PREDICTING SYSTEM PERFORMANCE BY INTERPOLATION USING A HIGH-DIMENSIONAL DELAUNAY TRIANGULATION

Tyler H. Chang

Dept. of Computer Science
Virginia Polytechnic Institute
& State University (VPI&SU)
Blacksburg, VA 24061
thchang@vt.edu

Layne T. Watson

Depts. of Computer Science,
Mathematics, and
Aerospace & Ocean Engineering
VPI&SU, Blacksburg, VA 24061

Thomas C. H. Lux

Dept. of Computer Science
VPI&SU, Blacksburg, VA 24061

Jon Bernard

Dept. of Computer Science, VPI&SU
Blacksburg, VA 24061

Bo Li

Dept. of Computer Science, VPI&SU
Blacksburg, VA 24061

Li Xu

Dept. of Statistics, VPI&SU
Blacksburg, VA 24061

Godmar Back

Dept. of Computer Science, VPI&SU
Blacksburg, VA 24061

Ali R. Butt

Dept. of Computer Science, VPI&SU
Blacksburg, VA 24061

Kirk W. Cameron

Dept. of Computer Science, VPI&SU
Blacksburg, VA 24061

Yili Hong

Dept. of Statistics, VPI&SU
Blacksburg, VA 24061

ABSTRACT

When interpolating computing system performance data, there are many input parameters that must be considered. Therefore, the chosen multivariate interpolation model must be capable of scaling to many dimensions. The Delaunay triangulation is a foundational technique, commonly used to perform piecewise linear interpolation in computer graphics, physics, civil engineering, and geography applications. It has been shown to produce a simplex based mesh with numerous favourable properties for interpolation. While

SpringSim-HPC, 2018 April 15-18, Baltimore, MD, USA;

computation of the two- and three-dimensional Delaunay triangulation is a well-studied problem, there are numerous technical limitations to the computability of a high-dimensional Delaunay triangulation. This paper proposes a new algorithm for computing interpolated values from the Delaunay triangulation without computing the complete triangulation. The proposed algorithm is shown to scale to over 50 dimensions. Data is presented demonstrating interpolation using the Delaunay triangulation in a real world high performance computing system problem.

Keywords: Delaunay triangulation, multivariate interpolation, performance, performance variability, high-dimensional data

1 INTRODUCTION

High performance computing (HPC) system performance is a topic of significant importance in the scientific community. There are many measures by which HPC performance can be evaluated, such as throughput, energy consumption, and compute time for a given task. Additionally, since each of these measures tends to fluctuate stochastically between independent runs, various summary statistics such as the mean, median, maximum, minimum observed value, or variance can be collected. Furthermore, each of these measures is generally affected by numerous factors, such as system specifications, system configuration, application type, and application dependent inputs. In this way, each measure can be thought of as a function of various system and application level parameters.

For a performance evaluator benchmarking a machine, each of these measures could easily be collected for a finite set of system and application "signatures" by running several applications over various machine configurations (with multiple runs at each configuration). After collecting the data, suppose an evaluator wanted to predict performance measures for some *different* signature(s). Assuming that there is an underlying relationship between the measured statistics and the set of system and application parameters that contribute to the signature, it is reasonable to make this prediction with a multivariate interpolatory model.

Given a set of data points, an interpolant is an approximation of the underlying function f that exactly matches all given values. A *multivariate* interpolant is characterized by the presence of more than one input variable. Therefore, a multivariate interpolant of f is a function over d-tuples of input parameters in the domain of f, where the domain of f has typically been mapped to a subset of $\mathbb{R}^d$. There are many parameters that can affect the performance of a HPC system, and therefore the dimension of the space $\mathbb{R}^d$ could be quite large.

A triangulation of a set of points P in $\mathbb{R}^d$ is a division of the convex hull of P, denoted $CH(P)$, into d-simplices with vertices in P. The Delaunay triangulation of P, denoted $DT(P)$, is a particular triangulation defined in terms of the empty circumsphere property, and is commonly used as an interpolatory mesh in fields such as geographic information systems (GIS), civil engineering, physics, and computer graphics. Two- and three-dimensional Delaunay triangulations (Delaunay triangulations of points in $\mathbb{R}^2$ and $\mathbb{R}^3$) are readily computable, but there are theoretical limitations to the computability of a high-dimensional Delaunay triangulation.

