HYPPO: A Hybrid, Piecewise Polynomial Modeling Technique for Non-Smooth Surfaces

Travis Johnston, Connor Zanin, and Michela Taufer
Department of Computer and Information Sciences
University of Delaware
Newark DE, 19711
Email: {travisj},{cnnrznn},{taufer}@udel.edu

***Abstract*—The number and diversity of tunable parameters in applications makes predicting settings that achieve optimal performance challenging. Complicating matters is the fact that resources are increasingly shared among computational tasks (for example, in cloud environments). Choosing *any* setting that yields near-optimal performance runs the risk of overusing shared resources. Building accurate models that capture the complicated interplay of parameters is crucial in order to maximize performance with minimal resource impact.**

Traditional techniques tend to fall short when modeling performance. One reason is that performance surfaces are often irregular but most traditional techniques are designed to produce smooth models. In this paper we introduce a hybrid modeling technique that combines the strengths of surrogate-based modeling (SBM) and k nearest-neighbor regression (kNN) into a single method called HYPPO. The hybrid method is a piecewise polynomial model composed of many small, local models. We demonstrate that HYPPO significantly improves overall prediction accuracy compared with SBM and kNN.

***Index Terms*—Multiple linear regression, k nearest neighbor, performance optimization, Apache-Spark.**

I. Introduction

Every parallel application has tunable parameters that affect the application's run time. Some of these parameters are resource related (e.g., the number of nodes or cores used or the amount of memory dedicated to the application). Other parameters affecting performance are software related (e.g., the number of MPI threads to create, the number of parallel map or reduce tasks in a MapReduce job, the method for partitioning the data for processing). Understanding how these parameters affect performance is a challenge for modern modeling techniques because the relationships are often complex and the performance surfaces are frequently not smooth but most techniques typically are designed to produce smooth (differentiable) models. Figure 1 shows an example of a non-smooth performance surface for the word count benchmark on a Spark platform; two parameters affecting data partitioning and memory allocation substantially impact the overall execution time. Being able to model complex performance landscapes like the one in Figure 1 without extensively sampling the parameter space is becoming increasingly important as resources are increasingly shared in data centers and the cloud and job schedulers consider more than just requested time for a job and number of CPUs. Traditional techniques such as surrogate-based modeling (SBM) take a global view of

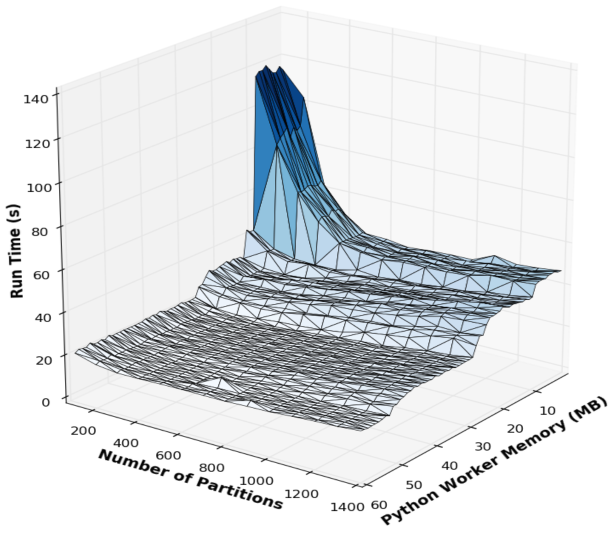


Fig. 1: Visualization of the non-smooth performance landscape generated by sampling 840 parameter settings on the word count benchmark on a Spark platform.

the data, building a single, complex surface that cannot capture surface discontinuities accurately. Other techniques such as k nearest-neighbor regression (kNN) take an intentionally local view of the data and build many simple local models, each of which describes a small portion of the space at a high sampling cost. When dealing with non-smooth performance surfaces, there is the need for combining the advantages of both traditional methods while mitigating their limits.

Motivated by this need, we introduce a general-purpose modeling method, HYPPO, designed to effectively and accurately model non-smooth performance surfaces without the need for extensive sampling. The contributions of this paper are threefold. First, we present the HYPPO modeling technique that exploits the locality of the kNN model while making its predictions as effective as those of SBM. Second, we provide a side-by-side comparison of the accuracy of the HYPPO model with SBM and kNN. Last, we present an in-depth analysis of the predictive power and cost of building a HYPPO model for a surface such as the one in Figure 1.

The rest of the paper is structured as follows: Section II describes two traditional modeling techniques and their limits;

Section III presents the HYPPO model; Section IV compares HYPPO with the two techniques and illustrates the predictive power of HYPPO; Section V describes related work; and Section VI concludes the paper.

II. Background and Limits

Two traditional techniques for modeling performance landscapes take orthogonal approaches: surrogate-based modeling (SBM) and k nearest-neighbor regression (kNN).

A. Surrogate-Based Modeling

Surrogate-based models are built by fitting a polynomial surface to sampled data points by using least squares regression. Suppose that we have n observed run times (data points) with various parameter settings for an application such as the Word Count example in Figure 1. The application settings, or application parameters, for each data point (e.g., number of partitions or Python worker memory) are independent variables and are stored in vectors $\vec{x}_1, ..., \vec{x}_n$. The performance metric, the dependent variable, for each data point (e.g., application run time or I/O bandwidth) is stored as $z_1, ..., z_n$.

