HH-Evolver: A System for Domain-Specific, Hyper-Heuristic Evolution

Achiya Elyasaf
Ben-Gurion University of the Negev, Be'er Sheva, Israel
achiya.e@gmail.com

Moshe Sipper
Ben-Gurion University of the Negev, Be'er Sheva, Israel
sipper@cs.bgu.ac.il

ABSTRACT

We present HH-Evolver, a tool for domain-specific, hyper-heuristic evolution. HH-Evolver automates the design of domain-specific heuristics for planning domains. Hyper-heuristics generated by our tool can be used with combinatorial search algorithms such as A^* and IDA^* for solving problems of a given domain. HH-Evolver has a rich GUI that enables easy operation, including: running experiments in parallel, pausing and resuming experiments, and saving them and analyzing the results. Implementing new domains and heuristics with HH-Evolver is easily accomplished.
Workshop: EvoSoft

Categories and Subject Descriptors

D.1.5 [**Object-oriented Programming**]; I.2.1 [**Applications and Expert Systems**]: Games; I.2.6 [**Learning**]: Parameter learning; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

Keywords

Genetic Algorithms, Genetic Programming, Heuristic, Hyper-Heuristic, Single-Agent Search, HeuristicLab, HH-Evolver

1. INTRODUCTION

One of the main approaches for solving problems, and in particular NP-Complete problems, is using a heuristic search. Heuristic search algorithms are based on the notion of approximating the distance of a given *state* to the problem's solution (or *goal*). Such approximations are found by means of a computationally efficient function, known as a *heuristic function*. By applying such a function to states reachable from the current one considered, it becomes possible to select more-promising alternatives earlier in the search process, possibly reducing the amount of search effort (typically measured in number of nodes expanded) required to solve a given problem. The putative reduction is strongly tied to the quality of the heuristic function used: employing a perfect function means simply "strolling" onto the solution (i.e., no search de facto), while using a bad function could render the search less efficient than totally uninformed search, such as breadth-first search (BFS) or depth-first search (DFS).

A heuristic function is said to be *admissible* if it never overestimates the distance to the goal. Thus, the higher the heuristic value, the closer it is to the true distance to goal. Using an admissible heuristic with an optimal search algorithm (e.g. A* or iterative deepening A*, IDA*) guarantees that any solution found will be optimal, i.e., with minimal solution length. Another interesting property of heuristic functions is that the maximum of several heuristic functions is also admissible function as well. In our work we do not seek optimal solutions and therefore we will not need the admissibility property during the evolutionary process, however, we will use this property in order to initialize our training set.

Combining several heuristics to get a more accurate one is considered one of the most difficult problems in contemporary heuristics research [5, 31]. Of course, if all of the heuristic functions are admissible and an optimal solution is what we are looking for, then we could use the max heuristic (which takes the heuristic function with the maximal value). However, when we do not need to guarantee optimality or when we do not use only admissible heuristics, then there may well be better combinations.

In this paper we present HH-Evolver, the first evolutionary system for the automation of the design of domain-specific heuristics.

1.1 Hyper-Heuristics

Within combinatorial optimization, the term hyper-heuristics was first used in 2000 [9] to describe heuristics to choose heuristics. This definition of hyper-heuristics was expanded later [5] to refer to an automated methodology for selecting or generating heuristics to solve hard computational search problems. In the process of hyper-heuristics learning, heuristics are used as building blocks. These heuristics can be of high level, usually complex and memory-consuming (e.g., landmarks and pattern databases), or even low-level heuristics that are usually intuitive and straightforward to implement and compute.

According to Burke et al [8] and Cowling et al [9] the idea of hyper-heuristics is also referred to as the problem domain barrier, between the low-level heuristics and the hyper-

Draft

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.
Copyright 2013 ACM 978-1-4503-1964-5/13/07 ...$15.00.

heuristics. The barrier signifies that the hyper-heuristic should be the same for any domain encountered while the heuristics that comprise the hyper-heuristic will be domain specific. We disagree with this view as elaborated in Section 1.2.2.