The rest of the paper is organized as follows. Section 2 will discuss background related to the theoretical advantages and computability of the Delaunay triangulation. Section 3 presents a new algorithm for interpolating high-dimensional data with the Delaunay triangulation. Section 4 contains performance data demonstrating the scalability of the proposed algorithm. Section 5 details the predictive performance of the proposed algorithm for a four-dimensional HPC system interpolation problem. Section 6 outlines future work.

2 BACKGROUND

2.1 The Delaunay Interpolant

The Delaunay triangulation is defined as the geometric dual of the Voronoi diagram, or equivalently, as any triangulation satisfying the empty circumsphere property defined below (**?**, **?**, **?**). See Figure 1 for a visual.

Definition 1. *A Delaunay triangulation $DT(P)$ of a set of points P in $\mathbb{R}^d$ is any triangulation of P such that for each d-simplex $s \in DT(P)$, the interior of the sphere circumscribing s contains no point in P.*

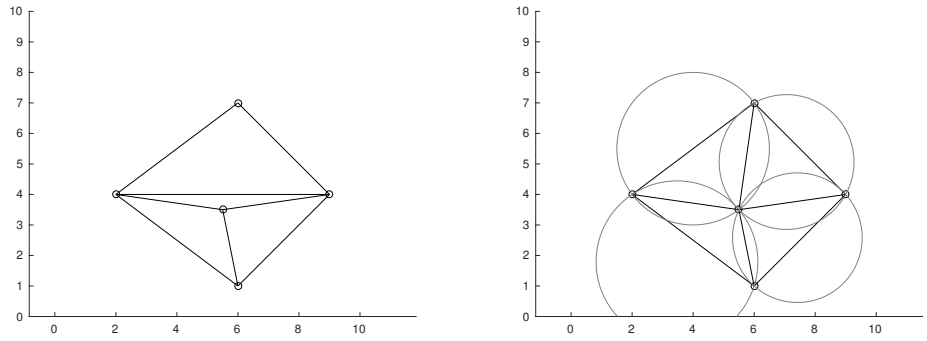

Figure 1: A triangulation in $\mathbb{R}^2$ (left) and the Delaunay triangulation (right).

The Delaunay triangulation exists for every nonempty finite set of points P in $\mathbb{R}^d$ that do not all lie in the same lower-dimensional linear manifold in $\mathbb{R}^d$ (or equivalently, if $CH(P)$ has nonzero volume). This is a direct corollary to the Delaunay triangulation's duality with the Voronoi diagram. Note that the existence of $DT(P)$ requires that n (the size of P) be greater than or equal to $d+1$, otherwise all n points will lie in the same $n-1$-dimensional linear manifold trivially. In the context of interpolation, the degenerate case where $DT(P)$ does not exist and $n \geq d+1$ can be seen as an over-parameterization of the underlying function f. The points are said to be in *general position* if the Delaunay triangulation exists, and no $d+2$ points in P lie on the same $d-1$-sphere; in this case the Delaunay triangulation of P is unique (**?**; **?**).

Given a triangulation T of P and function values $f_i = f(x_i)$ for all $x_i \in P$, f can be interpolated at any point $q \in CH(P)$ using a piecewise linear interpolant $\hat{f}$ defined as follows: Let s be a d-simplex in T with vertices $s_1, s_2, \ldots, s_{d+1}$ and $q \in s$. Then $q = \sum_{i=1}^{d+1} w_i s_i$, $\sum_{i=1}^{d+1} w_i = 1$, $w_i \geq 0$, $i = 1, \ldots, d+1$, and

$$\hat{f}(q) = f(s_1)w_1 + f(s_2)w_2 + \ldots + f(s_{d+1})w_{d+1}. \tag{1}$$

It is common to define $\hat{f}$ using $DT(P)$ since the Delaunay triangulation enjoys several properties that are considered optimal over all simplex based interpolation meshes (**?**, **?**). Consequently, the Delaunay triangulation finds wide use as a model for multivariable piecewise linear interpolation in fields such as GIS, civil engineering, physics, and computer graphics (**?**, **?**).

