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 distribution 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. (Helmer 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 un-related algorithms (e.g. (Howe et al. 2000; Gerevini, Saetti, and Vallati 2009; Fawcett et al. 2011; Helmer, Röger, and Karpus 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 $(V_p, O, s_0, g)$, 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 $(\text{pre}(o), \text{eff}(o))$, where pre(o) and 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 $\text{pre}(o) \subseteq s$. 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 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 $(S, A, p(s, a, s'), r(s, a), \gamma)$, 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 $\gamma$ 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 $\theta$ of policy $\pi$ 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 $\theta$:

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

where $\alpha$ 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 $\text{queue}$ (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(s₀, g, O, A, tᵣ)
    queue ← [s₀]
    while true do
        routine ← choose(A)
        t_start ← now()
        while now() − t_start < tᵣ do
            result ← routine(queue)
            if result ≠ in-progress(queue) then
                return result
            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 $V_i$. In Section 5 we introduce particular instantiations of $A$ and $V_i$. 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 $\theta$, 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:

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

where $\alpha$ 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.

$\epsilon$-greedy search  $\epsilon$-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 $\epsilon$ 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 set the 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 $h_{\text{best}}$ 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 $t_{\text{elapsed}}$ is the time elapsed from the beginning of the search and $t_{\text{max}}$ 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:

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

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 $r_{\text{ipc}}$, we cap the reward at 2 to prevent excessive premium for performance strictly better than that of reference planners. Formally:

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

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:

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

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.

\(^1\)http://icaps-conference.org/ipc2008/deterministic/
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.
<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>P</th>
<th>E</th>
<th>N</th>
<th>F</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBFS</td>
<td>29.46</td>
<td>15.57</td>
<td>28.73</td>
<td>24</td>
<td>28.27</td>
<td>126.03</td>
</tr>
<tr>
<td>c-gr.</td>
<td>27.23</td>
<td>10.88</td>
<td>21.87</td>
<td>30.64</td>
<td>30.33</td>
<td>140.95</td>
</tr>
<tr>
<td>RW</td>
<td>30.92</td>
<td>9.79</td>
<td>31.82</td>
<td>19.48</td>
<td>4.91</td>
<td>96.92</td>
</tr>
<tr>
<td>Local</td>
<td>37.18</td>
<td>21.89</td>
<td>32.87</td>
<td>20</td>
<td>23.9</td>
<td>135.84</td>
</tr>
<tr>
<td>DFS</td>
<td>12.95</td>
<td>37.45</td>
<td>16.92</td>
<td>7.7</td>
<td>0.13</td>
<td>75.15</td>
</tr>
<tr>
<td>Uni</td>
<td>24.07</td>
<td>25.71</td>
<td>26.79</td>
<td>20.75</td>
<td>32.44</td>
<td>129.76</td>
</tr>
<tr>
<td>L(I)</td>
<td>38.85</td>
<td>37.27</td>
<td>32.67</td>
<td>26.62</td>
<td>49.83</td>
<td>185.24</td>
</tr>
<tr>
<td>L(I)²</td>
<td>36.79</td>
<td>37.38</td>
<td>32.13</td>
<td>27.2</td>
<td>49</td>
<td>182.5</td>
</tr>
<tr>
<td>L(T)</td>
<td>30.58</td>
<td>36.56</td>
<td>19.54</td>
<td>26.37</td>
<td>46.84</td>
<td>159.89</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>P</th>
<th>E</th>
<th>N</th>
<th>F</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>GBFS</td>
<td>27.49</td>
<td>13.66</td>
<td>27.6</td>
<td>24</td>
<td>24.67</td>
<td>117.42</td>
</tr>
<tr>
<td>c-gr.</td>
<td>22.82</td>
<td>9.52</td>
<td>20.87</td>
<td>30.5</td>
<td>44.69</td>
<td>128.4</td>
</tr>
<tr>
<td>RW</td>
<td>19.26</td>
<td>8.41</td>
<td>24.27</td>
<td>19.38</td>
<td>3.95</td>
<td>75.27</td>
</tr>
<tr>
<td>Local</td>
<td>28.58</td>
<td>16.76</td>
<td>26.44</td>
<td>20</td>
<td>20.54</td>
<td>112.32</td>
</tr>
<tr>
<td>DFS</td>
<td>4.07</td>
<td>25.93</td>
<td>6.42</td>
<td>7.7</td>
<td>0.06</td>
<td>44.18</td>
</tr>
<tr>
<td>Uni</td>
<td>13.73</td>
<td>19.71</td>
<td>18.24</td>
<td>20.71</td>
<td>28.33</td>
<td>100.72</td>
</tr>
<tr>
<td>L(I)</td>
<td>30.7</td>
<td>26.18</td>
<td>25.13</td>
<td>26.55</td>
<td>44.28</td>
<td>152.84</td>
</tr>
<tr>
<td>L(I)²</td>
<td>30.55</td>
<td>26.51</td>
<td>24.57</td>
<td>27.1</td>
<td>43.59</td>
<td>152.32</td>
</tr>
<tr>
<td>L(T)</td>
<td>18.75</td>
<td>25.5</td>
<td>8.77</td>
<td>26.26</td>
<td>41.36</td>
<td>120.64</td>
</tr>
</tbody>
</table>

Table 2: IPC, IPC² 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, IPC² 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\(^1\).

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 (Valleti 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\(^2\). 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-

\[^1\]https://github.com/pgomoluch/fd-learn

\[^2\]http://www.cs.colostate.edu/~ipc2014/
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 as 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.
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


