Learning Domain-Specific Heuristics with Graph Convolutional Networks

Matheus Z. Marcon
Graduate Program in Computer Science - School of Technology
Pontifical Catholic University of Rio Grande do Sul - PUCRS
Porto Alegre, Brazil
matheus.marcon@edu.pucrs.br

Abstract

Heuristic functions are essential tools for improving the performance of search algorithms. Off-the-shelf functions work well on a variety of domains, but may however behave poorly in more complex scenarios. In this work, we propose the usage of Graph Convolutional Networks (GCNs) for exploiting the inherent graph-like structure of planning tasks and learning domain-specific heuristic values for non-consuming problems, which could later be generalized for harder instances.

Introduction

Automated Planning (AP) is a branch of Artificial Intelligence (AI) which seeks to compute sets of actions, or plans, that fulfill a given task. Planning can, however, become expensive in certain situations, making the selection of optimal heuristic functions of great importance.

An heuristic $h(S)$ informs the planner about cost estimates from a given state S to the goal S^G, guiding the search process towards more efficient paths. In order to function properly, heuristics need to be informative, returning cost values as close as possible to the true remaining cost. When tackling complex scenarios, off-the-shelf heuristics can nonetheless present scaling problems and lead to poor planning and performance results when compared to toy problems.

As such, domain-specific designs might be required. Designing heuristics in such a manner relies on accurate domain and search control knowledge by an expert, which is unfeasible for many real-world problems. Machine Learning (ML) techniques have been widely used as an alternative to human knowledge to circumvent these issues, as thoroughly reviewed in (Jiménez et al. 2012).

Deep Learning (DL) is a sub field of ML which has been popularized for its capabilities of working on unstructured data. In the field of AP, it has been previously used to find optimal search heuristics for given tasks (Sigurdson and Bulitko 2017; Loreggia et al. 2016).

Introduced in 2012 (Krizhevsky, Sutskever, and Hinton 2012), Convolutional Neural Networks (CNNs), Deep Learning's most used network architecture, consist of a data-driven approach for feature selection, achieving state-of-the-art performance in image and text data analysis.

Planning tasks can be naturally represented in graphs, where a graph G is composed of nodes N which represent states S, and edges ε representing the actions connecting each state. Graph Convolutional Networks (GCNs), graph analogs to CNNs introduced in (Defferrard, Bresson, and Vandergheynst 2016), are networks which take graph data as input. These networks compute topological or "neighbourhood" information about each data object, departing from the purely local bias of CNNs.

In this work, we propose the usage of GCNs for learning domain-specific heuristics for non-complex scenarios, with the aim of later generalizing obtained results for more demanding problems. To the best of our knowledge, this is the first work to tackle the generation of domain-specific heuristics by employing Deep Learning techniques.

Method

In this section we will detail our intended approach and further elaborate on the chosen methods.

Graph Convolutional Networks

Graph Convolutional Networks were first introduced in (Bruna et al. 2013) and elaborated in (Defferrard, Bresson, and Vandergheynst 2016), where Graph Signal Processing techniques were employed for computing graph convolutions in the spectral domain. Graphs are mathematical structures capable of encoding complex pairwise data relationships, which can be exploited with Graph Theory techniques.

In (Kipf and Welling 2016), a simplification of spectral GCNs is proposed, with interesting results for citation networks classification. This approach makes use of Chebyshev polynomial approximations to reduce computational demands of graph convolution operations.

Our proposed network for training consists of two Graph Convolutional layers with ReLU and Dropout, as described in Table 1.

The network is fed with plan graphs for a specific domain, where each node N represents a state S on the search tree. Each input graph is a subgraph of the whole tree, so that many different graphs are generated for a single problem.

Table 1: Proposed Network Architecture

Layer	Input	Output
Graph Conv	$\lvert F\rvert$	60
ReLU	-	-
Dropout	-	-
Graph Conv	60	1
ReLU	-	-

Performance was evaluated by comparing GCN heuristic value outputs with off-the-shelf heuristic functions. As metrics, we used the number of search expansions required for obtaining a plan and average plan length.

Dataset Generation

One of our challenges is creating a diverse enough dataset to contain the necessary semantic representations for the network to learn and generalize with confidence. Our approach was to create a high number of unique graphs which carry state transition topology and heuristic value information.