2.2 Related Works and Challenges

The two- and three-dimensional Delaunay triangulation (triangulations of points in $\mathbb{R}^2$ and $\mathbb{R}^3$) can be efficiently computed in $\mathcal{O}(n \log n)$ time (**?**). In order to compute an interpolated value $\hat{f}(q)$ for some two- or three-dimensional point $q \in CH(P)$, one must also locate the simplex in $DT(P)$ containing q. This simplex can be located using the "jump-and-walk" algorithm in sublinear expected time (**?**). Given this simplex, it is trivial to compute $\hat{f}(q)$ using (1).

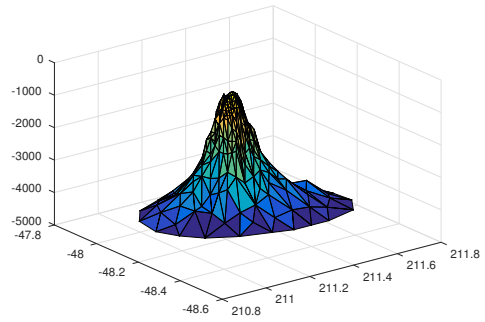

Figure 2: The 2D Delaunay triangulation of height data for a sea mountain is used to construct a piecewise linear model of the mountain in `Matlab`.

However, in higher dimensions, there is a theoretical limitation to the computability of the Delaunay triangulation. In the worst case, a d-dimensional Delaunay triangulation of n points could contain up to $\mathcal{O}(n^{\lceil d/2 \rceil})$ simplices (**?**). Therefore, the lower bound on the worst case time complexity for computing $DT(P)$ is $\mathcal{O}(n^{\lceil d/2 \rceil})$. Furthermore, in the worst case, $\mathcal{O}(n^{\lceil d/2 \rceil})$ space is required to store $DT(P)$. It is important to note that in the common case, significantly fewer than $n^{\lceil d/2 \rceil}$ simplices will be contained in $DT(P)$. However, the common case size still tends to grow exponentially with respect to d (**?**).

Current state-of-the art Delaunay triangulation techniques generally scale to six or seven dimensions for large input sets and focus on either minimizing compute time in the general case (**?**) or minimizing the storage overhead (**?**). Others have aimed to avoid the scalability problem by computing alternative simplical complexes with still favourable topological properties and improved scalability (**?**).

3 PROPOSED ALGORITHM

In order to interpolate over large data sets where the independent variables span many dimensions (such as system performance data sets), a more scalable approach to the Delaunay triangulation is required. Since it is intractable to compute the *entire* Delaunay triangulation in large dimensions, the following observation is useful: Given a set of points P in $\mathbb{R}^d$, one can compute the Delaunay interpolant $\hat{f}(q)$ at a point $q \in CH(P)$ knowing only the simplex $s \in DT(P)$ such that q is in s. It should be noted that q could lie on a face shared by multiple simplices in $DT(P)$. However, in these cases, $\hat{f}(q)$ will be the same regardless of the simplex chosen since only the vertices shared by *all* containing simplices will contribute to $\hat{f}(q)$. This observation reduces the problem of computing the entire Delaunay triangulation to that of computing a specific simplex in the Delaunay triangulation.

3.1 Algorithm Outline

It is possible to grow an initial Delaunay simplex using any point in P as the starting vertex. This approach is described as the basis for both *incremental construction* and *divide and conquer* algorithms in **?**. Given a point $q \in CH(P)$ to interpolate at and a simplex s defined by points in P, it must be the case that q is either in s, or q is across the hyperplane defined by at least one of the facets of s. Therefore, given any s not containing q, one can construct a new Delaunay simplex s^* closer to q by completing a facet of s that is shared with s^*. The technique used for completing an open facet of a simplex is outlined in **?**.

