Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Learning Classical Planning Strategies with Policy Gradient

Paweł Gomoluch, Dalal Alrajeh, Alessandra Russo
Department of Computing, Imperial College London
{pawel.gomoluch14,dalal.alrajeh,a.russo}@imperial.ac.uk

Abstract

A common paradigm in classical planning is heuristic forward search. Forward search planners often rely on simple best-first search which remains fixed throughout the search process. In this paper, we introduce a novel search framework capable of alternating between several forward search approaches while solving a particular planning problem. Selection of the approach is performed using a trainable stochastic policy, mapping the state of the search to a probability distribution over the approaches. This enables using policy gradient to learn search strategies tailored to a specific distributions of planning problems and a selected performance metric, e.g. the IPC score. We instantiate the framework by constructing a policy space consisting of five search approaches and a two-dimensional representation of the planner's state. Then, we train the system on randomly generated problems from five IPC domains using three different performance metrics. Our experimental results show that the learner is able to discover domain-specific search strategies, improving the planner's performance relative to the baselines of plain best-first search and a uniform policy.

1 Introduction

As a simple and complete search algorithm, best-first search forms the core of many modern classical planners (e.g. (Helmert 2006; Richter and Westphal 2010)). Approaches combining greedy best-first search (GBFS) with other planning techniques have largely been confined to sequentially attempting to solve the problem using two different search modes (e.g. (Hoffmann and Nebel 2001; Lipovetzky and Geffner 2017) or even an entire portfolio of potentially unrelated algorithms (e.g. (Howe et al. 2000; Gerevini, Saetti, and Vallati 2009; Fawcett et al. 2011; Helmert, Röger, and Karpas 2011; Cenamor, De La Rosa, and Fernández 2016)). Elsewhere, best-first search (BFS) has been combined with an auxiliary exploratory technique, triggered when the main GBFS fails to reach progress for a certain number of expansions (Xie, Müller, and Holte 2014; Lipovetzky and Geffner 2017).

In this work, we introduce a framework capable of systematically alternating between various forward search techniques, in the course of solving the planning problem. Unlike (Xie, Müller, and Holte 2014; Lipovetzky and Geffner 2017), we equip the planner with more than two techniques and do not manually specify the rules for choosing between them. The choice of the technique is made by the planner using a stochastic strategy, trained to maximize the planner's performance using reinforcement learning. With this approach, it is possible to train the planner for a particular domain or a selected performance objective, such as maximizing the IPC score or minimizing the time required to find a solution. By enabling the learner to discover domain-specific search strategies, this approach has the potential to cover the middle ground between general-purpose search algorithms designed to fit a variety of domains and the domain-specific solvers, hand-crafted by human experts. Moreover, through seamless integration of various performance objectives, it allows for automatic navigation of the trade-off between the time required to find plans and their cost, which is a central issue in satisficing planning.

To demonstrate applicability of our framework, we introduce a specific instantiation, which uses five different forward search approaches and a simple characterization of the planner's state in terms of the estimated distance to the goal (i.e. the heuristic value) and the remaining time available to the planner.

We then empirically evaluate the resulting system by training it on randomly generated problems from five IPC domains. Although our training scheme uses planning problems of relatively small size, we extend our evaluation to include larger, IPC-scale problems, which allows for testing the generality of the learned strategies with respect to the problem size.

2 Related work

In (Xie, Müller, and Holte 2014), GBFS is augmented with local search or random walks whenever the search does not yield progress for a certain number of node expansions. Progress of the search is determined by decrease of h_{min}, the lowest value of the heuristic function observed so far. In a similar way, GBFS can be combined with *width-based* search, which prunes out states which do not satisfy a novelty criterion (Lipovetzky and Geffner 2017). Both approaches rely on single addition to GBFS, which is triggered when the main search fails to find states with lower heuristic value for a certain number of expansions. Our framework allows for a larger number of search approaches and does

not explicitly distinguish between primary and backup ones. The strategy for choosing between them is subject to the learning process.

Related to our approach is also the concept of portfolio planners, especially those configured based on experience gathered on a set of training problems (Fawcett et al. 2011; Helmert, Röger, and Karpas 2011; Cenamor, De La Rosa, and Fernández 2016). The key difference between portfolio approaches and our work is that the operation of a portfolio planner involves running a number of independent searches, each with a different planning algorithm. In our approach a single search is performed, with the possibility of alternating between compatible search techniques, depending on the state of the search.

Learning from experience has long been used as a way of improving the planner's performance. In classical planning, the work in this area included learning macro actions (e.g. (Fikes, Hart, and Nilsson 1972; Coles and Smith 2007), control knowledge in the form of decision rules (e.g. (Leckie and Zukerman 1998; Yoon, Fern, and Givan 2008)) and heuristic functions (Yoon, Fern, and Givan 2008; Virseda, Borrajo, and Alcazar 2013; Garrett, Kaelbling, and Lozano-Perez 2016). A survey of learning methods for automated planning can be found in (Jimenez et al. 2012). To the best of our knowledge, none of the learning approaches attempted to construct a domain-specific composition of different search techniques.

