Learning Neural Priority Functions for Search-based Planning using Sufficient Conditions for Bounded Suboptimality without Re-openings

Anonymous submission

Abstract

Best-first search algorithms use a *priority function* to order a queue of nodes for a systematic exploration of the search space. The priority is typically a function of the cost-to-come from the start (g) and a lower-bound estimate of the cost-to-go to the goal (h), and takes the form of a linear combination. However, this linear form may not capture the true relationship between g and h for a given search problem, and designing an analytic function for this relationship is often impractical. While it is hard to provide guarantees with neural networks, they offer the flexibility to learn priority functions from data to improve search efficiency. Recently, it was shown that a priority function only needs to satisfy a set of sufficient conditions to certify bounded-suboptimal search without re-openings. In this paper, we explore the use of neural networks as priority functions, applying regularization using these sufficient conditions during the learning process. We integrate the learned neural priority function into BFS and Focal-search and evaluate its performance across multiple domains, including 2D and 3D navigation and the sliding tile puzzle.

Introduction

The priority function is central to best-first search (BFS) algorithms, allowing them to focus on the most promising paths and find solutions faster than exhaustive approaches. This ability to prioritize promising paths is essential for planning in large search spaces or time-sensitive problems. Priority functions can leverage domain knowledge to inform decision-making, but fully capturing this knowledge can be challenging. In such cases, neural networks (NN) can learn from data relevant contextual information to guide the priority function, further enhancing the effectiveness of BFS solutions. Recently, several works showed the potential of using NN to reduce the number of expanded nodes during the search (Bhardwaj, Choudhury, and Scherer 2017; Choudhury et al. 2018; Agostinelli et al. 2019; Yonetani et al. 2021; Takahashi et al. 2021). However, their performance in long-horizon and complex planning scenarios and their ability to maintain guarantees (i.e., bounded suboptimality) while operating efficiently remains unclear. Takahashi et al. (2021) introduced a novel loss function for heuristic learning. However, it may not consistently yield high-quality heuristics or address the challenges of finding bounded suboptimal solutions and avoiding the overhead of reopening nodes (where a node can be expanded again if a better path to it is found).

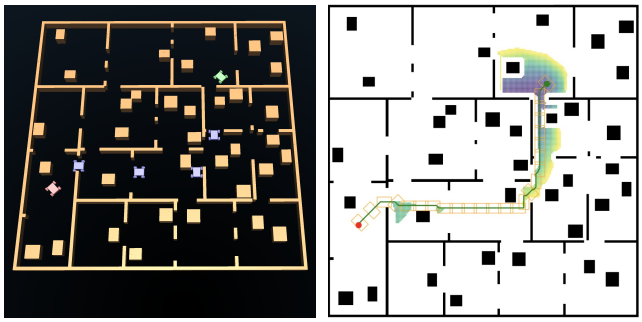


Figure 1: A 3D navigation problem: finding a minimum-cost path for a holonomic ground robot while maintaining computational efficiency using neural priority function.

Bounded suboptimal search (BSS) algorithms are crucial for practical problems that are too complex to solve optimally, offering a trade-off between solution quality and planning costs (e.g., node expansions). BSS algorithms use a queue data structure to store nodes ordered by a priority function. Weighted A* (wA*) (Pohl 1970), a BFS variant, is a widely used BSS algorithm for finding solutions within a predefined suboptimality bound using just a single queue to manage node exploration without requiring queue re-sorting (re-arranging the nodes in the queue based on new priorities) or node re-opening (Likhachev, Gordon, and Thrun 2003). More advanced algorithms (Pearl and Kim 1982; Thayer and Ruml 2008, 2011; Cohen et al. 2018; Aine et al. 2016) that outperform wA* in certain domains split their operation into two tasks: finding a solution and proving its bounded suboptimality. These algorithms typically employ multiple queues, with one queue – often called the *focal list* – dedicated explicitly to enforcing the suboptimality bound. However, unlike wA*, these approaches often require queue resorting and node reopening to guarantee bounded suboptimality, operations that impact both search performance (Sepetnitsky, Felner, and Stern 2016) and solution quality (Valenzano, Sturtevant, and Schaeffer 2014).