Polynomial models that fit the n data points contain one variable for each application parameter in the data. In the example above, we have two application parameters: the number of RDD partitions we call x and the amount of Python worker memory we call y. A degree d polynomial (with two application parameters) has the following form:

$$z(\vec{x}) = \beta_1 + \beta_2 x + \beta_3 y + \beta_4 x^2 + \beta_5 xy + \beta_6 y^2 + ... + \beta_D y^d.$$

The method of least squares regression is used to determine the *best* values of β_i. Least squares regression requires the construction of three matrices:

$$X = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & \cdots & y_1^d \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 & \cdots & y_2^d \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & x_n^2 & x_n y_n & y_n^2 & \cdots & y_n^d \end{bmatrix}$$

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \text{ and } \beta = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_D \end{bmatrix},$$

where β is a matrix containing the unknown coefficients of the polynomial. The coefficients contained in β are determined by solving the normal equations:

$$(X^T X)\beta = X^T Z \tag{1}$$

or equivalently by computing $\beta = (X^T X)^{-1} X^T Z$. We note that Equation 1 has a unique solution if and only if $X^T X$ is an invertible matrix. The number of data points collected, the manner in which the data is sampled, and the complexity (degree) of the polynomial all play a role in determining whether the matrix is invertible.

Using SBM, one can build many different polynomial surfaces to model the same application's performance—a different model for each degree. A natural question is which model is the best. To answer that question, we need a metric of *goodness*. One option is to build a model for each degree and then predict the values of the data points collected measuring the total sum of squares error (SSE):

$$SSE = \sum_{i=1}^{n} (z_i - z(\vec{x}_i))^2.$$

The problem is that increasing the degree of the model $z(\vec{x})$ never increases the SSE; effectively, this indicates that larger polynomial degrees are always better. Unfortunately, as the degree of the polynomial grows, the model becomes increasingly complex and tends to overfit the data, quickly becoming a poor predictor of unobserved points.

One common approach to remedy this problem, which was used in [1] and [2], is to apply k-fold cross validation to select the best polynomial. Cross validation begins by randomly partitioning the sampled data into k sets of equal size. The models are built by using $k-1$ of the sets (learning sets), and the models are scored by their accuracy in predicting the reserved set (testing set); the score is the SSE of the predictions. This process is repeated k times so that each set serves as the reserved testing set once. The process is then repeated, beginning with a new random partition of the data into k sets, as many times as desired. For each polynomial degree an average accuracy is determined from the total SSE when predicting points in the testing sets. The best polynomial (degree) is the one that had the best average SSE over all predictions. Once the *best* degree has been determined, the model is built by using a polynomial of that degree and all n data points.

B. k Nearest-Neighbor Regression

The k nearest-neighbor regression technique builds many simple models from local data. The technique is typically selected to model data when one believes that the data points nearest the point one wishes to model are the most relevant and that points farther away have little or no impact on the point in question. Suppose that we have n sampled data points from n runs of our application of interest. As with SBM, the application parameters for each data point are stored in the vectors $\vec{x}_1, ..., \vec{x}_n$. The performance metric for each data point is stored as $z_1, ..., z_n$. To predict the performance (e.g., the run time) of a specific choice of $\vec{x}$, the kNN regression selects the k nearest neighbors of $\vec{x}$ in the sample data, $\vec{x}_{i_1}, ..., \vec{x}_{i_k}$ with their associated run times $z_{i_1}, ..., z_{i_k}$. The run time of $\vec{x}$ is modeled as the average value of the run times of the nearest neighbors:

$$z(\vec{x}) := \frac{1}{k} \sum_{j=1}^{k} z_{i_j}.$$

The kNN model can be tuned in a variety of ways. One particularly important consideration is how distances between data points are measured. Often, the Euclidean distance is sufficient, however, choosing a different distance metric can be desirable. Ideally, the distance metric, d, should be a good indicator of impact or relevancy; in other words, if we want to model the performance at point $\vec{x}$ and $d(\vec{x}, \vec{x}_1) < d(\vec{x}, \vec{x}_2)$, then we hope that $\vec{x}_1$ has more impact on the performance at $\vec{x}$ than $\vec{x}_2$ (since it is closer). Essentially, the distance metric will determine which points are the nearest neighbors and therefore has a large impact on the kNN model. Another consideration is the choice of k. Are just a few data points sufficient (small k), or are more points needed? Instead of computing a simple average, one can compute a weighted average. The weighted average has the effect of smoothing out the model somewhat and assigning even more weight (or influence) to the nearest points. Note that the kNN model is equivalent to multiple linear regression using only k data points and forcing the polynomial to be degree 0 (i.e., constant). Our HYPPO model tunes this model by allowing the local model to be more complex, that is, by allowing the local polynomial models to have degree larger than 0.