3 Background

Classical planning Planning is the problem of finding sequences of actions which, when executed from a given initial state, lead to a state in which the planning goal is satisfied. Classical planning, in particular, relies on a known and perfect model of the environment, including a discrete set of deterministic actions. Formally, a classical planning task is given by a tuple $\langle V_p, O, s_o, g\rangle$, where V_p is a set of finite-domain variables, O is the set of operators, s_0 is the initial state, which is an assignment over the variables of V_p, and g is the goal, a partial assignment over the variables of V_p. Each operator $o \in O$ is itself specified with a tuple $\langle \mathrm{pre}(o), \mathrm{eff}(o)\rangle$, where $\mathrm{pre}(o)$ and $\mathrm{eff}(o)$ are both partial assignments over V_p, defining the preconditions and the effects of applying operator o, respectively. Operator o can be applied in state s if and only if $\mathrm{pre}(o) \subseteq s$. The result of applying operator o in state s, denoted as $o(s)$ is defined as an assignment over V differing from s by setting the variables covered by $\mathrm{eff}(o)$ accordingly.

A common classical planning approach is forward search. The search starts at the initial state and iteratively explores states reachable by sequentially applying operators of set O. A canonical example of forward search is *best-first search* (BFS), which always expands the node n with the lowest value under a specified evaluating function $f(n)$. Heuristic functions guide forward search by estimating the distance to the goal from any given node. Best-first search driven solely by the heuristic value of a node, $f(n) = h(n)$, is known as *greedy best-first search* (GBFS).

Reinforcement learning We adopt a reinforcement learning approach based on the standard Markov Decision Process (MDP) setting. An MDP is given by a tuple $\langle S, A, p(s, a, s'), r(s, a), \gamma\rangle$, where S is a set of states, A is a set of actions, $p(s, a, s') : S \times A \times S \to S$ is a function determining the probability of a transition to state s', given that action a is taken in state s, $r(s, a) : S \times A \to \mathbb{R}$ is a function yielding the expected reward for taking action a in state s and γ is the discount factor. The task of an agent operating in MDP is to maximize the discounted sum of rewards observed during an episode: $G = \sum_{t=1}^{T} \gamma^t r_t$, where T is the length of the episode. A *policy* of the agent is a (possibly stochastic) mapping from S to A. Since in this paper we focus on stochastic policies (search strategies), we assume a policy to be a function $\pi(s, a) = p(a|s)$, indicating the probability of agent selecting action a in state s, for every $s \in S, a \in A$.

Given the focus on stochastic policies, we employ a policy gradient method of learning them, based on the REINFORCE algorithm (Williams 1992; Sutton and Barto 1998). The core idea of REINFORCE is that unbiased samples of the gradient of the return G with respect to parameters θ of policy π can be computed using an expression that only depends on the current policy and a single return sample. Every state-action pair (s, a) occurring at time t in a particular episode, generates the following update of the policy parameters θ:

$$\Delta\theta = \alpha(G_t - b(s))\frac{\nabla_\theta \pi(s, a)}{\pi(s, a)} \tag{1}$$

where α is the learning rate, G_t is the discounted sum of rewards from time $t + 1$ to the end of the episode and $b(s)$ is the baseline for state s, for example computed as the average of returns observed for that state in previous episodes.

4 Approach

The planning approach proposed in this paper relies on the idea that the search algorithm does not have to be fixed throughout the process of solving a planning problem. A range of planning algorithms can interleave, provided that they can all be cast as operations processing common internal state of the planner. In the remainder of the paper, we assume that the algorithms are available to the planner in the form of a set of *routines*. The routines applied to the state of the planner perform an atomic step of the corresponding algorithm. For example, a step of GBFS can consist of selecting the node with the lowest heuristic value, expanding it and adding its children to the queue (open list).

Algorithm 1 outlines our framework for planning with alternating search routines. Typically for a planner, the algorithm's arguments include the initial state s_0, the goal g and the set of operators O. Additional parameters are the set of search routines A and time limit t_r. The algorithm initializes the state *queue* (the open list) with s_0.Then, it chooses a routine from set of routines A, applies it for a limited time of up to t_r, then chooses a routine again, and so on, until a plan is found. A single application of *routine* to the *queue* modifies the queue and returns the plan if a solution is found,

returns failure if the search space is exhausted without finding a plan and returns a special *in-progress* token otherwise. Note that the algorithm is presented in a simplified form, focusing on alternating between the routines. It hides details such as keeping track of already expanded states (closed list) and recording their ancestors for the purpose of extracting the plan.

Algorithm 1 Planning with alternating search routines

```
function PLANASR(s_0, g, O, A, t_r)
    queue ← [s_0]
    while true do
        routine ← choose(A)
        t_start ← now()                          ▷ current time
        while now() − t_start < t_r do
            result ← routine(queue)
            if result ≠ in-progress then
                return result                    ▷ a plan or failure
            end if
        end while
    end while
end function
```

The problem of choosing a routine given the state of the search can be modeled as an MDP where the state set S is the set of possible states of the search and the action set A is the set of search routines available to the planner. The states of the search are assignments over a set of finite-domain variables V_l. In Section 5 we introduce particular instantiations of A and V_l. The reward function can be any measure of planner's performance available after a single attempt at a given planning problem. In the simplest scenario, the reward can be defined as 1 if the planner solves the problem within a set time limit and 0 otherwise. Training with such a reward function would correspond to optimizing for coverage, i.e. the number of problems solved, disregarding the plan quality. The reward functions used in our experimental setup are discussed in Section 5.

