Learning Domain-Independent Heuristics over Hypergraphs

William Shen, Felipe Trevizan, Sylvie Thiébaux

The Australian National University
Learn domain-independent heuristics

- Learn entirely from scratch
  - Do not use hand-crafted features
    - e.g. Learning Generalized Reactive Policies using Deep Neural Networks [Groshev et al. 2018]
  - Do not rely on existing heuristics as input features
    - e.g. Action Schema Networks: Generalised Policies with Deep Learning [Toyer et al. 2017]
  - Do not learn an improvement for an existing heuristic
    - e.g. Learning heuristic functions from relaxed plans [Yoon et al. 2006]
Learn domain-independent heuristics

- Generalise to:
  - different initial states, goals
  - different number of objects
  - different domains
    - domains unseen during training

domain-independent!
STRIPS

$P = \langle F, A, I, G, c \rangle$

- $F$ is the set of propositions
- $A$ is the set of actions
  - Each action has preconditions, add-effects & delete-effects
- $I \subseteq F$ is the initial state
- $G \subseteq F$ is the goal states
- $c$ is the cost function

unstack(1, 2)

PRE: on(1, 2), clear(1) ...
EFF: holding(1) clear(2) ¬on(1, 2) ...
Hypergraph for the delete relaxation

- **Hyperedge**: edge that joins any number of vertices

The delete-relaxation $P^+$ of problem $P$ can be represented by a **hypergraph**

**Delete-Relaxation**: ignore delete effects for each action
The $h^{\text{add}}$ heuristic estimates the cost of the goal as the sum of the costs of each proposition. This assumes that achieving each proposition is independent.

- **Overcounting**
- **Non-admissible**

Mathematically, it is defined as:

$$h^{\text{add}}(s) = \sum_{g \in G} h^{\text{add}}(s; g)$$

where $h^{\text{add}}(s; g)$ is the cost of achieving proposition $g$. This heuristic is not admissible, meaning it may overestimate the true cost of reaching the goal.
$h_{\text{max}}$ heuristic

$\max_{g \in G} h_{\text{max}}(s; g) = \max_{g \in G} \text{cost of achieving } g$

- Estimate cost of goal as the most expensive goal proposition
- Admissible but not as informative as $h^{\text{add}}$
Learning Heuristics over Hypergraphs

- Learn a function $\oplus$ which better approximates shortest paths
Learning Heuristics over Hypergraphs

- Learn function $h: \text{hypergraph} \rightarrow \mathbb{R}$

![Hypergraph diagram]

<table>
<thead>
<tr>
<th>Input Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
</tr>
<tr>
<td>e.g. $\alpha$</td>
</tr>
<tr>
<td>Proposition</td>
</tr>
<tr>
<td>e.g. $e_2$</td>
</tr>
<tr>
<td>$v_1$</td>
</tr>
</tbody>
</table>
Hypergraph Networks (HGN)

- Our generalisation of *Graph Networks* [Battaglia et al. 2018] to hypergraphs

- **Hypergraph Network (HGN) Block**
  - Powerful and flexible building block
  - Hypergraph-to-Hypergraph mapping
  - Uses message passing to aggregate and update features with update/aggregation functions
Hypergraph Networks (HGN)

Figure from Battaglia et al. 2018

(a) Edge update
(b) Node update
(c) Global update

Analogous to Message Passing

Figure from Battaglia et al. 2018
STRIPS-HGN

\[ G_{\text{hid}}^{t-1} \rightarrow HGN_{\text{core}} \times M \rightarrow G_{\text{hid}}^t \]

\[ G_{\text{hid}}^0 \]

\[ G_{\text{inp}} \rightarrow HGN_{\text{enc}} \]

\[ G_{\text{out}} \rightarrow HGN_{\text{dec}} \]
STRIPS-HGN

Input features
Hypergraph structure

$G^{t-1}_{hid}$

$G^0_{hid}$

$HGN_{enc}$

$HGN_{core}$

$G^t_{hid}$

$HGN_{dec}$

$G^{t}_{hid} \times M$

$G_{out}$
STRIPS-HGN

Encoder Block

\[ G^{t-1}_{\text{hid}} \rightarrow \text{HGN}_{\text{core}}^x M \rightarrow G^t_{\text{hid}} \]

\[ G^0_{\text{hid}} \rightarrow \text{HGN}_{\text{enc}} \rightarrow G_{\text{inp}} \]

\[ \text{HGN}_{\text{dec}} \rightarrow G_{\text{out}} \]
STRIPS-HGN Encoder

Latent proposition and action features
STIRIPS-HGN Encoder

Multilayer Perceptrons

Latent proposition and action features
STRIPS-HGN

$G_{hid}^{t-1}$ → $HGN_{core}$ → $G_{hid}^t$

Initial Latent features

$G_{hid}^0$
STRIPS-HGN

Recurrent Latent features

Initial Latent features
STRIPS-HGN

Core Message Passing Block

Propagates information through the hypergraph!
STRIIPS-HGN Processing

Latent heuristic value!

