Bounded-Suboptimal Search with Learned Heuristics

Matias Greco[1] **Jorge A. Baier**[1,2]
[1] Pontificia Universidad Católica de Chile, Chile
[2] Instituto Milenio Fundamentos de los Datos, Chile
mogreco@uc.cl, jabaier@ing.puc.cl

Abstract

Reinforcement learning allows learning very accurate heuristics for hard combinatorial puzzles like the 15-puzzle, the 24-puzzle, and Rubik's cube. In this paper, we empirically investigate how to exploit these learned heuristics in the context of (deterministic) heuristic search with bounded suboptimality guarantees, using the learned heuristic for the 15 and 24-puzzle of DeepCubeA. We show that Focal Search (FS), in its most straightforward form, that is, using the learned heuristic to sort the focal list, has poor performance when compared to Focal Discrepancy Search (FDS), a version of FS that we propose that uses a discrepancy function to sort the focal list. This is interesting the best performing algorithm *does not use* the heuristic values themselves but just the ranking between the successors of the node. In addition, we show FDS is competitive with satisficing search algorithms Weighted A* and Greedy Best-First Search.

Introduction

Recent work has shown that reinforcement learning can learn very accurate heuristics estimators in different scenarios, such as domain-independent automated planning (Ferber, Helmert, and Hoffmann 2020) and combinatorial puzzles such as Rubik's cube and the sliding tile puzzle (Agostinelli et al. 2019).

Given an accurate learned heuristic function, a natural question to ask is *how to exploit such a function within a bounded-suboptimal search algorithm*; i.e, an algorithm that provides guarantees on the returned solution. This question is challenging because learned heuristics, even if highly accurate, cannot be assumed be admissible, preventing us from using well-known bounded-suboptimal algorithms such as Weighted A*, which rely on an admissible heuristic.

Recently, Araneda, Greco, and Baier (2021) studied various ways in which Focal Search (FS) (Pearl and Kim 1982), a bounded-suboptimal search framework, can be exploited when given a highly accurate learned *policy*. An important empirical finding of their work is that the notion of discrepancy, which given a state action sequence $s_0 a_0 s_1 a_1 \dots s_n$ counts the number of times a_i would have not been chosen by the learned policy at state s_i, results in best search performance. Their work, however, assumed no learned heuristic values were available.

In this paper, we study how to exploit a learned *heuristic* within a bounded-suboptimal search algorithm that uses a given admissible heuristic to provide suboptimality bounds. Following previous work, we study how FS can be used for this problem. In its original form, FS requires two inputs: an admissible heuristic function h, which is used to sort its open list, just like A* would, and a function $h_{\texttt{FOCAL}}$ which is used to sort the so-called FOCAL list. As such, given a learned inadmissible heuristic h', the most straightforward way to apply FS is by setting $h_{\texttt{FOCAL}} = h'$.

Here, however, we go beyond the straightforward setting, and inspired by the previous work of Araneda, Greco, and Baier, we propose Focal Discrepancy Search, which sorts FOCAL using a discrepancy score. To evaluate our algorithm, we used the learned heuristic of DeepCubeA (Agostinelli et al. 2019), which provides very accurate heuristic values, on the 15- and 24- puzzle. We compare our approach against classical bounded-suboptimal search algorithms, such as Weighted A*, and FS used in the straighforward setting described above. Also, we compare our approach against satisficing algorithms, which do not deliver suboptimality guarantees.

Our results show that Focal Discrepancy Search, which uses the learned heuristic to compute discrepancies, but which in practice *does not* exploit the heuristic values themselves during search, outperforms all other bounded suboptimality search algorithms. In addition, Focal Discrepancy Search is competitive relative to satisficing algorithms. Our experiments support that, when the objective is to exploit learned heuristics within bounded-suboptimal search, more effort should be put on learning an accurate ranking rather than an accurate cost-to-go estimation.

This is not the first work that has proposed the use of discrepancies in the context of machine learning and search. Discrepancies have been used in the context of automated planning with learned heuristics (Yoon, Fern, and Givan 2006, 2007). More recently, Cohen and Beck (2019) establish a relation between discrepancies and performance degradation for decoding neural sequence models. However, to the best of our knowledge, previous work has not studied heuristic discrepancies for bounded-suboptimal search.