Hyper-heuristics have been applied in many research fields, among them:

- Classical planning [13, 19, 22, 34].
- Classical NP-Complete domains, e.g., 2D and 3D bin-packing [6, 7], personnel scheduling [8, 9], traveling salesman [27], and vehicle routing [16, 29].
- Classical AI domains and puzzles, e.g., The Rush-Hour puzzle [17, 18], the game of FreeCell [10, 11, 32], and the Tile-Puzzle [2, 12].

The growing research interest in techniques for automating the design of heuristic search methods motivates the search for automatic systems for generating hyper-heuristics.

1.2 Hyper-Heuristic Systems

There are several hyper-heuristic systems in the literature.

1.2.1 Classical Planning Systems

A *planner* receives a domain description and a domain problem (or an instance), and outputs a solution (or a *plan*) for the given instance. The input is usually given in the *Planning Domain Definition Language* (PDDL) [23]. The planning community has created domain-independent planners since 1971 [15]. Traditionally the planners' heuristics were domain-independent as they do not know a priori the domain or the problem. However, many of the *International Planning Competitions* (IPC) had a special track for domain-independent planners that are capable of automatically making domain-specific adjustments. Some of these planners generate hyper-heuristics adjusted for the given domain. Yoon et al. [34] improved the *FF-Planner*'s heuristic [19] by extracting domain features from the FF's relaxed plan. These features were then used for learning the distance from the FF-Planner's heuristic to H^*. Others automatically learned the appropriate heuristic or heuristic combination for the input domain [13, 21, 22].

Since the domain is not known a priori, in all of the cases the heuristics used as building blocks were domain-independent heuristics. Thus, important domain-specific knowledge was being discarded.

1.2.2 Other Systems

Ochoa et al [26] proposed the *HyFlex* Java system for the development of cross-domain search methodologies and as a benchmark tool for research in hyper-heuristics and adaptive/autonomous heuristic search. The first *Cross-domain Heuristic Search Challenge* (CHeSC 2011) competition [4] used HyFlex-implemented domains and low-level heuristics as a benchmark for the competition. The challenge of the competition was to design a high-level search strategy that controls a set of problem-specific, low-level heuristics. The set of low-level heuristics was different for each problem domain, but the generated high-level strategy that controls the heuristics was to remain the same. Thus, domain-specific heuristics from several different domains were used to create domain-independent solvers. It should be noted that HyFlex was used as a benchmark and as an evaluation function for the hyper-heuristics generation phase, and it did not generate hyper-heuristics.

The approaches of HyFlex and CHeSC emanate from the problem domain-barrier view described in Section 1.1. As noted before, we find this approach to be problematic for a number of reasons:

1. The current definition of hyper-heuristic conflates two different definitions: The main definition—"an automated methodology for selecting or generating heuristics"; and a secondary definition—"the problem domain barrier"—which narrows the main definition to a sub-field of hyper-heuristics. A hyper-heuristic can be domain independent (as in Section 1.2.1) and it can be domain specific (as in [11]), while the "problem domain barrier" definition refers to domain independence alone. We argue that domain independence is not required where hyper-heuristics are concerned.
2. Assume the existence of n domains along with low-level heuristics for them. It might make sense to attempt to generate one hyper-heuristic, H, that will solve problems of any of these domains reasonably well. In this case we would say that H is domain specific to these n domains. We fail to see how H can apply successfully to domains other than the original n. H is doomed to under-perform in comparison with a domain-independent planner, since the latter will always apply strong and domain-independent heuristics on new domains instead of applying irrelevant, low-level heuristics that H will apply on problems of these domains.

   There is one exception to this argument: if H has some online learning mechanism to learn new domains on-the-fly it can considered as domain independent.
3. Checking the claims of the performance of the "domain-barrier" hyper-heuristics on hidden domains should have been an easy task. It can be achieved by comparing the performance of the winning CHeSC 2011 hyper-heuristics with the performance of domain-independent planners on hidden domains. However, here we face another problem: the number of hidden domains in CHeSC was 2, with the number of problem instances being 5. This is simply too small a test suite. It is also different than suites commonly used in the literature.
4. If we seek a domain-independent solver then, as presented in Section 1.2.1, it must be based on domain-independent heuristics only, and it must use a standard domain-independent language or system (e.g., PDDL, XCSP [30], JSR-331 [14]). An additional learning process can then be performed automatically to adapt the hyper-heuristic to the new domain encountered. This way the system will be truly domain independent. Currently, the community of hyper-heuristic researchers focuses on *Constraint Satisfaction Programming* (CSP) domains, though they do not use the aforementioned standards.

