A Graph Neural Network Framework for Grid-Based Simulation (GSimGNN)

Haoyu Tang, Wennan Long
Energy Resources Engineering, Stanford University
Stanford
{hytang, wennan}@stanford.edu

Abstract
Reservoir simulations are computationally expensive in the well control and well placement optimization. Generally, numerous simulation runs (realizations) are needed in order to achieve the optimal well locations. In this paper, we propose a graph neural network (GNN) framework to build a surrogate feed-forward model which replaces simulation runs to accelerate the optimization process. Our GNN framework includes an encoder, a process, and a decoder which takes input from the processed graph data designed and generated from the simulation raw data. We train the GNN model with 6000 samples (equivalent to 40 well configurations) with each containing the previous step state variable and the next step state variable. We test the GNN model with another 6000 samples and after model tuning, both one-step prediction and rollout prediction achieve a close match with the simulation results. Our GNN framework shows great potential in the application of well-related subsurface optimization including oil and gas as well as carbon capture sequestration (CCS).

Keywords
Graph Neural Network · Reservoir Simulation · Buckley-Leverett · Well Placement Optimization

1 Introduction
The management of subsurface flow-related resources requires running computationally expensive simulations. This includes oil/gas production, Carbon Capture, Sequestration, and groundwater management. The cost gets even higher when there are optimization problems involved. For example, for the well placement optimization (WPO) problem, the goal is often to search the best well locations for each well to achieve the highest overall profit or net present value (NPV). Here, thousands of simulation runs need to be performed to find the optimal solution. For a practical optimization project, the whole process of running a simulation may take a few weeks or even months. With these concerns, researchers are currently studying the surrogate modeling to replace the full simulation so that those projects can be performed within a more reasonable time frame. One baseline model that has been previously explored is the surrogate using a low-fidelity simulation model with machine-learning error correction [1]. This surrogate model achieves speedup for a factor of 10 to 100 with less than 1% difference in optimal NPV compared with running full simulation. This paper introduces a new surrogate model that can learn from a small set of realizations running from simulation and has the inductive capability to forecast the rest of the cases. This surrogate model is built based on Graph Neural Network (GNN) framework, so we term it as GSimGNN.

Unlike previous work [2][3] focusing on generating graph using a physical substance (particles or objects), GSimGNN directly encodes well locations and grids as nodes and build edges between their neighbors. Pfaff et al. [4] had a similar approach. They build graphs using meshes, and they create the concept of world edges to connect meshes that are close in the distance. But the problem they solved was using the finite element method (FEM), which is different from ours (finite volume method). Our study shows that the GSimGNN can learn the grid-based multiphase flow simulations with acceptable error for well optimization-related applications.
2 Simulated Data Generation Method

This study generates the database for training, validation, and testing using a simulator because of the broad application of simulation in practice and industry, as we mentioned in the previous section. Currently, we generate a $90 \times 90$ two-dimensional channelized reservoir model with oil and water. As shown in Fig. 1, we have the channelized permeability field, and the rest simulation parameters are described in Table 1.

To begin with, we set up the model as a Buckley-Leverret problem that contains incompressible flow and rock and straight-line relative permeability relationship (mobility ratio is 1 for both water and oil phase). This results in the fixed pressure field map after the transient state of the simulation (usually reaches a steady state after the first time step). Therefore, the only changing state variable during the simulation is saturation. The Buckley-Leverret problem is a classical problem widely studied in subsurface flow theory, making it a solid starting point in this study.

Table 1: Simulation Model Parameters

| No. | Property Name               | Value & Unit                  |
|-----|-----------------------------|-------------------------------|
| 1.  | Grid Number                 | $90 \times 90$               |
| 2.  | Grid Size                   | $32.8 \text{ m} \times 32.8 \text{ m}$ |
| 3.  | Porosity                    | 0.1                           |
| 4.  | Initial Water Saturation    | 0.2                           |
| 5.  | Initial Pressure            | 200 bar                       |
| 6.  | Injector BHP                | 300 bar                       |
| 7.  | Producer BHP                | 100 bar                       |
| 8.  | Total Time                  | 3000 days                     |
| 9.  | Single Time Step            | 20 days                       |
| 10. | Time Step Number per Simulation | 150                           |
| 11. | Number of Injector          | 5                             |
| 12. | Number of Producer          | 5                             |

One of the most critical applications of surrogate modeling in reservoir management is the well placement optimization (WPO), as it has significant impacts on the environment and economics. In the current study, we prepare the dataset by generating 100 different well configurations for all 10 wells (5 injectors and 5 producers as mentioned in Table 1. This means that in each configuration, the placement of all 10 wells is completely random and different from other configurations. By having this setup, our surrogate model can learn different scenarios from different well configurations and, therefore can be applied to WPO problems in practice.
A Graph Neural Network Framework for Grid-Based Simulation

3 GSimGNN Model Framework

3.1 Graph Generation

The simulation grids contain oil, water, and rock in a waterflooding reservoir simulation. In this study, the authors define a node as a grid shown in Figure 2. It indicates the conversion from simulation grids to GSimGNN nodes. Due to the pressure gradient, the fluid (oil and water) flows between two neighbor grids. Thus the edge is built between them. The figure shows three-by-three two dimensional grids with one injector and one producer setup.

