Specifying Goals to Deep Neural Networks with Answer Set Programming

Forest Agostinelli, Rojina Panta, Vedant Khandelwal
Department of Computer Science and Engineering, University of South Carolina, Columbia, South Carolina, USA
AI Institute, University of South Carolina, Columbia, South Carolina

Introduction

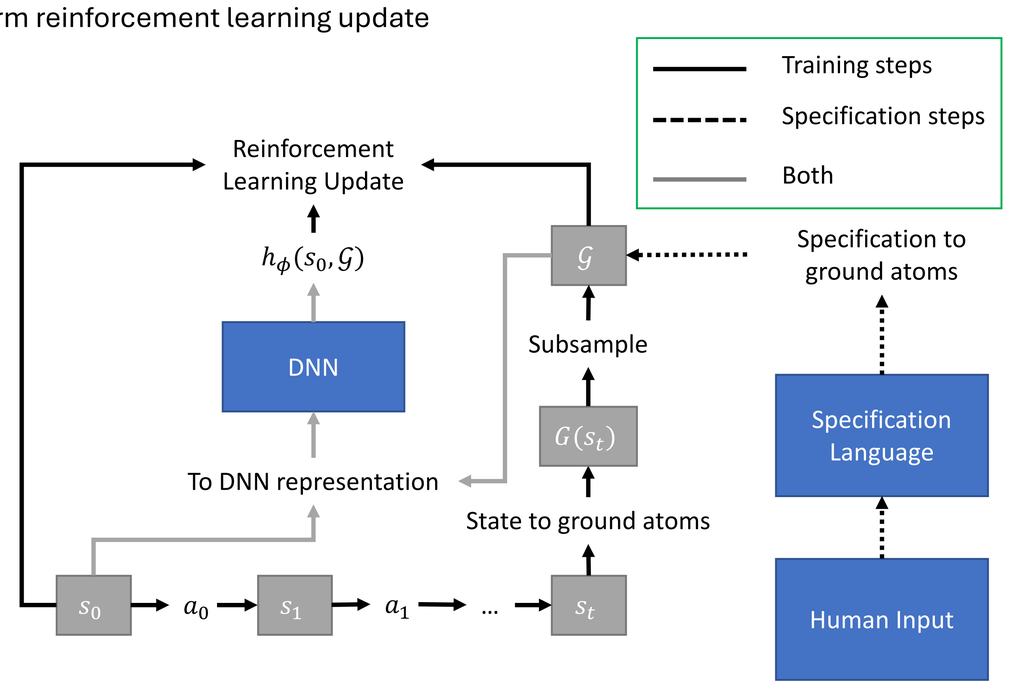
- Methods such as DeepCubeA train domain-specific heuristic functions in a largely domain-independent fashion [1]
- DeepCubeA assumes a pre-determined goal, variations assume a fully-specified goal state
- Therefore, specifying a set of goal states can be impractical
- We introduce DeepCubeA$_g$, which trains heuristic functions that generalize over goals represented as assignments
- We build on this with answer set programming to allow for more robust goal specification

States and Sets of States

- A state has a set of V variables $\{x_1, \dots, x_V\}$
  - Each variable, x_i, has domain, $D(x_i)$
- An assignment is a set $\{\dots, x_i = v_i, \dots\}$
  - $x_i = v_i$ represents x_i being assigned to value, $v_i \in D(x_i)$
- A state is a complete assignment
- A set of states is either a complete or partial assignment
- A state $s \in \mathcal{S}_A$ iff $A \subset s$
  - A is an assignment
  - $\mathcal{S}_A$ is the set of states represented by A

Training to Generalize Over Goals Represented as Assignments

- Generate start state
- Take a random walk between 0 and T steps
- Convert terminal state to its representation as an assignment
- Subsample assignment to obtain goal
- Perform reinforcement learning update



Performance when Reaching Goals Represented as Assignments

- Use heuristic function with batch A* search
- Compare to PDBs and fast downward planner with goal count, fast forward, and causal graph heuristics
- Compare for canonical goal state and randomly generated assignments as goals
  - Randomly generated assignment can be as small as the empty set
- 200 second time limit
- DeepCubeA$_g$ outperforms the fast downward planner and generalizes over goals without any re-training