In this paper, we propose a method for learning contextualized neural priority functions in the general form of $f(n) = \Phi(h(n, n_g), g(n), \text{context})$, where n is any search state and n_g is the goal. Unlike most prior approaches that

only learn the heuristic function h, we directly learn the priority f as a function of the current g-value and a given consistent heuristic estimate. This approach extends beyond traditional linear priority functions by learning a (contextualized) neural representation of the priority. To do this in a principled way, we regularize the learning of the neural priority using a set of recently developed sufficient conditions for a priority function to avoid costly node re-openings (Chen and Sturtevant 2019). By formulating these sufficient conditions as soft constraints in the NN optimization process, our approach aims to learn priority functions that balance the current search context with the desired priority function characteristics. We demonstrate the use of our neural priority function in both BFS and focal search algorithms, showing that it generalizes to new problem instances while improving efficiency and maintaining solution quality.

Preliminaries

wA* uses a queue ordered by the priority function $f(n) = g(n) + \epsilon \cdot h(n, n_g)$, where $\epsilon \in \mathbb{R}^{>1}$ is the allowed suboptimality factor. Since the priority function is a linear combination of g and h, the level curves (points (h, g) with equal priority), or isolines, are straight lines in the h-g plane. Due to the linear nature of these isolines, wA* maintains constant suboptimality along each portion of the path being explored. However, varying suboptimality along different portions of the path may lead to improved search performance. For example, in a 3D navigation problem (Fig.1), we might want higher suboptimality bounds in obstacle-free regions to search more greedily, while maintaining lower suboptimality in cluttered areas for a more systematic search.

Focal List

A lot of work has been done on developing BSS algorithms that improve upon wA*. Many of these algorithms introduce an additional queue, called a focal list, which decouples two tasks: finding a solution fast and guaranteeing its suboptimality bound. We explain how Focal search works as many bounded suboptimal search algorithms that use focal lists rely on the same core concept.

Focal search maintains two queues: OPEN, ordered by an admissible priority function f_1, and FOCAL, defined as: FOCAL $= \{n \in \text{OPEN} \mid f_1(n) \leq \epsilon \cdot \min_{n' \in \text{OPEN}} f_1(n')\}$ with a sub-optimality bound $\epsilon \geq 1$. The next node expanded is chosen from FOCAL according to some priority function f_2. This approach is powerful as it allows the use of any problem-specific priority function while maintaining suboptimality bounds, often dramatically improving performance. In the context of isolines, introducing the second priority function allows for different suboptimalities, shaped like curves, within the linear isoline bound. However, to guarantee bounded suboptimality, queue re-sorting can happen often, and node re-openings must be allowed (Cohen et al. 2018), unlike wA*, which is guaranteed to return a solution within the suboptimality bound without re-opening nodes[1].

[1] wA* may find a better solution when reopenings are permitted.

Sufficient Conditions for a Priority Function in BSS to Avoid Re-openings

While problem-specific priority functions can be beneficial, they risk requiring node re-openings since f_2 is unconstrained. Recent work has shown that any priority function in the form of $f(n) = \Phi(h(n, n_g), g(n))$ can be used in BFS to guarantee bounded suboptimality without re-openings so long as they meet certain properties (Chen and Sturtevant 2019). This enables the design of problem-specific priorities that allow varying suboptimality along different portions of the path.

Method

In this work, we propose learning a *neural priority function* $f(n) = \Phi_\theta(h(n, n_g), g(n), \phi(m))$ while incorporating the sufficient conditions as constraints in the learning process. Here θ denotes the parameters of the NN model and $\phi(m)$ is the encoding function that provides context based on the map m. Our method can be formalized as the following optimization

$$\text{find} \quad \Phi_\theta \tag{1a}$$

$$\text{s.t.} \quad \arg\min_{n \in \text{OPEN}} \Phi_\theta = n^* \tag{1b}$$

$$\Phi_\theta(0, \epsilon \cdot t, \phi(m)) = t \tag{1c}$$

$$\Phi_\theta(t, 0, \phi(m)) = t \tag{1d}$$

$$-\frac{\partial \Phi_\theta}{\partial h} \leq 0; -\frac{\partial \Phi_\theta}{\partial g} \leq 0 \tag{1e}$$

$$\frac{\partial \Phi_\theta}{\partial g} - \frac{\partial \Phi_\theta}{\partial h} \leq 0 \tag{1f}$$

