss rate curves on commodity systems for online optimizations, in: ACM Sigplan Notices, vol. 44, (no. 3) ACM New York, NY, USA, 2009, pp. 121–132.

[6] J. Wires, S. Ingram, Z. Drudi, N.J.A. Harvey, A. Warfield, Characterizing storage workloads with counter stacks, in: Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2014, USENIX Association, Broomfield, CO, 2014.

[7] C.A. Waldspurger, N. Park, A. Garthwaite, I. Ahmad, Efficient MRC construction with SHARDS, in: Proceedings of the 13th USENIX Conference on File and Storage Technologies, FAST'15, USENIX Association, Santa Clara, CA, 2015.

[8] E. Teran, Z. Wang, D.A. Jiménez, Perceptron learning for reuse prediction, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO, IEEE, 2016, pp. 1–12.

[9] X. Hu, X. Wang, L. Zhou, Y. Luo, C. Ding, Z. Wang, Kinetic modeling of data eviction in cache, in: 2016 USENIX Annual Technical Conference, USENIX ATC 16, USENIX Association, Denver, CO, 2016, pp. 351–364, URL https://www.usenix.org/conference/atc16/technical-sessions/presentation/hu.

[10] J. Fu, D. Arteaga, M. Zhao, Locality-driven MRC construction and Cache allocation, in: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, HPDC '18, ACM, New York, NY, USA, 2018, pp. 19–20, http://dx.doi.org/10.1145/3220192.3220461, URL http://doi.acm.org/10.1145/3220192.3220461.

[11] AccuSim: Accurate simulation of Cache replacement algorithms, 2020, URL https://engineering.purdue.edu/~ychu/accusim/.

[12] E.R. Lucas Filho, L. Odysseos, Y. Lun, F. Kebo, H. Herodotou, DITIS: A distributed tiered storage simulator, Infocommun. J. XIV (4) (2022) 18–25.

[13] J. Yang, PyMimircache, 2019, https://github.com/1a1a11a/PyMimircache. (Retrieved 17 April 2019).

[14] U. Ramer, An iterative procedure for the polygonal approximation of plane curves, Comput. Graph. Image Process. 1 (3) (1972) 244–256.

[15] M. Antunes, D. Gomes, R.L. Aguiar, Knee/elbow estimation based on first derivative threshold, in: Fourth IEEE International Conference on Big Data Computing Service and Applications, BigDataService, IEEE, Bamberg, Germany, 2018, pp. 237–240.

[16] V. Satopaa, J. Albrecht, D. Irwin, B. Raghavan, Finding a "Kneedle" in a haystack: Detecting knee points in system behavior, in: 31st International Conference on Distributed Computing Systems Workshops, IEEE, Minneapolis, MN, 2011, pp. 166–171.

[17] S. Salvador, P. Chan, Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms, in: 16th IEEE International Conference on Tools with Artificial Intelligence, IEEE, 2004, pp. 576–584.

[18] X. Tolsa, Principal values for the Cauchy integral and rectifiability, Proc. Amer. Math. Soc. 128 (7) (2000) 2111–2119.

[19] T. Estro, M. Antunes, P. Bhandari, A. Gandhi, G. Kuenning, Y. Liu, C. Waldspurger, A. Wildani, E. Zadok, Guiding simulations of multi-tier storage Caches using Knee detection, in: 31st Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS'23, IEEE Computer Society, Stony Brook, NY, 2023.

[20] N. Megiddo, D. Modha, ARC: A self-tuning, low overhead replacement Cache, in: Proceedings of the USENIX Conference on File and Storage Technologies, FAST'03, USENIX Association, San Francisco, CA, 2003, pp. 115–130.

[21] R. Goldman, Curvature formulas for implicit curves and surfaces, Comput. Aided Geom. Design 22 (7) (2005) 632–658, http://dx.doi.org/10.1016/j.cagd.2005.06.005.