Background

Learned Heuristics

Given a graph (S, E), where S is set of states and E is set of edges, a search problem is a tuple $(s_{start}, S, E, s_{goal})$ where s_{start} and s_{goal} are, respectively the initial and goal states. A heuristic function is a non negative function $h : S \to \mathbb{R}^{0,+}$ such that $h(s)$ estimates the cost of a path from s to s_{goal}. $h(s)$ is admissible iff $h(s) \leq h^*(s)$ for every $s \in S$, where h^* is the cost of a minimum-cost path from s to s_{goal}.

A learned heuristic is a function $h_\phi : s, \phi \to \mathbb{R}^{0,+}$, which maps a state s to a prediction of its cost-to-go, where ϕ is a set of trainable weights. During training, the weights ϕ are updated to minimize a loss function that represents the difference between $h_\phi(s)$ and $h^*(s)$. Usually, a learned heuristic is implemented as a deep neural network and could be trained with stochastic gradient descent and reinforcement learning or some form of supervision. Due to the inductive nature of the deep neural nets, it is not possible to ensure that the predicted value for a state s does not overestimate $h^*(s)$. For that reason, even if the h_ϕ is very accurate, assume it inadmissible. Thus use it in a typical bounded suboptimal heuristic search algorithms, such as Weighted A*, could not deliver guarantees in terms of solution quality.

Focal Search

Focal Search (FS) (Pearl and Kim 1982) is a well-known bounded suboptimal search algorithm. In addition to an admissible heuristic, it can guide the search using additional information. It uses two priority queues: `OPEN` which is sorted in ascending order by $f(s) = g(s) + h(s)$, where h is an admissible heuristic function; and `FOCAL` which is sorted by $h_{\texttt{FOCAL}}$, an arbitrary priority function, and contains a subset of `OPEN`. FS receives a parameter w to control the suboptimality of the solution. The `OPEN` list contains all generated and not yet expanded states. The `FOCAL` list contains every states in OPEN such that $f(s) \leq wf_{min}$, where f_{min} is the minimum f-value of a node in `OPEN`. At each iteration, it extracts from `FOCAL` a state s which minimizes $h_{\texttt{FOCAL}}$. Then it expands s. If a generated successor s' is within the bound, i.e., $f(s') \leq wf_{min}$, then s' is added to `FOCAL`. State s' is also added to `OPEN`. Since the value of f_{min} may increase during execution with a consistent heuristic, nodes that previously were added to `OPEN` but not to `FOCAL`, may have to be added to `FOCAL` when f_{min} increases.

Focal Discrepancy Search

As originally defined by Harvey and Ginsberg (1995) in the context of Depth-First Search, a *discrepancy* occurs over a path when at a certain state of the path s, the action taken does not lead to child of s with minimum h-value. This concept can be applied to a Best-First search algorithm, selecting for expansions the node which has a lower discrepancy. Discrepancies in Focal Search have been used before in the context of learned stochastic policies for deterministic domains (Araneda, Greco, and Baier 2021). The resulting algorithm, in every iteration, extracts from `FOCAL` the state that maximizes the probability that its path is a prefix of an optimal path. In our context, the definition of a discrepancy is based on preferred actions.

We define a preferred action at state s as the node with the most promising heuristic value between the successors, i.e.

$$pref\_state(s) = \text{argmin}_{s' \in succ(s)}\{h(s')\} \qquad (1)$$

Assuming that s is a state which during state has been reached via path $\sigma_s = s_1 s_2 \dots s_n$ (with $s_1 = s_{start}$ and $s_n = s$), we define $N_{\text{nonpref}}(\sigma_s)$ as the number of times along the path σ_s in which the state with best heuristic value was not taken (i.e, $\sum_{i=1}^{n-1}[pref\_state(s_i) \neq s_{i+1}]$, where $[A] = 1$ if Boolean expression A evaluates to true, and $[A] = 0$ otherwise). Thus, we define a discrepancy as:

$$h_{\text{disc\_best}}(s) = N_{\text{nonpref}}(\sigma_s). \qquad \textit{(FDS(best))}$$

Discrepancies were originally proposed for binary trees. Some researchers (e.g., Karoui et al. 2007) have considered counting discrepancies according to their successor rank in non-binary trees. In our experimental section we evaluate a variant of $h_{\text{disc\_best}}$, defined as:

$$h_{\text{disc\_rank}}(s) = \text{rank}(\sigma_s), \qquad \textit{(FDS(rank))}$$

where $\text{rank}(\sigma_s)$ adds up the h-value rank of each of the states s_{i+1}, for $i \in \{1, \dots, n-1\}$, where the h-value rank of s_{i+1} corresponds to the rank of s_{i+1} among all the children of s_i. We assume the rank of the best child is 0.