For facilitating the development of our model, we have restricted our experiments to varying instances of Blocksworld-4ops and Logistics domains. These two were selected as initial benchmarks for their differing degrees of complexity and homogeneity. We made use of the generator made available by Joerg Hoffmann[1] for creating 100 unique planning tasks for a given number of objects in a domain.

Training

For training our network, we create multiple subgraphs g for each task Π, wherein an heuristic value $\hat{h}_{i,g}$ is attributed to each subgraph node. This value is computed as the relaxed plan length from each state s to S^G. Planning is made using A^*_{GCN} search informed by hFF heuristic.

A^*_{GCN} consists of a slight modification on regular A^* which receives a state from which to start searching instead of the initial state of the task at hand. hFF is used as heuristic for its better cost-wise performance compared to optimal heuristics and for being well informed, even if not admissible. We used the values resulting from this process as ground-truth for our network.

In the case of more demanding instances, this approach showed itself too costly to compute. Thus, in such cases, we made a switch from A^*_{GCN} with hFF heuristic to Greedy Best-First Search (GBFS), so that our experiments could be performed in feasible time.

Each observed state is expanded until e expansions are reached relatively to the Root node, which is the first to be searched and expanded. The Root node receives the value of S_0 on the first iteration, and the value of a random observed node from the Node set during the following n iterations, each of which will generate one instance of g. This procedure is described below on Algorithm 1:

Algorithm 1 Generate graphs for each task Π

```
 1: Root ← $S_0$
 2: Nodes ← Set(Root)
 3: Solutions ← List()
 4: Plan ← $A^*_{GCN}(\Pi, Root)$
 5: ADD(Solutions, LENGTH(Plan))
 6: Closed ← Set(Root)
 7: Queue ← Set(⟨ Root, level ← 1⟩)
 8: for $i \leftarrow 1, n$ do          ▷ (create graphs from $n$ Roots.)
 9:     $s$, level ← POPLEFT(Queue)
10:     $S' \leftarrow List(\text{EXPAND}(s))$          ▷ (Get child states)
11:     for $s' \in S'$ do
12:         if $s' \notin$ Closed ∧ level $< e$ then
13:             Plan ← $A^*_{GCN}(\Pi, s')$
14:             ADD(Solutions, LENGTH(Plan))
15:             Nodes ← Nodes · $s'$
16:             Queue ← Queue · ⟨$s'$, level+1⟩
17:             Closed ← Closed · $s'$
18:     $g$ ← ADJACENCYMATRIX(Nodes, Solutions)
19:     $f$ ← GENERATEFEATURES(Nodes)
```

In the computed adjacency matrices, neighbouring nodes are attributed value 1, and node to goal adjacencies are given the nodes heuristic value. Goal nodes are attributed value 0.01, as values of zero represent no contiguity. For our experiments, we defined a maximum expansion level e as 3, and the number of generated graphs n as 100, resulting in approximately 10.000 graphs of varying sizes. We performed a simple 80/20 split on the generated data between training and test sets.

To make sense of this topological data, our network also receives as input F feature vectors of each node or state on the graphs. As we aim at avoiding the necessity for domain expert knowledge, we compose feature vectors as just binary representations of each possible domain predicate. Each binary value encodes a combination of predicate and object or group thereof. Taking for example an instance of Blocksworld domain with 2 objects, the resulting feature vector F has size 9, consisting of 2^1 bits for $CLEAR$, 2^1 for ON-TABLE, 2^2 for ON and 2^0 for HANDEMPTY.

Goal states S^G are encoded with the same procedure described above. Their representations are added to each node observed for a task, and the mean average for every value is taken in order to grant weighting to predicates in accordance with the goal.

Our network is trained by passing as input the generated graphs and feature vectors through the two layers of Graph Convolutions described in Table 1. Thus, the output is a prediction of heuristic value for each node on the input graph. Since ground-truth heuristic values $h_{i,g}$ are attributed to each node, we can define a loss function using *l2* regularized Mean Squared Error between ground-truth and predicted heuristics made for each graph node:

$$L_h(g) = \sum_{i=1}^{N} (\hat{h}_{i,g} - h^*_{i,g})^2 + \lambda \|\Theta\|_2, \qquad (1)$$

where $\hat{h}_{i,g}$ is the computed heuristic value for node *i* in subgraph g, h^*_i is the perfect heuristic for node *i*, and *N* is the total number of nodes. λ and Θ refer to the *l2* regularization

[1]http://fai.cs.uni-saarland.de/hoffmann/ff-domains.html

hyperparameters.

