Neural Network Heuristic Functions for Classical Planning: Reinforcement Learning and Comparison to Other Methods

Patrick Ferber[1,3] Florian Geißer[2] Felipe Trevizan[2]
Malte Helmert[1] Jörg Hoffmann[3]

[1]University of Basel, Switzerland
[2]Australian National University, Australia
[3]Saarland University, Germany

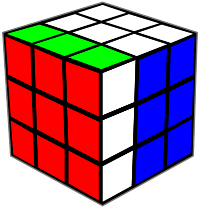
Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning
(2021)

Motivation

AlphaGo

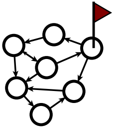
Silver et al. (2016)
Silver et al. (2017)
Silver et al. (2018)

Agostinelli et al. (2019)



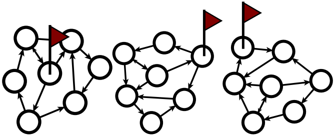
Neural Networks as Planning Heuristics

per-instance heuristics



- Ferber, Helmert, and Hoffmann (2020)
- Yu, Kuroiwa, and Fukunaga (2020)

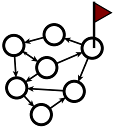
per-domain heuristics



- Shen, Trevizan, and Thiébaux (2020)
- Rivlin, Hazan, and Karpas (2020)
- Karia and Srivastava (2021)

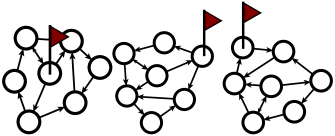
Neural Networks as Planning Heuristics

per-instance heuristics



- Ferber, Helmert, and Hoffmann (2020)
- Yu, Kuroiwa, and Fukunaga (2020)

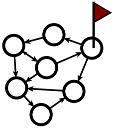
per-domain heuristics



- Shen, Trevizan, and Thiébaux (2020)
- Rivlin, Hazan, and Karpas (2020)
- Karia and Srivastava (2021)

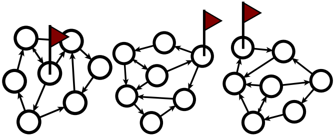
Neural Networks as Planning Heuristics

per-instance heuristics



- Ferber, Helmert, and Hoffmann (2020)
- Yu, Kuroiwa, and Fukunaga (2020)

per-domain heuristics



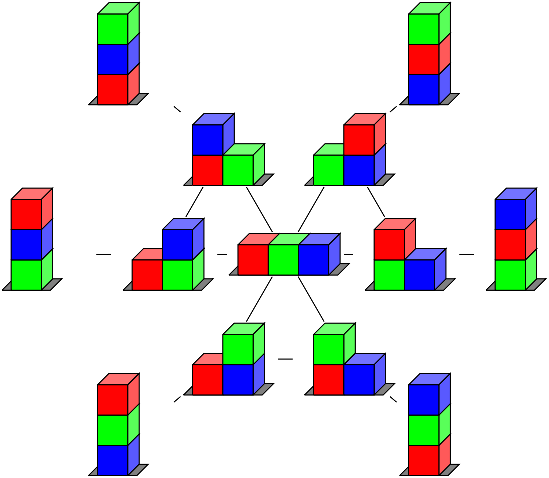
- Shen, Trevizan, and Thiébaux (2020)
- Rivlin, Hazan, and Karpas (2020)
- Karia and Srivastava (2021)

Contributions

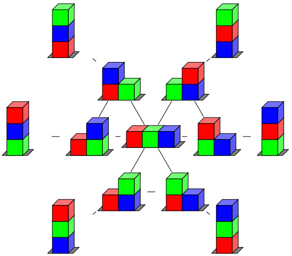
- three *per-instance* RL based heuristics
  - learned from scratch
  - only state as input
  - prove convergence to h^*

- comparison between state-of-the-art
  - neural network heuristics
  - model-based heuristics

Finite-Domain Representation (Helmert, 2009)



Finite-Domain Representation (Helmert, 2009)

