PhyGNNet: Solving spatiotemporal PDEs with Physics-informed Graph Neural Network

Longxiang Jiang
Liyuan Wang
Xinkun Chu
Yonghao Xiao
Hao Zhang
jianglx@whu.edu.cn
wangly_xjtu@163.com
neocosmos@163.com
xiao_yonghao@caep.com
linusec@163.com
Institute of Computer Application, China Academy of Engineering Physics
Mianyang, Sichuan, China

ABSTRACT
Partial differential equations (PDEs) are a common means of describing physical processes. Solving PDEs can obtain simulated results of physical evolution. Currently, the mainstream neural network method is to minimize the loss of PDEs thus constraining neural networks to fit the solution mappings. By the implementation of differentiation, the methods can be divided into PINN methods based on automatic differentiation and other methods based on discrete differentiation. PINN methods rely on automatic backpropagation, and the computation step is time-consuming, for iterative training, the complexity of the neural network and the number of collocation points are limited to a small condition, thus abating accuracy. The discrete differentiation is more efficient in computation, following the regular computational domain assumption. However, in practice, the assumption does not necessarily hold. In this paper, we propose a PhyGNNet method to solve PDEs based on graph neural network and discrete differentiation on irregular domain. Meanwhile, to verify the validity of the method, we solve Burgers equation and conduct a numerical comparison with PINN. The results show that the proposed method performs better both in fit ability and time extrapolation than PINN. Code is available at https://github.com/echowve/phygnnet.

CCS CONCEPTS
• Computing methodologies → Causal reasoning and diagnostics.

KEYWORDS
Physics-informed neural networks, Partial differential equation, Graph neural networks, Surrogate modeling

ACM Reference Format:
Longxiang Jiang, Liyuan Wang, Xinkun Chu, Yonghao Xiao, and Hao Zhang. 2023. PhyGNNet: Solving spatiotemporal PDEs with Physics-informed Graph Neural Network. In 2023 2nd Asia Conference on Algorithms, Computing and Machine Learning (CACML 2023), March 17–19, 2023, Shanghai, China. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3590003.3590029

1 INTRODUCTION
With the development of physics, biology, chemistry, and other fields, a large number of partial differential equations [3] have been accumulated in the related fields, the evolution process and results of specific problems can be obtained by solving the PDEs. However, solving PDEs is a challenging task, except for a few equations which exist analytical solutions, recent years, most of the equations are solved numerically [5]. Along with the rapid progress and popularization of deep learning, benefit from the excellent fit ability of neural networks, a series of novel methods have emerged nowadays.

The current representative method is PINN [9], which mostly adopts the fully connected network as the solver, and constructs a multi-objective loss function with the loss of PDE through automatic differentiation to optimize the network parameters. The solver takes time-domain coordinates tuple as input and predicts the corresponding solution, the differential between prediction and input is calculated through back-propagation. The method is widespread in solving fluid flow [1], heat conduction [13], and other problems. However, the current PINN method has two limitations. On the one hand, automatic differentiation is time-consuming, and the consumption is significantly increased when the amount of parameters of the fully connected network is large, or the differentiation is high-order. On the other hand, PINN approximates the real solution by piecewise fitting, to ensure the correctness of the solution, a large number of collocation points are required for iterative training, which further increases the time consumption of solving.
In addition to the above PINN method, some methods refer to numerical calculation and adopt discrete differences to approximate differentiation. Different from PINN, these methods take the solution at a certain time step as neural network input, the neural network outputs the predicted solution on a time interval behind the input, by minimizing the loss of PDE constructed by discrete difference, and the prediction tends to the real solution, which is an iterative method on time step. Geo et al [6] proposed a method to calculate spatial discrete differential using convolution kernel with fixed weights and adapt convolution neural network (CNN) as the solver. Similar to [6], [10] introduced Conv-LSTM neural network to solve spatiotemporal partial differential equations. Furthermore, with FV discretization scheme and two-point flux approximation [4], [11] applies CNN to solve transient Darcy flows.