Our above arguments notwithstanding, any of the CHeSC 2011 competitors (competition results [3], and the best solver [24]) can be considered as a hyper-heuristic generation system when used together with the HyFlex system.

It should be noted that all of the CHeSC entries generate hyper-heuristics for local search (i.e., they all improve existing solutions rather than finding novel ones). HH-Evolver is the first evolutionary system for designing domain-specific hyper-heuristics for classical planning domains, to be used with combinatorial search algorithms such as A^* and IDA^*.

1.3 HeuristicLab

HeuristicLab [33] is a GUI framework for heuristic and evolutionary algorithms. HeuristicLab provides a feature-rich software environment for heuristic optimization researchers and practitioners. It is based on a generic and flexible model layer and offers a graphical algorithm designer that enables the user to create, apply, and analyze heuristic optimization methods. A powerful experimenter allows HeuristicLab users to design and perform parameter tests even in parallel. The results of these tests can be stored and analyzed easily in several configurable charts. HeuristicLab is available under the GPL license.

HH-Evolver is built as a plug-in for HeuristicLab. HH-Evolver supplies hyper-heuristics encodings as well as interfaces for adding new domains to a system. Thus, executing experiments and analyzing the results is an easy task.

2. HH-EVOLVER

HH-Evolver is a hyper-heuristic generator for search domains. The HH-Evolver system receives as input: a domain, several heuristics for the domain, and a dataset of domain instances to be used partly as training set and partly as test set. HH-Evolver generates a population of random hyper-heuristics and trains them over generations against the training set. When used with a heuristic search algorithm, the individuals are required to produce near-optimal solutions to the instances encountered. The search-size (i.e., the number of nodes encountered during the search) should be small as well.

In Section 2.1 we describe the theory and the algorithm behind the system and in Section 2.2 we explain the steps that need to be taken in order to run HH-Evolver. Finally, in Section 2.3 we lay the foundations for implementing new domains for HH-Evolver.

2.1 The HH-Evolver Design

2.1.1 The Hyper-Heuristic-Based Genome

HH-Evolver distinguishes between heuristic functions according to their return value:

1. *Real number*: Heuristics that return a real-number value that represents an estimation of the distance of the current state to the goal. It can also estimate the difficulty or the complexity of the state (a larger value means a more difficult state).

2. *Boolean*: Heuristics that return a boolean value signifying whether a property of the domain exists for a given state.

The task of combining several heuristics typically involves solving two major sub-problems:

1. How to combine heuristics by *arithmetic* means, e.g., by summing their values or taking the maximal value.
2. Finding exact conditions (i.e., *logic* functions) regarding *when* to apply each heuristic, or combinations thereof—some heuristics may be more suitable than others when dealing with specific search states.

In order to properly solve these sub-problems, we designed three different genomic representations, all of which are thoroughly described in [11]. All of these representations use the heuristics given as input to HH-Evolver.

Standard GA. This type of hyper-heuristic only addresses the first problem of how to combine heuristics by arithmetic means. Each individual comprises n_r real values in the range $[0, 1]$, representing a linear combination of all n_r domain real-valued heuristics. Specifically, the heuristic value, H, designated by an evolving individual, is defined as $H = \sum_{i=1}^{n_r} w_i h_i$, where w_i is the ith weight specified by the genome, and h_i is the ith heuristic of the domain real-value heuristics.

GP. As we want to embody both combinations of estimates and application conditions we use standard GP-trees as described in [20]. The individual implementation extends the HeuristicLab `SymbolicExpressionTree` class and therefore it includes the `SymbolicExpressionTree` functions and terminals except for the `Variables Symbols` and the `Time Series Symbols`. In addition it includes the domain real-value and boolean heuristics.

Policies. The last genome also combines estimates and application conditions, using ordered sets of control rules, or *policies*. Policies have been evolved successfully with GP to solve search problems (e.g., [1] and [11, 18]).