[22] M. Antunes, J. Ribeiro, D. Gomes, R.L. Aguiar, Knee/elbow point estimation through thresholding, in: 6th IEEE International Conference on Future Internet of Things and Cloud, FiCloud, IEEE, Barcelona, Spain, 2018, pp. 413–419.

[23] G. Yu, L. Ma, Y. Jin, W. Du, Q. Liu, H. Zhang, A survey on knee-oriented multiobjective evolutionary optimization, IEEE Trans. Evol. Comput. 26 (6) (2022) 1452–1472, http://dx.doi.org/10.1109/tevc.2022.3144880.

[24] M. Antunes, H. Aguiar, D. Gomes, AL and S methods: Two extensions for L-method, in: 7th International Conference on Future Internet of Things and Cloud, FiCloud, IEEE, 2019, pp. 371–376.

[25] N. Beckmann, D. Sanchez, Talus: A simple way to remove cliffs in Cache performance, in: IEEE 21st International Symposium on High Performance Computer Architecture, HPCA, 2015, pp. 64–75.

[26] A. Cidon, A. Eisenman, M. Alizadeh, S. Katti, Cliffhanger: Scaling performance cliffs in web memory Caches, in: 13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 16, USENIX Association, Santa Clara, CA, 2016, pp. 379–392, URL https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/cidon.

