A Reinforcement Learning approach for the Cross-Domain Heuristic Search Challenge

Luca Di Gaspero, Tommaso Urli

DIEGM, Università degli Studi di Udine
via delle Scienze 208 – I-33100, Udine, Italy
luca.digaspero@uniud.it, tommaso.urli@uniud.it

Abstract

The *International Cross-Domain Heuristic Search Challenge* (hereinafter *CHeSC 2011*) [3] is an ongoing competition that prompts for the design of a generally applicable high-level strategy for the automatic selection of problem-specific low-level heuristics across different problem domains. We participate in the challenge with a Reinforcement Learning approach. In this paper we describe the current state of the general algorithm by outlining our design choices and we present the preliminary results achieved by this approach.

1 Introduction

CHeSC 2011 aims at fostering the development of methods for the automatic design of heuristic search methodologies, which are generally applicable to multiple problem domains. The competition is open from August 2010 to June 2011 and it is organized by the ASAP group at the University of Nottingham (UK) together with other partners (see [3] for more details). The competition sits on the underlying framework of hyper-heuristics [2], i.e., automatic methods to generate or select heuristics for tackling hard combinatorial problems. The main goal behind hyper-heuristics is to develop methods that work well over several classes of problems, rather than just one or few. In order to ease the implementation of hyper-heuristics and to let the participant compete on a common ground, the competition organizers released a software framework, called HyFlex, to be used as a basis for their algorithms. HyFlex is a Java API that provides the basic functionalities for loading problem instances, generating new solutions and applying low-level heuristics to existing ones. Low-level heuristics are treated as black-boxes, and only information about their "*family*" is known (e.g., *mutations*, *local search moves*, *cross-overs*, ...). Furthermore, it is allowed to tune them through a single parameter ("*intensity of mutation*" or "*depth of search*", depending on the specific heuristic family). Finally, the use of HyFlex imposes the hyper-heuristic code to compel to a fixed structure, which allows the automatic evaluation and comparison of the competing algorithms. The four problem domains considered in the competition are: *Boolean Satisfiability* (MAX-SAT), *1-Dimensional Bin Packing*, *Permutation Flow Shop Scheduling*, and *Personnel Scheduling*. In addition two hidden domains will be introduced in the final phase of the competition and the algorithms will be tested on five benchmark instances across all six domains. For space limits, we refer the reader to the CHeSC 2011 website [3] for further details, including the scoring system and the sources of the benchmark instances.

2 Reinforcement Learning for Low Level Heuristic Selection

The family of algorithms we propose for CHeSC 2011 is based on Reinforcement Learning (RL) [4], an unsupervised machine learning approach inspired by the idea of an agent learning by trial and error according to a positive or negative environment response to its actions. In the simplest case the agent tries to learn how to select the optimal action (i.e., the one which yields the better result in the long run) without prior knowledge about the environment.

In order to describe a RL agent we have to instantiate the following elements: (i) an *environment*, i.e., a set of states (as perceived from the agent's perspective) and a set of available actions; (ii) a *reward*, i.e., an *a posteriori* measure about the goodness of executing an action in a specific state of the environment;

(iii) a *policy*, i.e., the (learned) mapping from a state to the action that should be taken in order to maximize the total expected reward gained by the agent on the long run; (iv) a *learning function*, i.e., the strategy for updating the agent's knowledge about the policy. For clarity reason we describe the reward first.

Reward. The basic reward r is computed as the solution's Δ_{cost} before and after action application. However, additional transformations can be applied to this value (see **Learning function**). Moreover, we recently adopted a reward measure (Δ_{cost}/t) which also takes into account the time spent to apply an action providing the concept of *reward over time*.

Environment. Given that problem domains are virtually hidden by the HyFlex framework, at each decision step the only information available about the environment is the objective function value of the current solution (i.e., not the solution itself). Since an agent's interface with the environment is ultimately the environment state as perceived by the agent itself, we must define an explicit mapping from objective function values to states. Unfortunately, these values have a different scale depending on the problem domain at hand and their range evolves during the search, moreover they can't summarize properly an agent's history (i.e., they are not Markov states) since they are strictly related to the current solution.

For the reasons described above, the way a **state** s is represented within the agent knowledge is non-trivial. At first, the choice fell on a simplistic state representation based on the sign of the reward in the last n steps (i.e. a vector of values in $\{-1, 0, +1\}$ to represent a deteriorating, stable or improving objective function). This choice raised a number of questions, e.g. how to choose n? how to retain quantitative information about the reward? Therefore, in the final stages before the end of the competition, we resorted to a more adaptive state representation capturing the concept of *relative recent reward* obtained by the agent, i.e. the reward trend. By *relative* we mean that the reward is divided by the recent average of low rewards (with respect to a reference reward) received by the agent. The intuitive meaning of this operation is to obtain a reward measure which is both problem-independent and adaptive wrt. the search stage. With *recent* we mean that past experience is discounted, hence newer information is trusted more than older one. Thus, given an action application, the new state is computed as $s_{i+1} = \lfloor (s_i + \alpha * (r_i - s_i)) \rfloor$, where α represents a step size value (i.e. the agent's reactivity to a new reward) and r_i is the *relative reward* obtained in the last step.