C. Limits of Traditional Modeling Techniques

Both SBM and kNN have been successfully used for modeling an application's performance. However, both methods have drawbacks or limitations that can stifle their predictive power when applications exhibit irregular patterns (e.g., discontinuity performance values). The SBM suffers from two limits. First, the model produced by SBM is very smooth—that is, differentiable any number of times. In many modeling applications this feature is desirable; however, performance data is rarely smooth, and expecting a smooth model to accurately reflect a discontinuous performance surface is unrealistic. Second, SBM is a global model; every data point affects predictions of all other data points. This feature is desirable when there is a clear global trend; however, in our observations of performance data we do not see such tends. We therefore expect local models to be better predictors. The biggest limiting factor of kNN regression is that the local model it builds—a simple average—is too simple; it fails to capture even simple linear relationships among the observed data. Figure 2 demonstrates how these two methods compare when predicting a piecewise linear (and discontinuous) curve. While both methods manage

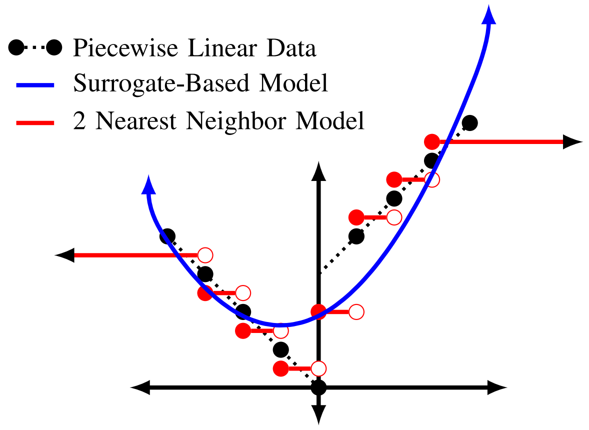


Fig. 2: Illustration of SBM and kNN applied to a simple, piecewise linear function.

to capture the coarse shape of the observed data (solid black points), neither method is a good predictor of the unseen data (dotted line). SBM produces a parabola that captures the general shape; however, the data is piecewise linear, not quadratic, so the parabola is not a good fit. The kNN model suffers because the local model is too simple (average) to recognize and capture the locally linear relationship.

III. HYPPO MODELING TECHNIQUE

The HYPPO modeling technique combines the advantages of both SBM AND kNN while mitigating their limits. To build the performance model with HYPPO, we start with n sampled data points, $\vec{x}_1, ..., \vec{x}_n$, with their associated run times, $z_1, ..., z_n$. The vector $\vec{x}_i$ represents the choice of application parameters, and the value z_i is an associated dependent performance value (e.g., the run time of the application). We want to predict the value of z (e.g., the run time) for a set of parameters $\vec{x}$. Following the technique of kNN, we first find the k nearest neighbors of $\vec{x}$ from among the sampled data points: $\vec{x}_{i_1}, ..., \vec{x}_{i_k}$. If $\vec{x}$ belongs to the data set, we do not include it among the nearest neighbors. Then, using the data of the k nearest neighbors, we follow the steps previously outlined and build a local surrogate-based model using a polynomial whose degree was chosen by the k-fold cross validation. Since k, our number of local samples, is small, we choose the number of folds to be equal to the number of nearest neighbors; the k data points are partitioned into k singleton sets. Because this partitioning is unique (up to order), it is done only once. Each of the k data points serves as the testing set (point) exactly once, and the models are built off of the remaining $k - 1$ points. This type of cross validation is commonly referred to as *leave-one-out* cross validation and typically is applied when the data size is small. We use this local surrogate-based model to predict the run time at $\vec{x}$. This local model is also used to predict any (and all) points in the parameter space whose k nearest neighbors are the same as the k nearest neighbors of $\vec{x}$. There may be many points, a few points, or just $\vec{x}$ itself.

The global surface we construct using the local models is piecewise polynomial, reminiscent of splines but without the guarantee of differentiability or even continuity. Essentially, the difference between this technique and traditional SBM is that instead of building a single polynomial of relatively high degree to model the entire space, we build many polynomials of relatively low degree, each of which models only a small local region. If we considered building only local models of degree 0, then this technique would be exactly the traditional kNN regression. Allowing the degree of the local model to be larger than 0 but still small allows the kNN model to become much more flexible and is a natural relaxation of kNN itself. Figure 3 demonstrates how the HYPPO method captures the data for the example in Figure 2. We see that HYPPO is able to capture the linear relationship of the data on either side of the discontinuity at $x = 0$, giving a perfect fit for much of the data. While it is not a perfect fit of all the data (as a whole), it is a clear improvement over SBM or kNN used in isolation (as showed in Figure 2).

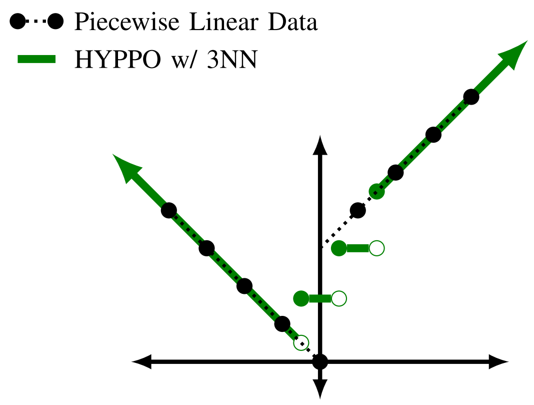


Fig. 3: Illustration of the HYPPO model generated with the same data as Figure 2.

