Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

CGPT: A CONDITIONAL GAUSSIAN PROCESS TREE FOR GREY-BOX BAYESIAN OPTIMIZATION

Mengrui (Mina) Jiang
Tanmay Khandait
Giulia Pedrielli

School of Computing and Augmented Intelligence
Arizona State University
699 S Mill Ave
Tempe, AZ 85281, USA

ABSTRACT

In black-box optimization problems, Bayesian optimization algorithms are often applied by generating inputs and measure values to discover hidden structure and determine where to sample sequentially. However, in some situations, information about system properties can be available, such as the trajectory of a dynamical system, discrete states executed during a simulation, the model generating the trajectories. In different learning tasks, we may know that the objective is the minimum of functions, or a network. In this paper we consider the case where the structure of the objective function can be encoded as a tree. In particular, each node of the tree performs a computation on the input and based on the outcome, a different branch is chosen. We propose the new Conditional Gaussian Process tree (CGPT) model for "tree functions" to embed the function structure and improving the prediction power of the Gaussian process. We utilize the intermediate information made available at the tree nodes, to formulate a novel likelihood for the estimation of the CGPT parameters under different levels of knowledge of the structure. We formulate the learning and investigate the performance of the proposed approach with a preliminary investigation. Our study shows that CGPT always outperforms a single Gaussian process model.

1 INTRODUCTION

Bayesian optimization (BO) is commonly used in complex systems to model expensive black-box functions (Frazier 2018). In their traditional implementation, BO procedures treat the function to be optimized as a black box function and, given an input location, the value of the function at that location is used to build a surrogate of the function, without using any information about the structure, the properties of the object being optimized. Intuitively, if more information is available to characterize the function, the search procedure should achieve better solutions (Astudillo and Frazier 2021a). A question arises on how to change the surrogate and/or the search procedure in a way that embeds such additional information. In this paper, we focus on the first approach, i.e., embedding the additional information in the surrogate model under the principle that improving the quality of the surrogate will result in improved search performance. In fact, while intuitively this enhanced surrogate should improve the prediction power, in practice models that embed additional structure are more complex to train, thus potentially overwhelming the potential advantage coming from the additional structure.

In this paper, we consider a particular class of functions whose structure can be encoded as a tree where nodes contain functions of the input (generally nonlinear non convex) and based on the value produced by the node operator, determine which child function to apply next (i.e., there are *branching conditions*). Given input $x \in \mathbb{X}$ to the root node, a specific *path* will be executed as a result of the function values at the intermediate nodes. Once arriving at a node, there can be multiple child nodes to go to based on different conditions. If the function value for the intermediate nodes is not a-priori (i.e., it is only available upon evaluation) the CGPT approach will fit a GP for each node in the tree. We assume the structure of the tree is known to us, so does the number of leaf nodes and all the intermediate layers and number of nodes. We will distinguish two scenarios: (1) the intermediate node information is known without simulation. In this case, given x, we will know which leaf function to apply; (2) the intermediate information can only be evaluated by simulation. In this case, fitting GP models on the nodes will give an estimation of the leaf function. In both scenarios, the threshold applied to the functions output at the intermediate nodes for branching are known. For this class of functions, we propose the novel conditional Gaussian process tree (CGPT) that embeds the tree structure with the intermediate condition information (true or predicted based on the scenario). This model leverages the information of the tree structure (number of nodes, levels), and threshold values for the branch selection, to build an improved predictor for the leaf functions. In this manuscript, we focus on the preliminary problem of single level trees, i.e., the case where a root function is defined and a number of possible branches can result from the root.

Example. Figure 1 illustrates the concept of a conditional tree function for a case with 10 conditions and 9 branches resulting in 6 leaf functions. We also provide a 1-dimensional toy case for a single layer tree with a root node and two leaf nodes. The function tree is the following:

$$f(x) = \begin{cases} y_1(x) := \sin(x) & \text{If } x \in B_1 = [0, 0.4] \\ y_2(x) := -\sin(x) & \text{If } x \in B_2 = (0.4, 1] \end{cases} \tag{1}$$

The tree function is discontinuous at $x = 0.4$ as a result of the condition. Figure 2 shows the true function, the prediction produced by a GP that ignores the branch structure and only evaluates input, outputs, and the predictions produced by our CGPT. We can see that our model outperforms the single GP especially in areas where function is discontinuous. The single GP is not able to handle the discontinuity and attempts to connect the jump which causes a big error, while CGPT fits the true function well and is robust to the magnitude of the discontinuity. The MSE is $1\text{e}-7$ for CGPT and $2\text{e}-3$ for the single GP model.

Conditional Gaussian processes are relevant to a broad range of applications. In particular, we present the example of program analysis in the context of Cyber Physical Systems.

Example (Program Analysis for Cyber Physical Systems). Monte-Carlo methods have recently gathered important attention in the field of analysis of software for Cyber Physical Systems. This is a special class of programs since input variables are typically (continuous) physical quantities and the program itself implements a controller. In these scenarios, the identification of bugs within the code can be translated into an optimization problem (Mathesen et al. 2019). In particular, after obtaining the system trajectory from the simulation of a specific *input* (e.g., initial conditions, control, environment), a monitor evaluates how far the trajectory is from a given unsafe set. Such distance is referred to as *robustness*. Negative robustness implies that the controller is eliciting unsafe behaviors. A very important question is: if we know the structure of the program can we identify bugs quicker. It is important now to note how a program execution can be seen as a tree, where at each node a function (known) is performed over the input. The result of this function determines the *path* we take in the code and the path results in a specific robustness function. This is precisely the context of CGPT. Most of the literature in program analysis focuses solely on sampling, ignoring the properties of the robustness. We argue that a better model of the robustness that reflects the tree structure is a better way to formulate the problem.