In the initial state, the volume of oil and water and pressure field stay constant, but when pressure changes due to the kick of sink or source term, the volume will change with time. For the arbitrary grid i, the mass conserved in the following equation (Equation 1).

$$\nabla \cdot \left( \sum_j x_{jr}^i \rho_j \mathbf{v}_j \right) + q^r = \frac{\partial}{\partial t} \left( \sum_j \phi x_{jr}^i \rho_j S_j \right),$$

where subscript $j$ indicates phases, superscript $r$ indicates components. $\rho_j$ is phase density, $x_{jr}^i$ is the mass fraction of component $r$ in phase $j$. $\mathbf{v}_j$ is the Darcy velocity (given below), $q^r$ represents the source/sink term, $t$ is time, $\phi$ is porosity, and $S_j$ is the phase saturation. The Darcy velocity for phase $j$, $\mathbf{v}_j$, is given by

$$\mathbf{v}_j = -\frac{k r_{jr}^i (S_j)}{\mu_j} \nabla p_j,$$

where $k$ is the absolute permeability tensor, $r_{jr}^i$, $\mu_j$ and $p_j$ denote relative permeability, viscosity and fluid pressure of phase $j$, respectively. In this study, the system has two components and two phases: water and oil. To solve water and oil saturation for the arbitrary grid. The node needs to have the features shown in the Table 2. The author used one-hot encoding to represent different types of grids shown in the Figure 4.

3.2 Graph Neural Network Framework for Simulation

The current section explains the GNN workflow trained with the graph data generated from simulation data. The overall structure of the GNN workflow contains an encoder, a processor, and a decoder (2), as is shown in Fig. 7. It takes the
A Graph Neural Network Framework for Grid-Based Simulation

Table 2: Node Features

| No. | Feature Name | Unit   |
|-----|--------------|--------|
| 1.  | Pressure     | Bar    |
| 2.  | Saturation   | -      |
| 3.  | Permeability | Darcy  |

Figure 4: One-hot encoding for different types of grids

input of each node feature and outputs the updated state variable (saturation in this study) of every node for the next time step.

To begin with, an encoder network is performed on every single node in the graph. The object of this process is to embed the features and project them to a standard dimension. The feature embedding can then be used for further treatment. Next, a processor is applied to the embedded features. This process involves the interaction of different nodes in the graph, and the interactions are computed among nodes based on their connection relationship. The computation is achieved by doing message passing and aggregation between nodes. The state’s dynamic transition among nodes is then completed during the process. The processor outputs the updated embedded features for the next step. Finally, a decoder is used to generate the final output. Similar to the encoder, the decoder is applied to every single node in the graph. It projects the embedded output back to the desired dimension as the final output.

Figure 5: GNN workflow with encoder, processor, and decoder.

3.3 GSImGNN Implementation Details

This section discusses the implementation details of our GSImGNN surrogate model for reservoir simulation. Our implementation is done using Pytorch Geometric (PyG) library and referencing the previous related work [2].

The input of the current study contains 7 different features, i.e., current pressure, current saturation, node permeability, node well bottom hole pressure (BHP, default value for non-well cells), one-hot encoding to identify different types of cells (with 3 dimensions). The output currently only contains future saturation as the pressure field is steady during the
A Graph Neural Network Framework for Grid-Based Simulation

Figure 6: Two-hops computation graphs for (a) corner node, (b) edge node and (c) inner node. PIA represents permutation invariant aggregation, and MLP represents multi-layer perception.

Buckley-Leverett setup simulation. Both input and output are assigned to each node within the graph during the graph generation.

The encoder of this study is a multi-layer perceptron (MLP) of 2 linear layers with an activation layer after each. As mentioned in the previous section, the encoder is performed on every single node of the graph to a standard dimension of 16.

Our processor is composed of 3 single-step processor. Each of the single-step processors first collects information from neighbor nodes. The collected information is then stacked with current node features before being processed by another MLP of 2 layers plus activation. The output of MLP is then aggregated with those from other neighbor nodes. The whole process is repeated 3 times for each node so that each node can be updated with enough information from neighbors as well as their neighbors.