IV. Results

In this section, we assess the predictive power and cost of HYPPO; we compare HYPPO with SBM and kNN.

A. Experimental Setup

As an example of non-smooth performance surface, we use the performance discontinuity associated to memory parameters in Figure 1. We generate performance data for the figure by running the word count benchmark on top of Apache-Spark. For our case study we consider two Spark parameters that affect the memory management of each run: `spark.python.worker.memory` and the number of partitions into which the data is split. The data set is approximately 5 GB containing 100k distinct words with repetition. Our experiments are carried out using four nodes of Comet, an XSEDE resource housed at the San Diego Supercomputer Center. Each node has 24 cores, a local SSD, and 128 GB of RAM. To evaluate the efficacy of our modeling, we initially sampled 840 points in the parameter space: Python worker memory from 1 MB to 60 MB in increments of 1 MB and the number of partitions from 100 to 1,400 in increments of 100. For the rest of the paper we call this the entire parameter space, or the exhaustive set of data. All the models we build are compared against this data for accuracy. The raw data is shown in Figure 1. Note that the relevance of the experiment is not in its data size or number of nodes used but in the nature of the performance pattern that shows the typical non-smooth performance landscape associated with the memory hierarchy of the system. While simple, this benchmark represents the type of performance discontinuity we can capture with HYPPO. Moreover, although we test HYPPO on two parameters for convenient visualization, our modeling techniques can easily scale to more parameters.

B. Modeling Complete Datasets

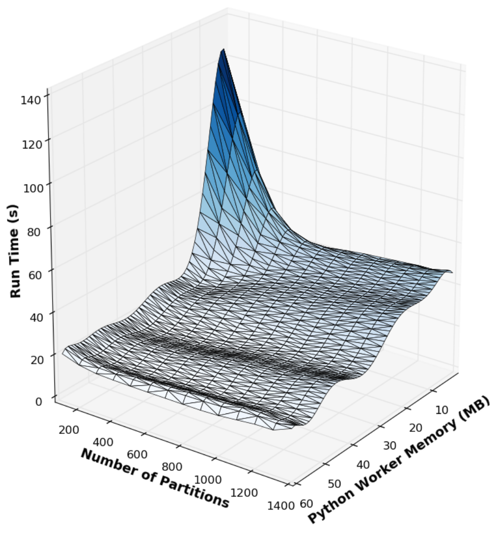
In this section we tackle the question of how well we can model the exhaustive performance space (in this case the run time of the Spark-based Word Count benchmark) when we have the entire data set at our disposal (in this case 840 data points). With that goal in mind we first construct the two traditional models: a traditional surrogate-based model and a model using k nearest-neighbor regression. We compare the two models with the HYPPO model.

We display each of the three models as follows. First, we plot the 3D model surface obtained from the method (SBM, kNN, or HYPPO). From these figures one can visually gauge the accuracy of the model by comparing it with the raw data in Figure 1. Second, we plot a black-gray-white figure that visually depicts where in the parameter space the models perform best (and worst). In these plots, there is a rectangle for each point in the parameter space. The rectangle is colored white (best) if the model differs from the raw data at that point by less than 5%. The rectangle is colored gray (good) if the model differs from the raw data by more than 5% but less than 10%. The rectangle is colored black (bad) if the model differs from the raw data by more than 10%. In all cases we compute the percentage error with

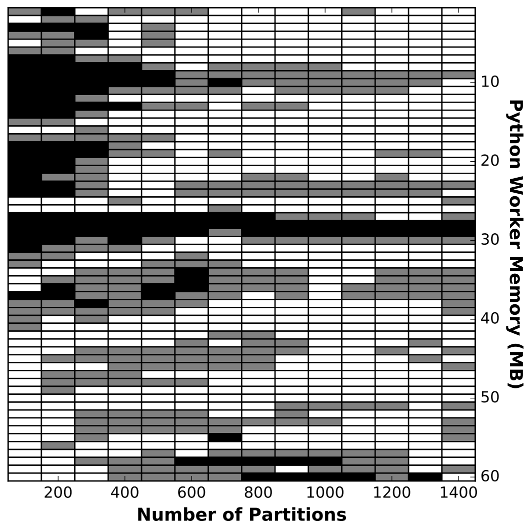
$$\%\text{Error} = 100 \cdot \frac{|z_{\text{real}} - z_{\text{model}}|}{z_{\text{real}}}.$$

Figure 4a shows the traditional surrogate-based model. In this case, the best surrogate-based model found is a degree 9 polynomial in two variables. The model surface constructed by using traditional surrogate-based modeling shares a number of features with the real data. For example, we note the prominent spike in run time when both the number of partitions and the Python worker memory are low. Additionally, the surrogate model produces two or three ridges similar to those seen in the real data. However, two major differences are evident. First, the surrogate model appears to be much smoother than the real data. This feature is essentially unavoidable because the surrogate model is a multivariate polynomial and therefore will be both continuous and differentiable. Second, the real data contains several regions that are relatively flat—for example, the region of the plot with the shortest run time. This feature is impossible to duplicate with a polynomial unless the degree is at most 1 (i.e., the polynomial is a hyperplane). Since the real data is clearly not planar, however, surrogate-based modeling will not result in a degree 1 polynomial.