In this paper, we present a method for solving PDEs in irregular domain based on GNN and discrete difference. This method divides the computational domain into meshes and treats the meshes as an undirected graph. The solution of PDE at the vertices of the mesh is predicted by message passing mechanism of the graph neural network. In addition, this paper proposes a discrete difference method for calculating laplace and gradient values on irregular grids based on Taylor expansion and least squares regression, which is used to construct PDE loss to constraint graph neural network to predict solution. What’s more, in this paper, we solve burgers equation using the method presented and conduct comparative numerical experiments with PINN. The results show that this method has better solution accuracy and time extrapolation ability.

2 METHOD

The general form of PDEs can be expressed as:

$$\frac{\partial u}{\partial t} + N[u, \ldots, \nabla u, \Delta u, \nabla \cdot u, \ldots ; \lambda] = 0$$

where, \(u \in \mathbb{R}^{d \times 1}\) denotes solutions which have \(d\) components in domain \(\Omega\), \(\frac{\partial u}{\partial t}\) is time derivative, \(\nabla u\) represents the gradient in space. \(\Delta u\) is Laplace item, which is equal to \(\nabla^2 u\). \(\lambda\) is parameters of the PDE. In addition, the Initial Condition (IC) and Boundary Condition (BC) have the following definitions:

$$F(u, \nabla u, \Delta u, \ldots ; t = 0) = 0$$

$$B(u, \nabla u, \Delta u, \ldots ; x \in \partial \Omega) = 0$$

where, \(\partial \Omega\) denotes the boundary area of \(\Omega\).

Given the above PDE equation and Initial/Boundary Conditions, a solution can be found with several approaches. However, in this paper, we aim to solve the problem with GNN. Similar to PINN, we yearn to devise a method that is unsupervised to solve PDEs.

The framework of our approach to solving the above equation is illustrated in Fig.1. We first divide the domain into an irregular mesh and express the mesh as an undirected graph, then assign the solution in a time step to the nodes of the graph, note that at the very first time step, the solution is the IC. The network takes the graph and encodes the node attributions and edges into features, updates features with message passing, and then decodes the features into solution for next time which behind the input time \(\Delta t\) interval, the procedure is detailed in Sec.2.1. When training, the BC is assigned to the predicted solutions to compute PDE loss, the solution is then detached from the calculation graph and fed into the network to predict solutions of all \(T\) time steps by repeating the above procedure, and the losses of \(T\) steps are cumulated to update the parameters of the network at once. The above process is repeated many times until the specified number of times is reached or the network converges to get final solutions.

2.1 Network

We regard the mesh as an undirected graph to train the network. That is, we express the grid points as graph nodes, and assign edges to nodes that are nearest neighbors to each other. Similar to [8], the features of nodes consist of the current solution and 2-dimensional one-hot code of node type, indicating whether a specific node is located on \(\partial \Omega\). The features of edges contain euclidean distance and the coordinate difference between sender and receiver nodes.

The framework of our network is shown in Fig.1. The network put the solution at time \(t\) to obtain the solution at the next time step \(t + \Delta t\). It mainly has three parts. The encoder transforms node and edges features mentioned above with MLP, the processor predicts latent feature variation of nodes via Graph Network [2] (GN) and the decoder decodes node features with MLP as correction of the input \(u_t\) to create final predicts. The dotted lines in the figure represent residual connection.

Specifically, the calculation process of GN in the framework contains edges update and nodes update steps. To illustrate the procedure, here, we define the edge \(e_{i,j}\) connected to two nodes with features \(v_i\) and \(v_j\) respectively. The GN conducts edges update step at first, which can be described as:

$$\tilde{e}_{i,j} = f_e(e_{i,j} || v_i || v_j)$$

where \(||\) denotes concatenating operator and \(f_e\) denotes MLP for edges. With the updated edge features, GN then update node features as:

$$\tilde{v}_i = f_n(v_i) \bigoplus \sum_{j \in \mathcal{N}(i)} \tilde{e}_{i,j}$$

and, \(f_n\) denotes MLP for nodes.