Using this methodology and a deterministic variant of the "walk" described in **?**, it is possible to continually advance toward the simplex containing q via an iterative process. Note, that for every d-simplex $s \in DT(P)$

and $q \in \mathbb{R}^d$, q is a *unique* affine combination of the vertices of s. Furthermore, if q is *in* s, then this affine combination will also be convex by the definition of a simplex. Therefore, one can terminate based on the nonnegativity of the affine weights for q with respect to the vertices of the current simplex s. Conveniently, upon termination, the convex weights for q are the exact weights needed to linearly interpolate at q within s, as defined previously in (1). Pseudo code for the proposed algorithm is provided below.

Algorithm.

Let P be an array of points in $\mathbb{R}^d$.
Let $q \in CH(P)$ be a point to interpolate at.
Let $f_i = f(x_i)$ be known for all $x_i \in P$, and let $f(s) = \left(f_{i_1}, f_{i_2}, \ldots, f_{i_{d+1}}\right)$ for d-simplex s with vertices $x_{i_1}, x_{i_2}, \ldots, x_{i_{d+1}}$.
Let MakeFirstSimplex(P) be a function that constructs an initial Delaunay simplex from points in P.
Let CompleteSimplex(σ, q, P) be a function that completes the facet σ with a point from P that is on the same side of the hyperplane containing σ as q.
Let AffineWeights(q, s) be a function that returns the affine weights that give q as a combination of the vertices of s.
Let MinIndex(w) return the index of the most negative element w_i.
Let DropVertex(s, i) return the facet of s that results from dropping vertex i.
The following algorithm computes the interpolant $\hat{f}(q)$ using the Delaunay triangulation of P.

s = MakeFirstSimplex(P);
w = AffineWeights(q, s);
while $w_i < 0$ for some $1 \leq i \leq d+1$ **do**
  i = MinIndex(w);
  σ = DropVertex(s, i);
  s = CompleteSimplex(σ, q, P);
  w = AffineWeights(q, s);
end while
return $\hat{f}(q) = \langle w, f(s) \rangle$ (inner product);

3.2 Complexity Analysis

3.2.1 Time Complexity

The construction of the first simplex can be formulated as a sequence of least squares (LS) problems ranging in size from $2 \times d$ to $d \times d$. Using `DGELS` from `LAPACK`, each LS problem can be solved in $\mathcal{O}(d^3)$ time. At all $d-1$ sizes, one must solve up to n LS problems, taking the point that produces the minimum residual as the next vertex. Therefore, the total computation time for the first simplex will be $\mathcal{O}(nd^4)$.

To complete a simplex (one iteration of the walk) requires at most n linear solves, performed with `DGESV` from `LAPACK` in $\mathcal{O}(nd^3)$ total time. In each iteration, a flip toward q is performed by dropping a vertex in s to get an open facet σ, then completing σ with a point on the opposite halfspace (as defined by the hyperplane containing σ). In the expected case, for uniformly distributed points, the simplex containing q is located in $\mathcal{O}(n^{1/d})$ iterations (**?**, **?**). Therefore, the worst case time complexity for performing the "walk" from the initial simplex to the simplex containing q is expected to be $\mathcal{O}(n^{1+\frac{1}{d}}d^3)$.

From the previous analysis, using the proposed algorithm to interpolate at m points from a set of n points in $\mathbb{R}^d$ will take $\mathcal{O}(mn^{1+\frac{1}{d}}d^3 + mnd^4)$ total time. This is a significant improvement over the non polynomial worst case for computing the complete triangulation.

3.2.2 Space Complexity

Recall from Section 3.1 that the size of the Delaunay triangulation grows exponentially with the dimension. Therefore, space complexity is equally as concerning as time complexity since, for large d, one cannot store the exponentially sized triangulation in memory. Another advantage of the proposed algorithm is that any computed simplex that does not contain any interpolation point can be discarded immediately.

Therefore, the required space for computing the Delaunay interpolant is reduced to:

- $\mathcal{O}(nd)$ space for the n input points in $\mathbb{R}^d$;
- $\mathcal{O}(md)$ space for storing the m interpolation points in $\mathbb{R}^d$, the m containing simplices of size $d+1$ each, and the m convex coordinate vectors of size $d+1$ each;
- $\mathcal{O}(d^2)$ space for storing the $d \times d$ matrices involved in performing linear solves;
- Other temporary storage arrays that require $\mathcal{O}(d)$ space.

