Evolving-Graph Gaussian Processes

David Blanco-Mulero [1] **Markus Heinonen** [2] **Ville Kyrki** [1]

Abstract

Graph Gaussian Processes (GGPs) provide a data-efficient solution on graph structured domains. Existing approaches have focused on static structures, whereas many real graph data represent a dynamic structure, limiting the applications of GGPs. To overcome this we propose evolving-Graph Gaussian Processes (e-GGPs). The proposed method is capable of learning the transition function of graph vertices over time with a neighbourhood kernel to model the connectivity and interaction changes between vertices. We assess the performance of our method on time-series regression problems where graphs evolve over time. We demonstrate the benefits of e-GGPs over static graph Gaussian Process approaches.

1. Introduction

Dynamic graphs provide rich representations for temporal structured data to model evolving relationships in the data. Structured data is often dynamic, such as human interactions (Casteigts et al., 2012), social networks (Trivedi et al., 2019) or brain interactions (Ofori-Boateng et al., 2020). These structures have been studied as dynamic graphs, time-varying structures (Le Bars et al., 2020), and *evolving graphs* (Bui-Xuan et al., 2002). However, the predominant methods for estimating dynamic graph evolution fail to account for model uncertainty (Sanchez-Gonzalez et al., 2018; Li et al., 2019; Zhu et al., 2019). This uncertainty can be crucial in dynamic systems where safety constraints are required (Hewing et al., 2020).

A natural choice to quantify uncertainty of a model are Gaussian processes (GPs) (Williams & Rasmussen, 2006). Pioneering attempts of applying Gaussian processes to model dynamics estimate autoregressive temporal transition functions with GPs in the Euclidean domain (Wang et al., 2005; Mattos et al., 2015). More recently, GPs have been applied to problems on graph-structured domains (graph GPs), such as signal processing over graphs (Walker & Glocker, 2019; Venkitaraman et al., 2020; Li et al., 2020) or link prediction (Opolka & Liò, 2020). Some of these works decompose the structured domain through graph convolutions (Walker & Glocker, 2019; Opolka & Liò, 2020) but their output is limited to the Euclidean domain. Others, provide an output in the graph-domain (Venkitaraman et al., 2020; Zhi et al., 2020) but are restricted to a vectorial input. To our knowledge, no Gaussian process models have been proposed to model temporal evolution of graph structures.

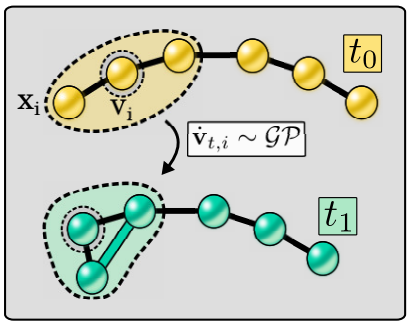


Figure 1: Evolving graph where the neighbourhood of the vertex $\mathbf{v}_i$ changes and the transition is modelled by an evolving-Graph Gaussian Process (e-GGP).

We present an autoregressive GP model that learns the transition function of vertices $f : \mathbf{v} \mapsto \dot{\mathbf{v}}$ on the graph domain G, which we call the evolving-Graph Gaussian Process (e-GGP), see Figure 1. We define a graph kernel that considers the neighbourhoods and attributes of the graph. We investigate the capability of our method to learn the graph evolution in simulated physical dynamical systems, where a measure of the simulation uncertainty can be required. Our results showcase the ability of e-GGP to learn accurate graph dynamics, and extrapolate the dynamics to graphs with unseen structures at test time.

2. Background

Graph-based Gaussian processes. There is a rich literature of different configurations of Gaussian processes

[1]School of Electrical Engineering, Aalto University, Espoo, Finland [2]Department of Computer Science, Aalto University, Espoo, Finland. Correspondence to: David Blanco-Mulero <david.blancomulero@aalto.fi>.

Time Series Workshop at 38^{th} International Conference on Machine Learning, 2021.

over graph domains. As previously mentioned, we can distinguish between two configurations. The first, maps from a vectorial input to an output in the graph domain $f : \mathbb{R}^D \to G$ (Venkitaraman et al., 2020; Zhi et al., 2020). The second, takes an input graph and maps it to the Euclidean domain $f : G \to \mathbb{R}$ (Ng et al., 2018; Walker & Glocker, 2019; Opolka & Liò, 2020; Borovitskiy et al., 2020). For a more thorough overview of graph GP methods see Supplementary Material Section A.1. These methods consider a fixed adjacency matrix of the graph input, or use convolution operations (van der Wilk et al., 2017), which narrows their input to a static graph structure.

Gaussian processes for time-series. The Gaussian process dynamical model (GPDM) models autoregressive Markov transitions of discrete-time random states with a Gaussian process (Wang et al., 2005). In latent modelling a Gaussian process interpolates the object trajectories in latent space (Damianou et al., 2011). More generally these models fall under recurrent Gaussian processes (Mattos et al., 2015). The GPAR extends multi-output Gaussian processes to time-series (Requeima et al., 2019). These models are limited to vector-valued inputs, and do not support non-Euclidean inputs.

The most relevant prior work to our aims is the deep Graph Gaussian process (DGPG) (Li et al., 2020), where the graph structure is auto-regressed before applying a conventional classification or regression step. However, their approach is limited to a fixed graph structure.