The search policy is parametrized by θ, with one parameter $\theta_{i,j}$ for every state-action pair (s_i, a_j). The probability of taking action a_j in state s_i is determined by softmax over the parameters associated with the state:

$$\pi(a_j|s_i) = \frac{e^{\theta_{i,j}}}{\sum_j e^{\theta_{i,j}}}$$

We train the weights using a variant of episodic REINFORCE (Williams 1992). To decrease the variance of the gradient sample we average the update over N episodes, during which the same stochastic policy is used on the same training problem. In our setting with just four states, a state is typically visited many times during a single episode and many (possibly different) actions are taken from it. The parameter update after every N episodes is therefore:

$$\Delta\theta = \alpha \frac{1}{N} \sum_{i=1}^{N} \sum_{s,a} \eta_i(s,a)(G_t - V(s)) \frac{\nabla_\theta \pi(s,a)}{\pi(s,a)} \quad (2)$$

where α is the learning rate, $\eta_i(s,a)$ is the number of times action a was taken from state s in episode i, G_t is the return observed at the end of the episode and $V(s)$ is a baseline for state s, computed as the average of all past returns for episodes passing through s.

5 Approach instantiation

In this section we describe an instantiation of the framework described in Section 4, which we then empirically evaluate in Section 6. The key components of the framework are the set of search routines A, the representation of the planner's state and the reward function, reflecting the chosen performance objective.

Another important choice is t_r, the time span over which the chosen routine continues to be applied. In this work we fix t_r to 100 ms, which enables a single routine to make substantial progress, while also allowing for strategies interleaving the routines in a fine-grained manner.

Search routines

Below, we describe the routines which we include in the set A of our instantiation and state how they can be integrated within the framework.

Greedy best-first search Plain greedy best-first search always expands the node with the lowest value of h. A single application of this routine consists of a single node expansion, followed by placing of all its ancestors in the queue.

ϵ-greedy search ϵ-greedy search was first considered in classical planning context by (Valenzano et al. 2014). Like greedy best-first search, one application of this routine performs a single node expansion. The difference is that with probability of ϵ a random node is selected from the queue with the probability of selection uniform across all the nodes. Throughout the paper we use $\epsilon = 0.2$.

Greedy search with random walks A variation of GBFS, following the expansion of node n with a single random walk of length l starting in n, provided that no decrease in heuristic value has been observed for the last s node expansions. All the nodes along the walk are added to the global queue. The walk stops as soon as a state with heuristic value lower than that of n is found. This method is inspired by (Xie, Müller, and Holte 2014), but throughout the paper we use parameters of $s = 5$ and $l = 20$, which makes the random walks much more frequent. This is to ensure that the routine is substantially different from plain GBFS and offers the learner a meaningful alternative.

Local search Local search is started from a node with the lowest h value, extracted from the global queue when the routine is selected. The node is used to initialize the local queue, which persists between subsequent calls of the routine. The search continues by expanding states from the local queue. When the time limit t_r expires, the local queue is merged into the global one (all the states from local queue

are inserted to the global one). If the local queue becomes empty before t_r expires, another node from the global queue is put in the local one, effectively starting a new local search.

Heuristic-guided depth-first search Depth-first search is performed using a local search stack. When the routine is selected, a node with the lowest h value is extracted from the global queue and placed on the stack. At every call, a node n is popped from the stack and expanded. Descendant nodes are put on the stack in order of decreasing h value, so that the node expanded at the next step is the descendant of n with the lowest h. The descendant nodes are also inserted in the global queue. If the stack becomes empty before t_r expires, another node from the global queue is put on the stack and the search continues.

State representation

As stated in Section 4, we the state space of our learner is the set of possible assignments over the variables of V_l. In the remainder of the paper, we consider V_l to be a set of two boolean variables $V_l = \{d, t\}$. We take d to be a binarized heuristic estimate of the distance to the goal:

$$d = \begin{cases} 0 & \text{if } h_{\text{best}} < h_0/2 \\ 1 & \text{otherwise} \end{cases}$$

where h_{best} is the lowest heuristic value recorded in the search so far and h_0 is the heuristic value of the initial state. The variable is a simple way of tracking the progress of the search. Similarly, t indicates how much time the planner has left:

$$t = \begin{cases} 0 & \text{if } t_{\text{elapsed}} < t_{\text{max}}/2 \\ 1 & \text{otherwise} \end{cases}$$

where t_{elapsed} is the time elapsed from the beginning of the search and t_{max} is the total time allocated for the search.

Using two binary state features allows for a compact tabular representation of the learned policies. For readability, we further refer to the states of the search as *near* ($d = 0$) or *far* ($d = 1$) and *early* ($t = 0$) or *late* ($t = 1$).

Reward functions

We consider three different reward functions, corresponding to three different ways of scoring performance of a planner. The first reward is based on IPC score, first used in IPC-2008[1]. This is a widely used measure of planners' performance, assigning higher scores to planners finding lower-cost solutions. For every solved problem, the planner receives score defined as follows:

$$z = \frac{c_{min}}{c}$$

where c is the cost of the plan returned by the planner and c_{min} is the cost of best known solution. For failed problems, planners receive a score of 0. Cost c_{min} is determined using a set of reference planners, each based on a single routine from set A and run with the same timeout as the trained planner.