This makes the total space complexity $\mathcal{O}(nd+md+d^2)$. Since no triangulation can exist unless $n > d$, the space complexity can be further reduced to $\mathcal{O}(nd+md)$, which is approximately the same size as the input.

3.3 Optimizations

There are two optimizations that are easily implemented to improve the performance of the proposed algorithm. First, one could identify the point $p \in P$ that is a nearest neighbor to q with respect to Euclidean distance in $\mathcal{O}(n)$ time and build the first simplex off of p instead of an arbitrarily chosen point. For interpolating at a single point, this typically leads to location of the simplex containing q in $\mathcal{O}(d \log d)$ iterations of the walk. Therefore, the expected time complexity is further reduced to $\Theta(mnd^4 \log d)$.

Furthermore, when interpolating many points, it is often the case that some simplex or simplices in the iterative process contain interpolation points that have not yet been resolved. With no increase in total time complexity, it is possible to check if the current simplex s contains *any* of the unresolved interpolation points. If the points being interpolated are tightly clustered, it is typical for them to all be contained in a small number of Delaunay simplices, significantly reducing total computation time.

3.4 Extrapolation

Up until this point, the proposed algorithm has only covered interpolation cases (when q is in $CH(P)$). Often, however, it is reasonable to make a prediction about some extrapolation points Z that are *slightly* outside $CH(P)$. In these cases, it is most reasonable to project each $z \in Z$ onto $CH(P)$ and interpolate at each projection $\hat{z}$, provided the residual $r = \|z - \hat{z}\|_2$ is small. The projection of z onto $CH(P)$ can be easily reformulated as an inequality constrained least squares problem, whose efficient solution is described in **?**.

4 RUNTIME ANALYSIS

A serial implementation of the proposed algorithm has been coded in ISO Fortran 2003 with extra machinery added for numerical stability and degenerate case handling. This code was tested for correctness against `Qhull` on both real world and pseudo-random data sets in up to four dimensions, with a mix of degenerate and general position input points. `Qhull` is an industry standard used for computing high-dimensional Delaunay triangulations in `Matlab`, `SciPy`, and `R`. An analysis of the `Qhull` algorithm is in **?**. After con-

firming correctness with lower dimensional data, run times were gathered for pseudo-randomly generated point sets in up to 64 dimensions.

The following run times were gathered on an Intel i7-3770 CPU @3.40 GHz running CentOS release 7.3.1611. All run times were averaged over a sample size of 20 runs, each performed on a unique pseudo-randomly generated input data set (generated with the Fortran intrinsic random number generator). Times were recorded with the Fortran intrinsic `CPU_TIME` function, which is accurate up to microsecond resolution. Table 1 details average run times for interpolating at uniformly spaced interpolation points using a five-dimensional input data set ranging in size from $n = 2000$ points to $n = 32,000$ points. Table 2 details average run times for interpolating at interpolation points that were clustered within a hypercube with 10% of the original point-set's diameter using a five-dimensional input data set ranging in size from $n = 2000$ points to $n = 32,000$ points. Table 3 details average run times for interpolating at a single point in $d = 2$ up to $d = 64$ dimensions over input data sets ranging in size from $n = 2,000$ points up to $n = 32,000$ points. For reference, the data in Tables 1 and 2 could be compared to the computation times for the complete five-dimensional Delaunay triangulation presented in **?**.

Table 1: Average runtime in seconds for interpolating at uniformly spaced interpolation points for 5D pseudo-randomly generated input points.

	$n = 2000$	$n = 8000$	$n = 16,000$	$n = 32,000$
32 interp. pts	0.3 s	2.7 s	9.6 s	35.7 s
1024 interp. pts	2.5 s	11.6 s	28.9 s	79.1 s

Table 2: Average runtime in seconds for interpolating at clustered interpolation points for 5D pseudo-randomly generated input points.