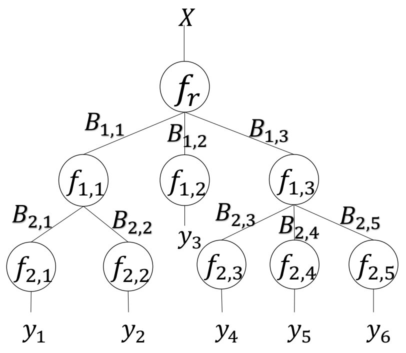


Figure 1: Illustration of a conditional tree function. $B_{i,j}$ represents the conditions of entering j-th node at the i-th level of the tree. Each node is a function of the type $f_{ij}(x)$ and the branch is chosen based on the value produced by $f_{ij}(x)$. For a given value of x only a leaf function is activated $y_h, h = 1,\dots,H$, $H = 6$ in the figure and the evaluation returns the corresponding value $y_h(x)$.

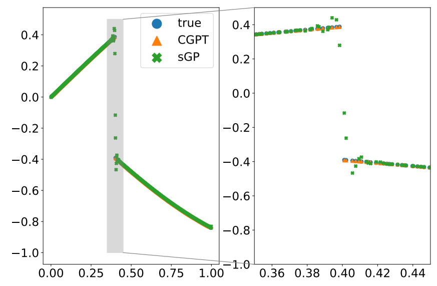


Figure 2: 1-*d* true sinusoidal function (circles) against the predictions produced by the single GP (asterisk) and the CGPT (triangles).

The remainder of the paper is structured as follows: Section 2 discusses several approaches in the literature that are related to the proposed work. Section 3 introduces the mathematical formulation and learning for the proposed model, for the different knowledge scenarios while the preliminary numerical analysis is presented in Section 4. Finally, Section 5 discusses the conclusions and future work.

2 RELATED LITERATURE

Bayesian optimization has become widely used for searches over expensive black-box functions. Applications are in areas such as manufacturing, biology, and machine learning (Brochu et al. 2010; Torun et al. 2018; Sano et al. 2020). While the ultimate objective is optimization, this paper focuses on surrogates modeling and learning. Interested readers may refer to (Shahriari et al. 2015; Frazier 2018; Greenhill et al. 2020) for a review of BO and surrogate models. As BO becomes more popular, the challenges of costly simulations arise, which motivate the research into a class of approaches that leverages *structure* of the objective function. This broad class of approaches is referred to as grey-box BO (Astudillo and Frazier 2021b).

In grey-box BO, additional information about the objective function can be available in different forms, e.g., derivative evaluations, function structures such as composition, nested evaluations or sequential evaluations. Astudillo and Frazier (2021a) models the objective using a function network structure which allows the usage of evaluations from the intermediate nodes. Similarly, composite structures have been dealt with by sequential GP modeling of nested nodes (Astudillo and Frazier 2019). Deep Gaussian Processes can also be interpreted as a network structure, where the number of nodes are related to the input dimensionality, while the output is given by evaluating all nodes in the previous layer (Sauer et al. 2022). Other works leverage the derivative evaluations as additional information from the objective function to improve the model predictions (Wu et al. 2017; Joy et al. 2020). Important effort has also been devoted to approaches that attempt to *learn* the structure of the problem. Cost-aware methods have also been recently proposed, which take into consideration the cost produced by the evaluation of functions under different settings (Lee et al. 2020; Guinet et al. 2020). In Multi-fidelity BO, complex structure dependencies and multiple objective functions and costs are used (Song et al. 2019; Zhang et al. 2017; Imani et al. 2019). More recently, Foumani et al. (2023) combines the cost-aware and multi-fidelity approaches to further utilize low-fidelity sources to learn more information about the objective. Part-X proposed in (Pedrielli et al. 2021) sequentially learns partitions of the input space in a way that allows for the estimation of the 0-level set of the function.

Our work differs from the above approaches in that the additional information about the objective function is a sequence of conditions imposed on the output of intermediate functions of the problem input along a known tree. The function network model in (Astudillo and Frazier 2021a) does not allow a node to have several paths as each node takes the output of the previous layer node by definition. On the other hand, our approach allows the output from the previous layer to enter different child branches based on the different conditions. Different from the composite structure introduced by (Astudillo and Frazier 2019), which is a one-step-composition, the setup of our problem is mode generalized in the sense that it can model multiple compositions if one looks at a single path given all the conditions. In fact, based on the path, completely different functions will be applied to the input.

Our model is also closely related to the class of trees typically used for classification and regression. Using Gaussian processes to fit tree nodes have been shown to improve regression performances by dealing with large datasets. Gaussian process trees have applications in multiple fields such as machine learning (Gramacy et al. 2004; Lee et al. 2015; Will et al. 2011) and complex systems (Fulgenzi et al. 2008; Gramacy and Lee 2008; Civera et al. 2020). Shen et al. (2005) reduces the training and prediction time of Gaussian process regressions using KD-trees on data structures. Similarly, transformations on data using a tree prior to GP approximations are proven in (Bui and Turner 2014) to accelerate the approximations. In classification problems, Achituve et al. (2021) develops the GP-tree model to scale with both large data and a big number of classes. These tree models with Gaussian processes have also shown to be used in many applications (Lee et al. 2015; Civera et al. 2020). Treed partitioning of Gaussian Process models was used in statistical inference by Gramacy and Lee (2008) where the input space was divided to fit independent GP models on each region. The overall prediction was based on an average of all possible trees using a Monte Carlo Markov Chain method. (Park 2022) proposed a Jump GP where local GPs are fit on sets of local data that are partitioned according to the response of training data, in order to adapt the model to problems where the input regions have discontinuity. Also, the classification and regression tree (CART) (Breiman et al. 2017) determines the classes of input according to their features. Similar to these approaches, our CGPT model inherits a tree structure, but different from the above mentioned approaches where classification is based on the input space, we partition the input according to the evaluations from the parent node, and the condition threshold is known as well as the tree structure.