Below *FDS(best)* and *FDS(rank)* denote the algorithm that results by setting $h_{\texttt{FOCAL}}$ to $h_{\text{disc\_best}}$ and $h_{\text{disc\_rank}}$.

Experimental Results

Our empirical evaluation seeks to evaluate the perfomance of the proposed approach with other bounded suboptimality search algorithms. On the other hand, we wanted to verify if the approach is also competitive with satisficing algorithms.

We use the pre-trained models of DeepCubeA (Agostinelli et al. 2019) for the 15- and 24- puzzle as the learned heuristic estimator. The pre-trained models are publicly available[1].

For the evaluations, we use the 100 Korf instances for the 15-puzzle (Korf 1985), and the 50 Korf's instances for the 24-puzzle (Korf and Felner 2002). The pre-trained models were trained using a different goal state that the goal state defined by Korf, i.e., in Korf's instances, the first position corresponds to empty tile, and in DeepCubeA corresponds to the last position. Despite that, it is possible to convert a state to evaluate the heuristic using the model, simply rotating the puzzle and remapping the tiles.

We evaluate two types of search algorithms that use the learned heuristic: bounded suboptimality search algorithms (BSS) and satisficing algorithms (SA). In BSS, we include *FDS(best)*, *FDS(rank)*, and Focal Search using the learned heuristic as h_{focal} (henceforth *FS(h_nn)*). We also include Weighted A* (*wA**) used with the (admissible) Linear Conflict heuristic (Hansson, Mayer, and Yung 1992). In SA, we include *wA** using the (non-admissible) learned heuristic in the same way it was used by DeepCubeA (i.e., setting

[1]https://github.com/forestagostinelli/DeepCubeA

Table 1: Results for the 15-puzzle with the algorithms using a suboptimality bound $w = 1.5$

	Coverage	Expansions (avg)	Cost (avg)
*wA**	100%	22100	56.67
FS(h_nn)	83%	10414	54.57
FDS(best)	100%	1478	55.47
FDS(rank)	100%	6542	55.45

Table 2: Results for the 24-puzzle with the algorithms using a suboptimality bound $w = 1.5$

	Coverage	Expansions (avg)	Cost (avg)
*wA**	68%	1519344	112.20
FS(h_nn)	96%	4465	111.5
FDS(best)	100%	137	110.26
FDS(rank)	100%	139	109.98

$w = 1.25$), Greedy Best-First Search (GBFS), and A* with preferred operators (PrefA*), a variant of Fast-Downward's search algorithm (Helmert 2006), where, intuitively, the action which leads to a child with minimum h-value is a preferred operator.

All algorithms were implemented in Python3, and the experiments were run on an Intel Xeon E5-2630 machine with 128GB RAM, using a single CPU core and one GPU Nvidia Quadro RTX 5000. We use a 30-minute timeout.

Tables 1 and 2 show a summary of the results obtained on the 15- and 24-puzzle, respectively. They show the percentage of problems that each algorithm can solve (coverage), the average number of expansions, and the average cost on 100 instances. On the 15-puzzle, *wA**, *FDS(best)*, and *FDS(rank)* solved all problems, but *FS(h_nn)* only solved 83% of the instances. In terms of expansions, *FDS(best)* outperforms *wA** in more than one order of magnitude.

Table 2 shows the results obtained on the 24-puzzle. In this domain, *FDS(best)* and *FDS(rank)* solved all problems, *FS(h_nn)* solved the 96% (48 instances), and *wA** solved only 56% (28 instances). In terms of expansions, *FDS(best)* and *FDS(rank)* outperforms *FS(h_nn)* by one order of magnitude, and *wA** by four orders of magnitude. We observe a substantial difference in the behavior of the algorithms between the 15- and 24-puzzle. We attribute this difference to the fact that the admissible heuristic is inaccurate in complex or challenging instances, and Focal Search needs to expand many states in order to prove the suboptimality bound.