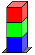


$\Pi = \langle V, O, I, g \rangle$

$V = \{ , , \}$

$dom( ) = \{$on , on , on $\}$

$O = \{$move from X to Y$\}$

$I =$ 

$g = \{ \mapsto$ on $\}$

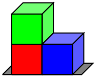
Progression & Regression

move from to

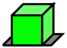
pre : $\{\ \mapsto$ on $\}$

eff : $\{\ \mapsto$ on $\}$

Progression



Regression



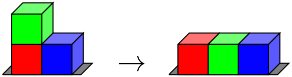
Progression & Regression

move from to

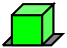
pre : { $\mapsto$ on }

eff : { $\mapsto$ on }

Progression



Regression



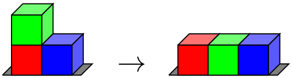
Progression & Regression

move from to

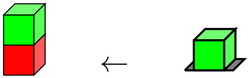
pre : { $\mapsto$ on }

eff : { $\mapsto$ on }

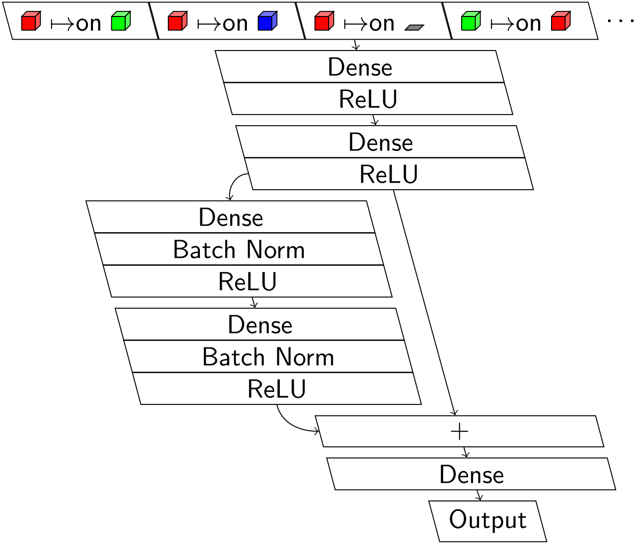
Progression



Regression



Residual Network (He et al., 2016)



Bootstrapping

```
def train(Π, NN, t_train):
    D = Buffer()

    while time() ≤ t_train:
        p = regression random walk(Π)
        s = complete to state (p)
        π = GBFS+NN(s)

        for s′ ∈ π:
            D.push(s′, distance(s′, goal(Π), π)

        NN = train(NN, D)
```

inspired by Bootstrap Learning of Arfaee, Zilles, and Holte (2011)

Bootstrapping

```
def train(Π, NN, t_train):
    D = Buffer()
    L = 5
    while time() ≤ t_train:
        p = regression random walk(Π, max_length=L)
        s = complete to state (p)
        π = GBFS+NN(s)

        for s′ ∈ π:
            D.push(s′, distance(s′, goal(Π), π)
        if frequently solves s:  L = 2 ∗ L
        NN = train(NN, D)
```

inspired by Bootstrap Learning of Arfaee, Zilles, and Holte (2011)

Bootstrapping

```
def train(Π, NN, t_train, t_search):
    D = Buffer()
    L = 5
    while time() ≤ t_train:
        p = regression random walk(Π, max_length=L)
        s = complete to state (p)
        π = GBFS+NN(s, timeout=t_search)
        if not π:  continue
        for s′ ∈ π:
            D.push(s′, distance(s′, goal(Π), π)
        if frequently solves s:  L = 2 ∗ L
        NN = train(NN, D)
```

inspired by Bootstrap Learning of Arfaee, Zilles, and Holte (2011)

Bootstrapping to Predict Expansions

```
def train(Π, NN, t_train, t_search):
    D = Buffer()
    L = 5
    while time() ≤ t_train:
        p = regression random walk(Π, max_length=L)
        s = complete to state (p)
        expansions = GBFS+NN(s,timeout=t_search)
        if not π: continue
        D.push(s, expansions)

        if frequently solves s: L = 2 * L
        NN = train(NN, D)
```

inspired by Bootstrap Learning of Arfaee, Zilles, and Holte (2011)

Approximate Value Iteration

```
def train(Π, NN, t_train):
    D = Buffer()
    while time() ≤ t_train:
        p = regression random walk(Π)
        s = complete to state (p)
        h = BellmanUpdate (s, NN)
        D.push(s, h)
        NN = train(NN, D)

def BellmanUpdate(s, NN):
    return 1 + min_{s' ∈ succ(s)} NN(s')
```

Algorithms

- h^{Boot} Bootstrapping
- h^{BExp} Bootstrapping with expansions
- h^{AVI} Approximate value iteration

- h^{SL} Ferber, Helmert, and Hoffmann (2020)
- h^{HGN} Shen, Trevizan, and Thiébaux (2020)