	$n = 2000$	$n = 8000$	$n = 16,000$	$n = 32,000$
32 interp. pts	0.2 s	2.2 s	8.4 s	33.0 s
1024 interp. pts	0.2 s	2.5 s	9.2 s	35.2 s

Table 3: Average runtime in seconds for interpolating at a single point in up to 64 dimensional space for pseudo-randomly generated input points.

	$n = 2000$	$n = 8000$	$n = 16,000$	$n = 32,000$
$d = 2$	0.1 s	1.7 s	6.8 s	27.0 s
$d = 8$	0.2 s	2.5 s	9.6 s	37.9 s
$d = 32$	1.4 s	9.5 s	29.7 s	101.1 s
$d = 64$	13.2 s	60.1 s	138.6 s	349.1 s

A keen reader may recall from Section 3.3 that with optimizations, linear scaling with respect to n (the input point size) is expected. However, in Table 3, a quadratic relationship is observed below 64 dimensions. This is actually an artifact of the implementation, which includes extra code to promote elegant error handling and numerical stability. Among other functions, the added code checks the diameter and closest-pair distance of the data-set in $\mathcal{O}(n^2)$ time and must perform at least one check for rank deficiency using the singular-value decomposition of a $d \times d$ matrix. If the input is assumed to consist of legal values with all points in general position, then these checks can be dropped, significantly improving the run time.

5 PREDICTIVE ACCURACY

There are many theoretical advantages to interpolating using the Delaunay triangulation. To demonstrate its predictive power, in this section, the code described in Section 4 is applied to an HPC system performance problem.

5.1 Problem Summary

For this problem, the performance measure chosen is *throughput variance*, which is a quantifier of performance variability. Performance variability refers to the inherent "jitter" observed in performance values between multiple independent runs of an identical application. This variability can be of concern, particularly in large scale systems such as HPC and Cloud systems (**?**, **?**, **?**). Specifically for this work, throughput variability is considered with respect to various instances of the `IOzone` benchmark being run over a cluster of identical machines with identical system and application level parameters. Unless otherwise stated, all system and application level parameters are assumed to be fixed at some reasonable value.

5.2 Data Collection

Data for these experiments has been gathered at Virginia Tech on a homogeneous cluster of shared-memory nodes running Ubuntu 14.04 LTS on a dedicated 2TB HDD. Each node is a 2 socket, 4 core hyperthreaded Intel Xeon E5-2623 v3 (Haswell) processor with 32 GB DDR4. Data has been gathered by running the `IOzone` benchmark. The `IOzone` benchmark measures read/write throughputs by reading and writing files of configurable size, broken up into configurable record sizes, utilizing a configurable number of threads. For more information on `IOzone`, see `www.iozone.org`. Up to 13 different variations of read and write tasks can be tested, but in this paper only the `fread` task is considered. To generate each data point, the `IOzone` benchmark has been run independently 40 times with identical settings, and the variance has been computed for the maximum throughput of the `fread` task using

$$\sigma^2 = \frac{\sum_{i=1}^{40}(t_i - \mu)^2}{39} \tag{2}$$

where t_i represents the throughput for each of the 40 runs, and μ is the mean observed throughput over all 40 runs. The variance is modeled as a function of four system and application parameters, chosen because of their relevance to the `IOzone` benchmark. These parameters are thread count, CPU frequency, file size, and record size. Note that the parameters thread count, file size, and record size are specifiable as `IOzone` inputs, while CPU frequency must be manually set for each run using system tools. To avoid biasing the data, the CPU cache has been purged between each run of an `IOzone` test.

For each parameter, several values have been chosen spanning a range of reasonable values, and the observed variance has been calculated for each combination of settings using (2). For the data presented, the values chosen are in Table 4. Note that some combinations of parameters are not viable (specifically, the record size cannot be greater than the file size). These combinations have been omitted when collecting data. Observe that there are 6 valid combinations of file and record size, 9 thread counts, and 16 distinct CPU frequencies. This results in $6 \times 9 \times 16 = 864$ total data points.