$$\frac{\partial \Phi_\theta}{\partial h} + \frac{\partial \Phi_\theta}{\partial g} \leq 2 \tag{1g}$$

where n^* is a node on the optimal path from n_s to n_g and Eq. 1c-1g are the sufficient conditions for a priority function to certify bounded suboptimality in a BFS without reopening states (Chen and Sturtevant 2019). Intuitively, Eq. 1c-1d states that a node with $g = \epsilon \cdot t$ and $h = 0$ should have equivalent priority as a node with $g = 0$ and $h = t$, since both lead to an ϵ-suboptimal path, Eq. 1e constrains the function to be monotonic with respect to g and h, Eq. 1f requires that Φ_θ grows faster with increases in h than with equivalent increases in g and Eq. 1g bounds the rate of change of the isolines. To learn such a function, we relax the problem by defining the set of conditions 1c - 1g as soft constraints, which we then incorporate into the loss function.

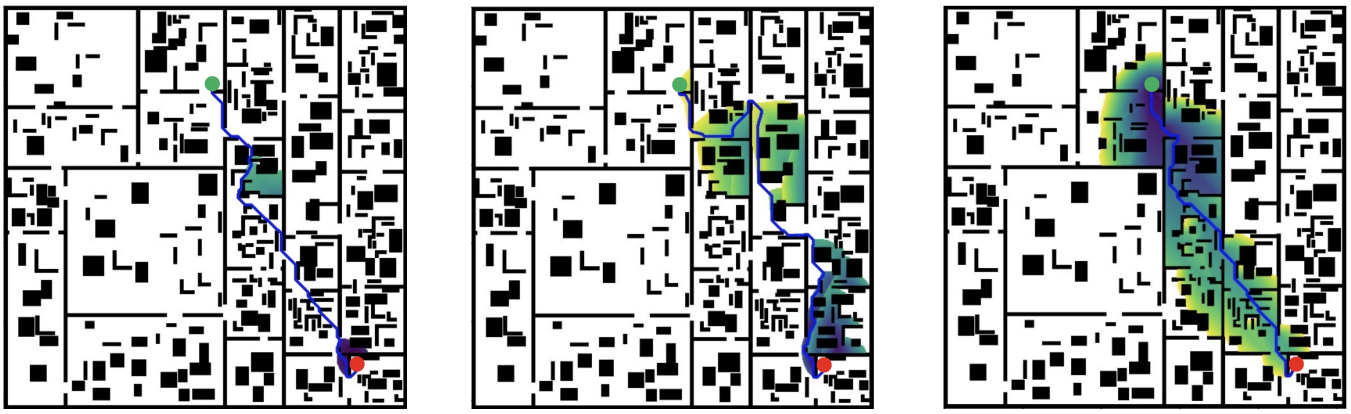
Loss Function

To train our neural priority function Φ_θ to satisfy the sufficient conditions while accurately predicting optimal path costs, we employ a composite loss function. The total loss $\mathcal{L}$ is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \lambda_1 \mathcal{L}_1 + \lambda_2 \mathcal{L}_2 + \lambda_3 \mathcal{L}_3$$

The MSE loss is computed between the network's output and the target optimal path cost:

$$\mathcal{L}_{\text{MSE}} = \mathbb{E}_{(x, f^*(n)) \in \mathcal{D}} \left[||\Phi_\theta(x) - f^*(n)||^2 \right]$$



(a) Neural priority function learning with our loss function (CST).

(b) Neural priority function learning with only MSE.

(c) A* baseline, using a priority of $g(n) + h(n)$.