3. Evolving Graphs using Gaussian Processes

We consider a temporally evolving, or *dynamic*, undirected graph $G_t = \langle \mathcal{V}_t, \mathcal{E}_t \rangle$, where $\mathcal{V}_t$ is the set of vertices and $\mathcal{E}_t \subset \mathcal{V}_t \times \mathcal{V}_t$ is the set of edges at discrete timepoints $t \in N$ of the graph time-series $G = \{G_0, \dots, G_N\}$. For notation simplicity we consider the number of vertices in each graph to be the same $|\mathcal{V}_t| = M$.

The vertices $\mathbf{v}_t \in \mathcal{V}$ reside in a D-dimensional space. For our objective of learning physics simulations, a vertex at time t is represented as $\mathbf{v}_t := (\mathbf{x}_t, \Delta\mathbf{x}_{t-1}, \phi(\mathbf{v}_t))$ over the Cartesian coordinates $\mathbf{x}$, and static attributes $\phi(\mathbf{v})$, which describe node properties or types. We assume that edges are a function of vertices and that the vertices-to-edges function $\mathbf{g} : 2^{\mathcal{V}} \to \mathcal{E}$ is defined by a distance function $d(\mathbf{v}_i, \mathbf{v}_j)$. We adopt the L2-norm distance between the vertices Cartesian coordinates. Therefore, an edge exists if the distance between two nodes is lower than a connectivity threshold R_{nn}, see Supplementary Material Section B.

We assume the graph dynamics are governed by

$$G_{t+1} = T(G_t) \tag{1}$$

driven by the graph transition function $T : G \to G$, where the graph transition is defined by a transition function of node attributes $\mathcal{V}_{t+1} = T(\mathcal{V}_t)$ and the edges are given by the nodes-to-edge function. Our goal is to learn the evolution function T. Hence, we opt to model and learn the node evolution in the discrete timestep Δt by:

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \underbrace{\mathbf{f}(N^{0,1}(\mathbf{v}_t))}_{\Delta\mathbf{v}_t}, \tag{2}$$

with a velocity function $\mathbf{f} : s(G) \to \mathbb{R}^F$ mapping a sub-tree $s(G)$ into coordinate velocities $\mathbb{R}^F$. The sub-tree $N^{0,1}(\mathbf{v}_t)$ consists of a node $\mathbf{v}_t$, or $N^0(\mathbf{v}_t)$, and its neighbours $N^1(\mathbf{v}_t)$. To simplify the notation in the following, we will use $\mathbf{f}(\mathbf{v}_t)$ to denote the mapping. This formulation allows us to handle node insertion and deletion.

3.1. Prior and likelihood

We model the velocity transitions with a vector-valued Gaussian process (GP) prior (Williams & Rasmussen, 2006; Alvarez et al., 2012)

$$\mathbf{f}(\mathbf{v}) \sim \mathcal{GP}(\mathbf{m}(\mathbf{v}), K(N^{0,1}(\mathbf{v}), N^{0,1}(\mathbf{v}'))), \tag{3}$$

which implies that the transition is a stochastic process, whose mean and covariance are parameterised by the mean function $\mathbf{m}(\cdot)$ and the kernel similarity $K(\cdot, \cdot)$ as

$$\mathbb{E}[\mathbf{f}(\mathbf{v})] = \mathbf{m}(\mathbf{v}), \tag{4}$$

$$\mathrm{cov}[\mathbf{f}(\mathbf{v}), \mathbf{f}(\mathbf{v}')] = K(N^{0,1}(\mathbf{v}), N^{0,1}(\mathbf{v}')). \tag{5}$$

For tractability, we assume different velocity predictions are independent, however sharing the same kernel, which simplifies the kernel into a scalar function. Furthermore, for any subset of nodes $\mathbf{v}_1, \dots, \mathbf{v}_M$ the corresponding transitions are jointly Gaussian

$$p(\Delta\mathbf{v}_1, \dots, \Delta\mathbf{v}_M) = \mathcal{N}(\Delta\vec{\mathbf{v}} | \vec{\mathbf{m}}, \mathbf{K}), \tag{6}$$

where $\Delta\vec{\mathbf{v}} \in \mathbb{R}^{MF}$ and $\vec{\mathbf{m}} \in \mathbb{R}^{MF}$ stack all vertices and means to column vectors, and $\mathbf{K} \in \mathbb{R}^{MF \times MF}$ is the block kernel matrix consisting of $M \times M$ blocks $K(\mathbf{v}_i, \mathbf{v}_j)$ of size $F \times F$. The key property of Gaussian processes is that similar nodes have similar transitions, as specified by the similarity function K.

We assume a dataset

$$\mathcal{D} = \left\{ (\mathbf{v}_{t,i}, \Delta\mathbf{v}_{t,i}) \right\}_{t=1,i=1}^{N,M} \tag{7}$$

of observed graph states G_t that are described in terms of the M vertices at N timepoints. Our observation model induces a likelihood

$$p(\mathcal{D}|\mathbf{f}) = \prod_{t=1}^{N} \mathcal{N}\left(\mathbf{v}_{t,i} | \mathbf{f}, \Sigma\right), \tag{8}$$

where $\mathbf{f}$ is the estimated node evolution following equation (2), and $\Sigma \in \mathbb{R}^{F \times F}$ represents the numerical variance required for numerical stability.