Figure 1 shows the results (cumulative runtime, cumulative expansions, and cumulative suboptimality) obtained by BSS on each domain. We use a square mark to indicate the search algorithm failed to solve a particular instance. In that case, we consider the suboptimality of such a solution to be w times the optimal cost (for the 24-puzzle problems we use the optimal cost reported by (Korf and Felner 2002). In addition, when a problem is not solved we do not increment the cumulative runtime and cumulative expansions.

On the 15-puzzle, *FDS(best)* outperforms *wA** by one order of magnitude with respect to number of expansions. Nevertheless, both require a similar time. This is due to the fact that expansions are slower when using h_nn; in fact, they are one order of magnitude slower. We observe that *FDS(best)* loses advantage over *wA** for the last 30 instances. This may be due to the fact that many search states have the same f-values and many expansion are needed to make progress in the search.

On the 24-puzzle, the plot shows that *FDS(best)* and *FDS(rank)* outperform *wA** by four orders of magnitude regarding the number of expansions. Also, we observe that *FDS(best)* and *FDS(rank)* outperform *FS(h_nn)* in more than one order of magnitude.

Figure 2 shows the results obtained by satisficing algorithms and FDS(best) (with suboptimality bound 1.5 and 2.0) for the 15 and 24-puzzle. We included FDS(best) because it was the best performing among the BSS algorithms. On the 15-puzzle, the figure shows that *GBFS* and *PrefA** make almost the same number of expansions as the solution cost in all instances. On the other hand, *wA*(h_nn)* makes more expansions but produces better solutions. We observe that, in the first 30 instances, all algorithms perform similarly, but as the problem becomes more complex, FDS and *wA*(h_nn)* require more expansions. On 24-puzzle domain, we observe that all algorithms perform expansions within the same order of magnitude. Remarkably, we observe that FDS(best), a bounded suboptimality algorithm with different properties, is very competitive with satisficing algorithms, and as the suboptimality bound is enlarged, perform similar expansions and produce similar solutions.

In summary, the results show that *FDS(best)*, which is a straightforward way to include discrepancies in the `FOCAL` list, outperforms all other BSS algorithms in terms of expansions and solution quality, and it is very competitive with (incomplete and without guaranties) satisficing algorithms.

Discussion

Once that an accurate heuristic estimator has been learned, the next step is to determine how to use and exploit it within a search procedure. The most straightforward method to use a learned heuristic is to follow the heuristic estimator to the most promising state, for example by using it directly with GBFS. However, by doing so, we obtain a solution that does not have any theoretical guarantees.

A classical question in AI is how to combine inductive knowledge, such as a learned heuristic, with a symbolic procedure. In this paper we aims at exploiting such inductive knowledge acquired with reinforcement learning within a heuristic search algorithm. In addition, our algorithm needs a user-given admissible heuristic that is necessary to provide suboptimality guarantees and complement the power of a learned heuristic with theoretical bounds.

The literature has also studied how to take advantage of heuristics search to enables the rapid learning of heuristics and policies (Orseau and Lelis 2021), which may be the next step of this research.

Conclusions and Future Work

In this paper, we present Focal Discrepancy Search, a method that uses Focal Search to exploit a given learned

