Extended Abstract: The Heuristic Search Research Framework

Meir Goldenberg
The Jerusalem College of Technology
Israel
mgoldenbe@gmail.com

Abstract

This paper is an extended abstract of the recently published journal article (Goldenberg 2017b).

The *Heuristic Search Research Framework* is a software framework for implementing heuristic search algorithms and exploring their properties. The framework uses policy-based design to enable efficient, flexible and well-organized implementations. In addition, the framework provides effective tools for exploring the properties of the implemented algorithms. The extensive online documentation includes a video demo.

Introduction

Studying heuristic search algorithms involves serious challenges at all stages: implementation, exploration and evaluation of performance relative to competitors for a large number of problem instances and parameter settings. The *Heuristic Search Research Framework* aims to aid the researcher in surmounting these challenges. The framework is written in C++11 and uses policy-based design (Alexandrescu 2001). This paper is an extended abstract of the journal paper that introduced the framework (Goldenberg 2017b). The video tutorial and the extensive online documentation (Goldenberg 2017a; 2017c) enhance the original paper, which references the online documentation extensively.

The challenges

A researcher implementing heuristic search algorithms has to face the fact that an algorithm like A* (Hart, Nilsson, and Raphael 1968) may have hundreds of variants. Naive approaches to implementing these variants result in complicated code with many deeply nested conditional statements. This code will inevitably prove to be buggy and inefficient. Furthermore, the code blocks pertaining to various variants mingle together and are extremely difficult to locate and maintain despite all attempts at organizing them. All these qualities become much more pronounced with each new implemented variant, soon making adding a new variant too cumbersome.

Once an algorithm is implemented, the researcher can perform measurements, but is utterly limited in his ability to acquire insights into the algorithm's properties by means of direct observation of algorithm's behavior.

The framework

The framework addresses the challenge of the multiplicity of variants by promoting policy-based design (Alexandrescu 2001). Using this technique lets the researcher represent facets specific to a given algorithmic variant as *policies*, which can be implemented separately from the algorithm and each other. Compile-time techniques enable run-time performance comparable or even exceeding that of a hand-crafted implementation. This has been demonstrated by comparing the performance of a generic implementation of IDA* (Korf 1985) with that of the famous implementation of Korf for solving instances of the sliding-tile puzzle. As far as performance characteristics are concerned, this work may be seen as an extension of the work by Burns et al. (2012). In addition, each algorithmic variant is uniquely identified by the policies that comprise it. This may give rise to a unique way of systematizing the algorithms.

The framework provides tools for direct observation of algorithm's behavior, thereby enabling insights that can inspire theoretical derivations and complement the researcher's intuition and the collected statistics about solving sets of problem instances. In particular, the researcher can visualize an algorithm in action by defining visualizable algorithmic events and domain layouts. The latter ability is not limited to grid-based domains and is complemented by the automated layout feature. Needless to say, one can use the resulting visualizations as effective examples for conference and classroom presentations.

We refer the reader to the original paper and the online documentation (Goldenberg 2017b; 2017c) for the detailed description of the framework, examples of its use and a brief comparison with the related projects cited[1] in the following references section.

[1]Whenever the year of publication was not available, the year of the last commit was used instead.

References

Alexandrescu, A. 2001. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley Professional.

Burns, E. A.; Hatem, M.; Leighton, M. J.; and Ruml, W. 2012. Implementing fast heuristic search code. In *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*.

Ethan Burns. 2016. Research Code for Heuristic Search. https://github.com/eaburns/search.

Goldenberg, M. 2017a. A video demo of the Heuristic Search Research Framework. https://youtu.be/cElxLWve1Zw.

Goldenberg, M. 2017b. The heuristic search research framework. *Knowl.-Based Syst.* 129:1–3.

Goldenberg, M. 2017c. The online documentation of the Heuristic Search Research Framework. jct.ac.il/~mgoldenb.

Hart, P.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4:100–107.

Jeremy Siek; Lie-Quan Lee; and Andrew Lumsdaine. 2001. The Boost Graph Library. https://www.boost.org/doc/libs/1_67_0/libs/graph/doc/index.html.

Jordan Thayer. 2016. OCaml Search Code. https://github.com/jordanthayer/ocaml-search.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artif. Intell.* 27(1):97–109.

Malte Helmert. 2016. Fast Downward. http://www.fast-downward.org/.

Matthew Hatem. 2015. Combinatorial Search for Java. https://github.com/matthatem/cs4j.

Nathan Sturtevant. 2015. HOG2. https://github.com/nathansttt/hog2.

Robert Holte. 2014. PSVN. https://era.library.ualberta.ca/downloads/7m01bn08g.