2.2 Discrete format

The above PDE equation consists of three derivative operators, namely \(u_t\), \(\nabla u\), and \(\Delta u\). The three operators can be approximated by the following discrete format.

For operator \(u_t\), it can be approximated with backward difference on time \(t\), denoted as:

$$\frac{\partial u}{\partial t} \approx \frac{u_{t+\Delta t} - u_t}{\Delta t}$$

where \(\Delta t\) denotes the single time interval in the time evolution process, a superparameter.

Operators \(\nabla u\) and \(\Delta u\) are differential on spatial area, when the area \(\Omega\) is divided into irregular grids, the operators can be defined on the grids as well. For simplicity, we only take two-dimension as an example to illustrate the solution method of the operator, for others, the same.

According to first-order Taylor expansion the value at \(u_{x+\Delta x}\) can be approximated with the value at \(u_x\) as:

$$u_{x+\Delta x} \approx u_x + \nabla u_x \Delta x$$
Figure 1: The framework of our proposed method. Given initial and boundary conditions, we predict the solutions of multi time steps with the network consisting of an encoder, processer, and decoder blocks, where the processer is a graph neural network block.

Then, given $u_{k+\Delta x}$ and $u_x$, the gradients can be solved with least squares regression as:

$$\nabla u_x = (A^T A)^{-1} A^T U$$  \hspace{1cm} (8)

where, $A \in \mathbb{R}^{m \times 2}$ is coordinate differences matrix with $m$ nearest neighbors, denoted as:

$$\begin{bmatrix}
\Delta x_1, \Delta y_1 \\
\Delta x_2, \Delta y_2 \\
\vdots \\
\Delta x_{m-1}, \Delta y_{m-1} \\
\Delta x_m, \Delta y_m
\end{bmatrix}$$  \hspace{1cm} (9)

and $U \in \mathbb{R}^{m \times d}$ is value differences matrix with $m$ nearest neighbors, denoted as:

$$U = [u_{k+\Delta x_1} - u_x; u_{k+\Delta x_2} - u_x; \cdots; u_{k+\Delta x_m} - u_x]$$  \hspace{1cm} (10)

And, the Laplace value follow the form as:

$$\Delta u_i = \sum_{j \in N(u_i)} w_{ij} (u_j - u_i)$$  \hspace{1cm} (11)

which indicates that the laplacian value is a weighted difference of values of nearest neighbors, and the weights $w$ are only related to the nearest neighbor structure of the graph, therefore, here, we determine the weights based on a group of test function as following:

We expect to select representative functions to solve the weights, due to the linearity of the laplacian operator, here, we choose the basic function of the second-order Taylor expansion as $\{x, y, xy, x^2, y^2\}$. Given the coordinates of a point and its neighbors, the function values and laplacian values of the group can be numerically obtained. It is worth noting that to ensure numeric stability, all the coordinates are subtracted from the point coordinates. With the above test function values and corresponding Laplacian values, $w$ can also be solved with least squares regression as:

$$w_i = (\tilde{A}^T \tilde{A})^{-1} \tilde{A}^T \tilde{u}$$  \hspace{1cm} (12)

where $\tilde{A} \in \mathbb{R}^{5 \times m}$ is test function value difference matrix, and $\tilde{u} \in \mathbb{R}^{5 \times 1}$ is a vector filled with corresponding laplacian values.