Finally, mixture of Gaussian processes are share aspects with our study (Stachniss et al. 2008; McDowell et al. 2018; Li et al. 2019). Rasmussen and Ghahramani (2001) uses a divide-and-conquer strategy to fit infinite experts of GPs on smaller data sets and achieve predictions respectively. Similarly, Meeds and Osindero (2005) adapts the infinite mixture framework by Rasmussen and Ghahramani (2001) to improve model efficiency. Gaussian Mixture Model (Reynolds 2009) takes the weighted average of several GPs from the branches as the final prediction. Our model differs in that the input can only enter one path at a time, whereas input of GMM can "enter" multiple models at the same time. Moreover, the goal of our work is to find the optimal path of sampling as opposed to the aforementioned approaches used for regression.

To further articulate the difference between our proposed model and the ones in the literature, we summarize and categorize the above methods into the following three kinds:

(1) Learned partitions via treed GPs: local GPs are fit to separate regions for regression and classification. The trees are learnt, thus cannot directly address problems where the objective function structure is encoded as a given tree with known branch conditions.
(2) Grey-box models with networks structure: a network of functions where the input of a child node is the output of the parent node. This model does not allow encoding conditional (binary) conditions. As a result, no "path" structure can be defined.
(3) Grey-box models with condition-directed paths: a given tree structure with known conditions, encoded as functions of the input, that determine the branch structure. Each path is associated with a different output function. This is the scenario supported by our novel CGPT model.

Our proposed model belongs to the third category, which specifies a given tree structure with paths that depend on conditions. The first category focuses on the task of learning and the tree structure in this case needs to be learned, such as in discontinuous processes, trees are used as a mechanism to sequentially separate the space, whereas we are given the problem encoded as a tree. Trees that are built through learning cannot serve as a structure for simulation, which is what the proposed model is able to achieve. On the other hand, we are also not to be confused with the grey-box models which have a network structure. This category of models contains a flow of the full input from one node to another, thus is incapable of incorporating the concept of "path" into the problem, unlike our model which has known conditions that direct only partial input flow and formulate multiple flows into different paths.

3 STATISTICAL MODELING

In this paper we will start considering a single level tree, where there is one root node and b child nodes. The notation will consider a single index, simplifying the, more general, notation used previously in Figure 1. Let function f_r be modeled by a GP in the root node; the evaluations that result from the root GP are referred to as y_r. Given b branches, each with associated functions $f_1,\dots,f_b$, an input will be associated to a branch according to the intervals $B_1,\dots,B_b$ on the root function value $\mathbf{y}_r$, thus forming b, potentially disconnected, sets of inputs $S_1,\dots,S_b$. Namely, the function can be written as:

$$f(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}), & \text{if} \quad \mathbf{y}_r = f_r(\mathbf{x}) \in B_1 \\ \dots \\ f_b(\mathbf{x}), & \text{if} \quad \mathbf{y}_r = f_r(\mathbf{x}) \in B_b \end{cases}$$

We divided the theoretical derivations of the problem into two phases (scenarios) according to how much information about the root node is available ahead of evaluation of the location value:

(1) The value of the root function is given for all locations *a-priori*, i.e., without requiring evaluation. Equivalently, given and input x, we know what path in the tree corresponds to the location;
(2) The value root test function is not given for all locations *a-priori*, but we are given a Gaussian Process prediction for each location. Equivalently, given and input x, we can predict which path in the tree corresponds to the location;

3.1 Scenario 1: Known *a-priori* Root Function

When the root node test function is known, we can simply define the likelihood function for each branch-associated Gaussian processes i, with $i = 1,\dots,b$, namely:

$$L_i(\mathbf{y}|\mathbf{x},\theta_i) = \sum_{j=1}^{n_i} \log\left[p\left(\mathbf{y}_i|\mathbf{y}_r,\mathbf{X}_j\right)\right] = -\frac{1}{2}\left(y_i-\mu_i\right)^T\left(K_i+\eta_i^2 I_{n_i}\right)^{-1}\left(y_i-\mu_i\right) -$$
$$-\frac{1}{2}\log|K_i+\eta_i^2 I_{n_i}| - \frac{n_i}{2}\log(2\pi), i = 1,\dots,b$$

where n_i is the number of samples in branch i, η_i is the model noise associated with the i-th branch GP. For each of the Gaussian processes, we use the best linear unbiased predictor from (Santner et al. 2013)

$$\hat{\mu}_i = \left(\mathbf{1}^T\mathbf{K}_i^{-1}\mathbf{1}\right)^{-1}\mathbf{1}^T\mathbf{K}_i^{-1}\mathbf{f}_i$$

where $\mathbf{f}_i$ is the true function output associated with branch i. In this paper we adopted the squared exponential kernel for the variance-covariance matrix K_i, namely:

$$K_{i,j,k} = \prod_{l=1}^{d} e^{-(\theta_{i,l}|x_{i,jl}-x_{i,kl}|)^2}$$

where (j,k) are the indexes of the cell in the matrix and refer to locations (x_j,x_k) within the region of the input space S_i implied by branch i, and θ is the d-dimensional smoothing length-scale parameter. When the root function is known a-priori, the branch Gaussian processes can be estimated individually using the form above. The predictor will be:

$$\hat{y}(x) = \sum_{i}^{b} I\left(y_r \in B_i\right)\left(\hat{\mu}_i + r_i^T K_i^{-1}\left(y_i - \hat{\mu}_i\right)\right),$$

where $\mathbf{r}_i = r_j(\mathbf{x}_j)$ for each $r_j(\mathbf{x}_j) = \text{Corr}(Z_i(\mathbf{x}_j), Z(\mathbf{x}_j))$ and $j = 1, \ldots, n_i$ (Santner et al. 2013), for each $Z_i(\mathbf{X})$ realized from a Gaussian process corresponding to branch i. That is, for each new sample, if the evaluation of the root function determines that the sample falls in branch i, the corresponding branch i GP will be used for the prediction.