The structure of our policies is the same as the one in [11]:

$RULE_1$: IF $Condition_1$ THEN $Value_1$
.
.
.
$RULE_N$: IF $Condition_N$ THEN $Value_N$
$DEFAULT$: $Value_{N+1}$

where $Condition_i$ and $Value_i$ represent conditions and estimates, respectively.

Policies are used by the search algorithm in the following manner: The rules are ordered such that we apply the first rule that "fires" (meaning its condition is true for the current state being evaluated), returning its *Value* part. If no rule fires, the value is taken from the last (default) rule: $Value_{N+1}$.

Thus, with N being the number of rules used, each individual in the evolving population contains N *Condition* GP trees and $N + 1$ *Value* sets of weights used for computing linear combinations of heuristic values. The number of rules is a user parameter, though our experience with different domains has shown that 5 rules suffice.

For *Condition* we used the GP trees described above, and for *Value* we used the Standard GA also described above.

2.1.2 Genetic Operators

HeuristicLab differentiates between the algorithm used (i.e., the order of the genetic operators used) and the individual representation (i.e., GP-tree or GA). HH-Evolver uses the HeuristicLab standard genetic algorithm for the algorithmic part and the individuals' representations described in Section 2.1.1.

Since standard GA individuals extend the HeuristicLab

`RealVector` representation, any of the `RealVector` crossover or mutator operators can be used. GP individuals extend the HeuristicLab `SymbolicExpressionTree` and thus any of the `SymbolicExpressionTree` crossover or mutator operators can be used.

For policies, there are 6 operators for crossover and mutation, all described in [11].

For both GP-trees and policies, crossover can be performed between nodes of the same type (using *Strongly Typed Genetic Programming* [25]).

2.1.3 Training and Test Sets

The `DomainData` parameter, which is defined in the domain, defines a dataset of domain instances to be used and the size of the training and test sets. The dataset can be written in code and compiled or it can be imported as a csv (comma-separated values) file. The file is in the following format:

Dataset Name
Dataset Description
$Instance_1, h^*_1, h_{1,1}, \ldots, h_{1,n}$
$\vdots$
$Instance_m, h^*_m, h_{m,1}, \ldots, h_{m,n}$

where $Instance_i$ is the representation of instance i; $h_{i,j}$ is the value of heuristic j when applied to instance i; and h^*_i is the true distance of instance i to the solution. If the true distance is not known an estimation can be used (e.g., the maximum of all admissible heuristics).

HH-Evolver includes an automatic dataset generator.

2.1.4 Fitness

HH-Evolver currently includes the following fitness evaluators:

1. Least-square evaluator.
2. Complete heuristic-search evaluator.
3. Multi-Evaluator, which includes the previous ones.

The least-square evaluator calculates the least-square distance from the hyper-heuristic value to the h^* value for the training set instances. The lower the distance, the better the hyper-heuristic.

The complete heuristic-search evaluator tries to solve $K > 1$ training set instances with a time limit of S seconds for each instance (K and S are user parameters). The individual's fitness then equals:

$$f_s = \frac{1}{K}\sum_{i=1}^{K}(w_L * SolutionLength + w_S * SearchSize) + w_{\#}(1 - \frac{\#SolvedInstances}{K}).$$

Where w_L, w_S and $w_{\#}$ are the components' weights in the final score.

Finally, the multi-evaluator executes a different evaluator according to a given probability.

There is a known tradeoff between the quality of the solution (i.e., the solution length) and the amount of search effort (i.e., the search size) [28]. This tradeoff is given as a parameter for the user to define. The least-square evaluator pushes towards a more-precise heuristic function that will lead to a larger search size. The complete heuristic search evaluator includes parameters to control this tradeoff more precisely. Since running a complete search for all individuals may be too long, we suggest running the least-square evaluator more often than the complete search evaluator.