3.2. Posterior

The predictive posterior distribution of our Gaussian process model is conveniently tractable under the Gaussian likelihood model (8) as (Williams & Rasmussen, 2006)

$$p(\mathbf{f}_*|\mathcal{D}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \Sigma_*), \tag{9}$$

$$\boldsymbol{\mu}_* = \mathbf{K}_*(\mathbf{K}+\boldsymbol{\Sigma})^{-1}\Delta\vec{\mathbf{v}}, \tag{10}$$

$$\Sigma_* = \mathbf{K}_{**} - \mathbf{K}_*(\mathbf{K}+\boldsymbol{\Sigma})^{-1}\mathbf{K}_*^T, \tag{11}$$

where $\mathbf{K} \in \mathbb{R}^{NMF\times NMF}$, $\mathbf{K}_* \in \mathbb{R}^{N_*MF\times NMF}$ and $\mathbf{K}_{**} \in \mathbb{R}^{N_*MF\times N_*MF}$ for a test set consisting of N_* graphs with M nodes. The posterior effectively interpolates the velocity predictions from the observed velocities of the training graphs.

3.3. Kernel between attributed sub-trees

We now define a kernel that can measure the similarity between vertices and their transitions. We propose to use a kernel on the graph that measures the similarity between attributed sub-trees $s(G)$, $K : s(G)\times s(G) \to \mathbb{R}$. For simplicity, we refer to the kernel as $K(\mathbf{v}_i, \mathbf{v}_j)$. The kernel requires the attributed vertices to be in the same space. Therefore, we define a vertex mapping function $\varphi_\mathbf{v}$ which maps the attributed vertex to an Euclidean-space $\varphi_\mathbf{v} : \mathbb{R}^D \mapsto \mathbb{R}^E$. In our case, $\varphi_\mathbf{v}$ discards the static attributes of the vertices.

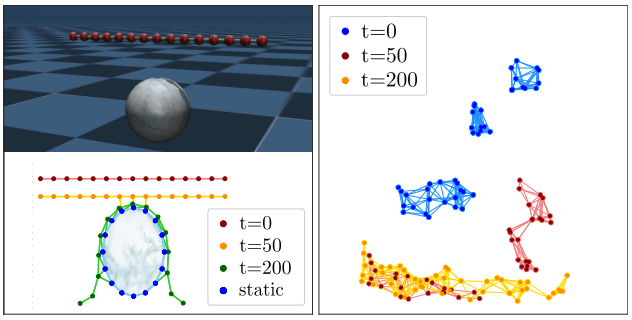
The kernel input sub-trees are rooted at the vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ with 1-hop neighbourhood. We measure the similarity between the attributed roots using the *root kernel* $k_r : \mathbb{R}^E \times \mathbb{R}^E \to \mathbb{R}$. In order to share information with the root and compare the structures, we compute a kernel between all the attributed leaves in the neighbourhood $N_i^1 = N^1(\mathbf{v}_i)$. The similarity between the leaves is measured by the *leaf kernel* $k_l : \mathbb{R}^E \times \mathbb{R}^E \to \mathbb{R}$. Both the *root* and the *leaf kernel* operations are performed in the mapped space $\mathbb{R}^E$. We express the neighbourhood kernel $k_{nn}(\mathbf{v}_i, \mathbf{v}_j)$ as

$$k_{nn}(\mathbf{v}_i, \mathbf{v}_j) = \frac{1}{L}\sum_{\mathbf{x}_i\in N_i^1}\sum_{\mathbf{x}_j\in N_j^1} k_l(\varphi_\mathbf{v}(\mathbf{x}_i), \varphi_\mathbf{v}(\mathbf{x}_j)), \tag{12}$$

where L is a normalisation constant that denotes the product between the number of leaves in N_i^1 and N_j^1. Then, the kernel between two attributed vertices is defined by

$$K(\mathbf{v}_i, \mathbf{v}_j) = k_r(\varphi_\mathbf{v}(\mathbf{v}_i), \varphi_\mathbf{v}(\mathbf{v}_j)) + k_{nn}(\mathbf{v}_i, \mathbf{v}_j). \tag{13}$$

We restrict the kernel to the 1-hop neighbourhood for scalability and plan to investigate the effect of the neighbourhood in future work.



(a) (b)

Figure 2: Physics simulations: (a) graph interaction environment in MuJoCo simulation and 2-D approximation, (b) evolving isolated sub-graphs environment, showing graphs connectivity for different timepoints t.

4. Experimental results

4.1. Experiments description

We evaluate the performance of e-GGPs to learn the node transition function $\Delta\mathbf{v}_t = \mathbf{f}(\mathbf{v}_t)$ in two physics simulations where graphs evolve over time: 1) graph interaction (GI) environment, and 2) evolving isolated sub-graphs (EIs) environment. The GI environment simulates a rope on free fall that gets in contact with a static ball using MuJoCo (Todorov et al., 2012). The EIs environment simulates a set of 2D fluid particles using Taichi (Hu et al., 2018). Both simulations are represented in 2-D, where the joints of the rope, the static ball and the particles define the set of nodes $\mathbf{p}_t = \mathbf{v}_t \in \mathcal{V}$, see Figure 2. Thus, the experiments consist of multiple sub-graphs where connectivity varies over time.

The purpose our experiments is twofold: (i) investigate the capability of the methods to address the varying connectivity and (ii) validate that e-GGP can learn the evolution of the graph by capturing the interaction changes between vertices.

4.2. Baselines and training details

We consider three Gaussian process methods as baselines: GP regression (GPR) using exact inference, GPAR and DGPG. The first two baselines are GP functions that model transition functions in the Euclidean domain. We use baselines in the Euclidean domain to demonstrate the potential of graph-structured information compared to approaches restricted to a vector-valued input. For these we take the vertices attributes vector $\mathcal{V}$ as input. Then, we learn a mapping $\mathbf{f} : \mathbb{R}^D \to \mathbb{R}^F$. The mapping of DGPG is defined as $\mathbf{f} : G \to \mathbb{R}^F$. Since DGPG does not handle node insertions we assume the adjacency matrix to be equal to the initial state graph adjacency for any given timepoint $A_t = A_0$.