[1] http://icaps-conference.org/ipc2008/deterministic/

During the training phase, we deviate from traditional IPC score in two ways. First, we do not include the trained planner in the reference set, allowing for a situation where $c < c_{min}$ an so $z > 1$. This way, the learner observes rewards higher than 1 for finding solutions with costs strictly lower than any of the reference planners. Second, if the trained planner solves the problem but none of the reference planners does, we set the reward to a fixed value of 2. The choice of value 2 is motivated by the fact that, during a competition, solving a problem unsolved by others yields a net advantage of 1, which is 1 more than in the case when a solution with cost equal to the best competitor is found, when both planners receive one point. Both of these changes aim at providing the reward the ability to distinguish between situations where the learner's performance is as good as the reference planners' and the cases when it is strictly better. If a problem is solved neither by the learner nor by any of the reference planners, the reward is not defined and the episode does not generate a parameter update. Formally,

$$r_{ipc} = \begin{cases} \frac{c_{min}}{c_L} & \text{if both } c_{min} \text{ and } c_L \text{ are defined} \\ 0 & \text{if only } c_{min} \text{ is defined} \\ 2 & \text{if only } c_L \text{ is defined} \end{cases}$$

where c_L is the cost of the plan computed by the trained strategy.

The modified version of IPC score is only used in the training phase. Naturally, during the evaluation of the trained policies, the learned planner is included in the reference set, treating it on par with any other planner included in the comparison and effectively capping its score for any single problem at 1.

The second reward function we consider is based on squared IPC score. The motivation is to further increase emphasis on the cost of the plan. For example, a planner returning a plan two times more expensive than the best plan known would only receive $(\frac{1}{2})^2 = \frac{1}{4}$ points. Given the modifications we introduced in r_{ipc}, we cap the reward at 2 to prevent excessive premium for performance strictly better than that of reference planners. Formally:

$$r_{ipc2} = \min(r_{ipc}^2, 2)$$

Finally, we consider a reward based solely on the time used to find a solution. The learner receives a reward equal to the proportion of spared time to the total time allocated for the problem:

$$r_{time} = \begin{cases} \frac{T-t}{T} & \text{if solved} \\ 0 & \text{otherwise} \end{cases}$$

where T is the total time allocated for the planner and t is the actual time elapsed before the plan is returned. Note that another reward function disregarding the plan cost could be derived from coverage (1 if the problem is solved and 0 otherwise). This however would ignore the computation time as long as it falls within the limit. The time-based formulation retains more information in the reward signal and incentivizes finding a solution as quickly as possible.

Transport

IPC reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	**0.98**	0.01	~ 0	~ 0	~ 0
near late	0.01	~ 0	**0.97**	0.02	~ 0
far early	~ 0	~ 0	~ 0	**1.00**	~ 0
far late	0.01	0.01	**0.94**	0.01	0.02
IPC^2 reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.01	**0.98**	~ 0	~ 0	~ 0
near late	0.14	**0.22**	**0.60**	0.04	~ 0
far early	**0.96**	0.02	0.01	~ 0	~ 0
far late	0.02	0.02	**0.91**	0.05	0.01
Time reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.01	0.01	**0.95**	0.01	0.01
near late	0.04	0.03	0.05	0.09	**0.78**
far early	0.01	~ 0	**0.97**	0.01	0.01
far late	0.19	0.17	**0.23**	0.19	**0.21**

Parking

IPC reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	~ 0	~ 0	~ 0	~ 0	**0.99**
near late	0.01	0.01	0.01	0.02	**0.94**
far early	~ 0	~ 0	~ 0	~ 0	**0.99**
far late	0.2	0.2	0.2	0.2	0.2
IPC^2 reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.2	0.02	0.01	0.15	**0.80**
near late	0.01	0.01	0.01	0.02	**0.96**
far early	~ 0	~ 0	0.01	~ 0	**0.98**
far late	0.19	0.19	0.19	0.19	**0.25**
Time reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	~ 0	~ 0	~ 0	0.01	**0.98**
near late	0.19	0.19	0.18	0.19	**0.25**
far early	0.01	0.01	0.01	0.01	**0.97**
far late	0.2	0.2	0.2	0.2	0.2

Elevators

IPC reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.05	**0.58**	0.03	**0.32**	0.02
near late	0.01	~ 0	0.01	**0.98**	~ 0
far early	0.02	0.01	**0.92**	0.02	0.04
far late	0.11	0.09	0.11	0.16	**0.53**
IPC^2 reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.01	~ 0	**0.98**	~ 0	~ 0
near late	0.01	0.01	**0.97**	0.01	~ 0
far early	0.01	0.03	**0.45**	**0.51**	~ 0
far late	0.14	0.14	0.19	**0.28**	**0.25**
Time reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.02	0.01	0.09	0.08	**0.79**
near late	0.07	0.04	**0.30**	**0.38**	**0.21**
far early	0.03	0.03	0.04	0.04	**0.86**
far late	0.19	0.19	**0.21**	0.20	**0.21**

No-mystery

IPC reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	~ 0	**0.93**	~ 0	0.06	0.01
near late	**0.94**	0.05	0.01	~ 0	~ 0
far early	0.01	**0.86**	0.01	0.01	0.10
far late	0.18	0.19	**0.24**	0.20	0.19
IPC^2 reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.06	**0.87**	0.01	0.02	0.04
near late	0.01	**0.99**	~ 0	~ 0	~ 0
far early	0.01	**0.92**	0.02	0.03	0.02
far late	0.19	**0.22**	0.20	0.20	0.20
Time reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	0.01	**0.97**	0.01	~ 0	0.01
near late	**0.42**	**0.38**	0.09	0.05	0.06
far early	0.01	**0.96**	0.01	0.01	0.01
far late	0.2	0.2	0.2	0.2	0.2