Table 4: Values for adjusted parameters. Note, the frequency 3.001 GHz results from overclocking.

	values
file size (KB)	64, 256, 1024
record size (KB)	32, 128, 512
thread count	1, 2, 4, 8, 16, 32, 64, 128, 256
frequency (GHz)	1.2, 1.4, 1.5, 1.6, 1.8, 1.9, 2.0, 2.1, 2.3, 2.4, 2.5, 2.7, 2.8, 2.9, 3.0, 3.001

The relationship between variance and each individual problem dimension has been observed to exhibit highly nonlinear behavior, making a simple linear fit an extremely poor solution to this problem. The Delaunay interpolant is better able to accommodate this nonlinearity since the Delaunay interpolant is only piecewise linear.

5.3 Model Evaluation

From the collected data, the following points in (3) have been selected for testing the Delaunay prediction.

$$\begin{aligned} q_1 &= (\text{fsize} = 256, \text{rsize} = 128, \text{threads} = 2, \text{freq} = 1.9) \\ q_2 &= (\text{fsize} = 256, \text{rsize} = 128, \text{threads} = 2, \text{freq} = 2.9) \\ q_3 &= (\text{fsize} = 1024, \text{rsize} = 32, \text{threads} = 4, \text{freq} = 2.9) \end{aligned} \tag{3}$$

Note that because of the restriction on valid combinations of file and record size, it is not possible to select points strictly inside the convex hull. Consequently, all the points in (3) are on the boundary of the convex hull. Various percentages of the remaining points are used by the Delaunay interpolant to predict the value of the throughput variance $f(q_i)$ for all q_i in (3). For each of these "training percentages," up to 200 Delaunay interpolations are calculated using different pseudo-random samplings from the complete set of data points, with a bias toward uniformly distributed samplings. These samplings are constrained in that the prediction of $f(q_i)$ cannot be based off a sampling that includes q_i, and q_i cannot be outside the convex hull of the selected points. Also, repeated use of the same sampling in a single batch of 200 samplings has been forbidden. Note that because of the constraints, in some cases, less than 200 samplings could be gathered.

Figures 3, 4, and 5 show box plots of the relative errors observed when using the Delaunay interpolant to predict q_1, q_2, and q_3 at each training percentage. The relative error was computed using: $\left|\hat{f}(q_i) - f(q_i)\right| / f(q_i)$ for $i = 1, 2, 3$.

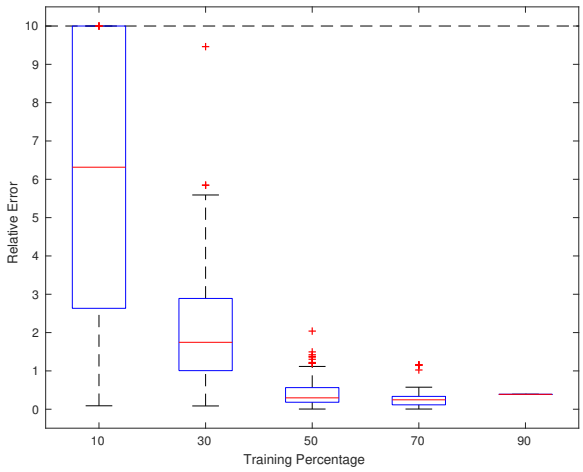

Figure 3: Box plot of the relative error for up to 200 variance predictions at q_1 (3) using the Delaunay interpolant with various percentages of the total available data. Note: Relative errors greater than 10 (1000%) have been truncated.

Using 90% of the 863 remaining data points (after excluding the point q_i that is being interpolated at) the Delaunay interpolant is seen to be fairly accurate, taking into consideration the difficulty of the problem. The Delaunay interpolant remains relatively accurate using as little as 50% of the remaining data. Note that for the large training percentages, the boxes in Figures 3, 4, and 5 are very narrow. This is because large samplings have a relatively low probability for dropping a vertex from the Delaunay simplex containing q_i, and unless some vertex of the containing Delaunay simplex is dropped, the interpolated value will not change for the Delaunay interpolant.