Table 1: Average results of GPR, GPAR, DGPG and e-GGP on test sets (lower is better) for different training data sizes N.

		Graph interaction				Evolving isolated sub-graphs			
N	**Metric**	**GPR**	**GPAR**	**DGPG**	**e-GGP**	**GPR**	**GPAR**	**DGPG**	**e-GGP**
10	RMSE	0.121	0.143	0.413	**0.049**	0.294	0.283	0.337	**0.182**
	MAPE	0.418	0.537	2.456	**0.249**	0.873	**0.726**	2.696	0.777
	NLL	67.63	**-9.16**	9.51	10.44	43.37	-47.942	22.24	**-58.042**
15	RMSE	0.149	0.209	0.275	**0.031**	0.276	0.277	0.325	**0.183**
	MAPE	0.410	0.397	1.873	**0.230**	**0.671**	**0.671**	3.135	0.853
	NLL	102.49	-7.67	2.20	**-17.56**	103.82	-46.764	20.12	**-78.58**
20	RMSE	0.162	0.250	0.222	**0.036**	0.257	0.253	0.300	**0.160**
	MAPE	0.426	0.466	1.455	**0.148**	0.612	**0.594**	2.998	0.766
	NLL	126.97	-6.84	-2.18	**-29.69**	4.17E6	**-48.52**	9.731	8.77E5

The kernel hyper-parameters are trained using automatic relevance determination (ARD) and optimised by minimising the negative marginal log-likelihood (MLL) (Williams & Rasmussen, 2006). For all the methods we used as optimizer Adam (Kingma & Ba, 2015). We use a Radial Basis Function kernel for the *root* and *leaf* kernels k_r and k_l in e-GGP and GPR. In DGGP we use the Matérn32 kernel and $Z = 5$ inducing points. For more details about the training see Supplementary Material Section B.

4.3. Results

We investigate the error of the node evolution predictions. In order to evaluate the methods ability to capture the evolving graph information we limit the amount of data provided in training. We select the most informative training points as described in Supplementary Material Section B. To evaluate the prediction performance and confidence we measure the root mean squared error (RMSE), mean absolute percentage error (MAPE) and negative log-likelihood (NLL). The test datasets on the GI environment include an offset between the rope and the static ball from the training data to evaluate the generalisation capabilities of the methods. We provide additional results in Supplementary Material Section C.

The results for the GI environment on Table 1 show that e-GGP outperforms the baselines when data is scarce. The different orders of magnitude in the RMSE provide evidence that the method we propose benefits of the evolving graph-structured information captured by the kernel. These results also provide evidence that e-GGP can generalise better to unseen data under limited training samples. Furthermore, e-GGP presents low RMSE for the EIs environment, which shows that the model can generalise to unseen sub-graph structures, i.e. sub-graphs that have not been seen during training. When increasing the sample size to $N = 20$ the confidence drops while keeping low RMSE and MAPE. This suggests that the model is not able to provide a confident estimate under sparse data, which will be investigated in the future. The DGPG method is limited to a fixed graph size and cannot handle node insertions, which is shown by the higher error results.

In summary, the results provide evidence that: (i) e-GGP can predict the node evolution under varying connectivity, and (ii) learning the interactions between the evolving vertices is beneficial as e-GGP outperforms the baselines in the limited data setting.

5. Conclusion

In this paper we proposed evolving-Graph Gaussian Processes (e-GGPs), an autoregressive graph-GP model that learns the evolving structure of dynamic graphs via the node transitions. Our model is able to learn the node transitions as well as the interactions of the structure via an attributed subtree kernel that incorporates information from each node's neighbourhood. The experimental results demonstrate that our model is able to capture evolving graph connectivity, overcoming this limitation of current methods. We provide evidence of the importance of capturing the dynamic connectivity and investigated the performance of our method in scarce data. Our experiments showed that e-GGP outperforms other approaches by making use of the rich information in the evolving graph-domain.

Our model opens GPs to dynamic graphs which is of great interest in real graph problems. However, our model suffers from poor scalability, which we plan to investigate via variational inference. We also plan to study the application of e-GGP to real problems that require both data-efficiency and uncertainty, such as dynamic systems with safety constraints, where other approaches such as neural networks are limited.

Acknowledgements

This work was supported by the Academy of Finland, decision 317020.

References

Alvarez, M. A., Rosasco, L., and Lawrence, N. D. Kernels for vector-valued functions: a review, 2012.

Borovitskiy, V., Azangulov, I., Terenin, A., Mostowsky, P., Deisenroth, M. P., and Durrande, N. Matern Gaussian Processes on Graphs. *arXiv:2010.15538 [cs, stat]*, October 2020. arXiv: 2010.15538.

Bui-Xuan, B.-M., Ferreira, A., and Jarry, A. Computing shortest, fastest, and foremost journeys in dynamic networks. Technical Report RR-4589, INRIA, October 2002.

Casteigts, A., Flocchini, P., Quattrociocchi, W., and Santoro, N. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

Damianou, A., Titsias, M., and Lawrence, N. Variational gaussian process dynamical systems. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31, pp. 7576–7586. Curran Associates, Inc., 2018.

Hewing, L., Kabzan, J., and Zeilinger, M. N. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2020.

Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., and Jiang, C. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4): 150, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Le Bars, B., Humbert, P., Kalogeratos, A., and Vayatis, N. Learning the piece-wise constant graph structure of a varying ising model. In *International Conference on Machine Learning*, pp. 675–684. PMLR, 2020.

Li, N., Li, W., Sun, J., Gao, Y., Jiang, Y., and Xia, S.-T. Stochastic deep gaussian processes over graphs. *Advances in Neural Information Processing Systems*, 33, 2020.

Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B., and Torralba, A. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Mattos, C. L. C., Dai, Z., Damianou, A., Forth, J., Barreto, G. A., and Lawrence, N. D. Recurrent gaussian processes. *arXiv preprint arXiv:1511.06644*, 2015.

Ng, Y. C., Colombo, N., and Silva, R. Bayesian semi-supervised learning with graph gaussian processes. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31, pp. 1683–1694. Curran Associates, Inc., 2018.

Ofori-Boateng, D., R.Gel, Y., and Cribben, I. Nonparametric anomaly detection on time series of graphs. 2020. doi: 10.1101/2019.12.15.876730.

Opolka, F. L. and Liò, P. Graph Convolutional Gaussian Processes For Link Prediction. In *Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020.

Requeima, J., Tebbutt, W., Bruinsma, W., and Turner, R. E. The gaussian process autoregressive regression model (gpar). In *AISTATS*, pp. 1860–1869, 2019.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4470–4479, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Trivedi, R., Farajtabar, M., Biswal, P., and Zha, H. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*, 2019.

van der Wilk, M., Rasmussen, C. E., and Hensman, J. Convolutional gaussian processes. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 2849–2858. Curran Associates, Inc., 2017.

Venkitaraman, A., Chatterjee, S., and Handel, P. Gaussian processes over graphs. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5640–5644, 2020. doi: 10.1109/ICASSP40776.2020.9053859.

Walker, I. and Glocker, B. Graph convolutional Gaussian processes. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6495–6504. PMLR, 09–15 Jun 2019.

Wang, J. M., Fleet, D. J., and Hertzmann, A. Gaussian process dynamical models. In *NIPS*, volume 18, pp. 3. Citeseer, 2005.

Williams, C. K. and Rasmussen, C. E. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Zhi, Y.-C., Ng, Y. C., and Dong, X. Gaussian Processes on Graphs via Spectral Kernel Learning. *arXiv:2006.07361 [cs, eess, stat]*, October 2020. arXiv: 2006.07361.

Zhu, C., Storandt, S., Lam, K.-Y., Han, S., and Bi, J. Improved dynamic graph learning through fault-tolerant sparsification. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7624–7633. PMLR, 09–15 Jun 2019.

Supplementary Material: Evolving Graph Gaussian Processes

A. Analysis of e-GGP

A.1. Overview of graph Gaussian processes

There is a rich literature of different configurations of Gaussian processes over graph domains. Graph-output Gaussian processes consider functions $f : \mathbb{R}^D \to G$, where the $|\mathcal{V}|$ outputs have dependencies according to an underlying output graph (Venkitaraman et al., 2020; Zhi et al., 2020). In semi-supervised Gaussian processes a graph relationship between inputs is assumed to propagate label information within the graph from labelled to unlabelled inputs (Ng et al., 2018). Borovitskiy et al. (2020) adapts Matérn Gaussian processes for graph node labelling. The graph convolutional Gaussian processes consider functions $f : G \to \mathbb{R}$, where the label of an entire input graph G is predicted (Walker & Glocker, 2019; Opolka & Liò, 2020) with convolution operations (van der Wilk et al., 2017).

Our method learns the transition function of vertices $f : \mathbf{v} \mapsto \dot{\mathbf{v}}$ and infers the new edges using a nodes-to-edge function $\mathbf{g} : 2^{\mathcal{V}} \to \mathcal{E}$, learning the graph transition. We provide an overview of the different graph-GP methods as well as their mapping function to highlight the scope of our work on Table A.1.

Table A.1: Overview of different graph-GP methods, their mapping functions $\mathbf{f}$ and whether an static graph is assumed or not. We denote each model by the name addressed in the paper or by the formulation they propose.

	Input domain			Output domain				
Model	$\mathbb{R}^D$	$\mathcal{V}$	$\mathcal{G}$	$\mathbb{R}^D$	$\mathcal{Y}$	$\mathcal{G}$	Time-series	Reference
GPDM	✓	-	-	✓	-	-	✓	Wang et al. (2005)
GGP	-	✓	-	-	✓	-	-	Ng et al. (2018)
GCGP	-	-	✓	-	✓	-	-	Walker & Glocker (2019); Opolka & Liò (2020)
DGPG	-	-	✓	✓	-	-	-	Li et al. (2020)
Matérn Graph GP	-	-	✓	✓	-	-	-	Borovitskiy et al. (2020)
GPG	✓	-	-	-	-	✓	-	Venkitaraman et al. (2020)
Graph-output GP	✓	-	-	-	-	✓	-	Zhi et al. (2020)
Ours	-	-	✓	-	-	✓	✓	

B. Supplementary training details

B.1. Experiments data sets

The number of nodes in each of the graph data sets is provided in Table B.1. The connectivity hyper-parameter R_{nn} and the maximum number of neighbours K_{nn} are selected based on the environment definition.

Table B.1: Description of train and test data for the experiments described in Section 4. N denotes the horizon, $|\mathcal{V}| = M$ the number of nodes in the graph and D the attributes of the nodes.