Figure 4b depicts which regions are modeled well with the surrogate-based model using the entire data set. We observe that 55.8% of the space is white (i.e., modeled accurately within 5% error) and 87.3% of the space is either white or gray (i.e., modeled accurately within 10% error). Notice, however, that the lines corresponding to Python worker memory equal to 10 MB and 30 MB are not modeled well. These lines correspond to sudden drops in run time that were observed the raw data. We refer to these as ridges; and in the raw data, these areas of the surface clearly are not smooth. Since polynomials are very smooth, it is not surprising that this polynomial surface fails to capture the ridge phenomena. Additionally, the raw data in the region of the Python worker memory of at least 35 MB (up to 60 MB) is relatively flat. Only linear polynomials have any regions that are flat. Again, it is not surprising that this nonlinear polynomial fails to capture well

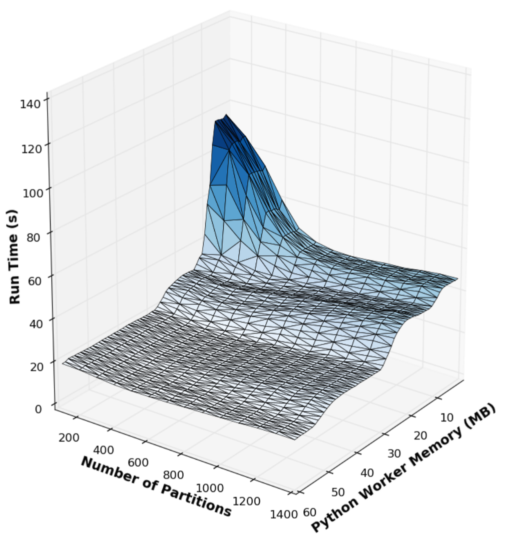


(a) The performance landscape produced by traditional surrogate-base modeling technique

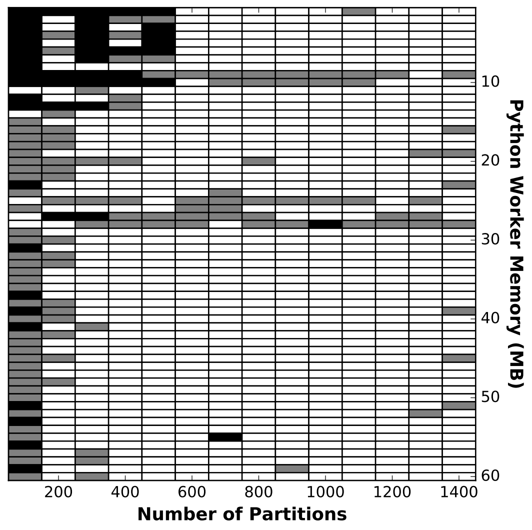


(b) The accuracy over the entire parameter space of the traditional surrogate-based model

Fig. 4: Traditional surrogate-based model, a bivariate polynomial of degree 9, built from the exhaustive data set with its associated accuracy. The model is within 5% of the raw data on 55.8% of the parameter space (white) and within 10% of the raw data on 87.3% of the parameter space (white or gray).



(a) Performance landscape produced by traditional k nearest-neighbor model with $k = 10$



(b) Accuracy over the parameter space of k nearest neighbors with $k = 10$

Fig. 5: Traditional k nearest-neighbor model with $k = 10$ built from the exhaustive data set with its associated accuracy. The model is within 5% of the raw data on 79.7% of the parameter space (white) and within 10% of the raw data on 94.1% of the parameter space (white or gray).

that flat region.

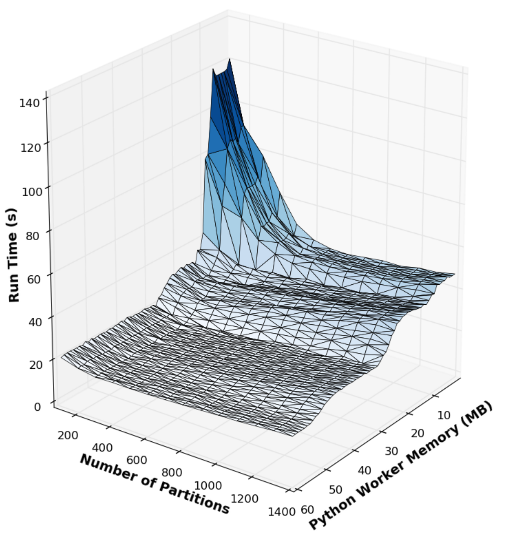
Figure 5a shows the traditional k nearest-neighbor regression model using $k = 10$. This model captures the major features of the exhaustive data we collected. We again see the spike when both variables are relatively small; we see two distinct ridges; and we see the relatively flat regions that appear in the raw data. Figure 5b clearly shows that over the flat regions, the kNN model performed significantly better than did SBM; this performance is explained by the fact that kNN is computing an average value over a region that is nearly constant. Overall, the kNN model predicted 79.7% of the parameter space within 5% of the raw data and 94.1% of the parameter space within 10% of the raw data. These results are an improvement over SBM, suggesting that locality plays an important role in the performance landscape. However, this model encountered the same problem as did SBM along the ridges—albeit somewhat less pronounced. The kNN model also significantly underestimated the values of the peak (when both parameters were small). This result can be expected, however, because kNN will always underestimate the largest values (because it averages smaller values to make the prediction).