Floortile

IPC reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	~ 0	**1.00**	~ 0	~ 0	~ 0
near late	**0.42**	**0.55**	~ 0	0.01	~ 0
far early	0.01	**0.98**	~ 0	~ 0	~ 0
far late	0.2	0.2	0.2	0.2	0.2
IPC^2 reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	~ 0	**1.0**	~ 0	~ 0	~ 0
near late	0.02	**0.97**	~ 0	~ 0	~ 0
far early	0.01	**0.97**	0.01	0.01	0.01
far late	0.2	0.2	0.2	0.2	0.2
Time reward					
	GBFS	ϵ-greedy	RW	Local	DFS
near early	~ 0	**1.00**	~ 0	~ 0	~ 0
near late	**0.79**	0.19	0.01	0.01	0.01
far early	0.01	**0.97**	0.01	0.01	0.01
far late	0.2	0.2	0.2	0.2	0.2

Entries in the table are the probabilities of selecting a given routine in a given search state, learned separately for each of the domains and reward functions. Probabilities > 0.2 highlighted in **bold**. The impact of different rewards is most visible in *Transport* and *Elevators* domains, where time-based reward shifts the policies towards depth-first search. On the other hand, changing the reward has little effect in *No-mystery* and *Floortile* domains, where the more aggressive routines are potentially harmful, and in *Parking*, which admits DFS without big impact on the plan cost.

Table 1: The policies learned for each of the planning domains and reward functions.

IPC score						
	T	P	E	N	F	Sum
GBFS	29.46	15.57	28.73	24	28.27	126.03
ϵ-gr.	27.23	10.88	21.87	**30.64**	**50.33**	140.95
RW	30.92	9.79	31.82	19.48	4.91	96.92
Local	37.18	21.89	**32.87**	20	23.9	135.84
DFS	12.95	**37.45**	16.92	7.7	0.13	75.15
Uni	24.07	25.71	26.79	20.75	32.44	129.76
L(I)	**38.85**	**37.27**	**32.67**	26.62	**49.83**	**185.24**
L(I^2)	36.79	**37.38**	**32.13**	27.2	49	182.5
L(T)	30.58	**36.56**	19.54	26.37	46.84	159.89
IPC^2 score						
	T	P	E	N	F	Sum
GBFS	27.49	13.66	**27.6**	24	24.67	117.42
ϵ-gr.	22.82	9.52	20.87	**30.5**	**44.69**	128.4
RW	19.26	8.41	24.27	19.38	3.95	75.27
Local	28.58	16.76	26.44	20	20.54	112.32
DFS	4.07	**25.93**	6.42	7.7	0.06	44.18
Uni	13.73	19.71	18.24	20.71	28.33	100.72
L(I)	**30.7**	**26.18**	25.13	26.55	**44.28**	**152.84**
L(I^2)	**30.55**	**26.51**	24.57	27.1	43.59	**152.32**
L(T)	18.75	25.5	8.77	26.26	41.36	120.64
Time score						
	T	P	E	N	F	Sum
GBFS	23.66	14.27	14.83	**20.58**	17.13	90.47
ϵ-gr.	21.48	10.15	10.44	**21.58**	**38.47**	102.12
RW	39.93	9.35	21.28	16.77	3.96	91.29
Local	33.87	22.45	19.22	15.98	14.02	105.54
DFS	36.88	**48.6**	**25.41**	5.44	0.21	116.54
Uni	34.55	27.75	20.25	17.18	19.71	119.44
L(I)	33.97	**47.69**	20.64	**21.12**	**38.39**	161.81
L(I^2)	29.27	46.98	21.11	**21.56**	37.9	156.82
L(T)	**41.32**	47.13	**24.8**	**20.83**	37.02	**171.1**

Table 2: IPC, IPC^2 and time score of the learned planning strategies and relevant baselines on 60 test problems randomly generated for each of the five domains: *Transport* (T), *Parking* (P), *Elevators* (E), *No-mystery* (N) and *Floortile* (F). Average of 10 test runs. L(I), L(I^2) and L(T) are the strategies trained with rewards based on IPC, IPC^2 and time-based score, respectively. Values within 1 point from the highest one highlighted in **bold**.