Figure 2: Comparison of our proposed loss function for training a neural priority function versus training with only MSE and the A* baseline. The green dot represents the start node, the red dot indicates the goal node, and the blue line shows the found path. The colored regions depict the expanded nodes during the search, with darker colors indicating lower f-values. As shown, our proposed learning method results in fewer expansions while maintaining solution quality.

where $x = (n, n_g, g(n), h(n, n_g), \phi(m))$ and $f^*(n)$ is the target priority value. To enforce the gradient constraints from equations 1e-1g, we compute the gradients of the output with respect to the cost-to-come and heuristic inputs:

$$\nabla_g \Phi_\theta = \frac{\partial \Phi_\theta}{\partial g}, \quad \nabla_h \Phi_\theta = \frac{\partial \Phi_\theta}{\partial h}$$

These gradients are used to formulate the penalty terms:

$$\mathcal{L}_1 = \mathbb{E}\left[\text{ReLU}(-\nabla_g \Phi_\theta) + \text{ReLU}(-\nabla_h \Phi_\theta)\right] \quad \text{(From 1e)}$$
$$\mathcal{L}_2 = \mathbb{E}\left[\text{ReLU}(\nabla_g \Phi_\theta - \nabla_h \Phi_\theta)\right] \quad \text{(From 1f)}$$
$$\mathcal{L}_3 = \mathbb{E}\left[\text{ReLU}(\nabla_g \Phi_\theta + \nabla_h \Phi_\theta - 2)\right] \quad \text{(From 1g)}$$

The weights λ_i follow a linear schedule:

$$\lambda_i(t) = \lambda_i^{\text{start}} + (\lambda_i^{\text{end}} - \lambda_i^{\text{start}})(t/T)^\alpha$$

where t is the current epoch, T is the total number of epochs, and α is the growth rate. We choose the range of λ_i values such that $\lambda_i \mathcal{L}_i \ll \mathcal{L}_{\text{MSE}}$ throughout training to ensure the gradient penalties act as soft constraints without dominating the primary learning objective. Then, as training progresses and λ_i values increase, we increasingly enforce the gradient constraints. This approach helps avoid local optima that might satisfy the constraints but poorly approximate the target values.

The conditions from equations 1c and 1d are implicitly satisfied via $\mathcal{L}_{\text{MSE}}$, since all the data points along a path have the same f^* values.

Data Collection

Let $\mathcal{M} = \{m_1, \ldots, m_k\}$ be a set of k generated maps. Then the encoding function $\phi : \mathcal{M} \to \mathbb{R}^d$ is a domain-specific autoencoder, where d is the dimension of the latent space. For each map m_i, we generate a set of N start n_{start} goal n_{goal} pair nodes queries $Q_i = \{q_{i,1}, ..., q_{i,N}\}$. For each query $q_{i,j}$, we run forward and backward optimal search between n_{start} and n_{goal} and cache g values for all expanded states. Then, the target function is the optimal path cost from the start node to the goal node through a specific node n: $f(n) = g_{\text{forward}}(n) + g_{\text{backward}}(n)$.

The resulting dataset $\mathcal{D}$ consists of indexed tuples grouped by map $\mathcal{D} = \{(m_i, X_i)\}_{i=1}^k$, where X_i represents all the j samples from map m_i. Each sample in X_i is a tuple containing a node, the goal node, the cost-to-come, heuristic value, the map encoding, and the optimal cost from start to goal through the node $(x_{i,j}, f^*(n_{i,j}))$.

Experiments

To evaluate the proposed neural priority function, we created a set of experiments in three different planning domains: two-dimensional grid navigation with a point robot (Fig. 2), three-dimensional grid navigation with a holonomic robot (incorporating orientation) with a rectangular footprint (Fig. 1), and the sliding tile puzzle.

We benchmark our learned neural priority function against wA*, which inflates the heuristic by the factor of allowed suboptimality and uses a priority key of $g(n)+\epsilon \cdot h(n)$; Dynamic Potential Search (DPS), that uses an adaptive priority of $\frac{B \cdot f_{min} - g(n)}{h(n)}$ where B is the required suboptimality factor and f_{min} is the minimum f-value in OPEN; Multi-Heuristic A* (MHA*), which employs multiple heuristic queues while ensuring bounded suboptimality through an admissible anchor heuristic; and Optimistic Search (OS), which uses two queues to first find a solution quickly with a greedy approach, and then attempts to prove its bounded suboptimality.

The neural priority function is implemented as a fully-connected neural network. For spatial domains (2D/3D navigation), we incorporate environmental context through latent encodings generated by a U-Net architecture (Ronneberger, Fischer, and Brox 2015). For the sliding tile puzzle, we em-

Table 1: Performance comparison of search algorithms on the 2D (XY) and 3D (x,y,orientation) (XYT) domains. **SC**: Solution cost relative to A*, **ER**: State expansion ratio compared to A*, **RR**: State re-opening rate relative to total state expansions in percentages [%]. **-R** algorithms indicate that re-openings were allowed. **-F** algorithms indicate Focal Search. **-N** indicate neural priority functions; **MSE** indicates the priority function was trained using exclusively MSE loss, while **CST** indicates that soft constraints were applied in conjunction with MSE loss.