Inference

Our method presents important differences than regular heuristics concerning inference. Since our network input is composed of graphs, and not simple states, each node to be evaluated must be expanded, so that we can use its neighbourhood topology data. Thus, evaluating node N consists of applying once more Algorithm 1 for generating its adjacency matrix. N is expanded e levels, as done during training.

Again as in training, feature vectors F are generated for each node in the graph, resulting in a NxF feature matrix, alongside the NxN adjacency matrix. Both matrices are then fed to the network, which outputs $\hat{h}$. This set of operations might at first seem overly complex for being performed during inference time. However, only a single node is being expanded at each inference, and heuristic computations for each node are done in constant time. Thus, our method is able to perform robustly.

Experiments

As previously mentioned, our experiments were restricted to different instances of Blocksworld-4ops and Logistics domains. More specifically, we selected a couple varying number of objects for each domain. Regarding Blocksworld, we tested instances with 3, 5 and 6 blocks. For Logistics, we used 2-1-2-1, 2-2-2-2 and 3-3-3-3 configurations of cites, city-sizes, airports and packages respectively.

In every execution, we compare the performance of our approach to Blind and LM-Cut heuristics. A Blind heuristic is the equivalent of using no heuristic, or a uninformed search. LM-Cut is the current state-of-the-art heuristic for optimal planning (Helmert and Domshlak 2009).

Apart from domain-specificity, we must also treat within domain variations in the number of objects. The size of feature vectors F vary according to the number of objects. As such, a model trained with a larger number of objects, and therefore larger F size can be used to infer heuristics of smaller instances, as long as F vectors are padded accordingly.

We also performed a variation of Algorithm 1 by expanding nodes inversely, starting from S^G and finding their predecessors. The idea is that our heuristic could gain more relevant information when prioritizing expanding nodes topologically closer to S^G.

When evaluating small instances, our approach behaves very closely to LM-Cut. In Figure 1, we can see the behaviour of our model for solving Logistics 2-1-2-1 tasks regarding mean number of expansions required for computing a plan. We infer these instances using models trained with Logistics 2-2-2-2 and Logistics 3-3-3-3.

Regarding the model trained with 2-2-2-2 instances, the number of expansions of our approach, noted as GCN, is basically the same as LM-Cut, both being way below the results for Blind search. Results for the model trained with larger instances, however, show relevant worsening, with the GCN number of expansions being comparable to that Blind search.

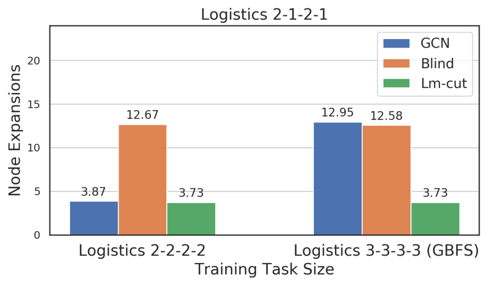

Figure 1: Number of expansions using different models for Logistics 2-1-2-1 inferences.

In the smaller instance of the Blocksworld domain, contaning 3 blocks, our approach shows the same behaviour, as can be seen of Figure 2. When using the model trained with instances of the same size of the ones inferred, expansions are similar to LM-Cut. When using padded models trained with larger instances, the results show once more a larger number of expansions, gradually approximating that of Blind search.

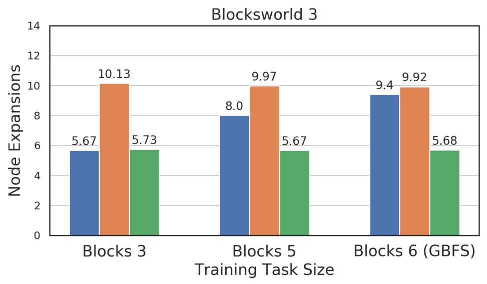

Figure 2: Number of expansions using different models for Blocksworld 3 inferences.

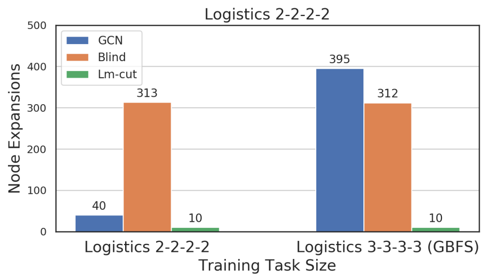

Figure 3: Number of expansions using different models for Logistics 2-2-2-2 inferences.