2.3 PDE loss construction

The objective of our approach is to solve PDE on an irregular domain with GNN when the PDE equation, IC, and BC are explicitly presented. In PINN, the PDE equation, IC, and BC are softly satisfied when minimizing a multi-objective loss function. However, the loss function requires tuning the weight parameters of multiple losses carefully to avoid falling into local minima. In this paper, we construct a loss function with the same methodology as [10], given $u_1$, the network predicts $u_{T+\Delta t}$, the loss value of grid points has the
exact predict (our) predict (pinn) difference (our) difference (pinn) 
t=0.1 s

t=0.4 s

Figure 2: The results of burgers equation with \( u, v \) components at different time steps. The predicted results are compared with the exact analytical solutions and the difference is also presented.

following form:

\[
R(u_{t+\Delta t}^\theta; u_t, \theta) = \left| \frac{u_{t+\Delta t}^\theta - u_t}{\Delta t} \right| + N[u_{t+\Delta t}^\theta, \cdots, \nabla u_{t+\Delta t}^\theta] \|_2 
\] (13)

and the loss function to optimize is:

\[
L(u_{t+\Delta t}^\theta; u_t, \theta) = \frac{1}{N} \sum_{s=1}^{T} R(u_{t+\Delta t}^\theta; u_t, \theta) 
\] (14)

Notice that when training, we update the network parameters by cumulating gradients of multi time-steps. For each time step, the input is the network prediction of the last time step, and the IC is as input at the very first step of the network. Also, similar to [10], the BC is hard assigned when organizing loss. In detail, the values of boundary nodes in prediction are assigned according to the BC before PDE loss construction.

3 EXPERIMENTS

In this section, we conduct numerical experiments on burgers equation [12] to solve the propagation and reflection of waves to evaluate our proposed method.

3.1 Setup

As aforementioned, there are MLPs in the encoder, processor, and decoder of the neural network. In our experiments, the MLPs are with two hidden layers, each with 128 neurons, and the ReLU activation function is applied to transform the output of the input layer and hidden layer. For the variable parameters in discrete format, the time interval \( \Delta t \) is set to 0.001. The computational domain is set to a disk with 0.5 radius and (0.5, 0.5) as center. The learning rate of our method is set to \( 1 \times 10^{-4} \).

3.2 Burgers Equation

Here, we consider the two-dimensional burgers equation as example, which has the following form:

\[
\begin{align*}
    u_t + u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} &= \frac{1}{R} \Delta u = 0 \\
    v_t + u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} &= \frac{1}{R} \Delta v = 0
\end{align*}
\] (15)

where the parameter \( R \) is the Reynolds number, which controls the wave dynamics. Specifically, To verify the accuracy of our method, we choose to solve the equation with the exact solution same as
Table 1: the aRMSE at different time steps of Burgers Equation.

| Step | 1     | 100   | 200   | 300   | 400   |
|------|-------|-------|-------|-------|-------|
| PINN | 2.06e-04 | 2.01e-04 | 3.73e-04 | 8.60e-04 | 1.69e-03 |
| OUR  | 1.09e-06 | 3.09e-05 | 5.67e-05 | 8.00e-05 | 1.02e-04 |

[12], that is, the solution is shown below:

\[
\begin{aligned}
    u(x, y, t) &= \frac{3}{4} \frac{1}{4 (1 + e^{R(-t-4x+4y)})/32} \\
    v(x, y, t) &= \frac{3}{4} \frac{1}{4 (1 + e^{R(-t-4x+4y)})/32}
\end{aligned}
\]  \hspace{1cm} (16)

The IC is the exact solution at time \( t = 0 \) and the boundary value changes along with time \( t \) based on the solution. The parameter \( R \) is set to 80. We train the network to solve the PDE at the first 10 time steps and expect the model to have the ability to infer the subsequent solutions. In detail, we set \( T = 10 \) in this situation and repeat the training process 10000 epochs, taking the model with minimal PDE loss to evaluate performance.