IPC score						
	T	P	E	N	F	Sum
GBFS	0	5.89	**11.56**	**8**	3.62	29.07
ϵ-gr.	0	3.84	**11.68**	**7.72**	4.87	28.11
RW	0.92	4.55	9.88	6.91	2.6	24.86
Local	**2**	9.98	**11.63**	6.98	3.64	34.23
DFS	0	8.88	7.47	**7.77**	0	24.12
Uni	0	6.21	**10.81**	**7**	3.61	27.63
L(I)	0	**12.27**	**11.78**	**7.72**	5.7	**37.47**
L(I^2)	0	**12.09**	**11.15**	**7.72**	**6.72**	**37.68**
L(T)	0.48	7.59	7.67	**7.72**	4.76	28.22
IPC^2 score						
	T	P	E	N	F	Sum
GBFS	0	5.79	**11.15**	**8**	3.29	28.23
ϵ-gr.	0	3.7	**11.39**	**7.48**	4.75	27.32
RW	0.85	4.15	8.05	6.83	2.29	22.17
Local	**2**	8.42	9.78	6.96	3.32	30.48
DFS	0	7.38	4.63	**7.55**	0	19.56
Uni	0	4.9	8.8	**7**	3.29	23.99
L(I)	0	**10.28**	9.6	**7.48**	**5.48**	**32.84**
L(I^2)	0	**10.69**	9.32	**7.48**	**6.48**	**33.97**
L(T)	0.23	6.5	4.82	**7.48**	4.55	23.58
Time score						
	T	P	E	N	F	Sum
GBFS	**0**	4.03	10.35	**7.77**	3.99	26.14
ϵ-gr.	**0**	2.95	10.07	**7.86**	4.96	25.84
RW	**0.72**	3.05	10.3	6.3	2.97	23.34
Local	**0.56**	8.49	**10.81**	**6.92**	3.97	30.75
DFS	**0**	8.47	**11.67**	**7.32**	0	27.46
Uni	**0**	5.51	**10.76**	6.46	3.98	26.71
L(I)	**0**	**10.85**	10.65	**7.86**	**5.93**	**35.29**
L(I^2)	**0**	**10.33**	**11.02**	**7.73**	**6.9**	**35.98**
L(T)	**0.13**	7.28	**11.64**	**7.89**	4.95	31.89

Table 3: IPC, IPC^2 and time score of the learned planning strategies and relevant baselines on IPC-11 problems under 60 second time limit. *Transport* (T), *Parking* (P), *Elevators* (E), *No-mystery* (N) and *Floortile* (F) domains. L(I), L(I^2) and L(T) are the strategies trained with rewards based on IPC, IPC^2 and time-based score, respectively. Values within 1 point from the highest one highlighted in **bold**.

6 Evaluation

The learning planner was implemented on the basis of the Fast Downward planning system (Helmert 2006). The source code is available online[2].

We tested the system on five IPC domains of *Transport*, *Parking*, *Elevators*, *No-mystery* and *Floortile*. This is the set of domains used in the learning track of IPC 2014 (Vallati et al. 2015), with the exception of the *Spanner* domain. We excluded *Spanner* because it was designed not to work well with delete-relaxation heuristics, such as the FF heuristic, used throughout all of our experiments. We fixed the time allocated for solving a single problem to 5 seconds. For each of the domains we generated 1000 training problems using the problem generators from the learning track of IPC 2014[3]. The parameters passed to the problem generators were selected to match the timeout of 5 seconds: for each of the domains we aimed at parameters for which the resulting problems will prove challenging but possible to solve. More precisely, we searched for generator configurations, for which about 50% of the problems could be solved in 5 seconds by the baseline planner using GBFS guided by FF heuristic (Hoffmann and Nebel 2001) with unary operator costs, which we used as the heuristic function throughout all the experiments. Training on substantially larger problems would be difficult in the current framework, because of the time required to complete a single episode (that is, solve the planning problem) and the sparser reward (more actions contributing to a single solution).

For every combination of domain and reward function, we trained the policy on a single CPU for 48 hours. During this time, problems were sampled randomly from the training set and attempted $N = 5$ times, before the policy was updated according to Equation 2 with $\alpha = 0.02$. Table 1 shows the policies learned for each of the domains and reward func-

[2]https://github.com/pgomoluch/fd-learn

[3]http://www.cs.colostate.edu/~ipc2014/

tions. Every entry of the tables indicates the probability of selecting the routine given by the column in the state given by the row.

In the majority of the states, the learned policy approaches a deterministic one, with the probability of choosing the dominant routine exceeding 0.9. For some states, however, the policy remains nondeterministic. This is often the case for states visited infrequently during the training process. The most extreme example is the *far late* state, which did not occur at all when training on *Parking* with the IPC or time reward. To shed some light on the learning process, in Figure 1 we plot the probabilities of choosing particular routines in selected states, as functions of the number of training episodes.

In the *Transport* domain the preferred routine changes over the course of the search. For example, under the IPC reward, GBFS is preferred when substantial progress towards the goal has been made and there is still a lot of time (the *near early* state). However, when the time starts to run out, the planner switches to BFS augmented with random walks (the *near late* state). The learned policies vary depending on the selected reward function. The relatively conservative routines of GBFS and ϵ-greedy search are chosen in two states under the IPC^2 reward, one state under the plain IPC reward, and in none of the states in the case of time-based reward. In fact, with the time reward the learner goes as far as choosing DFS in the *near early* state. Indeed, in this domain DFS allows for rapid progress towards the goal, at the cost of generating strongly suboptimal plans. The impact of changing the objective is even more visible in the *Elevators* domain, where time reward results in frequent selection of DFS, while less aggressive options of local search and random walks are preferred under the two remaining rewards.

Since, in *Parking*, the use of DFS comes without substantial increase of the plan costs, the learner chooses it very consistently, irrespective of the selected reward function. The opposite occurs for *No-mystery* and *Floortile*: since aggressive exploration can be harmful in these domains (because of dead ends), the learner confined itself to GBFS and ϵ-greedy search. Again, this resulted in very similar policies learned across all three reward functions.