Puzzle	Solver	Path Cost	% Solved	% Opt	Nodes	Secs	Nodes/Sec
RC (Canon)	PDBs$^+$	**20.67**	**100.00%**	**100.0%**	**2.05E+06**	**2.20**	**1.79E+06**
	DeepCubeA	21.50	**100.00%**	60.3%	6.62E+06	24.22	2.90E+05
	DeepCubeA$_g$	22.03	**100.00%**	35.00%	2.44E+06	41.99	5.67E+04
	FastDown (GC)	-	0.00%	0.0%	-	-	-
	FastDown (FF)	-	0.00%	0.0%	-	-	-
	FastDown (CG)	-	0.00%	0.0%	-	-	-
RC (Rand)	DeepCubeA$_g$	15.22	**99.40%**	-	1.91E+06	32.24	5.19E+04
	FastDown (GC)	7.18	32.80%	-	2.67E+06	13.79	1.41E+05
	FastDown (FF)	6.49	31.20%	-	4.87E+05	13.83	2.93E+04
	FastDown (CG)	7.85	33.80%	-	1.12E+06	11.62	5.81E+04
15-P (Canon)	PDBs	**52.02**	**100.00%**	**100.0%**	**3.22E+04**	**0.002**	**1.45E+07**
	DeepCubeA	52.03	**100.00%**	99.4%	3.85E+06	10.28	3.93E+05
	DeepCubeA$_g$	**52.02**	**100.00%**	**100.0%**	1.81E+05	2.61	6.94E+04
	FastDown (GC)	36.75	0.80%	0.80%	9.05E+07	102.11	8.66E+05
	FastDown (FF)	52.75	80.80%	24.80%	2.92E+06	42.11	6.93E+04
	FastDown (CG)	41.95	4.40%	1.20%	2.00E+07	80.58	2.47E+05
15-P (Rand)	DeepCubeA$_g$	**33.98**	**100.00%**	-	**1.11E+05**	**1.60**	**6.16E+04**
	FastDown (GC)	14.92	38.00%	-	1.61E+07	18.77	5.46E+05
	FastDown (FF)	32.66	89.20%	-	1.24E+06	17.39	5.65E+04
	FastDown (CG)	20.45	51.20%	-	3.90E+06	21.41	1.20E+05
24-P (Canon)	PDBs	**89.41**	**100.00%**	**100.00%**	8.19E+10	4239.54	**1.91E+07**
	DeepCubeA	89.49	**100.00%**	96.98%	6.44E+06	19.33	3.34E+05
	DeepCubeA$_g$	90.47	**100.00%**	55.24%	**3.38E+05**	**5.22**	6.48E+04
	FastDown (GC)	-	0.00%	0.00%	-	-	-
	FastDown (FF)	81.00	1.01%	0.40%	2.68E+06	89.84	2.91E+04
	FastDown (CG)	-	0.00%	0.00%	-	-	-
24-P (Rand)	DeepCubeA$_g$	66.28	**99.60%**	-	3.10E+05	4.91	6.16E+04
	FastDown (GC)	9.86	10.00%	-	9.54E+06	11.88	4.27E+05
	FastDown (FF)	26.35	26.00%	-	5.99E+05	19.57	2.41E+04
	FastDown (CG)	13.75	12.60%	-	1.42E+06	14.42	6.85E+04
Sokoban	DeepCubeA	32.88	**100.00%**	-	**5.01E+03**	2.71	1.84E+03
	DeepCubeA$_g$	**32.02**	**100.00%**	-	1.80E+04	0.95	1.79E+04
	FastDown (GC)	31.94	99.80%	-	3.17E+06	5.93	5.85E+05
	FastDown (FF)	33.15	**100.00%**	-	2.92E+04	**0.32**	**7.49E+04**
	FastDown (CG)	33.12	**100.00%**	-	4.43E+04	0.51	7.25E+04

Table 1: Comparison of DeepCubeA$_g$ with optimal solvers based on pattern databases (PDBs) that exploit domain-specific information and the domain-independent fast downward planning system with the goal count (GC), fast forward (FF), and causal graph (CG) heuristics. Comparisons are along the dimensions of solution length, percentage of instances solved, percentage of optimal solutions, number of nodes generated, time taken to solve the problem (in seconds), and number of nodes generated per second. For the Rubik's cube and sliding tile puzzles, experiments are done on canonical goal states (Canon) and randomly generated goals (Rand). For testing DeepCubeA on Sokoban, we report numbers obtained from the DeepCubeA GitHub repository.