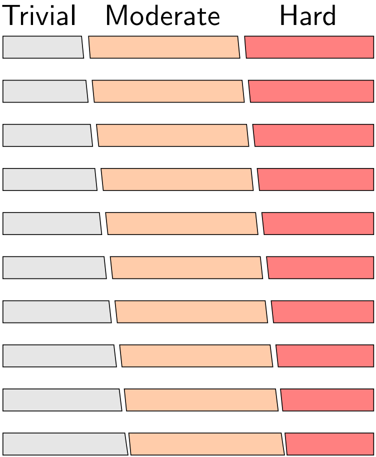
- h^{FF} Hoffmann and Nebel (2001)
- *LAMA* Richter and Westphal (2010)

Benchmarks (Ferber, Helmert, and Hoffmann, 2020)

- Blocksworld
- Depots
- Grid
- NPuzzle
- Pipesworld-NT
- Rovers
- Scanalyzer
- Storage
- Transport
- Visitall

Benchmarks (Ferber, Helmert, and Hoffmann, 2020)

- Blocksworld
- Depots
- Grid
- NPuzzle
- Pipesworld-NT
- Rovers
- Scanalyzer
- Storage
- Transport
- Visitall



Validation (Moderate Tasks)

Domain	h^{Boot}	+V	h^{BExp}	+V	h^{AVI}	+V
blocks	0.0	+18.0	0.0	+0.0	0.0	+0.0
depots	31.7	+28.6	17.1	+15.0	**43.7**	+11.0
grid	**100.0**	+0.0	**100.0**	+0.0	51.0	+0.0
npuzzle	**27.0**	+1.0	0.0	+0.0	1.0	+0.0
pipes-nt	36.2	+21.6	**51.2**	+17.2	21.4	+28.8
rovers	**36.5**	+11.7	15.2	+6.6	34.2	+10.8
scanalyzer	33.3	+0.0	59.7	+11.0	**66.7**	+0.6
storage	**89.0**	+0.0	61.0	-3.5	67.0	+2.5
transport	**83.8**	+16.2	79.5	+20.5	70.0	+17.5
visitall	**17.0**	+38.3	0.0	+0.0	0.0	+0.0

Table: Performance of our algorithms without validation and performance change due to validation (+V).

Coverage (Moderate Tasks)

Domain	h^{Boot}	h^{BExp}	h^{AVI}
blocks	**18.0**	0.0	0.0
depots	**60.3**	32.7	54.7
grid	**100.0**	**100.0**	51.0
npuzzle	**28.0**	0.0	1.0
pipes-nt	57.8	**68.4**	50.2
rovers	**48.2**	21.8	45.0
scanalyzer	33.3	**70.7**	67.3
storage	**89.0**	57.5	69.5
transport	**100.0**	**100.0**	87.5
visitall	**55.3**	0.0	0.0

Coverage (Moderate Tasks)

Domain	h^{Boot}	h^{BExp}	h^{AVI}	h^{SL}	h^{HGN}
blocks	18.0	0.0	0.0	80.4	**100.0**
depots	60.3	32.7	54.7	**90.3**	0.0
grid	**100.0**	**100.0**	51.0	93.0	0.0
npuzzle	**28.0**	0.0	1.0	0.0	0.3
pipes-nt	57.8	68.4	50.2	**92.2**	7.6
rovers	**48.2**	21.8	45.0	26.0	14.0
scanalyzer	33.3	70.7	67.3	**82.7**	11.0
storage	**89.0**	57.5	69.5	24.5	0.0
transport	**100.0**	**100.0**	87.5	99.2	94.7
visitall	55.3	0.0	0.0	0.0	**100.0**

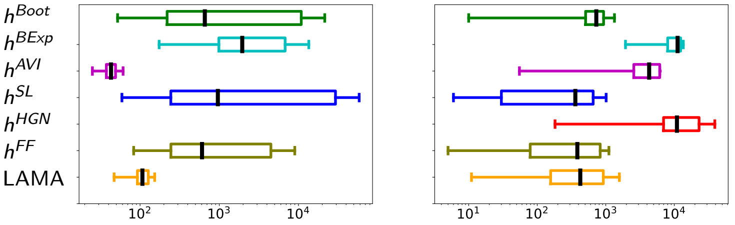
Coverage (Moderate Tasks)