Metric	A*	WA*-R	WA*	DPS-R	DPS	N MSE-R	N MSE	N MSE-F-R	N MSE-F	N CST-R	N CST	N CST-F-R	N CST-F
SC-XY	1.00	1.06	1.14	1.04	1.09	1.04	1.07	1.04	1.07	1.00	1.03	1.00	1.03
ER-XY	1.00	0.16	0.07	0.22	0.20	0.19	0.12	0.19	0.12	0.08	0.07	0.08	0.07
RR-XY	—	62.8	—	47.6	—	68.8	—	68.7	—	15.5	—	15.5	—
SC-XYT	1.00	1.30	1.35	1.26	1.31	1.01	1.01	1.01	1.01	1.00	1.01	1.00	1.01
ER-XYT	1.00	0.92	0.58	1.51	0.84	0.97	0.71	0.99	0.73	0.63	0.60	0.63	0.61
RR-XYT	—	34.2	—	66.9	—	26.3	—	18.6	—	3.70	—	1.20	—

Table 2: Performance comparison across algorithms on the 24-puzzle and 48-puzzle instances. **SC**: Solution cost relative to A*, **ER**: State expansion ratio compared to A*, **RR**: State re-opening rate relative to total state expansions in percentages [%], **SR**: Success rate in achieving lower expansions relative to A* in percentages [%]. **-R**: algorithms indicate that re-openings were allowed. **-F** algorithms indicate Focal Search; MSE/CST: Neural Network variants with MSE/Constraint loss.

Metric	A*	WA*-R	WA*	DPS-R	DPS	OS	IMHA*-R	IMHA*	SMHA*	MSE-R	MSE	MSE-F-R	MSE-F	CST-R	CST	CST-F-R	CST-F
SC-24	1.00	1.58	1.60	1.52	1.53	1.16	1.10	1.13	1.17	1.01	1.02	1.01	1.02	1.00	1.00	1.00	1.00
ER-24	1.00	0.36	0.32	0.29	0.28	0.54	0.42	0.34	0.32	0.37	0.37	0.38	0.37	0.36	0.36	0.36	0.36
RR-24	—	5.12	—	6.27	—	—	2.28	—	—	0.14	—	0.07	—	0.05	—	0.00	—
SR-24	100	76	80	72	85	82	90	90	93	97	97	97	97	99	99	99	99
SC-48	1.00	1.28	1.34	1.42	1.44	1.14	1.21	1.25	1.20	1.01	1.01	1.01	1.01	1.00	1.00	1.00	1.00
ER-48	1.00	0.39	0.32	0.40	0.37	0.28	0.29	0.27	0.31	0.32	0.33	0.32	0.33	0.32	0.32	0.32	0.32
RR-48	—	8.65	—	26.0	—	—	3.39	—	—	0.07	—	0.05	—	0.02	—	0.01	—
SR-48	100	84	88	80	90	88	96	96	97	100	100	100	100	100	100	100	100

ploy a direct representation, as the domain structure does not benefit from spatial encoding.

We also evaluate the impact of introducing the sufficient conditions for BSS without re-openings as soft constraints into the learning process over exclusively MSE loss.

We set a consistent suboptimality bound of $\epsilon = 3.0$ across all BSS algorithms. For MHA*, we distribute the suboptimality evenly by setting its bounds to $\epsilon_1 = \epsilon_2 = \sqrt{3}$, maintaining the same total bound of 3.0. We evaluate our learned neural priority function in two configurations: a single-queue BFS, and integrated into Focal Search. For both configurations, we permit node re-openings, a policy choice whose impact on solution quality we analyze separately. For each domain, we evaluate the algorithms' performances on 100 distinct problems and report the means in Table 1 and Table 2.