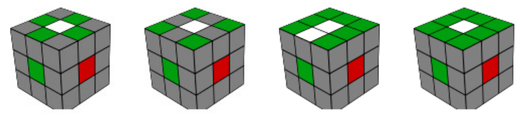
Specifying Goals with Answer Set Programming

- Answer set programming (ASP) is a form of first-order logic programming [2]
- An answer set program defines a set of stable models, where a stable model is a set of ground atoms
- We represent a goal (i.e. an assignment) as a set of ground atoms in first-order logic
- A state, s, is a goal state with respect to an answer set program, Π, iff s is a subset of some stable model of Π
- An answer set program consists of
  - Background knowledge
  - A set of rules with `goal` in the head, a headless rule so goal is true in all stable models
  - A choice rule with no body that contains ground atoms that represent all possible assignments of values to variables
- Goal specification often only requires a few lines of code

```
cross(F, CrossCol):- face(F), color(CrossCol), #count{Cbl: edge_cbl(Cbl), onface(Cbl, CrossCol, F)} = 4.

spot(F, BCol) :- color(BCol), face(F), face_col(F, FCol), dif_col(FCol, BCol), #count{Cbl: onface(Cbl,
BCol, F), edge_or_corner(Cbl)} = 8.

face_same(F) :- face_col(F, FCol), #count{Cbl : onface(Cbl, FCol, F)}=9.
canon :- #count{F : face_same(F)}=6.
```

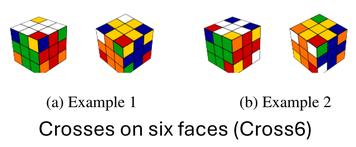


(a) Cross (b) X (c) Cup (d) Spot

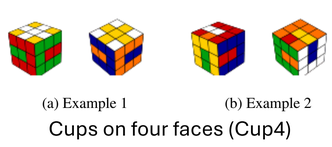
Performance when Reaching Goals Represented as Answer Set Programs

- Sample assignments from answer set program specification, Π
- Search for goal state, if not found, ban that assignment and sample another from Π
  - Use batch weighted A* search with a budget of 50 iterations
- Repeat until a goal state is found or until no more stable models are found
- Sokoban goals not always reachable

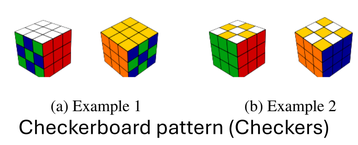
Goal	Path Cost	% Solved	# Models	Model Time	Search Time
Rubik's Cube (Canon)	24.41	100%	1	0.37	4.39
Rubik's Cube (Cross6)	13.11	100%	1	0.41	2.14
Rubik's Cube (Cup4)	24.33	100%	42.5	34.65	374.11
Rubik's Cube (CupSpot)	17.99	100%	27.68	38.66	241.08
Rubik's Cube (Checkers)	23.85	100%	1	0.49	4.2
Sokoban (Immov)	35.15	100%	6.37	6.83	16.16
Sokoban (BoxBox)	33.77	88%	1.89	0.58	6.08
Sokoban (AgentInBox)	34.42	77%	1.26	0.38	4.09



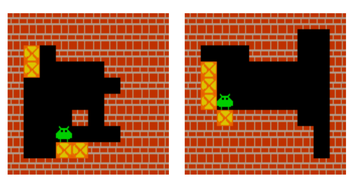
(a) Example 1 (b) Example 2
Crosses on six faces (Cross6)



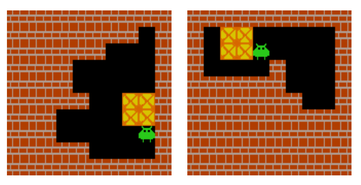
(a) Example 1 (b) Example 2
Cups on four faces (Cup4)



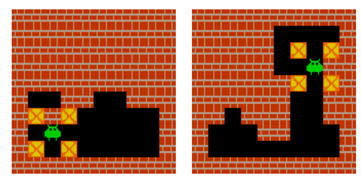
(a) Example 1 (b) Example 2
Checkerboard pattern (Checkers)