3.2 Scenario 2: Known Root GP

In the case of known root Gaussian process, we use a pre-trained Gaussian process on a set of given data and consider the hyperparameters to be known in the training phase of CGPT. Hence, these hyperparameters will remain fixed in the subsequent steps of the learning process, i.e., for the branch Gaussian processes training, denoted as conditioning on θ_r. The probability of a sample going into a specific branch is induced by the root GP. It follows intuitively that there is no optimization or any change regarding the root GP during the maximum likelihood optimization of each of the branch Gaussian processes.

For a specific branch i, the log-likelihood can be formalized in this way:

$$\begin{aligned} L_i\left(\mathbf{y}|\mathbf{x}, \theta_i, \theta_r\right) &= \sum_{j=1}^{n_i} \log\left[p\left(\mathbf{y}_i|\mathbf{y}_r, \mathbf{X}_j\right) p\left(\mathbf{y}_r \in B_i|X_j, \theta_r\right)\right] = \sum_{j=1}^{n_i} \log\left[p\left(\mathbf{y}_i|\mathbf{y}_r, \mathbf{X}_j\right) p_j\right] \\ &= -\frac{1}{2}\left(y_i - \mu_i\right)^T \left(K_i + \eta_i^2 I_{n_i}\right)^{-1}\left(y_i - \mu_i\right) - \frac{1}{2}\log\left|K_i + \eta_i^2 I_{n_i}\right| - \frac{n_i}{2}\log\left(2\pi\right) + n_i log(p_j) \end{aligned}$$

Since for each sample, with the pre-trained GP hyperparameters, the probabilities induced are no more than just constants during the optimization phase, we express the constant probabilities as p_j in the above likelihood where the subscript j indicates each sample. The final predictor is given by

$$\begin{aligned} \hat{y}(x) &= \sum_{i}^{b} p\left(\mathbf{y}_r \in B_i\right)\left(\hat{\mu}_i + r_i^T K_i^{-1}\left(y_i - \hat{\mu}_i\right)\right) \\ &= \sum_{i}^{b} p\left(c_i^{\ell} \le \mathbf{y}_r < c_i^u | \theta_{pt}\right)\left(\hat{\mu}_i + r_i^T K_i^{-1}\left(y_i - \hat{\mu}_i\right)\right) \\ &= \sum_{i}^{b}\left(\Phi\left(\frac{c_i^u - \mu_r}{\sigma_r}|\theta_{pt}\right) - \Phi\left(\frac{c_i^{\ell} - \mu_r}{\sigma_r}|\theta_{pt}\right)\right)\left(\hat{\mu}_i + r_i^T K_i^{-1}\left(y_i - \hat{\mu}_i\right)\right) \end{aligned}$$

where c_i^{ℓ} and c_i^u are the lower and upper bounds of the interval defining the region B_i for the i-th branch, and θ_{pt} represent the hyperparameters from the pre-trained (pt) root Gaussian process, which remain unchanged during the training and prediction stages.

Extension to Multi-level trees We provide the procedure of fitting a 2-level tree with one root node, two child nodes, one of which has another two child nodes, referring to nodes f_r, $f_{1,1}$, $f_{1,2}$, $f_{2,1}$ and $f_{2,1}$ in Figure 1. The function can be formulated as:

$$f(\mathbf{x}) = \begin{cases} f_{2,1}(\mathbf{x}), & \text{if} \quad f_r(\mathbf{x}) \in B_{1,1} \quad \text{and} \quad f_{1,1}(\mathbf{x}) \in B_{2,1} \\ f_{2,2}(\mathbf{x}), & \text{if} \quad f_r(\mathbf{x}) \in B_{1,1} \quad \text{and} \quad f_{1,1}(\mathbf{x}) \in B_{2,2} \\ f_{1,2}(\mathbf{x}), & \text{if} \quad f_r(\mathbf{x}) \in B_{1,2} \end{cases}$$

The modeling for the root and first level of child nodes is the same as mentioned above for both scenarios of single-level tree. When it reaches the second level of child nodes, node one of level one becomes the parent node. Thus, according to the given conditions, input that entered node $f_{1,1}$ will be partitioned again into nodes $f_{2,1}$ and $f_{2,1}$ according to the intermediate output from the GP fit on node $f_{1,1}$. Two additional GPs will then be fit on these two nodes. Given an input X, the weight of final prediction associated with leaf node output y_1 will be the multiple of the probability of it entering node $f_{1,1}$ and $f_{2,1}$.

4 PRELIMINARY ANALYSIS

Experimental setup. The preliminary numerical experiments will be presented in a one-depth tree with 2, 4, and 8 child nodes for both the learning scenarios presented in Section 3. The same 2-dimensional samples generated using a Latin Hypercube Sampling on the interval $[0,1]$ were used across all the experiments. We kept the number of test samples to be 2000 for all models. For both scenarios, the root test function is a 2-d parabola: $f(x_1,x_2) = x_2 - x_1^2$. The child node test functions are multiples of the tetramodal function:

$$f(x_1,x_2) = -5(1-(2x_1-1)^2)(1-(2x_1-1)^2)(4+2x_1-1) \times (0.05^{(2x_1-1)^2} - 0.05^{(2x_2-1)^2})^2.$$

Section 4.4 presents a practical test case with known root test function. Data used is 2000 samples generated using uniform sampling. We investigate both *balanced* and *imbalanced* partitions: in balanced cases the measure associated to the branches as volume of the partition is similar across branches, in the imbalanced case there are branches with different measure.