The HYPPO model generates a piecewise polynomial surface and does not suffer from the same limitations as the traditional methods do. One strong advantage of HYPPO is that when the real data is flat, the method can choose to use a low-degree polynomial to build a local model and when the local region is more complex, the method may choose a higher-degree polynomial. Thus, HYPPO provides the flexibility to model both smooth, flat regions and more complex regions. The resulting model built with the hybrid technique is shown in Figure 6a. The model is remarkably similar to the real data. The most notable difference is that the model appears somewhat smoother than the data—for example, at $(700, 55)$ the raw data has a small peak (possibly noise) that is smoothed out in the model. We note that when we choose the k nearest neighbors of a point, if the point was sampled—which is the case for every point when we have all the data—then we exclude that point from the nearest-neighbors set; building the local model for point x never uses knowledge of the run time of x to build the local model. This model is built by using the 10-nearest neighbors. We initially chose 10 because it would provide enough points to build polynomials of degree 0, 1, or 2. Later, we explore the effect that the specific choice of k has on the results.

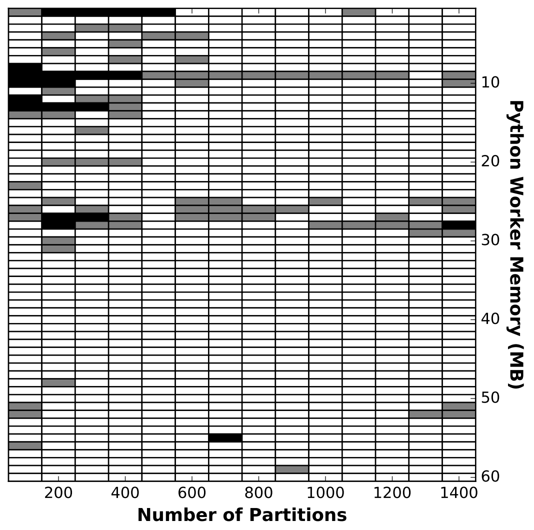
Figure 6b shows how accurate the HYPPO model is over the entire parameter space. Clearly, our hybrid model captures the performance landscape much more accurately than the traditional surrogate-based model does. Our HYPPO model also outperforms the kNN model, although this result is less obvious from the figure. We observe that 89.2% is modeled very accurately by the HYPPO model as opposed to 55.8% for the surrogate-based model or 70.9% for the kNN method and that 97.6% of the space is accurately modeled within 10% of the observed run time as opposed to 87.3% for the surrogate-based model or 94.1% for the kNN method. These results show that the potential accuracy of the HYPPO model, using complete data, is much higher than that of either traditional method.

C. Modeling Incomplete Datasets

In this section we investigate how the accuracy of the model improves with the number of random samples. For each n in $\{32, 48, 64, 80, 96, ..., 256\}$ we select n data points at random from our exhaustive data set, simulating what would occur had only those n data points been sampled. We then construct the three models: traditional surrogate-based model (SBM), traditional k nearest neighbors (kNN) with $k = 10$, and the HYPPO model with $k = 10$. For each of these models we compute the percentage of the parameter space that is accurately modeled. Specifically, we use the models to predict each point in the exhaustive data set of 840 points, and we compute what fraction of the predictions are within 5% of the observed run times and what fraction of the predictions are within 10% of the observed run times. For each n, we repeat this process 100 times and compute the average accuracy of the 100 different models. Figure 7 shows the accuracy of the HYPPO model the at 5% and 10% threshold levels. In the figure, the median accuracy is plotted along with error bars that capture 90% of the models created. We observe that the average accuracy improves quickly when only a few samples have been taken; but after 112–128 samples (out of 840) we observe little improvement with additional sampling.



(a) Performance landscape produced by the HYPPO model using the 10 nearest neighbors



(b) Accuracy over the parameter space of the HYPPO model

Fig. 6: HYPPO model built from $k = 10$ nearest neighbors built from the exhaustive data set with its associated accuracy. The model is within 5% of the raw data on 89.2% of the parameter space (white) and within 10% of the raw data on 97.6% of the parameter space (white or gray).