To evaluate the learned policies, for every domain we randomly generated 60 test problems, using the same generator parameters as for the training problems. Table 2 shows performance of the learned planners, measured by IPC, IPC^2 and time-based score respectively. For comparison, the baselines of uniform random policy and each of the routines on its own are included. The strategies learned with the corresponding reward function consistently outperform most of the baselines, reaching the highest or nearly-highest scores. In particular, for *Transport* the scores obtained when learning with relevant reward are higher than those reached with any of the routines on their own. In *Elevators*, the learner scores a bit lower than the best single routine (local search for IPC, GBFS for IPC^2 and DFS for the time score), but remains ahead of most others. On *Parking*, *No-mystery* and *Floortile* all the strategies are comparable to the single best routine, which is expected given the policies learned in these domains.

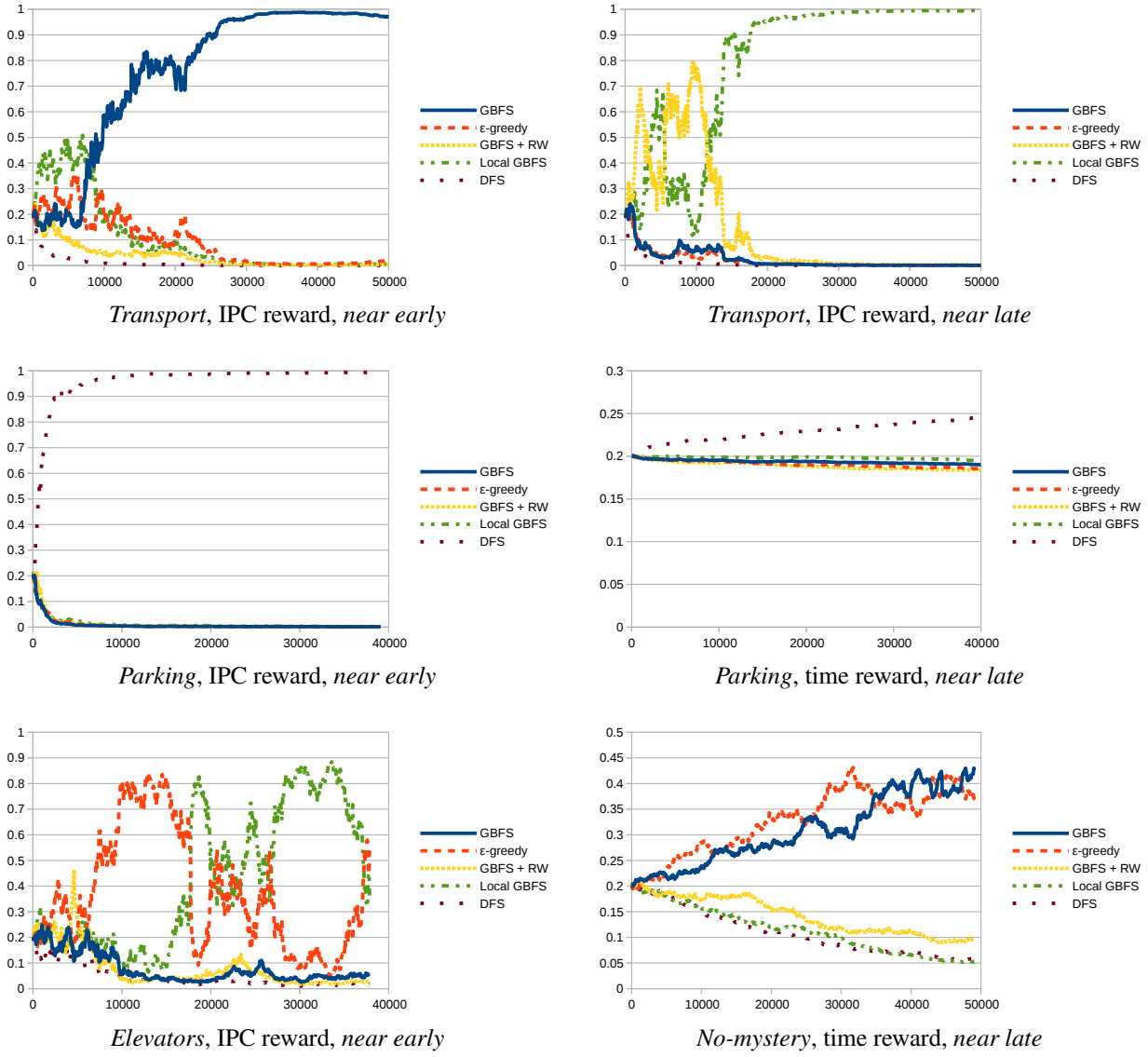
A real-world application of a learning planner would aim at training the system on problems representative of the ones encountered in actual operation, e.g. by recording past problems. However, to check whether the learned policies can also be useful on problems larger than the training ones, we performed an additional series of experiments. We used the problem sets from IPC-2011, which is the last edition in which the five domains occurred together in the satisficing track. Initially, we conducted the experiments with the standard per-problem time limit of 30 minutes. The value of timeout T, used for computing the time score was adjusted accordingly to 1800s. However, the *Parking* and *Elevators* problems turned out to be very easy even for the plain GBFS(FF), which only failed one of 20 *Parking* problems. GBFS with random walks also failed one of the problems, but otherwise all of the *Parking* and *Elevators* problems were solved by all of the planners. In this situation, the learned planners were unable to match GBFS, which generally returned solutions of lower cost. This is expected given the added depth-bias of the other routines and therefore the strategies relying on them.