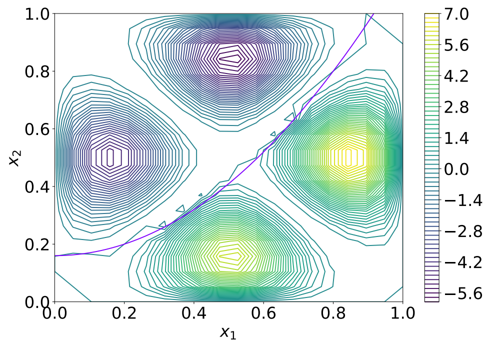
Analysis. For each case (number of nodes, scenario, and balance), we plot the true function and the associated contour overlaid with the root function splitting levels. We then show the performance for the balanced and imbalanced cases for both scenarios. For the practical test case, we experimented both balanced and imbalance cases for scenario 1 by the problem setup. We compare the median of relative error ($\tilde{re}$) and MSE of the CGPT model against the single GP model (sGP), across models and cases, summarized in Table 1 and 2.

4.1 2-node Case

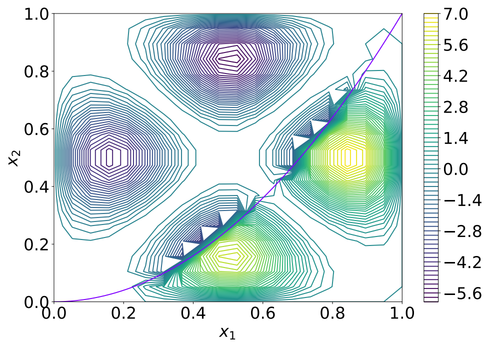
The first case has two branches, with leaf functions being the tetramodal and negative tetramodal, respectively. Figure 3(a) shows the contour plots of the root splitting at $y_r = 0.16$ for the balanced case, and Figure 3(b) shows the case for the $y_r = 0$ split, i.e., the imbalanced data.

Scenario 1. In the case of a-priori known root function, the relative error appears to be similar for both single GP and CGPT, however, the median relative error achieved by our model is one order of magnitude smaller than the one achieved by the single GP (See Table 1). Moreover, we observed that CGPT performance is robust to the imbalanced setting, while the single GP performance deteriorates. In fact, we observed that the relative errors are strongly concentrated around 0 for CGPT in both balanced and imbalanced cases, and larger errors are located in regions where the true function values change sign (see Figure 4). The performance of CGPT is confirmed by the results in Table 2, where we can see how the CGPT achieves orders of magnitude lower MSE.

Scenario 2. In this scenario, no a-priori information of the root function is given. Instead, we use a pre-trained Gaussian process for the root node, so the hyperparameters of the root GP are fixed during the model fitting. Similar performance and behaviors are observed for this case in terms of comparison with the single GP (See Table 1). While we see from Table 2 that the performance of CGPT deteriorates for the imbalanced case, it is still orders of magnitude superior to the single GP.

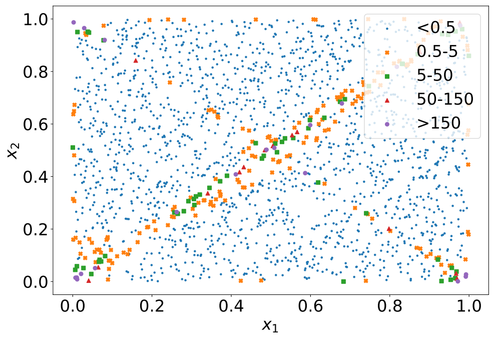


(a) Contours with split at $y_r = 0.16$.

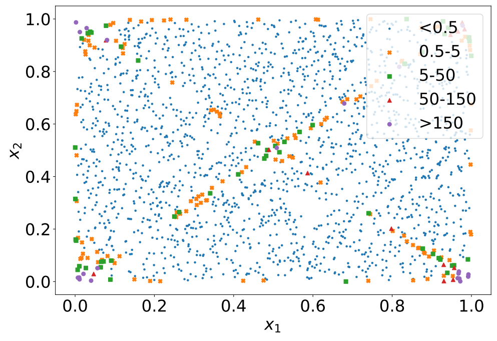


(b) Contours with split at $y_r = 0$.

Figure 3: 2-node true function contour plots overlaid with splitting levels at root node.



(a) *re* distribution of single GP in input space.



(b) *re* distribution of CGPT in input space.

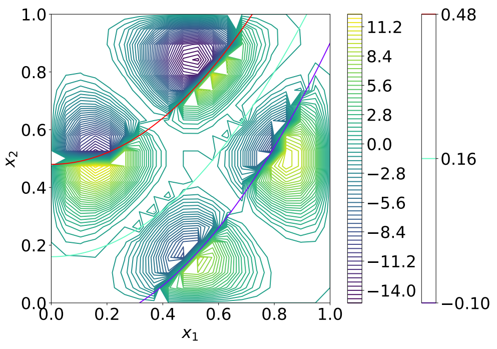
Figure 4: Comparison of *re* distribution in input space for the 2-node tree, balanced case.

4.2 4-node Case

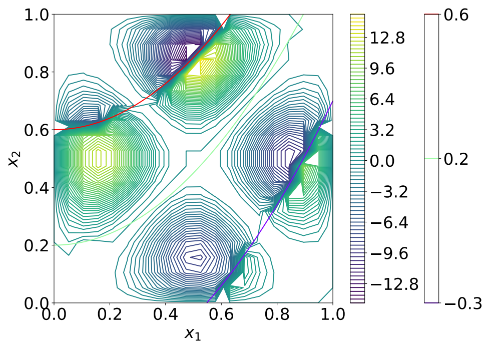
We explore CGPT performance extended to 4 child nodes in one depth. The results are presented by scenario, and divided into balanced and imbalanced cases. For the balanced data, the root function has split values at $\{-0.1, 0.16, 0.48\}$, and the imbalanced split values are at $\{-0.3, 0.2, 0.6\}$. The root function and branch function forms are the same as the 2-node case, with the branch functions multipliers set to $\{-1.5, 1.5, -2.5, 2.5\}$ times the original tetramodal function at nodes $i = 1, \dots, 4$, respectively. We imposed large values of the multiples in order to see how both models would perform with severe function rate changes. The corresponding true functions and contour plots for root splits for balanced and imbalanced partitions are presented in Figure 5 for the balanced and imbalanced cases.