Heuristics. For the 2d and 3d navigation domains, we use Euclidean distance as our admissible and consistent heuristic for WA*, DPS, and OS. For the sliding puzzles, we employed the Manhattan Distance (MD) combined with Linear Conflicts (LC) as the primary admissible and consistent heuristic function, defined as $h_0 = \text{MD} + \text{LC}$. To enhance the performance of MHA* algorithms, we generated two additional heuristics by incorporating the number of Misplaced Tiles (MT) with random weights, ensuring a suboptimality bound of 3. Specifically, each additional heuristic h_i was computed as $h_i = r_1 \times \text{MD} + r_2 \times \text{LC} + r_3 \times \text{MT}$, where r_1, r_2, r_3 are randomly selected weights between 1.0 and 3.0.

We observe that the learned priority function with soft constraints produces superior solution quality to WA*, DPS, MHA*, and OS while performing comparably on state expansions, though it may achieve slightly inferior state expansion rates in certain scenarios. The neural priority function also produces the highest success rates of the tested algorithms in returning solutions while requiring less state expansions than A*, as can be seen in Fig. 2.

Allowing re-openings can allow solution quality refinement at the cost of state expansions. With the learned priority functions, near-optimal solutions mean little possible benefit in solution quality from allowing re-openings, though the neural priority function with soft constraints dominates performance on re-openings relative to learning node priorities directly with MSE loss without additional constraint, particularly on the 2D and 3D domains.

Conclusion

In this work, we proposed an approach for learning neural priority functions that improves the efficiency of search-based planning algorithms. Unlike common approaches that focus on learning heuristic functions, our method learns a priority as a function of the g-value, a consistent heuristic value, and problem-specific context. By incorporating sufficient conditions for bounded suboptimal search without re-openings as regularizers during training, we demonstrated across three domains that our approach maintains low solution costs while improving computational efficiency. Our results also show that when allowing node re-openings, our regularization leads to fewer re-openings while maintaining nearly identical solution quality. Future work could address the computational overhead of neural network inference through simplified architectures and selective inference strategies. Despite this limitation, our work establishes a promising direction for incorporating machine learning into search algorithms while considering their theoretical guarantees and desired properties.

References

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(8): 356–363.

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2016. Multi-heuristic a. *The International Journal of Robotics Research*, 35(1-3): 224–243.

Bhardwaj, M.; Choudhury, S.; and Scherer, S. A. 2017. Learning Heuristic Search via Imitation. *CoRR*, abs/1707.03034.

Chen, J.; and Sturtevant, N. R. 2019. Conditions for avoiding node re-expansions in bounded suboptimal search. *puzzle*, 40: 39–753.

Choudhury, S.; Bhardwaj, M.; Arora, S.; Kapoor, A.; Ranade, G.; Scherer, S.; and Dey, D. 2018. Data-driven planning via imitation learning. *The International Journal of Robotics Research*, 37(13-14): 1632–1672.

Cohen, L.; Greco, M.; Ma, H.; Hernández, C.; Felner, A.; Kumar, T. S.; and Koenig, S. 2018. Anytime Focal Search with Applications. In *IJCAI*, 1434–1441.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In Thrun, S.; Saul, L.; and Schölkopf, B., eds., *Advances in Neural Information Processing Systems*, volume 16. MIT Press.

Pearl, J.; and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*, (4): 392–399.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3): 193–204.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597.

Sepetnitsky, V.; Felner, A.; and Stern, R. 2016. Repair policies for not reopening nodes in different search settings. In *Proceedings of the International Symposium on Combinatorial Search*, volume 7, 81–88.

Takahashi, T.; Sun, H.; Tian, D.; and Wang, Y. 2021. Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29(1): 764–772.

Thayer, J. T.; and Ruml, W. 2008. Faster than Weighted A*: An Optimistic Approach to Bounded Suboptimal Search. In *ICAPS*, 355–362.

Thayer, J. T.; and Ruml, W. 2011. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, volume 2011, 674–679.

Valenzano, R.; Sturtevant, N.; and Schaeffer, J. 2014. Worst-Case Solution Quality Analysis When Not Re-Expanding Nodes in Best-First Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1).

Yonetani, R.; Taniai, T.; Barekatain, M.; Nishimura, M.; and Kanezaki, A. 2021. Path Planning using Neural A* Search. In Meila, M.; and Zhang, T., eds., *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 12029–12039. PMLR.