2.2 Executing HH-Evolver

HH-Evolver source and compiled dll files can be found at `http://achiya.elyasaf.net/research/HH-Evolver`. The dll files should be placed inside the HeuristicLab folder. Once HeuristicLab is running, a new hyper-heuristic genetic algorithm should be created with a hyper-heuristic problem loaded. The following parameters should be initialized before starting the evolutionary process:

1. The `Domain` parameter, with one of the existing domains (currently, the *15-Puzzle* and the game of *FreeCell*).
2. The `Dataset` parameter, which is defined in the `DomainData` parameter, inside the `Domain` parameter (see Figure 1).
3. The search algorithm, with one of the existing heuristic search algorithms (currently, *Weighted* A^* and *WeightedIDA**).

Figure 1 shows a screenshot of HeuristicLab loaded with a hyper-heuristic GP-tree generator for the game of FreeCell. The reader is welcome to test this setting and learn from it by downloading and opening the saved configuration file at `http://achiya.elyasaf.net/research/HH-Evolver`.

2.3 Implementing New Domains for HH-Evolver

A domain definition comprises information regarding: 1) How to represent a state in the domain; 2)how to get the neighbors of a domain state; and 3) the domain heuristics. Thus, there are four classes that must be inherited in order to implement a new domain for HH-Evolver (depicted in Figures 2 and 3):

- `DomainState`: Represents a state in the domain.
- `Domain`: Represents the domain. This class contains all information regarding the domain (e.g., possible heuristics, possible actions).
- `Action`: Represents an action in the domain. An action transforms one domain state to another (e.g., selecting a route in the vehicle routing problem).
- `RealValueHeuristic` / `ConditionalHeuristic`: Represents a domain heuristic that returns a real-value number or a boolean value.

The `HyperHeuristicDomains` project contains full examples for implementing new domains.

2.3.1 Inheriting the DomainState Class

The `DomainState` class represents a state in the domain. Any class that inherits this class must implement the following functions for the heuristic search algorithm to work:

- **bool isGoal()**: Returns true if the state is a goal state and false otherwise.
- **bool Equals(object obj)**.
- **int GetHashCode()**: A hash code for states. The hash code should be a bijection, though it is not an obligation.