(a) Example 1 (b) Example 2
All boxes immovable (Immov)



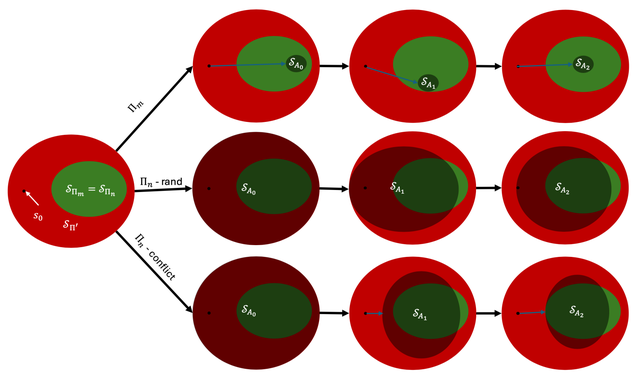
(a) Example 1 (b) Example 2
Box of boxes (BoxBox)



(a) Example 1 (b) Example 2
Agent in four boxes (AgentInBox)

Future Work

- Usage of negation as failure can result in reaching sampling an assignment, A, whose set of states, $\mathcal{S}_A$, contains states that are not goal states
- This paper proposed randomly searching for larger assignments that are stable models
- This does not account for why a state is not a goal state
- Future work will develop conflict-driven search to handle this [3]
- Preliminary results show that conflict-driven search with negation as failure results in finding shorter paths while also taking less time



Goal	Op	Cost	Solve	Itr	Node	Reach	¬Goal	Secs / Spec	Secs / Path	Secs
RC:Π^1_m	-	11.5	70	**3.3**	**33.4**	7.7	**0.0**	12.8	7.5	564.9
RC:Π^1_n	Rand	1.7	99	7.2	63.0	87.8	69.1	**0.1**	1.0	95.5
	Conf	**1.3**	**100**	5.4	36.3	**99.3**	52.4	**0.1**	**0.1**	**6.0**
24p:Π^1_m	-	24.6	**100**	9.2	92.4	**100.0**	**0.0**	**0.2**	0.2	42.5
24p:Π^1_n	Rand	3.2	**100**	4.3	33.6	**100.0**	38.7	**0.2**	**0.0**	**6.6**
	Conf	**2.5**	**100**	**4.1**	**31.6**	**100.0**	22.1	**0.2**	**0.0**	**6.6**
24p:Π^2_m	-	83.7	**100**	9.2	91.9	50.4	**0.0**	0.9	1.8	250.2
24p:Π^2_n	Rand	17.1	**100**	10.2	92.1	100.0	85.5	**0.1**	**0.1**	21.7
	Conf	**12.9**	**100**	**8.7**	**77.1**	**100.0**	79.7	**0.1**	**0.1**	**17.1**

Table 2: Comparison of monotonic and non-monotonic specifications with random and conflict-driven specialization operators for non-monotonic specifications. Comparisons are along the dimensions of average path cost, percentage solved, average number of iterations, average number of nodes generated, the average percentage of specified assignments reached with A* search, the average percentage of reached assignments that were not goal states, the average number of seconds it took to do a single specialization, the average number of seconds it took to find a single path (whether or not it was successful), and the average number of overall seconds it took to find a solution. RC:Π^1_m: All stickers on the white face are different than the center sticker. RC:Π^1_n: There does not exist a sticker on the white face that matches the center sticker. 24p:Π^1_m: The sum of row 0 is even. 24p:Π^1_n: It is not true the sum of row 0 is odd. 24p:Π^2_m: The sum of all rows is even. 24p:Π^2_n: There does not exist a row whose sum is odd.

References

[1] Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." *Nature Machine Intelligence* 1.8 (2019): 356-363.
[2] Brewka, Gerhard, Thomas Eiter, and Mirosław Truszczyński. "Answer set programming at a glance." *Communications of the ACM* 54.12 (2011): 92-103.
[3] Agostinelli, Forest "A conflict-driven approach for reaching goals specified with negation as failure." *ICAPS HAXP Workshop (2019)*

Code

Paper code:
https://github.com/forestagostinelli/SpecGoal

Built on DeepXube code base
pip install deepxube
https://github.com/forestagostinelli/deepxube