Scenario 1. Similar performance as those in the 2-node case and behaviors were observed for this case in terms of comparison with the single GP. However, we can observe from Table 1 how the increased number of splits leads to deterioration of performance of the single GP, while CGPT is robust to the splits. Table 2 confirms this observation and while the MSE performance of CGPT deteriorates, it is still orders of magnitude superior to the single GP in both balanced and imbalanced cases.

Scenario 2. The results for the case of no a-priori knowledge of the root function is similar, as shown in both tables. As the number of nodes increases and more drastic the difference between branch output values, it is more clear that the CGPT has much better performance compared to the single GP. Moreover, CGPT remains robust to changes in the experimental setting.



(a) Contours with root levels, balanced case.

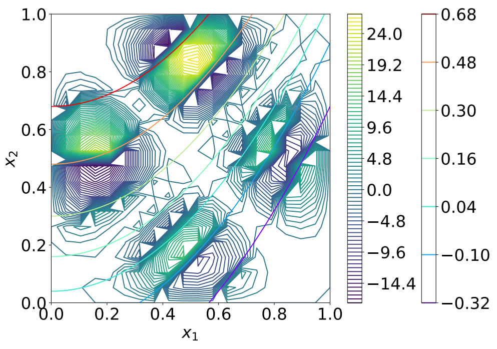


(b) Contours with root levels, imbalanced case.

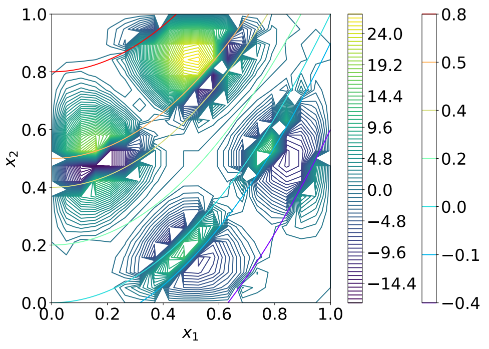
Figure 5: 4-node true function contour plots overlaid with splitting levels at root node.

4.3 8-node Case

In this case, for each of the branch, in addition to the four functions used in the previous 4-node case, we add four more multiples of the tetramodal function, with multipliers $\{-3.5, 3.5, -4.5, 4.5\}$, with the objective to maximize the rate of variation of the leaf node evaluations at the boundaries. The true function and contour plots with splitting level sets for both balanced and imbalanced data is shown in Figure 6. For balanced partition (Figure 6(a)), the splitting levels are at $\{-0.32, -0.1, 0.04, 0.16, 0.3, 0.48, 0.68\}$, while for the imbalanced case (Figure 6(b)) we set the splits at levels $\{-0.4, -0.1, 0, 0.2, 0.4, 0.5, 0.8\}$. The performance for the single GP is much more deteriorated as the structure of the model gets more complex.



(a) Contours with root levels, balanced case.



(b) Contours with root levels, imbalanced case.

Figure 6: 8-node true function contour plots overlaid with splitting levels at root node.

Scenario 1. Similar performance as those in the 4 node case and behaviors were observed for this case in terms of comparison with the single GP (See Table 1). Again, we can observe how the increased number of splits leads to deterioration of performance of the single GP, while CGPT is robust to the splits. Table 2 confirms this observation and while the performance of CGPT deteriorates, it is still orders of magnitude superior to the single GP in both balanced and imbalanced cases.

Scenario 2. Similar performance is observed (refer to Table 1 and 2) for the case of no a-priori knowledge of the root function. As the number of nodes increases and more drastic the difference between branch output values, it is more clear that the CGPT has much better performance compared to the single GP. Moreover, CGPT remains robust to changes in the experimental setting. The single GP has poor accuracy in both scenarios. Larger spread of values were observed, meaning that the relative error is deviated from 0, thus having bad model performance.

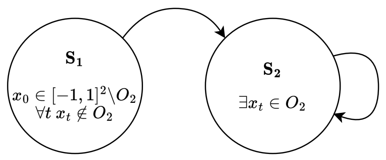

$\mathbf{S_1}$
$x_0 \in [-1,1]^2 \backslash O_2$
$\forall t \; x_t \notin O_2$
$\mathbf{S_2}$
$\exists x_t \in O_2$

Figure 7: Kripke Structure for the 2−state Automaton

4.4 Analyzing a Cyber Physical System

As a second test case, we consider the 2-state hybrid automaton proposed in (Dokhanchi et al. 2015) and is shown in Fig. 7. The search is performed over the initial state defined in a 2-dimensional space $O = [-1,1]^2$. Based on the initial state, the hybrid automaton follows the dynamics shown in Fig. 7. If at any point in time the state of the automaton is in region $O_2 = [0.85, 0.95]^2$, the system is in discrete state S_2; otherwise, it is in state S_1, defined by the region $O_1 = O \setminus O_2$ (see Fig. 7). Note that the system has different, continuous, dynamical behavior in each discrete state. If the system starts in S_1 and enters state S_2, it switches the dynamics, resulting in different robustness functions and system behaviors.

The input space and the region of the states are modified to adjust for the examination of both balanced and imbalanced partitions, while the setup remains the same. As seen in the last two columns of Table 1 and 2, CGPT achieves performance twice better than single GP for both balanced and imbalanced cases. By setup, the automaton system belongs to scenario 1 where root test function is known, in this case, the root function is an identity function of the input.