In addition, based on DeepXDE [7] framework, we construct a PINN baseline as a comparison of our method, that is, we build an MLP model with 4 hidden layers, each layer containing 20 neurons and we choose the Tanh as activation function. When training, for a fair comparison, the computation area, and train time steps are the same as the items in our approach, we randomly sample \( 1.2 \times 10^5 \) collocation points and train the model with two stages, in the first stage, we train the model \( 1 \times 10^4 \) epochs by Adam optimizer with \( 1 \times 10^{-4} \) learning rate and then, in the second stage, the L-BFGS optimizer is adopted to further minimize the loss, which repeats \( 1 \times 10^5 \) epochs.

The results are demonstrated in Tab.1, which contains the aRMSE (accumulated Root Mean Square Error) [8, 10] at different time steps. Note that the model is built at the very first 10 steps, thus the results are an extrapolation of the model. As shown in the Table, our method maintains a low level of error over time, demonstrating the better fitting ability and long-time generalization performance of the model compared with the PINN approach. Besides, we also visualize our results in Fig.2. In the Figure, the first two rows in the figure are the results at 0.1s and the remaining indicates the results at 0.4s. As we can observe, the predicted patterns are close to the exact solutions, and the differences are near zero.

4 CONCLUSION

In this paper, we proposed a method to solve PDEs with a graph neural network on an irregular computational domain. That is, specifying the status of a time step, we supervised the network to predict the solution of the next time step by minimizing a PDE loss. Compared with the typical PINN methods, our approach doesn’t rely on automatic differentiation, the differential term is constructed with spatial and time differences. In particular, to obtain spatial differences on an irregular computational domain, we propose an approach that approximates gradients and laplacian values with least squares regression. The experiments conducted on the burgers equation show that our method performs better in fit ability and time extrapolation in contrast to the PINN method.

REFERENCES

[1] Muhammad M Almajid and Moataz O Abu-Al-Saud. 2022. Prediction of porous media fluid flow using physics informed neural networks. Journal of Petroleum Science and Engineering 208 (2022), 105265.
[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261 (2018).
[3] Haim Brezis and Haim Brézis. 2011. Functional analysis, Sobolev spaces and partial differential equations. Vol. 2. Springer.
[4] Zhangxin Chen, Guanren Huan, and Yuanle Ma. 2006. Computational methods for multiphase flows in porous media. SIAM.
[5] John R Dormand. 2018. Numerical methods for differential equations: a computational approach. CRC press.
[6] Han Gao, Luning Sun, and Jian-Xun Wang. 2021. PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. J. Comput. Phys. 428 (2021), 110079.
[7] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. 2021. DeepXDE: A deep learning library for solving differential equations. SIAM Rev. 63, 1 (2021), 206–228. https://doi.org/10.1137/19M1274067
[8] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. 2020. Learning mesh-based simulation with graph networks. arXiv preprint arXiv:2010.03409 (2020).
[9] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics 378 (2019), 686–707.
[10] Pu Ren, Chengping Rao, Yang Liu, Jian-Xun Wang, and Hao Sun. 2022. PhyCRNet: Physics-informed convolutional-recurrent network for solving spatiotemporal PDEs. Computers and Mathematics with Applications 60, 3 (2020), 608–626. https://doi.org/10.1016/j.camwa.2020.08.048
[11] Zhao Zhang. 2022. A physics-informed deep convolutional neural network for simulating and predicting transient Darcy flows in heterogeneous reservoirs without labeled data. Journal of Petroleum Science and Engineering (2022), 110179.
[12] Hongqing Zhu, Huazhong Shu, and Meiyu Ding. 2010. Numerical solutions of two-dimensional Burgers’ equations by discrete Adomian decomposition method. Computers & Mathematics with Applications 60, 3 (2010), 840–848.
[13] Navid Zobeiry and Keith D Hamfeld. 2021. A physics-informed machine learning approach for solving heat transfer equation in advanced manufacturing and engineering applications. Engineering Applications of Artificial Intelligence 101 (2021), 104232.