Updated proposition and action features
STRIPS-HGN

\[ G_{hid}^{t-1} \xrightarrow{} HGN_{core} \xrightarrow{\times M} G_{hid}^{t} \]

\[ G_{hid}^{0} \]

\[ HGN_{enc} \]

\[ G_{inp} \]

\[ HGN_{dec} \]

\[ G_{out} \]

Updated Latent features
STRIPS-HGN

\[ G^{t-1}_{\text{hid}} \rightarrow HGN_{\text{core}} \rightarrow G^t_{\text{hid}} \]

\[ G^0_{\text{hid}} \rightarrow HGN_{\text{enc}} \]

\[ G_{\text{inp}} \]

\[ HGN_{\text{dec}} \]

\[ G_{\text{out}} \]
STRIPS-HGN

\[ G^{t-1}_{hid} \rightarrow HGN_{core} \rightarrow G^t_{hid} \]

\[ G^0_{hid} \xrightarrow{\times M} HGN_{enc} \]

\[ HGN_{dec} \]

\[ G_{inp} \rightarrow G_{out} \]

Updated Latent features
STRIPS-HGN

Decoder Block
STRIPS-HGN Decoder

$u^t_{\text{hid}}$  

$V^t_{\text{hid}}$  

$E^t_{\text{hid}}$  

Edge block  |  Node block  |  Global block

Decoded heuristic value (real number)
Training a STRIPS-HGN

- **Input Features** - learning from scratch
  - Proposition:
    - [proposition in current state, proposition in goal state]
  - Action: [cost, #preconditions, #add-effects]

- **Generate Training Data**
  - Run an optimal planner for a set of training problems
  - Use the states encountered in the optimal plans
  - Aim to learn the optimal heuristic value

- **Train using Gradient Descent, treat as regression problem**
Experimental Results

- Evaluate using A* Search

- Baseline Heuristics
  - $h^{add}$ (inadmissible), $h^{max}$, blind and Landmark Cut (admissible)

- STRIPS-HGN: $h^{HGN}$
  - Train and evaluate on a single CPU core
  - Run core block 10 times (i.e., $M = 10$)
  - Powerful generalisation but slower to compute
### Evaluation on domains we trained on

<table>
<thead>
<tr>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Zenotravel</strong></td>
<td><strong>Blocksworld</strong></td>
</tr>
<tr>
<td>10 small Training Problems</td>
<td>18 larger Testing Problems</td>
</tr>
<tr>
<td>2-3 cities</td>
<td>4-20 balls</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Gripper</strong></th>
<th><strong>Gripper</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>3 small Training Problems</td>
<td>10 small Training Problems</td>
</tr>
<tr>
<td>1-3 balls</td>
<td>4-5 blocks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Blocksworld</strong></th>
<th><strong>Blocksworld</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>100 larger Testing Problems</td>
<td>6-10 blocks</td>
</tr>
</tbody>
</table>

- Train and evaluate **a single network** on 3 domains.
- **Training time:** 15 min
Blocksworld (trained on)
Train on Zenotravel, Gripper & Blocksworld

95% confidence interval shown for $h^{HGN}$ over 10 repeated experiments.
Gripper (trained on)
Train on Zenotravel, Grippeer & Blocksworld

Number of Nodes Expanded by A*

Deviation from Optimal Plan Length with A*
### Evaluation on domains we did **not** train on

**Training**

- **Zenotravel**
  - 10 small Training Problems
  - 2-3 cities

- **Gripper**
  - 3 small Training Problems
  - 1-3 balls

<table>
<thead>
<tr>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zenotravel</td>
<td>Blocksworld</td>
</tr>
</tbody>
</table>

- 50 Testing Problems
- 4-8 blocks

- **Train a single network on 2 domains. Evaluate on new unseen domain.**
- **Training time:** 10 min
Blocksworld (not trained on)
Train on Zenotravel and Gripper only.
Future Work

- **Speeding up a STRIPS-HGN**
  - Slow to evaluate - bottleneck
  - Optimise Hypergraph Networks implementation
  - Take advantage of multiple cores or use GPUs for parallelisation

- **Improve Generalisation Performance**
  - Use richer set of input features
  - Careful study of hyperparameter space, similar to [Ferber et al. 2020]
Thanks!