[27] Z. Liu, H.W. Lee, Y. Xiang, D. Grunwald, S. Ha, eMRC: Efficient miss rate approximation for multi-tier caching, in: 19th USENIX Conference on File and Storage Technologies, FAST 21, USENIX Association, 2021, URL https://www.usenix.org/conference/fast21/presentation/liu.
[28] J.O. Agushaka, A.E.-S. Ezugwu, L.M. Abualigah, S.K. Alharbi, H.A.E.-W. Khalifa, Efficient initialization methods for population-based metaheuristic algorithms: A comparative study, Arch. Comput. Methods Eng. (2022).
[29] B. Kazimipour, X. Li, A.K. Qin, A review of population initialization techniques for evolutionary algorithms, in: 2014 IEEE Congress on Evolutionary Computation, CEC, 2014, pp. 2585–2592, http://dx.doi.org/10.1109/CEC.2014.6900618.
[30] D. Freitas, L.G. Lopes, F.M. Dias, Particle swarm optimisation: A historical review up to the current developments, Entropy 22 (2020).
[31] Z. Ma, G.A.E. Vandenbosch, Impact of random number generators on the performance of particle swarm optimization in antenna design, in: 2012 6th European Conference on Antennas and Propagation, EUCAP, 2012, pp. 925–929, http://dx.doi.org/10.1109/EuCAP.2012.6205998.
[32] S.J. Mousavirad, A.A. Bidgoli, S. Rahnamayan, Tackling deceptive optimization problems using opposition-based DE with center-based Latin hypercube initialization, in: 2019 14th International Conference on Computer Science & Education, ICCSE, 2019, pp. 394–400, http://dx.doi.org/10.1109/ICCSE.2019.8845360.
[33] N.Q. Uy, N.X. Hoai, R. McKay, P.M. Tuan, Initialising PSO with randomised low-discrepancy sequences: The comparative results, in: 2007 IEEE Congress on Evolutionary Computation, 2007, pp. 1985–1992, http://dx.doi.org/10.1109/CEC.2007.4424717.
[34] A. Ashraf, S. Pervaiz, W. Bangyal, K. Nisar, A.A. Ag Ibrahim, J. Rodrigues, D. Rawat, Studying the impact of initialization for population-based algorithms with low-discrepancy sequences, Appl. Sci. 11 (2021) 8190, http://dx.doi.org/10.3390/app11178190.
[35] M. García-Arnau, D. Manrique, J. Ríos, A. Rodríguez-Patón, Initialization method for grammar-guided genetic programming, Knowl.-Based Syst. 20 (2) (2007) 127–133, http://dx.doi.org/10.1016/j.knosys.2006.11.006, AI 2006, URL https://www.sciencedirect.com/science/article/pii/S0950705106001973.
[36] G. Zhang, L. Gao, Y. Shi, An effective genetic algorithm for the flexible job-shop scheduling problem, Expert Syst. Appl. 38 (4) (2011) 3563–3573, http://dx.doi.org/10.1016/j.eswa.2010.08.145, URL https://www.sciencedirect.com/science/article/pii/S095741741000953X.
[37] T. Tometzki, S. Engell, Systematic initialization techniques for hybrid evolutionary algorithms for solving two-stage stochastic mixed-integer programs, IEEE Trans. Evol. Comput. 15 (2011) 196–214.
[38] C.C. Aggarwal, Outlier Analysis, second ed., Springer Publishing Company, Incorporated, 2016.
[39] L. Cheng, J. Xiong, L. He, Non-Gaussian statistical timing analysis using second-order polynomial fitting, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 28 (1) (2008) 130–140.
[40] Z. Cao, V. Tarasov, S. Tiwari, E. Zadok, Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems, in: USENIX Annual Technical Conference,, ATC, Boston, MA, 2018, pp. 893–907.
[41] D. Chicco, G. Jurman, The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation, BMC Gen. 21 (2020).
[42] M. Li, X. Yao, Quality evaluation of solution sets in multiobjective optimisation: A survey, ACM Comput. Surv. 52 (2) (2019) http://dx.doi.org/10.1145/3300148.
[43] Y. Cao, B.J. Smucker, T.J. Robinson, On using the hypervolume indicator to compare Pareto fronts: Applications to multi-criteria optimal experimental design, J. Statist. Plann. Inference 160 (2015) 60–74.
[44] K. Tan, T. Lee, E. Khor, Evolutionary algorithms for multi-objective optimization: Performance assessments and comparisons, in: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), vol. 2, 2001, pp. 979–986, http://dx.doi.org/10.1109/CEC.2001.934296.
[45] P. Godefroid, S. Khurshid, Exploring very large state spaces using genetic algorithms, in: J.-P. Katoen, P. Stevens (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 266–280.
[46] A. de Perthuis de Laillevault, B. Doerr, C. Doerr, Money for nothing: Speeding up evolutionary algorithms through better initialization, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15, ACM, 2015, http://dx.doi.org/10.1145/2739480.2754760.
[47] AnandTech: Hardware news and tech reviews since 1997, 2023, www.anandtech.com.
[48] Tom's hardware: For the hardcore PC enthusiast, 2023, www.tomshardware.com.
[49] A. Verma, R. Koller, L. Useche, R. Rangaswami, FIU traces (SNIA IOTTA trace set 390), in: G. Kuenning (Ed.), SNIA IOTTA Trace Repository, Storage Networking Industry Association, 2009, URL http://iotta.snia.org/traces/block-io?only=390.
[50] D. Narayanan, A. Donnelly, A. Rowstron, MSR Cambridge traces (SNIA IOTTA trace set 388), in: G. Kuenning (Ed.), SNIA IOTTA Trace Repository, Storage Networking Industry Association, 2007, URL http://iotta.snia.org/traces/block-io?only=388.
[51] S. Kukkonen, J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in: 2005 IEEE Congress on Evolutionary Computation, vol. 1, IEEE, 2005, pp. 443–450.
[52] J.H. Holland, Adaptation in Natural and Artificial Systems, The MIT Press, 1992, http://dx.doi.org/10.7551/mitpress/1090.001.0001.
[53] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948 vol.4, http://dx.doi.org/10.1109/ICNN.1995.488968.
[54] R. Storn, On the usage of differential evolution for function optimization, in: Proceedings of North American Fuzzy Information Processing, 1996, pp. 519–523, http://dx.doi.org/10.1109/NAFIPS.1996.534789.
[55] S. Rahnamayan, H.R. Tizhoosh, M.M. Salama, A novel population initialization method for accelerating evolutionary algorithms, Comput. Math. Appl. 53 (10) (2007) 1605–1614, http://dx.doi.org/10.1016/j.camwa.2006.07.013.