Summary. We see that from the multiple cases across different number of nodes, our CGPT model is robust under either balanced and imbalanced models. The median of relative error for CGPT model is dominantly better than the sGP under all cases. Plotting the distribution of the relative error across the input space (not shown here in the interest of space), we see that the larger errors are consistently distributed around the areas where the true function has high rates of variation. This corresponds to our intuition that additional information about partition improves the tree model performance whereas the single GP keeps deteriorating as the splits increase. The relative errors can be unusually large due to the denominators that are really close to zero, but we expect to achieve good results of the standard error of *re* as we macro-replicate each experiment.

Table 1: Relative error median ($\tilde{re}$) calculated for single GP model and CGPT model across all experiments.

		2 node		4 node		8 node		CPS	
		balanced	imbalanced	balanced	imbalanced	balanced	imbalanced	balanced	imbalanced
sGP		0.011	0.043	0.122	0.065	0.439	0.498	0.807	0.770
CGPT	known root function	0.005	0.005	0.006	0.005	0.007	0.007	0.407	0.368
	known root GP	0.005	0.005	0.005	0.004	0.005	0.006	*	*

Table 2: Mean squared error (MSE) calculated for single GP model and CGPT model across all experiments.

		2 node		4 node		8 node		CPS	
		balanced	imbalanced	balanced	imbalanced	balanced	imbalanced	balanced	imbalanced
sGP		0.004	0.167	2.361	1.900	13.848	15.363	0.494	0.484
CGPT	known root function	0.0004	0.0004	0.002	0.002	0.008	0.008	0.200	0.209
	known root GP	0.0002	0.018	0.002	0.0004	0.006	0.013	*	*

5 CONCLUSIONS AND FUTURE WORK

We propose the conditional Gaussian Process Tree (CGPT) for conditional tree functions with single level trees. The model can embed information on the tree structure and the branching condition values that describe the structure of the function to model. We provide the model form and the estimation procedure for the scenarios where the root function is known *a-priori* and the case where a surrogate for the root function is known. To demonstrate the efficiency of our CGPT, we present a set of preliminary numerical results with a different number of leaf nodes. Our model consistently outperforms the single Gaussian process model showing little to no deterioration as the balance of the branches changes and as the number of nodes increases. These are promising results; we will next investigate how to embed the estimation of the root GP a one-shot likelihood optimization. We will study multiple tree levels. Another important aspect would be to integrate linear functions and convex functions at the intermediate nodes and see how to exploit such knowledge. Moreover, we will investigate search criteria to couple with CGPT for optimization.

6 ACKNOWLEDGEMENTS

This work is supported in part by NSF grants #2046588, #2134256, and by DARPA ARCOS program under contract #FA8750-20-C-0507, and Lockheed Martin funded contract #FA8750-22-9-0001.

REFERENCES

Achituve, I., A. Navon, Y. Yemini, G. Chechik, and E. Fetaya. 2021. “Gp-Tree: A Gaussian Process Classifier for Few-Shot Incremental Learning”. In *International Conference on Machine Learning*, 54–65. PMLR.

Astudillo, R., and P. Frazier. 2019. “Bayesian Optimization of Composite Functions”. In *International Conference on Machine Learning*, 354–363. PMLR.

Astudillo, R., and P. Frazier. 2021a. “Bayesian Optimization of Function Networks”. *Advances in Neural Information Processing Systems* 34:14463–14475.

Astudillo, R., and P. I. Frazier. 2021b. “Thinking Inside the Box: A Tutorial on Grey-Box Bayesian Optimization”. In *Proceedings of the 1994 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, 1–15. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. 2017. *Classification and Regression Trees*. Routledge.

Brochu, E., V. M. Cora, and N. De Freitas. 2010. “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning”. *arXiv preprint arXiv:1012.2599*.

Bui, T. D., and R. E. Turner. 2014. “Tree-Structured Gaussian Process Approximations”. *Advances in Neural Information Processing Systems* 27.

Civera, M., G. Boscato, and L. Z. Fragonara. 2020. “Treed Gaussian Process for Manufacturing Imperfection Identification of Pultruded GFRP Thin-Walled profile”. *Composite Structures* 254:112882.

Dokhanchi, A., A. Zutshi, R. T. Sriniva, S. Sankaranarayanan, and G. Fainekos. 2015. “Requirements Driven Falsification with Coverage Metrics”. In *2015 International Conference on Embedded Software (EMSOFT)*, 31–40. Institute of Electrical and Electronics Engineers.

Foumani, Z. Z., M. Shishehbor, A. Yousefpour, and R. Bostanabad. 2023. “Multi-Fidelity Cost-Aware Bayesian Optimization”. *Computer Methods in Applied Mechanics and Engineering* 407:115937.

Frazier, P. I. 2018. “A Tutorial on Bayesian Optimization”. *arXiv preprint arXiv:1807.02811*.

Fulgenzi, C., C. Tay, A. Spalanzani, and C. Laugier. 2008. “Probabilistic Navigation in Dynamic Environment Using Rapidly-Exploring Random Trees and Gaussian Processes”. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1056–1062. IEEE.

Gramacy, R. B., H. K. Lee, and W. G. Macready. 2004. “Parameter Space Exploration with Gaussian Process Trees”. In *Proceedings of the Twenty-First International Conference on Machine Learning*, 45.

Gramacy, R. B., and H. K. H. Lee. 2008. “Bayesian Treed Gaussian Process Models with an Application to Computer Modeling”. *Journal of the American Statistical Association* 103(483):1119–1130.

Greenhill, S., S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh. 2020. “Bayesian Optimization for Adaptive Experimental Design: A Review”. *IEEE access* 8:13937–13948.

Guinet, G., V. Perrone, and C. Archambeau. 2020. “Pareto-Efficient Acquisition Functions for Cost-Aware Bayesian Optimization”. *arXiv preprint arXiv:2011.11456*.