We defined a possible **action** a in a given state as the choice of the heuristic family to be used, plus an intensity (or depth of search) value in the quantized set of values $0.2, 0.4, 0.6, 0.8, 1.0$. Once the family has been determined, a random heuristic belonging to that family is chosen and applied to the current solution with the specified intensity (or depth of search). The action application yields a new solution and a reward. A special action performs a solution restart on an agent (without resetting its learned policy). Low-level heuristics belonging to the cross-over family require to operate on two solutions. For this reason we keep a number of independent agents, each one with its own solution, and use them for breeding when needed. In our current set-up we evolve simultaneously 4 agents. This number of agents was chosen empirically to provide, within the settings imposed by the competition, a good balance between intensification and diversification.

Policy. Moving to the policy, each state-action pair is assigned an *action value* π, which represent a degree of suitability of the action in the given state. We experimented with two classical reinforcement learning policies, i.e., *softmax* (using a Boltzmann distribution with an exponential cooling schedule) and *ϵ-greedy* action selection strategies as ways to balance the exploration-exploitation problem. The ϵ-greedy selection policy scored overall better and was chosen to participate in the competition.

Learning function. As for the learning function, we have used various techniques to update action values. In the simplest case, the action value for a specific state at the decision stage k is the discounted average of the rewards (i.e. Δ_{cost}) received so far, according to the following formula: $\pi(s,a)_{k+1} = \pi(s,a)_k + \gamma * (r_k - \pi(s,a)_k)$. The discount factor γ is needed to tackle non-stationary problems. The

second method is identical to the first one, except for the fact that the Δ_{cost} is compared with a reference reward ($r_k = (\Delta_{cost})_k - r^*_{k-1}$) to yield the reward, as it happens in reinforcement comparison methods [4]. The last learning function we have tried is SARSA [4], an on-line temporal-difference method in which the value of an action is also dependent on the stored value of the next action, thus realizing a sort of look-ahead strategy.

3 Preliminary Results

We have evaluated the different variants under various settings on the whole benchmark set of CHeSC 2011. The results are presented in Table 3, they represent the scores against the default competition hyper-heuristics. Because of space limits we report only the top-10 configurations, selected through a racing technique [1]. We also report the results of an uninformed random walk hyper-heuristic for comparison (last row).

State	n	α	γ	**Rew.**	**Policy**	ϵ	**Score**	**SAT**	**Bin P.**	**Pers. S.**	**Flow-s.**
Trend	n/a	0.1	0.1	Δ_{cost}	ϵ-greedy	0.10	241.00	96.00	64.00	53.50	27.50
Trend	n/a	0.1	0.9	Δ_{cost}	ϵ-greedy	0.01	240.00	98.00	73.00	57.00	12.00
Trend	n/a	0.1	0.1	Δ_{cost}/t	ϵ-greedy	0.10	237.75	92.25	65.00	48.00	32.50
Trend	n/a	0.1	0.9	Δ_{cost}/t	ϵ-greedy	0.10	237.50	65.50	71.00	61.00	40.00
Trend	n/a	0.1	0.1	Δ_{cost}	ϵ-greedy	0.01	236.00	98.00	72.00	48.00	18.00
Trend	n/a	0.1	0.1	Δ_{cost}/t	ϵ-greedy	0.10	235.08	91.58	66.00	35.50	42.00
Trend	n/a	0.1	0.1	Δ_{cost}	ϵ-greedy	0.10	232.00	96.00	64.00	39.00	33.00
Trend	n/a	0.9	0.9	Δ_{cost}/t	ϵ-greedy	0.01	232.00	96.00	76.00	41.50	18.50
Hist.	1	n/a	0.10	Δ_{cost}/t	ϵ-greedy	0.01	231.75	94.25	74.00	57.00	6.50
Hist.	0	n/a	0.10	Δ_{cost}	Softmax	n/a	231.25	93.25	67.00	44.50	26.50
Rand.	n/a	n/a	n/a	n/a	n/a	n/a	51.00	1.00	18.00	19.00	13.00

4 Conclusions and Future Work

This is an ongoing work which aims at investigating the use of learning-based approaches for combinatorial optimization. Although the results are only preliminary (at the time of writing our best algorithm scored 11^{th} in the competition *Leaderboard*), our experiments suggested that, if given enough time to learn, learning-based algorithms can be effective for solving combinatorial optimization problems. As a future work, we plan to investigate additional state/action representations, different machine learning techniques, and meta-learning techniques to improve the generality over different problem classes.

References

[1] M. Birattari, T. Stutzle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 11–18, 2002.

[2] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-Heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, chapter 16, pages 457–474. 2003.

[3] CHeSC Organising committee. CHeSC: cross-domain heuristic search competition, 2011.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, 1998.