In Figure 8 we compare the performance of the HYPPO model with the SBM at the 5% threshold (i.e., we compare the fraction of the predictions that are within 5% of the

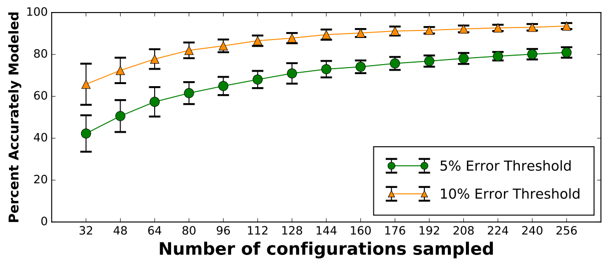


Fig. 7: Fraction of the parameters space that is modeled well using the HYPPO method, as the number of sampled points increases

observed run time). We make two important observations

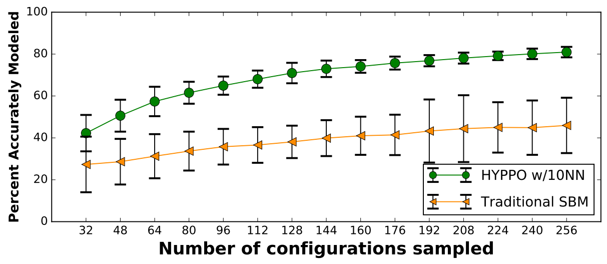


Fig. 8: Comparison of the overall accuracy of traditional surrogate-based modeling and HYPPO as the number of sampled points increases

from this figure. First, the typical performance of the HYPPO method is much better than the typical performance of SBM, as witnessed by the larger median accuracy of the HYPPO model. Second, the variation is much smaller with the HYPPO models (i.e., the error bars are much tighter). This means that on average, the HYPPO models are more consistently accurate than the surrogate-based models are.

In Figure 9 we compare the performance of the HYPPO model with the traditional k nearest-neighbor model taking $k = 10$. As before, the accuracy is compared at the 5%

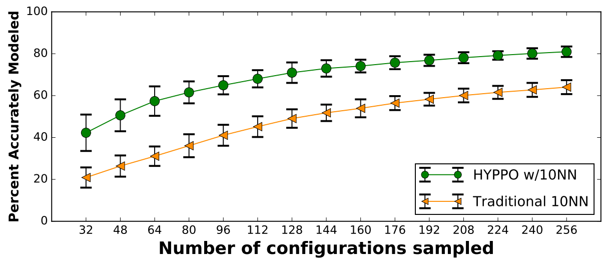


Fig. 9: Comparison of the overall accuracy of the traditional k nearest-neighbor model (with $k = 10$) and the HYPPO method as the number of sampled points increases.

threshold. We observe in the figure that the HYPPO model is consistently more accurate than the traditional kNN regression model is. Notice that in this case the error bars of kNN and HYPPO are both fairly tight. This implies that the accuracy of both models is consistent over the 100 random samples. This is quite a contrast from the variation from model to model of the surrogate-based models.

D. Impact of k on HYPPO's Accuracy

In all our previous results we demonstrated how the HYPPO model with $k = 10$ outperforms the traditional methods. One might naturally ask how much impact the choice of $k = 10$ had on the accuracy of the HYPPO models and whether the choice of k mattered at all. Figure 10 shows how the accuracy of HYPPO models varies as we vary k. In this

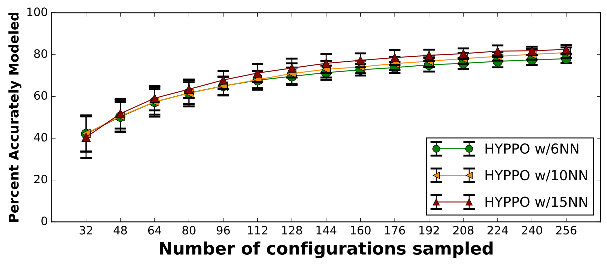


Fig. 10: Comparison of the accuracy of HYPPO with three values of k: 6, 10, and 15.

figure, as in Figure 7, Figure 8, and Figure 9, we build 100 models from n random samples of the exhaustive data for $n \in \{32, 48, 64, 80, ..., 256\}$. From the figure we clearly see that the choice of k from among 6, 10, and 15 has little impact on the accuracy of the typical models they build.

We emphasize that extreme choices of k will have a detrimental impact on the accuracy of the HYPPO model. For example, if k is very small (e.g., 3 or less), then the HYPPO method becomes equivalent to the traditional kNN. The reason is that X^TX (from Equation 1; that is, the normal equations) will not be invertible if the degree of the local polynomial is larger than 0 (i.e., constant). On the other hand, if k is very large (e.g., close to n), then the HYPPO method nearly resembles the traditional surrogate-based model. In either case, we have previously observed that the performance in terms of accuracy degrades.

The choice of k also affects the computational cost of querying the HYPPO model. The larger the choice of k, the greater the computational cost incurred by querying the model. Specifically, the choice of k alters the sizes of the matrices in Equation 1, the normal equations. X is a $k \times D$ matrix instead of an $n \times D$ matrix with full data, where D is the number of monomials in the polynomial of degree d. In addition, the choice of k determines the range of degrees of local polynomials that can be built, which increases the cost of the cross validation; the choice of k also affects the number of partitions in leave-one-out cross validation. Figure 11 illustrates the rate of growth of the model querying time as a function of k. To construct this figure, we build the HYPPO model using the exhaustive data set and query the