	N	R_{nn}	K_{nn}	M_{TRAIN}	D_{TRAIN}	M_{TEST}	D_{TEST}
GRAPH INTERACTION	500	0.043	2	31	7	31	7
ISOLATED EVOLVING SUB-GRAPHS	200	0.08	20	44	8	44, 55, 66	8

Graph interaction connectivity. We define two objects: an elastic rope and a static ball. The distance between the nodes of each object was set to $d(\mathbf{v}_i, \mathbf{v}_j) = 0.03$. However, because of the elasticity of the material in MuJoCo (Todorov et al., 2012), we found out experimentally that using a connectivity lower than $R_{nn} = 0.043$ would lead to disconnected nodes in the graph that are physically connected in the simulation. Considering that each of the objects is internally connected, we use that information to build the graph connectivity. Therefore, we considered the maximum number of neighbours affecting a node to be $K_{nn} = 2$, as the internal connectivity was already taken into account.

The attributes of the nodes are set to $\mathbf{v}_t = \{x_t, y_t, z_t, \Delta x_{t-1}, \Delta y_{t-1}, \Delta z_{t-1}, \phi(\mathbf{v}_t)\}$, where $\phi(\mathbf{v}_t) = 0$ if the node belonged to the ball and $\phi(\mathbf{v}_t) = 1$ if it was part of the rope.

Isolated evolving sub-graphs connectivity. We randomise for each test the initial conditions of the particles (Hu et al., 2018). For each of the tests, we initialise a random number of blocks of particles, where the initial position and velocity are also randomised. This lead to a different evolution of the connectivity, as shown in Figure B.1.

The resolution used in our Taich-MPM environment was set to 16. We found out experimentally that particles that are separated by a distance of $R_{nn} = 0.08$ or lower had an effect on each other so as to be considered neighbours. We decided to use $K_{nn} = 20$ to provide the maximum information from the nearby particles.

In this environment, the nodes are defined as $\mathbf{v}_t = \{x_t, y_t, \Delta x_{t-1}, \Delta y_{t-1}, d(x_t, x_{\text{max}}), d(y_t, y_{\text{max}}), d(x_t, x_{\text{min}}), d(y_t, y_{\text{min}})\}$, where x_{max}, x_{min}, y_{max}, y_{min} refer to the boundaries of the water container.

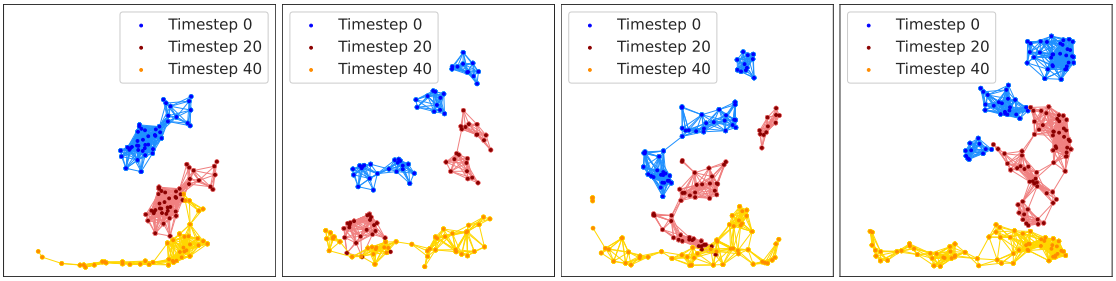


Figure B.1: Visualisation of the evolving isolated sub-graphs initial configuration, evolution and connectivity for (a) training data $N = 44$, (b) test data $N = 44$, (c) test data $N = 55$, (d) test data $N = 66$.

B.2. Kernel selection

The attributes of the nodes live in the coordinate space. Therefore, we decided to use the Radial Basis Function (RBF) stationary kernel as the *roof* and *leaf* kernels for e-GGP as well as for the baseline of GPR. We note to the reader that different combinations of kernels are possible when using e-GGP. As an example, one could choose to use an RBF *root* kernel and a Matérn 52 *leaf* kernel. The decision of the kernel to use for the root nodes and the neighbourhood should be based on the expert knowledge.

For the *graph interaction environment*, even though the environment is approximated in 2-D, the nodes are attributed with 3-dimensional coordinates and velocities. The models are not affected by the third dimension as the kernel hyper-parameters will reject the non-informative dimensions as training is done using automatic relevance determination (ARD) (Williams & Rasmussen, 2006).

B.3. Implementation details

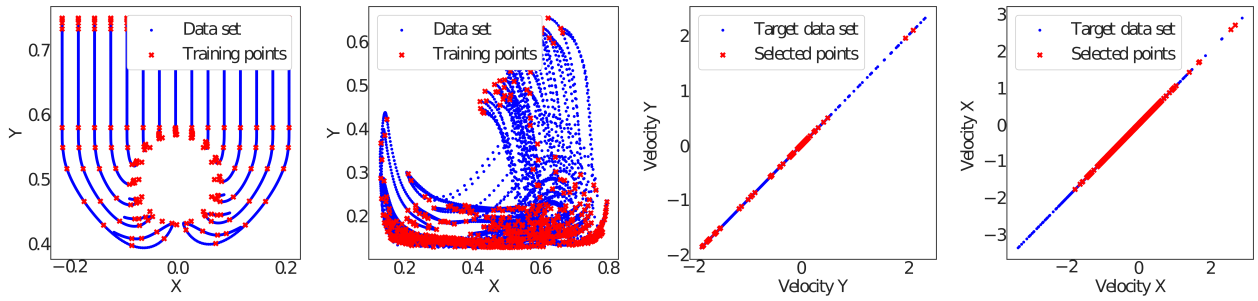
e-GGP implementation. We provide an open-source implementation of e-GGP[1]. Our implementation is based on GPyTorch (Gardner et al., 2018). The implementation of the *root* and *leaf* kernels was done so that the stationary kernels distance was scaled by a length-scale parameter θ_d per dimension $d \in D$, and a noise parameter σ^2.