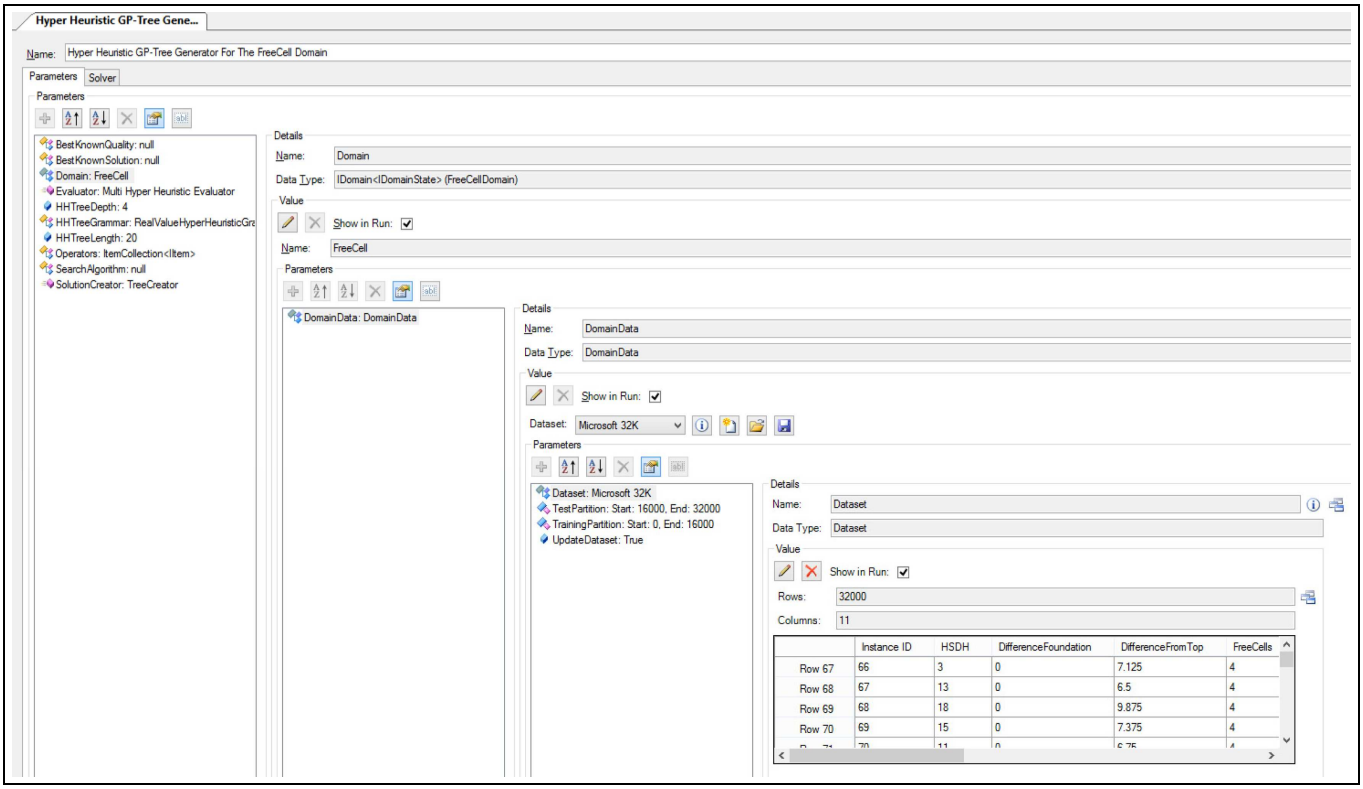

Figure 1: HeuristicLab loaded with a `Hyper-Heuristic GP-tree Creator` for the game of FreeCell

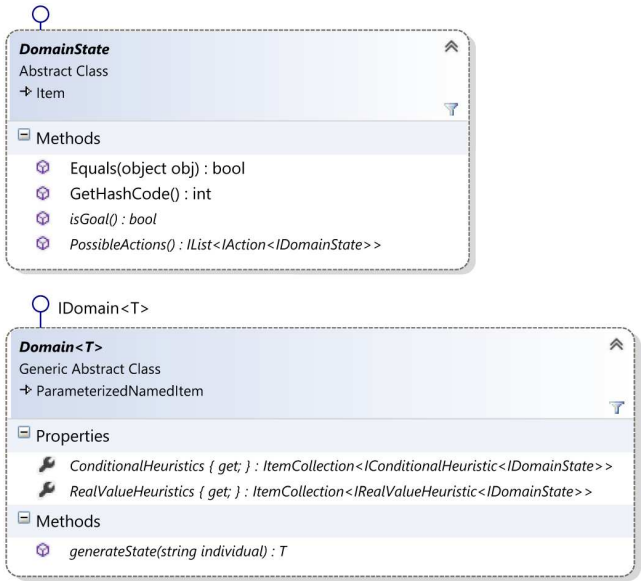

Figure 2: `Domain` and `DomainState` abstract classes.

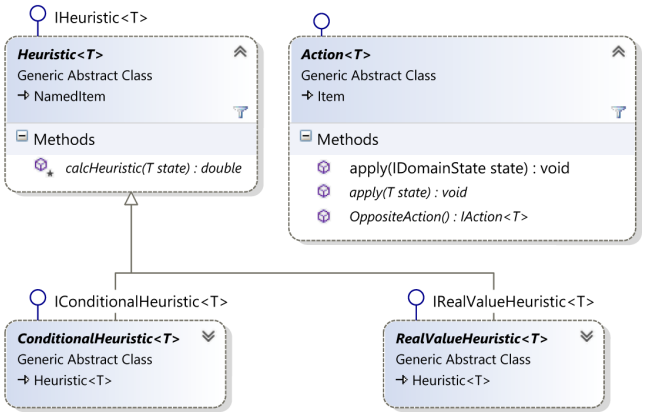

Figure 3: `Action` and `Heuristics` abstract classes.

2.3.2 Inheriting the Domain Class

The `Domain` class holds two heuristics lists—one for `RealValueHeuristics` and one for `ConditionalHeuristics`. Inheriting the `Domain` class requires implementing these getters as well as a domain state generator function. The function receives an encoding of a domain state and returns an instance of a `DomainState`.