Imani, M., S. F. Ghoreishi, D. Allaire, and U. M. Braga-Neto. 2019. “MFBO-SSM: Multi-Fidelity Bayesian Optimization for Fast Inference in State-Space Models”. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 33, 7858–7865.

Joy, T. T., S. Rana, S. Gupta, and S. Venkatesh. 2020. “Fast Hyperparameter Tuning Using Bayesian Optimization with Directional Derivatives”. *Knowledge-Based Systems* 205:106247.

Lee, D., H. Park, and C. D. Yoo. 2015. "Face Alignment Using Cascade Gaussian Process Regression Trees". In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4204–4212.

Lee, E. H., V. Perrone, C. Archambeau, and M. Seeger. 2020. "Cost-aware Bayesian Optimization". *arXiv preprint arXiv:2003.10870*.

Li, L.-L., J. Sun, C.-H. Wang, Y.-T. Zhou, and K.-P. Lin. 2019. "Enhanced Gaussian Process Mixture Model for Short-Term Electric Load Forecasting". *Information Sciences* 477:386–398.

Mathesen, L., S. Yaghoubi, G. Pedrielli, and G. Fainekos. 2019. "Falsification of Cyber-Physical Systems with Robustness Uncertainty Quantification through Stochastic Optimization with Adaptive Restart". In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 991–997. IEEE.

McDowell, I. C., D. Manandhar, C. M. Vockley, A. K. Schmid, T. E. Reddy, and B. E. Engelhardt. 2018. "Clustering Gene Expression Time Series Data Using an Infinite Gaussian Process Mixture Model". *PLoS Computational Biology* 14(1):e1005896.

Meeds, E., and S. Osindero. 2005. "An Alternative Infinite Mixture of Gaussian Process Experts". *Advances in Neural Information Processing Systems* 18.

Park, C. 2022. "Jump Gaussian Process Model for Estimating Piecewise Continuous Regression Functions". *Journal of Machine Learning Research* 23(278):1–37.

Pedrielli, G., T. Khandait, S. Chotaliya, Q. Thibeault, H. Huang, M. Castillo-Effen, and G. Fainekos. 2021. "Part-x: A Family of Stochastic Algorithms for Search-Based Test Generation with Probabilistic Guarantees". *arXiv preprint arXiv:2110.10729*.

Rasmussen, C., and Z. Ghahramani. 2001. "Infinite Mixtures of Gaussian Process Experts". *Advances in Neural Information Processing Systems* 14.

Reynolds, D. A. 2009. "Gaussian Mixture Models.". *Encyclopedia of Biometrics* 741(659-663).

Sano, S., T. Kadowaki, K. Tsuda, and S. Kimura. 2020. "Application of Bayesian Optimization for Pharmaceutical Product Development". *Journal of Pharmaceutical Innovation* 15:333–343.

Santner, T. J., B. J. Williams, and W. I. Notz. 2013. *The Design and Analysis of Computer Experiments*. Berlin: Springer Science & Business Media.

Sauer, A., R. B. Gramacy, and D. Higdon. 2022. "Active Learning for Deep Gaussian Process Surrogates". *Technometrics*:1–15.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. 2015. "Taking the Human out of the Loop: A Review of Bayesian Optimization". *Proceedings of the IEEE* 104(1):148–175.

Shen, Y., M. Seeger, and A. Ng. 2005. "Fast Gaussian Process Regression Using Kd-Trees". *Advances in Neural Information Processing Systems* 18.

Song, J., Y. Chen, and Y. Yue. 2019. "A General Framework for Multi-Fidelity Bayesian Optimization with Gaussian Processes". In *The 22nd International Conference on Artificial Intelligence and Statistics*, 3158–3167. PMLR.

Stachniss, C., C. Plagemann, A. J. Lilienthal, and W. Burgard. 2008. "Gas Distribution Modeling Using Sparse Gaussian Process Mixture Models". In *International Conference on Robotics Science and Systems, Robotics: science and systems, 2008, Zürich, Switzerland, June 25-28, 2008*, Volume 4, 310–317. MIT Press.

Torun, H. M., M. Swaminathan, A. K. Davis, and M. L. F. Bellaredj. 2018. "A Global Bayesian Optimization Algorithm and its Application to Integrated System Design". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26(4):792–802.

Will, J., L. Peel, and C. Claxton. 2011. "Fast Maritime Anomaly Detection Using Kd-Tree Gaussian Processes". In *IMA Maths in Defence Conference*.

Wu, A., M. C. Aoi, and J. W. Pillow. 2017. "Exploiting Gradients and Hessians in Bayesian Optimization and Bayesian Quadrature". *arXiv preprint arXiv:1704.00060*.

Zhang, Y., T. N. Hoang, B. K. H. Low, and M. Kankanhalli. 2017. "Information-Based Multi-Fidelity Bayesian Optimization". In *NIPS Workshop on Bayesian Optimization*, 49. Journal of Machine Learning Research JMLR. org Cambridge, MA.

AUTHOR BIOGRAPHIES

MENGRUI (MINA) JIANG is a Ph.D. student in the School of Computing and Augmented Intelligence at Arizona State University. Her research interest is primarily in Bayesian optimization. Her email address is mjiang42@asu.edu.

TANMAY KHANDAIT is a Ph.D candidate and graduate research assistant in School of Computing and Augmented Intelligence at ASU. He graduated with Masters in Computer Science from Arizona State University. His research interests include machine learning, verification of cyber-physical systems, computer vision, and bilevel optimization. Email: tkhandai@asu.edu.

GIULIA PEDRIELLI is Associate Professor in the School of Computing and Augmented Intelligence at Arizona State University. She is interested in the area of stochastics and simulation based optimization, and deals with applications in biomanufacturing, power systems, supply chains, safety critical systems. Her email address is giulia.pedrielli@asu.edu.