Learning Neural Search Policies for Classical Planning

Paweł Gomoluch*, Dalal Alrajeh*, Alessandra Russo*, Antonio Bucchiarone†

*Imperial College London, UK; †Fondazione Bruno Kessler, Trento, Italy

Problem statement

Forward search planners typically rely on a single search procedure which remains fixed throughout the process of solving a problem.
In this work, we aim to:

- **Construct a planner which can adapt its search approach while solving a problem, based on the state of the search.**
- **Automatically learn search policies tailored to given problem distributions and performance objectives (such as the IPC score).**

Planning with parametrized search

We first introduce a parametrized search routine, which combines a number of search techniques including greedy and ϵ-greedy best-first search, local search and random walks. The character of the search depends on the values of the routine's parameters:

- ϵ – the probability of selecting a random node from the open list;
- S – the number of expansions without progress necessary to trigger a random walk;
- R – the number of random walks following a single node expansion;
- L – the length of a random walk;
- C – the number of node expansions in the global-local cycle;
- c – the proportion of local search in the global-local cycle.

The (simplified) algorithm below outlines a planner based on the parametrized search routine.

```
function PLANNER($s_0, g, O$)
    $global\_open \leftarrow [s_0]$
    while true do
        $\epsilon, S, R, L, C, c \leftarrow$ set_search_parameters()
        for $i = 1 \ldots (1-c) \cdot C$ do
            STEP($global\_open, O, g, R, L$)
        end for
        $local\_open \leftarrow$ [pop($global\_open$)]
        for $i = 1 \ldots c \cdot C$ do
            STEP($local\_open, O, g, R, L$)
        end for
        merge $local\_open$ into $global\_open$
    end while
end function

function STEP($open, O, g, R, L$)
    $s \leftarrow$ pop($open, \epsilon$)
    if $s \in g$ then
        $plan \leftarrow$ extract_plan($s$)
        return $plan$    ▷ return from PLANNER
    end if
    $successor\_states \leftarrow$ expand($s, O$)
    add($open, successor\_states$)
    if $expansions\_without\_progress > S$ then
        for $i = 1 \ldots R$ do
            $walk\_states \leftarrow$ random_walk($s, L$)
            add($open, walk\_states$)
        end for
    end if
    return in_progress    ▷ return to the PLANNER loop
end function
```

The parameters are set at every iteration, which allows for adaptation of the search approach.

The search interleaves between global and local expansions, with their numbers defined by the parameters.

Each node expansion can be followed by a number of random walks if the search is not progressing with respect to the lowest observed heuristic value [2].

References

[1] Shie Mannor, Reuven Rubinstein, and Yohai Gat. "The Cross Entropy Method for Fast Policy Search". In: *ICML*. 2003.

[2] Fan Xie, Martin Müller, and Robert Holte. "Adding Local Exploration to Greedy Best-First Search in Satisficing Planning". In: *AAAI*. 2014.

Representation of the planner's state

To capture the information about the state of the search, we consider features such as, among others:

- the heuristic value of the initial state $h(s_0)$;
- the lowest heuristic value encountered within the search $h_{\min}$;
- the time elapsed since the search started;
- the number of node expansions performed since the last change in the value of $h_{\min}$.

Learning search policies

Our neural search policies are feed-forward neural networks, mapping the representation of the planner's state to the values of the search routine's parameters. To learn policies best suited to a given problem distribution and performance objective, we employ the Cross-Entropy Method (CEM) [1] and introduce a normal distribution over the policy parameters $\mathcal{N}$.

We also evaluate a simpler approach, in which the search routine's parameters are optimized directly and remain fixed throughout the search, without regard to the current search state.

```
function TRAIN($\mathcal{P}, u, r, n, m$)
    initialize $\mu$ and $\Sigma$
    for $i = 1 \ldots u$ do
        $p_1 \ldots p_r \leftarrow \mathcal{P}$
        $\theta_1 \ldots \theta_n \leftarrow \mathcal{N}(\mu, \Sigma)$
        for $j = 1 \ldots n$ do
            for $k = 1 \ldots r$ do
                run policy $\theta_j$ on $p_k$, record plan cost $c_{j,k}$
            end for
        end for
        $G_1 \ldots G_n \leftarrow$ compute IPC score for $\theta_1 \ldots \theta_n$
        sort $\theta_1 \ldots \theta_n$ by scores $G_1 \ldots G_n$ (highest first)
        $\mu \leftarrow (1-\alpha)\mu + \alpha \cdot \text{mean}(\theta_1 \ldots \theta_m)$
        $\Sigma \leftarrow (1-\alpha)\Sigma + \alpha \cdot \text{covariance}(\theta_1 \ldots \theta_m)$
    end for
    return $\mu$
end function
```

At every iteration, r problems and n policies are sampled randomly. Vectors θ store the parameters of the neural network or the search routine's parameters directly.

The policies are evaluated in order to select the m best performing ones. The distribution is then updated to increase the likelihood of the best policies.

Results and future work

For evaluation, we train both the neural search policies (*NSP*) and the directly optimized variant (*Optimized*) for each domain separately and compare against planners based on each of the involved search techniques on its own, as well as a handcrafted, fixed combination of all the techniques (*Mixed*).

	Elevators	Floortile	No-mystery	Parking	Transport	Sum
GBFS	14.67	2.24	8.18	9.24	2.6	36.93
ϵ-GBFS	13.07	2.64	8.93	7.44	2.7	34.78
GBFS+RW	14.63	0.47	6.78	7.95	3.6	33.42
Local search	15.97	1.91	7.15	11.85	4.48	41.36
Combined	11.6	1.25	6.69	6.14	2.9	28.58
Optimized	14.64	**3.5**	8.86	**13.81**	**5.39**	46.18
NSP	**16.37**	3.28	**9.04**	12.93	5.12	**46.74**

The table includes the IPC scores obtained on held-out test sets of 20 problems of increasing difficulty (average over 10 randomly generated sets per domain). The trained approaches generally outperform the baselines. The benefit of the state-dependency is clear for the *Elevators* domain. However, there are also cases where the simpler variants performs better, despite being strictly less expressive. This effect is likely due to the fact that, with less parameters, optimization can better explore the policy space. Future work will aim to improve optimization for the more complex case. Other directions include further extensions of the parametrized search routine and more detailed representations of the planner's state.