2.3.3 Inheriting the Action Class

The `Action` class represents an action that takes one `DomainState` and transforms it to another. The following functions need to be implemented when inheriting the class:

- **void apply(T state)**: Applies the action to a state.
- **void OppositeAction()**: Returns the opposite action of the current action. The opposite action is required for backtracking during the search process.
- **void applyOppositeAction(T state)**: Applies the opposite action to a state.

2.3.4 Inheriting the RealValueHeuristic and the ConditionalHeuristic Classes

Any heuristic function should be implemented as a class that inherits the `RealValueHeuristic` or the `ConditionalHeuristic` classes. These abstract classes require implementing the `double calcHeuristic(T state)`. The function receives a `DomainState` instance and returns the heuristic value for that state. In case the heuristic class inherits the `ConditionalHeuristic` class, it should return a positive value if the heuristic value for the given state is true, and a negative value otherwise.

3. DISCUSSION AND FUTURE WORK

We presented HH-Evolver, the first evolutionary system for designing domain-specific hyper-heuristics for classical planning domains. These hyper-heuristics can be used by combinatorial search algorithms such as A^* and IDA^* for solving domain problems.

HH-Evolver has a rich GUI that enables easy operation, including: running experiments in parallel, pausing and resuming experiments, saving them and analyzing the results. Implementing new domains and heuristics with HH-Evolver is done by inheriting four classes.

There are several possible extensions to our work, including:

1. Add classical planning domains to HH-Evolver (i.e., domains from the International Planning Competitions (IPC)).
2. Publish HH-Evolver results for experiments in these domains.
3. Extend HH-Evolver to support local-search, constraint-satisfaction domains such as bin packing, personnel scheduling, traveling salesman, and vehicle routing.
4. Extend HH-Evolver to the field of domain-independent planning.

We wish to end by encouraging the reader to experience HH-Evolver firsthand by pointing their browser to `http://achiya.elyasaf.net/research/HH-Evolver`.

Acknowledgments

Achiya Elyasaf is partially supported by the Lynn and William Frankel Center for Computer Sciences. This research was supported by the Israel Science Foundation (grant no. 123/11).

4. REFERENCES

[1] R. Aler, D. Borrajo, and P. Isasi. Using genetic programming to learn and improve knowledge. *Artificial Intelligence*, 141(1–2):29–56, 2002.

[2] S. J. Arfaee, S. Zilles, and R. C. Holte. Bootstrap learning of heuristic functions. In *Proceedings of the 3rd International Symposium on Combinatorial Search (SoCS2010)*, pages 52–59, 2010.

[3] Automated Scheduling, Optimisation and Planning (ASAP) research group. The first cross-domain heuristic search challenge results. `http://www.asap.cs.nott.ac.uk/external/chesc2011/results.html\#Detailed_Explanation`, 2011.

[4] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, B. McCollum, G. Ochoa, A. J. Parkes, and S. Petrovic. The cross-domain heuristic search challenge – an international research competition. In *Proceedings of the 5th international conference on Learning and Intelligent Optimization*, LION'05, pages 631–634, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In M. Gendreau and J. Potvin, editors, *Handbook of Meta-Heuristics 2nd Edition*, pages 449–468. Springer, 2010.

[6] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Trans. Evolutionary Computation*, 14(6):942–958, 2010.

[7] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20(1):63–89, 2012.

[8] E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *J. Heuristics*, 9(6):451–470, 2003.

[9] P. I. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. K. Burke and W. Erben, editors, *PATAT*, volume 2079 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2000.

[10] A. Elyasaf, A. Hauptman, and M. Sipper. GA-FreeCell: Evolving Solvers for the Game of FreeCell. In N. Krasnogor et al., editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1931–1938, Dublin, Ireland, 12-16 July 2011. ACM.

[11] A. Elyasaf, A. Hauptman, and M. Sipper. Evolutionary design of FreeCell solvers. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(4):270 –281, Dec. 2012.

[12] A. Elyasaf, Y. Zaritsky, A. Hauptman, and M. Sipper. Evolving solvers for FreeCell and the sliding-tile puzzle. In D. Borrajo, M. Likhachev, and C. L. López,

editors, *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SoCS 2011, Castell de Cardona, Barcelona, Spain, July 15-16, 2011*. AAAI Press, 15-16 July 2011.