[1]https://github.com/dblanm/evolving-ggp

Graph definition. The graph is built as follows. We take the attributed nodes at a timepoint $\mathcal{V}_t$ and build a k-d tree. We define the neighbourhood in the tree by the nearest K neighbours K_{nn}, with distance lower than the threshold R_{nn}, to the queried node. The K_{nn} parameter can also be seen as the maximum degree of each node. The neighbours denoted by the parameters K_{nn} and R_{nn} define the edges of the graph $\mathcal{E}_t$, thereby forming the graph $G_t = \langle \mathcal{V}_t, \mathcal{E}_t \rangle$ at the given timepoint t.

Training points selection. We select points from the training data set based on the rate of change of the target. Therefore, if the target is the velocity Δx_t, we select the points with highest acceleration $\Delta^2 x_t$. In order to avoid points that are close in the temporal dimension, we restrict the time between samples to be at least $\Delta t = 20$ timepoints far from each other. In case that there are no more data points fitting the temporal constraint, the time difference is constantly reduced until the requested number of training points is satisfied.

We show the coordinate space for each environment in Figure B.2. The blue markers show the full set of training points whereas the red markers show the limited selected training points, in this case $N = 20$. The Figure B.2 also shows a target for each of the environments. Note that each node covers a different part of the state-space.



(a) (b) (c) (d)

Figure B.2: Coordinate state space in (a) GI environment and (b) IEs environment showing full data set (blue) and selected $N = 20$ training points (red). Selected training targets (c) Δy_t and (d) Δx_t in each environment respectively.

Hardware used for training All the experiments were performed in CPU. For training e-GGP we used a shared server with a CPU Xeon E5-2680 v2. All the other models were trained in a personal laptop.

Training time We provide information about the approximate training time of e-GGP in Table B.2 for different number of training points. We trained e-GGP using Adam with a learning rate of $l_r = 0.1$ for 150 iterations. We note that the computational complexity highly depends on the number of nodes in the graph M and the maximum connectivity for a node. We would like to highlight that our implementation of the kernel used in e-GGP is not ideal and the time taken shown in Table B.2 can be highly reduced.

Table B.2: Training time of e-GGP for the experiments using a CPU Xeon E5-2680 v2.

	N	M	W	D	t
GRAPH INTERACTION	5	31	4	7	14MIN
	10	31	4	7	1H 20MIN
	15	31	4	7	2H 55MIN
	20	31	4	7	4H 24MIN
ISOLATED EVOLVING SUB-GRAPHS	5	44	20	8	33MIN
	10	44	20	8	2H 16MIN
	15	44	20	8	5H 42MIN
	20	44	20	8	14H 7MIN

C. Supplementary results

In Section 4, we provided the results for the Graph Interaction environment. In the Isolated Evolving Sub-graphs environment, because of the limitation of DGPG to predict test inputs with a different graph structure, we only provided results for a single test set. Below, we extend those results and detail the results for each test set in the experiments. We also provide an ablation study in the GI environment.

C.1. e-GGP ablation study

C.2. Graph connectivity ablation study

We evaluated two variants of our method. The first one assumes that the graph connectivity is fixed (F. e-GGP), using the same adjacency matrix during inference and predictions $A = A_0$. The second one, has knowledge of the evolving graph structure (E. e-GGP) and uses a different adjacency A_t as the graph evolves over time.

We first compare the results for each of the unseen test sets for Δx_t and Δy_t in Table C.1 and Table C.2 respectively. We can see that the F. e-GGP variant performs slightly better for the offsets close to the training data. However, the evolving version (E. e-GGP) clearly outperforms the fixed variant shown in Table C.2. This supports our hypothesis and highlights the relevance of accounting for the graph evolution.

Table C.1: Results of fixed (F. e-GGP) and evolving (E. e-GGP) variants in GI environment for Δx_t with $N = 15$.

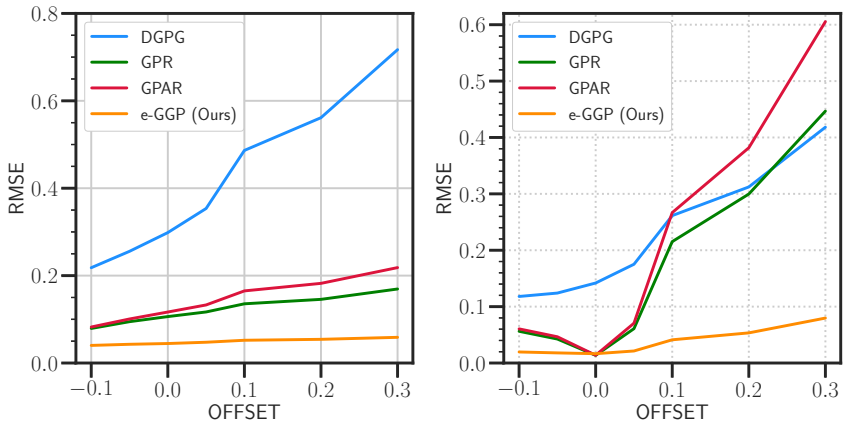
	RMSE		MAPE		NLL	
OFFSET	F. E-GGP $\times 10^{-2}$	E. E-GGP $\times 10^{-2}$	F. E-GGP $\times 10^{12}$	E. E-GGP $\times 10^{12}$	F. E-GGP	E. E-GGP
-0.1	**0.810**	0.879	**4.14**	4.52	**-57.8**	-54.0
-0.05	**0.890**	0.956	**4.28**	4.80	**-56.7**	-52.7
0	**0.980**	1.04	**3.89**	3.92	**-54.8**	-50.4
0.05	**1.10**	1.14	4.57	**4.17**	**-52.5**	-48.3
0.1	**1.40**	1.41	10.8	**9.80**	**-47.1**	-41.4
0.2	**1.56**	**1.56**	16.1	**15.4**	**-44.7**	-38.3
0.3	**1.87**	**1.87**	29.5	**29.4**	**-43.2**	-37.5