For this reason, we decided to reduce the time limit to one minute (again, T was adjusted to 60s). This setting was intended to reflect more closely the relative difficulty of the training setting, in which, as stated at the beginning of this section, only about half of the problems could be solved in time by GBFS with FF heuristic. Indeed, with one minute timeout GBFS solved 6 of the *Parking* problems and 12 of the *Elevators* problems. The scores are reported in Table 3. The strategies trained with IPC and IPC^2 rewards remain competitive, although their advantage over the baselines is not as large as on problems generated from the training distributions. The strategies learned with the time-based reward generalize worse, but on aggregate remain ahead of the baselines in terms of the obtained time score.

7 Conclusion and future work

In this paper, we introduced a planning framework capable of alternating between various search techniques while solving a problem. We modeled the problem of choosing the planning technique a reinforcement learning problem. Further, we provided an instantiation of the framework, defining a set of compatible search routines and a high-level representation of the planner's state. The resulting system was trained and evaluated on five planning domains and with three different performance measures. The experimental results show that the learned strategies obtain good performance on all five domains. Furthermore, despite being trained on relatively small problems, the strategies were also useful on larger problems, provided that the time interplay between problem size and time constraint roughly matched the training setting.

Our future work will investigate more complex representations of the planner's state and ways to improve practical sample efficiency of the learner. Another research direction is to extend the set of routines available to the learner. Possible additions include local search guided by other heuristic functions, width-based search (Lipovetzky and Geffner 2012), and stochastic rollouts driven by preferred operators.



Transport, IPC reward, *near early*

Transport, IPC reward, *near late*

Parking, IPC reward, *near early*

Parking, time reward, *near late*

Elevators, IPC reward, *near early*

No-mystery, time reward, *near late*

Figure 1: Policies for selected states and reward functions, changing with the number of training episodes. The policy for *near early* state in *Transport* domain under the IPC reward (top left) chooses GBFS nearly deterministically. DFS is discarded particularly early because in *Transport* it leads to plans of very high cost. For the *near late* state (top right), the learner oscillates between greedy search with random walks and local search, before committing to the latter. The policy for *near early* state in *Parking* domain under the IPC reward (middle left) quickly commits to DFS, which is typical for the *Parking* domain. However, for *near late* state and the time reward (middle right), the policy remains close to uniform despite preference for DFS. This is because the state is rarely visited, providing few samples. For *near early* state in *Elevators* domain under IPC reward (bottom left) the learner does not find a stable policy and oscillates between local and ϵ-greedy search, which both seem reasonable choices. For *near late* state in *No-mystery* domain under time reward (bottom right), the policy remains nondeterministic after 48h of training, choosing either of the preferred routines with similar probability. GBFS and ϵ-greedy search are generally preferred over more aggressive routines in the *No-mystery* domain, which was designed to enforce nearly-optimal solutions.

References

Cenamor, I.; De La Rosa, T.; and Fernández, F. 2016. The IBaCoP Planning System: Instance-Based Configured Portfolios. *Journal of Artificial Intelligence Research* 56:657–691.

Coles, A., and Smith, A. 2007. Marvin: A Heuristic Search Planner with Online Macro-Action Learning. *Journal of Artificial Intelligence Research* 28:119–156.

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Domain-Specific Configuration using Fast Downward. *ICAPS 2011 Workshop on Planning and Learning* 28–35.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* 3(1972):251–288.

Garrett, C. R.; Kaelbling, L. P.; and Lozano-Perez, T. 2016. Learning to Rank for Synthesizing Planning Heuristics. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 3089–3095.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An Automatically Configurable Portfolio-based Planner with Macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A Baseline for Building Planner Portfolios. *ICAPS 2011 Workshop on Planning and Learning* 28–35.

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

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:263–312.

Howe, A. E.; Dahlman, E.; Hansen, C.; Scheetz, M.; and Mayrhauser, A. v. 2000. Exploiting Competitive Planner Performance. In *Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning*, ECP '99, 62–72. Berlin, Heidelberg: Springer-Verlag.

Jimenez, S.; De La Rosa, T.; Fernandez, F.; and Borrajo, D. 2012. A Review of Machine Learning for Automated Planning. *The Knowledge Engineering Review* 27(4):433–467.

Leckie, C., and Zukerman, I. 1998. Inductive learning of search control rules for planning. *Artificial Intelligence* 101:63–98.

Lipovetzky, N., and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*, 540–545.

Lipovetzky, N., and Geffner, H. 2017. Best-First Width Search: Exploration and Exploitation in Classical Planning. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. 3590–3596.

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

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press.

Valenzano, R.; Sturtevant, N. R.; Schaeffer, J.; and Xie, F. 2014. A Comparison of Knowledge-Based GBFS Enhancements and Knowledge-Free Exploration. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Vallati, M.; Chrpa, L.; Grzes, M.; McCluskey, T. L.; Roberts, M.; and Sanner, S. 2015. The 2014 International Planning Competition: Progress and Trends. *AI Magazine* 36(3):90–98.

Virseda, J.; Borrajo, D.; and Alcazar, V. 2013. Learning heuristic functions for cost-based planning. *Preprints of the ICAPS'13 PAL Workshop on Planning and Learning* 6–13.

Williams, R. J. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8:229–256.

Xie, F.; Müller, M.; and Holte, R. 2014. Adding Local Exploration to Greedy Best-First Search in Satisficing Planning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27-31, 2014, Québec City, Québec, Canada.*, 2388–2394.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning Control Knowledge for Forward Search Planning. *The Journal of Machine Learning Research* 9:683–718.