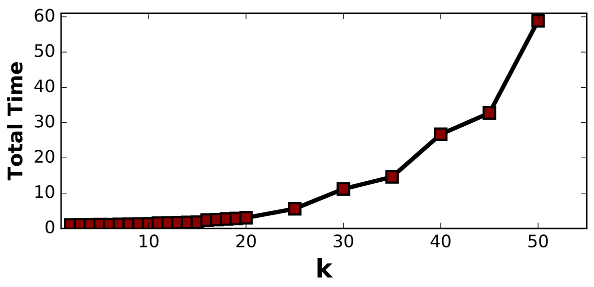


Fig. 11: Illustration of the relative cost of querying the HYPPO model 840 times as the choice of k increases.

model at all 840 points. We do this over a range of k values and plot the resulting run time. The run time is normalized so that when $k = 2$, the run time is considered a single time unit (i.e., a HYPPO model with $k = 40$ requires approximately 30 times longer to query than a HYPPO model with $k = 2$). From the figure we conclude that we can safely leave k at 10 (at least in our 2D case) and build accurate models without paying undue computational cost.

V. Related Work

Considerable work has been done on performance modeling; see, for example, [1], [2], [3], [4], [5], and [6]. Typically some variation of multiple linear regression or surrogate-based modeling (and gradient descent methods) is used to search for a single optimal parameter setting. While these models have been shown to work well to predict a single optimal setting, our primary goal is to accurately model the performance over a wide range of parameters, so that an optimal setting can be chosen from among many optimal settings, resulting in minimal resource requirements. As previously demonstrated, traditional techniques for modeling do not perform as well as HYPPO when the goal is to understand the complex interaction of parameters.

Our HYPPO technique fits into the broad class of regression algorithms called local linear regression [7]. Methods such as LOWESS [8] are used in many data visualization settings to draw smooth curves connecting data or complicated, but smooth, trend lines. In a similar light our model can be viewed as weighted local linear regression using a non-smooth weighting function, specifically an indicator weighting function, 1 if the data point is one of the k nearest and 0 otherwise. As far as we can determine, this particular method of weighting has not been used for local linear regression. Typical weighting functions (e.g., Gaussian weighting functions) are smooth. Using smooth weighting functions results in a smooth model surface. In our case, we do not want a smooth model surface because the real data we model is not smooth.

In at least one case [9] surrogate-based modeling, or linear regression, was used in conjunction with k nearest-neighbor regression. The combination of these two techniques in that work is significantly different from our HYPPO model, however, in that it used surrogate-based modeling to determine the best distance metric to use for the k nearest neighbors. Indeed, one direction in which HYPPO might be improved would be to use a modeling technique such as the one proposed in [9] in order to more methodically and systematically scale the distance metric.

VI. Conclusions

In this paper we introduce HYPPO, a generals-purpose modeling method designed for modeling non-smooth surfaces. For the case study considered in this paper, we show that the HYPPO technique improves the overall accuracy of models from 55.8% in the case of SBM or 79.7% in the case of kNN to 89.2% when all 840 data points were available to build the model. We also show that the HYPPO technique significantly outperforms either SBM or kNN when limited data sets are used. In particular, when sampling less than 12% of the total data, the HYPPO model is able to predict accurately more than 60% of the 840 data points. In contrast, SBM accurately predicts only about 35% of the points, and kNN accurately predicts only about 40%.

Acknowledgments

This work is supported by NSF grants #1446794 and #1513025. XSEDE resources, supported by NSF grant ACI-1053575, were used to obtain the performance data. The authors want to thank Dr. Pietro Cicotti from SDSC for his support with the XSEDE resources. We recognize the importance of reproducibility and want our work to be used and improved upon. To that end, we have made all our code and run-time data available on github at https://github.com/TauferLab.

References

[1] M. Matheny, S. Herbein, N. Podhorszki, S. Klasky, and M. Taufer, "Using Surrogate-Based Modeling to Predict Optimal I/O Parameters of Applications at the Extreme Scale," in *Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, December 2014.

[2] T. Johnston, M. Alsulmi, P. Cicotti, and M. Taufer, "Performance Tuning of MapReduce Jobs Using Surrogate-Based Modeling," in *Proceedings of the International Conference on Computational Science (ICCS)*, 2015.

[3] P. K. Lakkimsetti, "A Framework for Automatic Optimization of MapReduce Programs Based on Job Parameter Configurations," Master's thesis, Kansas State University, August 2011.

[4] K. Wang, X. Lin, and W. Tang, "Predator – An Experience Guided Configuration Optimizer for Hadoop MapReduce," in *Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, December 2012, pp. 419–426.

[5] D. Wu, "A Profiling and Performance Analysis Based Self-Tuning System for Optimization of Hadoop MapReduce Cluster Configuration," Master's thesis, Vanderbilt University, May 2013.

[6] Y. Jin, "Surrogate-Assisted Evolutionary Computation: Recent Advances and Future Challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2009.

[8] W. S. Cleveland, "Robust locally weighted regression and smoothing scatterplots," *Journal of the American Statistical Association*, vol. 74, pp. 829–836, 1979.

[9] H. Hirose, Y. Soejima, and K. Hirose, "NNRMLR: a combined method of nearest neighbor regression and multiple linear regression," *IIAI International Conference on Advanced Applied Informatics*, pp. 351–356, 2012.