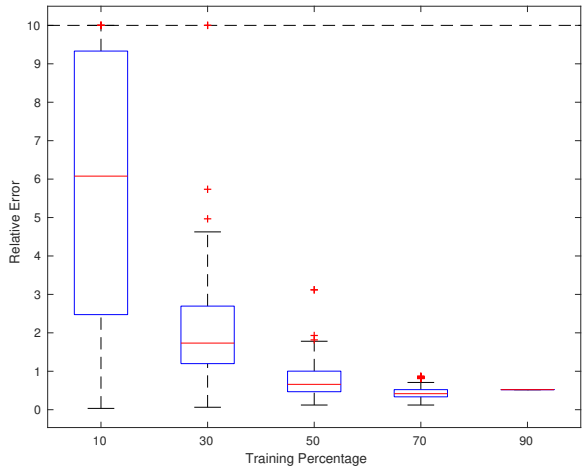

Figure 4: Box plot of the relative error for up to 200 variance predictions at q_2 (3) using the Delaunay interpolant with various percentages of the total available data. Note: Relative errors greater than 10 (1000%) have been truncated.

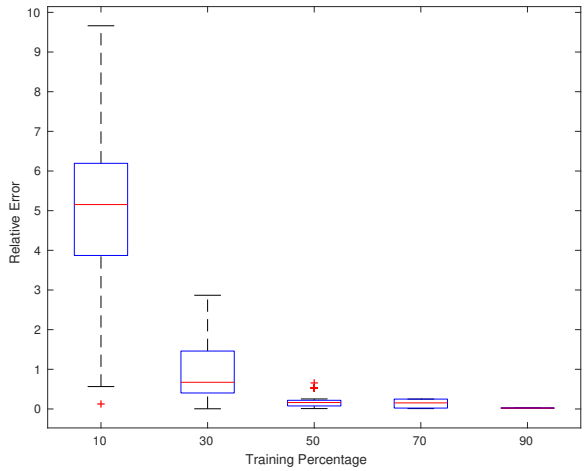

Figure 5: Box plot of the relative error for up to 200 variance predictions at q_3 (3) using the Delaunay interpolant with various percentages of the total available data.

6 CONCLUSION AND FUTURE WORK

In this paper, a new Delaunay triangulation algorithm is proposed for interpolating in point clouds in arbitrary dimension d. This is achieved by computing a relatively small number of simplices from the complete Delaunay triangulation in polynomial time. The proposed algorithm scales linearly with respect to the size of the input in the expected case, regardless of the dimension. The described algorithm was demonstrated on a system performance problem, where it was used to make accurate predictions about throughput variance. It should be noted that the Delaunay triangulation is not limited to interpolation and is widely used in mesh generation, principle component analysis, and topological data analysis. Its geometric dual, the Voronoi diagram, can also be used to make rapid nearest neighbor queries and is the basis for another interpolation technique, *natural neighbor interpolation*. In future work, the methods used to construct and locate a single

Delaunay simplex could potentially be extended for other Delaunay triangulation applications. In particular, knowledge of a Voronoi cell can be achieved by computing the star of simplices incident at a given vertex.

REFERENCES

AUTHOR BIOGRAPHIES

TYLER H. CHANG is a Ph.D. student at VPI&SU studying computer science under Dr. Layne Watson.

LAYNE T. WATSON (Ph.D., Michigan, 1974) has interests in numerical analysis, mathematical programming, bioinformatics, and data science. He has been involved with the organization of HPCS since 2000.

THOMAS C. H. LUX is a Ph.D. student at VPI&SU studying computer science under Dr. Layne Watson.

JON BERNARD is a Ph.D. student at VPI&SU studying computer science under Dr. Kirk Cameron.

BO LI is a senior Ph.D. student at VPI&SU studying computer science under Dr. Kirk Cameron.

LI XU is a Ph.D. student at VPI&SU studying statistics under Dr. Yili Hong.

GODMAR BACK (Ph.D., University of Utah, 2002) has broad interests in computer systems, with a focus on performance and reliability aspects of operating systems and virtual machines.

ALI R. BUTT (Ph.D., Purdue, 2006) has interests in cloud computing and distributed systems.

KIRK W. CAMERON (Ph.D., Louisiana State, 2000) has interests in computer systems design, performance analysis, and power and energy efficiency.

YILI HONG (Ph.D., Iowa State, 2009) has interests in engineering statistics, statistical modeling, and data analysis.