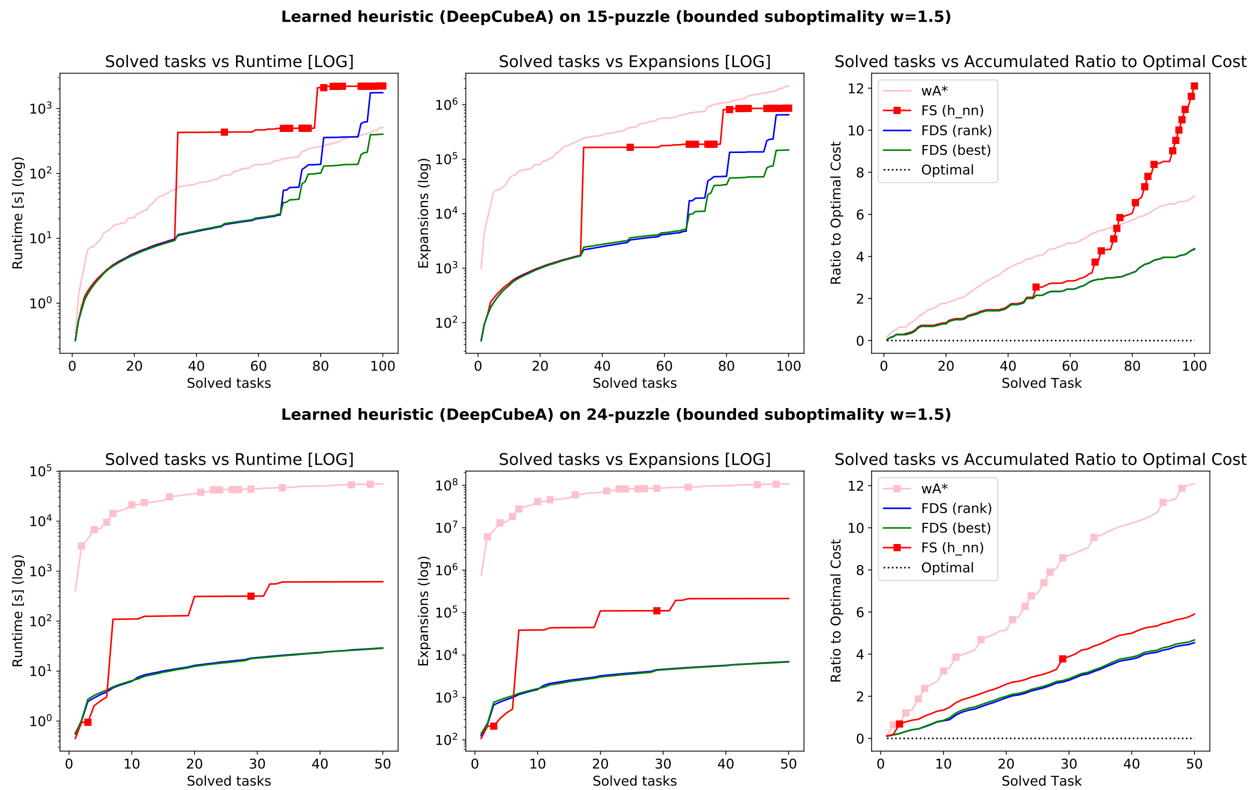

Figure 1: Results in 15- and 24-puzzle for bounded suboptimality search algorithms