Like the encoder, our decoder is another MLP of 2 linear layers with an activation layer after each. The decoder would be applied to each node to project them back to the output dimension which is 1 in the current study (future saturation).

In the graph, we have three different nodes based on the number of neighbors: nodes in the corner, nodes on the edges, and inner nodes. Take the graph in Figure 6 for example, the computation graphs for each of nodes are shown in the Figure 6.

4 Results and Discussion

As mentioned in the previous section and Table 1, we generated multiple simulation cases with different random well locations for 5 injectors and 5 producers. Each generated well configuration is simulated for 3000 total simulation days.
with 150-time steps. There are in total 100 well configurations generated in the current study which includes a total of 15000-time steps for GNN to learn. The dataset is split into 6000 samples for training (equivalent to 40 simulation cases), 3000 samples for validation and model tuning (equivalent to 20 simulation cases), and 6000 samples for testing (equivalent to 40 simulation cases).

We need to mention here that our GNN model is the original setup for predicting one time step during the training stage. An excellent one time step prediction can be achieved quite easily without much tuning during the testing stage. However, as the simulator performs multiple-step predictions, we also need to do rollout predictions with GNN (the next step prediction is based on the previous step prediction instead of given hard data). This could introduce accumulated errors that result in deviations from simulation results after multiple steps (in our case, more than 50 steps of rollout would usually introduce obvious errors). The problem can be resolved by using L1 loss (on top of the original L2 loss) plus Gaussian noise during the training stage.

Next, we demonstrate the GNN performance during the rollout with several figures. Here, we selected 5 out of 40 test case results for presentation purposes. We present both saturation maps and oil/water well rates. As is shown in Fig. 8 we plotted the saturation field map at 1500 days and 3000 days for simulation results, GNN results, and their difference. The difference between GNN results and simulation results at 1500 days ranges from 0.05 to -0.15 and 3000 days ranges from 0.1 to -0.2. As mentioned in the previous paragraph, the error accumulates during the rollout which explains the pattern of increasing error shown in the difference map. In visual, the saturation shape of both time steps generated from GNN matches well with the simulation results. This indicates that our GNN model learns to simulate multiphase flow in 2D. In Fig. 9 we pick three producers (with relatively higher rates) to show the ability of our GNN model to predict well rates which can be used for the well control and placement optimization. The red line (simulation results) and black line (GNN results) align well with each other with a close trend near the end of the simulation. Similar patterns can be observed for the water rate as well. For other saturation maps (Fig. 10, 12, 14, 16) and well rates (Fig. 11, 13, 15, 17), we have similar observations.

Figure 8: Saturation fields for Case 1 of 1500 days (upper row) and 3000 days (lower row) for prediction from simulation (left), GNN (middle), and their difference (right).
A Graph Neural Network Framework for Grid-Based Simulation

Figure 9: Well rates for Case 1 of oil (upper row) and water (lower row) for well 1 (left), well 2 (middle), well 3(right).

Figure 10: Saturation fields for Case 2 of 1500 days (upper row) and 3000 days (lower row) for prediction from simulation (left), GNN (middle), and their difference (right).
Figure 11: Well rates for Case 2 of oil (upper row) and water (lower row) for well 1 (left), well 2 (middle), well 3(right).

Figure 12: Saturation fields for Case 3 of 1500 days (upper row) and 3000 days (lower row) for prediction from simulation (left), GNN (middle), and their difference (right).
A Graph Neural Network Framework for Grid-Based Simulation

Figure 13: Well rates for Case 3 of oil (upper row) and water (lower row) for well 1 (left), well 2 (middle), well 3(right).

Figure 14: Saturation fields for Case 4 of 1500 days (upper row) and 3000 days (lower row) for prediction from simulation (left), GNN (middle), and their difference (right).
Figure 15: Well rates for Case 4 of oil (upper row) and water (lower row) for well 1 (left), well 2 (middle), well 3(right).

Figure 16: Saturation fields for Case 5 of 1500 days (upper row) and 3000 days (lower row) for prediction from simulation (left), GNN (middle), and their difference (right).
Figure 17: Well rates for Case 5 of oil (upper row) and water (lower row) for well 1 (left), well 2 (middle), well 3 (right).
References

[1] Haoyu Tang and Louis J Durlofsky. Use of low-fidelity models with machine-learning error correction for well placement optimization. *arXiv preprint arXiv:2111.02960*, 2021.

[2] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[3] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR, 2018.

[4] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.