In Figure 3, we can see results for the inference of Logistics 2-2-2-2 task instances. Using the model trained with same sized instances, our approach shows four times more expansions as the state-of-the-art, but largely outperforms Blind search, which shows it was able to learn informative domain representations. The model trained with larger instances and GBFS, however, provides little to no information. The usage of GBFS algorithms surely plays a part in this results, but consisted of the only feasible approach for computing graphs in this instance size.

Nevertheless, usage of GBFS was not the only issue presented by our approach. In Figure 4, we see the results for inferences on Blocksworld-5 instances, which show negative results for our approach. In different models, inferences performed more expansions than Blind search.

In order to try and circumvent these results, we performed the inverse computation of the graphs, in the procedure previously described. This approach caused overfitting in the training model, which translated in an even larger number of expansions during inference. Again, usage of GBFS presented the worst results, performing over one thousand mean expansions during planning.

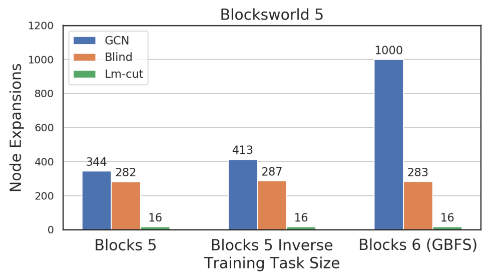

Figure 4: Number of expansions using different models for Logistics Blocksworld 5 inferences.

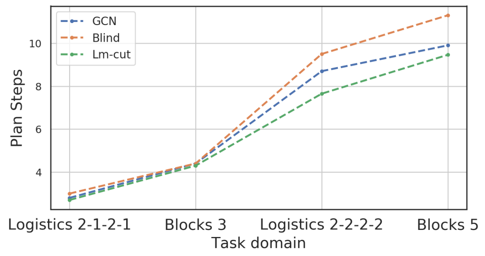

Figure 5: Average number of steps for each analyzed heuristic on different domains.

As additional analysis, we can see on Figure 5 the average number of steps on the plans computed by each heuristics on each of the previously mentioned domain instances. The solutions achieved by the different heuristics are similar, with our approach being slightly closer to LM-Cut than Blind search, even on domains where a larger number of expansions were required.

Training time among all performed experiments averaged on 2 hours duration, employing a Nvidia Tesla K40c. Our work used the Pytorch implementation of GCNs made available by Kipf and Welling on GitHub[2].

Conclusions and Future Work

In this work, we have investigated the use of Graph Convolutional Networks for learning domain-specific heuristics, a very pressing matter for facilitating planning in complex scenarios. Our approach has thus far failed to achieve good results on the selected tasks. Nonetheless, our results provide many insights on the challenges regarding heuristics learning and the usage of Deep Learning Algorithms for this goal. Even if the heuristics learned for complex tasks were not satisficing, we believe the results achieved for smaller instances show that better results could have been accomplished with further experimentation.

The biggest issue with the current approach is its high memory requirements during graph generations. Methods for guiding node expansions toward more promising nodes, such as to avoid dead-ends for example, could aid at generating more informative graphs while reducing time costs.

Regarding our network, more effort could be made on fine tuning the used hyperparameters. Additionally, The GCN implementation provided by Kipf and Welling could also not be optimal for our approach. Their approach relies on mathematical simplifications for improving performance, which could be affecting our results. Since our approach's bottleneck is not located on the network, different implementations which perform complete spectral convolutions with focus on graphs of varying sizes could be employed. We will continue to investigate these and further possible improvements.

[2]https://github.com/tkipf/pygcn

References

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2013. Spectral networks and locally connected networks on graphs. *CoRR* abs/1312.6203.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR* abs/1606.09375.

Helmert, M., and Domshlak, C. 2009. Lm-cut: Optimal planning with the landmark-cut heuristic.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(4):433–467.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. 1097–1105.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. 2016. Deep learning for algorithm portfolios. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, 1280–1286. AAAI Press.

Sigurdson, D., and Bulitko, V. 2017. Deep learning for realtime heuristic search algorithm selection.