Domain	h^{Boot}	h^{BExp}	h^{AVI}	h^{SL}	h^{HGN}	h^{FF}	*LAMA*
blocks	18.0	0.0	0.0	80.4	**100.0**	98.8	**100.0**
depots	60.3	32.7	54.7	**90.3**	0.0	98.0	**100.0**
grid	**100.0**	**100.0**	51.0	93.0	0.0	96.0	**100.0**
npuzzle	**28.0**	0.0	1.0	0.0	0.3	97.5	**100.0**
pipes-nt	57.8	68.4	50.2	**92.2**	7.6	82.4	**99.4**
rovers	**48.2**	21.8	45.0	26.0	14.0	84.2	**100.0**
scanalyzer	33.3	70.7	67.3	**82.7**	11.0	98.3	**100.0**
storage	**89.0**	57.5	69.5	24.5	0.0	**48.0**	**38.5**
transport	**100.0**	**100.0**	87.5	99.2	94.7	98.5	**100.0**
visitall	55.3	0.0	0.0	0.0	**100.0**	93.3	**100.0**

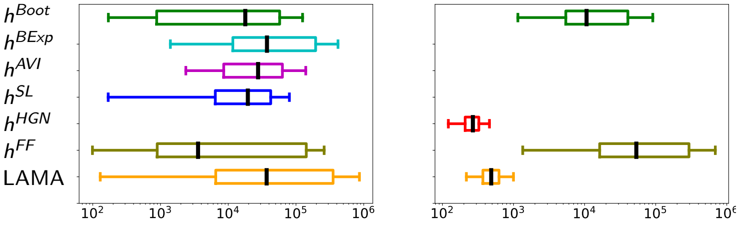
Coverage (Hard Tasks)

Domain	h^{Boot}	h^{BExp}	h^{AVI}	h^{SL}	h^{HGN}	h^{FF}	$LAMA$
blocks	0.0	0.0	0.0	0.0	**50.0**	61.6	**96.8**
depots	8.3	4.3	12.9	**35.4**	0.0	36.0	**82.6**
grid	87.8	**95.0**	70.5	60.2	0.0	53.2	**100.0**
npuzzle	0.0	0.0	0.0	0.0	0.0	33.2	**86.5**
pipes-nt	23.4	19.1	8.0	**48.7**	0.0	27.4	**69.3**
rovers	2.8	0.8	**6.5**	1.5	0.3	13.9	**100.0**
scanalyzer	3.3	0.0	**60.7**	60.0	0.0	98.0	**100.0**
storage	**27.2**	13.2	15.8	0.0	0.0	**13.8**	**11.5**
transport	0.0	0.0	**2.4**	0.0	0.0	0.0	**92.8**
visitall	28.0	0.0	0.0	0.0	**100.0**	74.0	**100.0**

Expansions



(a) Grid (b) Scanalyzer



(c) Storage (d) Visitall

Conclusion

- three new per-instance RL heuristics

- large scale comparison to previous work
  - trained heuristics highly complementary
  - in general, model-based heuristics win
  - all our RL heuristics superior in Storage

Paper &
Supplement

References I

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. Nature Machine Intelligence 1: 356–363. doi:10.1038/s42256-019-0070-z.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning Heuristic Functions for Large State Spaces. Artificial Intelligence 175: 2075–2098.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Neural Network Heuristics for Classical Planning: A Study of Hyperparameter Space. In Giacomo, G. D., ed., Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020), 2346–2353. IOS Press.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. ISSN 1063-6919. doi:10.1109/CVPR.2016.90.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. Artificial Intelligence 173: 503–535.

References II

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research 14: 253–302.

Karia, R.; and Srivastava, S. 2021. Learning Generalized Relational Heuristic Networks for Model-Agnostic Planning. In Leyton-Brown, K.; and Mausam, eds., Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021). AAAI Press.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. Journal of Artificial Intelligence Research 39: 127–177.

Rivlin, O.; Hazan, T.; and Karpas, E. 2020. Generalized Planning With Deep Reinforcement Learning. In ICAPS 2020 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL), 16–24.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning Domain-Independent Planning Heuristics with Hypergraph Networks. In Beck, J. C.; Karpas, E.; and Sohrabi, S., eds., Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020), 574–584. AAAI Press.

References III

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature 529(7587): 484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362(6419): 1140–1144.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the Game of Go Without Human Knowledge. Nature 550(7676): 354–359.

References IV

Yu, L.; Kuroiwa, R.; and Fukunaga, A. 2020. Learning Search-Space Specific Heuristics Using Neural Network. In ICAPS 2020 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP), 1–8.