[13] C. Fawcett, E. Karpas, M. Helmert, G. Roger, and H. Hoos. Fd-autotune: Domain-specific configuration using fast-downward. In *Proceedings of ICAPS-PAL 2011*, 2011.

[14] J. Feldman et al. JSR-331: Constraint programming API website. `http://jcp.org/en/jsr/summary?id=331`.

[15] R. E. Fikes and N. J. Nilsson. Strips: a new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd international joint conference on Artificial intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[16] P. Garrido and M. C. R. Rojas. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *J. Heuristics*, 16(6):795–834, 2010.

[17] A. Hauptman, A. Elyasaf, and M. Sipper. Evolving hyper heuristic-based solvers for Rush Hour and FreeCell. In A. Felner and N. R. Sturtevant, editors, *Proceedings of the 3rd Annual Symposium on Combinatorial Search, SoCS 2010, Stone Mountain, Atlanta, Georgia, USA, July 8-10, 2010*, pages 149–150. AAAI Press, 8-10 July 2010.

[18] A. Hauptman, A. Elyasaf, M. Sipper, and A. Karmon. GP-Rush: using genetic programming to evolve solvers for the Rush Hour puzzle. In G. Raidl, F. Rothlauf, G. Squillero, R. Drechsler, T. Stuetzle, M. Birattari, C. B. Congdon, M. Middendorf, C. Blum, C. Cotta, P. Bosman, J. Grahl, J. Knowles, D. Corne, H.-G. Beyer, K. Stanley, J. F. Miller, J. van Hemert, T. Lenaerts, M. Ebner, J. Bacardit, M. O'Neill, M. D. Penta, B. Doerr, T. Jansen, R. Poli, and E. Alba, editors, *GECCO'09: Proceedings of 11th Annual Conference on Genetic and Evolutionary Computation Conference*, pages 955–962, New York, NY, USA, 2009. ACM.

[19] J. Hoffmann and B. Nebel. The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.*, 14(1):253–302, May 2001.

[20] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[21] J. Levine and D. Humphreys. Learning action strategies for planning domains using genetic programming. In G. R. Raidl, J.-A. Meyer, M. Middendorf, S. Cagnoni, J. J. R. Cardalda, D. Corne, J. Gottlieb, A. Guillot, E. Hart, C. G. Johnson, and E. Marchiori, editors, *EvoWorkshops*, volume 2611 of *Lecture Notes in Computer Science*, pages 684–695. Springer, 2003.

[22] J. Levine, H. Westerberg, M. Galea, and D. Humphreys. Evolutionary-based learning of generalised policies for AI planning domains. In F. Rothlauf, editor, *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO 2009)*, pages 1195–1202, New York, NY, USA, 2009. ACM.

[23] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical report, New Haven, CT: Yale Center for Computational Vision and Control, July 1998.

[24] M. Misir, K. Verbeeck, P. D. Causmaecker, and G. V. Berghe. An intelligent hyper-heuristic framework for chesc 2011. In *Proceedings of the 6th international conference on Learning and Intelligent Optimization*, LION'12, pages 461–466, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[26] G. Ochoa, M. R. Hyde, T. Curtois, J. A. V. Rodríguez, J. D. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic, and E. K. Burke. Hyflex: A benchmark framework for cross-domain heuristic search. In J.-K. Hao and M. Middendorf, editors, *European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2012)*, LNCS, pages 136–147, Heidelberg, 2012. Springer.

[27] M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, 2005.

[28] J. Pearl. *Heuristics*. Addison–Wesley, Reading, Massachusetts, 1984.

[29] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & OR*, 34(8):2403–2435, 2007.

[30] O. Roussel and C. Lecoutre. XML representation of constraint networks: Format XCSP 2.1. *CoRR*, abs/0902.2362, feb 2009.

[31] M. Samadi, A. Felner, and J. Schaeffer. Learning from multiple heuristics. In D. Fox and C. P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 357–362. AAAI Press, 2008.

[32] M. Sipper. *Evolved to Win*. Lulu, 2011. available at `http://www.lulu.com/`.

[33] S. Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Johannes Kepler University, Linz, Austria, 2009.

[34] S. W. Yoon, A. Fern, and R. Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9:683–718, 2008.