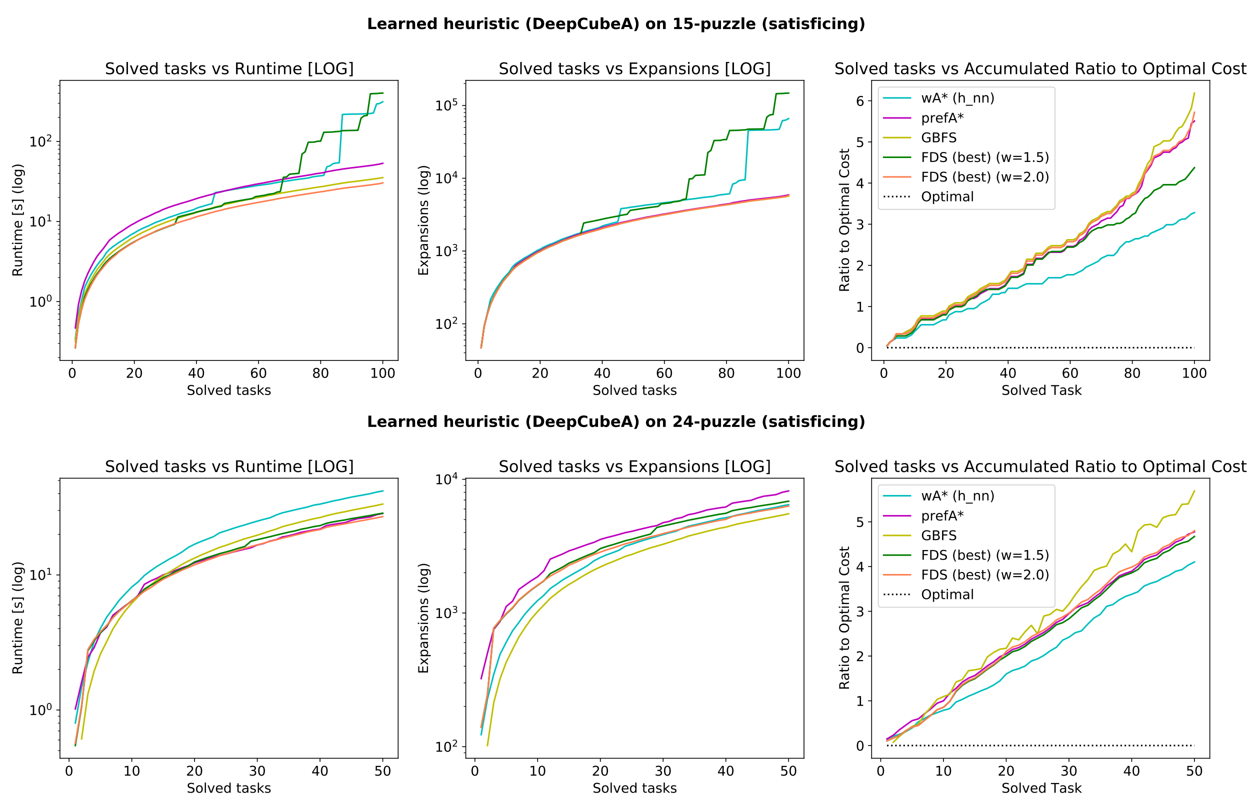

Figure 2: Results for 15- and 24-puzzle for satisficing algorithms and FDS(best), a bounded suboptimality search algorithm.

heuristic. We show that it is possible to exploit an (inadmissible) learned heuristic together with an admissible heuristic, in a bounded suboptimality search procedure that provides suboptimality guarantees. This idea has already been exploited in the context of learned policies and in this paper we show a simple way of adapting it to the context of learned heuristics. Our experiments were built over DeepCubeA, a recent framework that learns very accurate heuristics for puzzle games with Reinforcement Learning. We show that FDS outperforms others bounded-suboptimality search algorithms, such as *wA**, up to four orders of magnitude, and it is competitive with satisficing algorithms which do not provide suboptimality guaranties. Also, we show that FDS outperforms *FS(h_nn)*, which uses the heuristic value to sort the `FOCAL` list. We perform experiments using a suboptimality bound, but it can be also applied to bounded cost. An important conclusion is that our results suggest that it may be more important to learn to rank successors in order to compute discrepancies rather than learning cost-to-go values. As future work, we seek to move this approach to a parallel/concurrent algorithm that could exploit the GPU resources to compute the heuristic values.

Acknowledgements

Matias Greco was supported by the National Agency for Research and Development (ANID) / Doctorado Nacional / 2019 - 21192036.

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.

Araneda, P.; Greco, M.; and Baier, J. 2021. Exploiting Learned Policies in Focal Search. In *Proceedings of the 14th Symposium on Combinatorial Search (SoCS)*. URL https://arxiv.org/abs/2104.10535.

Cohen, E.; and Beck, J. C. 2019. Empirical Analysis of Beam Search Performance Degradation in Neural Sequence Models. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, 1290–1299. PMLR. URL http://proceedings.mlr.press/v97/cohen19a.html.

Ferber, P.; Helmert, M.; and Hoffmann, J. 2020. Reinforcement Learning for Planning Heuristics. In *Proceedings of the 1st Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL)*, 119–126. University_of_Basel.

Hansson, O.; Mayer, A.; and Yung, M. 1992. Criticizing solutions to relaxed models yields powerful admissible heuristics. *Information Sciences* 63(3): 207–227.

Harvey, W. D.; and Ginsberg, M. L. 1995. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26: 191–246.

Karoui, W.; Huguet, M.; Lopez, P.; and Naanaa, W. 2007. YIELDS: A Yet Improved Limited Discrepancy Search for CSPs. In Hentenryck, P. V.; and Wolsey, L. A., eds., *4th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 4510 of *LNCS*, 99–111. Springer.

Korf, R. E. 1985. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence* 27(1): 97–109.

Korf, R. E.; and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial intelligence* 134(1-2): 9–22.

Orseau, L.; and Lelis, L. H. S. 2021. Policy-Guided Heuristic Search with Guarantees. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 12382–12390. AAAI Press.

Pearl, J.; and Kim, J. H. 1982. Studies in Semi-Admissible Heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4(4): 392–399.

Yoon, S. W.; Fern, A.; and Givan, R. 2006. Learning Heuristic Functions from Relaxed Plans. In Long, D.; Smith, S. F.; Borrajo, D.; and McCluskey, L., eds., *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 162–171. AAAI.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Using Learned Policies in Heuristic-Search Planning. In Veloso, M. M., ed., *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2047–2053.