Table C.2: Results of fixed (F. e-GGP) and evolving (E. e-GGP) variants in GI environment for Δy_t with $N = 15$.

	RMSE		MAPE		NLL	
OFFSET	F. E-GGP $\times 10^{-2}$	E. E-GGP $\times 10^{-2}$	F. E-GGP $\times 10^{-1}$	E. E-GGP $\times 10^{-1}$	F. E-GGP	E. E-GGP
-0.1	2.52	**2.30**	3.98	**3.79**	-13.91	**-18.14**
-0.05	2.42	**2.36**	3.98	**3.79**	**-17.98**	-16.00
0	**2.17**	2.31	3.32	**2.86**	**-31.15**	-20.89
0.05	**2.25**	2.45	1.74	**1.73**	**-33.17**	-23.30
0.1	3.61	**3.31**	**1.04**	**1.04**	**-18.80**	-17.10
0.2	4.63	**3.98**	5.01	**4.69**	-11.09	**-12.10**
0.3	6.84	**5.46**	0.861	**0.732**	0.20	**-3.97**

C.3. Results on scarce data for the graph interaction environment

In this section, we provide additional results for the GI environment. We show in Figure C.1 the RMSE results for different offsets of the rope. The rope offset used for training was $Z = 0$. We can see that e-GGP outperforms the other methods for both $N = 10$ and $N = 20$ training points.



(a) (b)

Figure C.1: Comparison of one-step ahead prediction of RMSE results for GPR, GPAR, DGPG and e-GGP in the graph interaction experiment for (a) 10 training points and (b) 20 training points.

C.4. Results on scarce data for the isolated evolving sub-graphs environment

We provide the comparison of e-GGP and the baselines GPR, GPAR for the node velocity prediction for each target Δx_t and Δy_t in Table C.3 and Table C.4 respectively. We discussed in Section 4.3 the averaged results of e-GGP. In here, we can see that for both targets e-GGP has consistent low RMSE, MAPE and NLL. These results show that our model is capable of learning the transitions more accurately, regardless of the target.

Table C.3: Results for GPR, GPAR and e-GGP in evolving isolated sub-graphs test datasets for the target Δx_t.

		RMSE			MAPE			NLL		
N	M	GPR	GPAR	E-GGP	GPR	GPAR	E-GGP	GPR	GPAR	E-GGP
5	55	0.162	0.194	**0.107**	1.27	2.06	**1.09**	**-40.173**	242.025	3.542
	66	0.306	0.307	**0.148**	3.03	4.95	**2.23**	**-22.676**	435.562	89.792
10	55	0.148	0.172	**0.110**	0.97	1.32	**0.88**	**-43.425**	83.373	3.574
	66	0.397	0.277	**0.154**	**2.33**	3.31	2.37	**-18.255**	370.486	193.102
15	55	0.133	0.126	**0.096**	0.86	0.86	**0.79**	-49.657	146.950	**-90.342**
	66	0.325	0.295	**0.152**	2.41	2.56	**2.16**	-30.424	490.559	**-74.796**
20	55	0.117	0.111	**0.086**	0.78	**0.65**	<u>0.70</u>	-49.822	5.2×10^6	**-96.042**
	66	0.334	0.309	**0.194**	2.04	**1.76**	2.10	-30.945	1.07×10^7	1.98×10^6

Table C.4: Results for GPR, GPAR and e-GGP in evolving isolated sub-graphs test datasets for the target Δy_t.

		RMSE			MAPE			NLL		
N	M	**GPR**	**GPAR**	**E-GGP**	**GPR**	**GPAR**	**E-GGP**	**GPR**	**GPAR**	**E-GGP**
5	55	0.403	0.396	**0.264**	**2.52**	3.25	3.39	**-21.474**	92.63	-0.256
	66	0.643	0.751	**0.567**	4.48	**3.70**	6.07	**5.156**	167.1	200.3
10	55	0.414	0.429	**0.318**	1.38	1.16	**1.07**	-47.420	**-114.1**	-110.4
	66	**0.647**	0.654	0.713	3.54	8.73	**1.39**	-13.558	**-55.81**	-26.29
15	55	0.435	**0.391**	0.418	1.07	**0.93**	1.48	-42.35	**-111.38**	-50.755
	66	0.645	0.632	**0.596**	4.87	**1.45**	1.83	-9.78	**-44.65**	95.468
20	55	0.383	0.407	**0.253**	1.00	**0.78**	1.09	2.39×10^5	**-50.321**	1.17×10^6
	66	0.596	0.668	**0.435**	3.04	**0.96**	2.19	3.65×10^5	**-19.752**	2.47×10^6