CENTRO DE CIENCIAS EXATAS E TECNOLOGICAS - CCE

DEPARTAMENTO DE INFORMATICA

PROJECT:

ON SELECTING HEURISTIC FUNCTIONS FOR DOMAIN-INDEPENDENT PLANNING.

Marvin Abisrror Zarate
MSc Student in Computer Science

Levi Henrique Santana de Lelis
(Advisor)

Santiago Franco
(Co-Advisor)

VIÇOSA - MINAS GERAIS
JULY - 2015

Contents

1 Introduction

In the last few decades, Artificial Intelligence has made significant strides in domain-independent planning. Some of the progress has resulted from adopting the heuristic search approach to problem-solving, where use of an appropriate heuristic often means substantial reduction in the time needed to solve hard problems. Initially heuristics were hand-crafted. These heuristics required domain-specific expertise and often much trial-and-error effort.

Recently, techniques [Haslum et al., 2007; Edelkamp, 2007; Nissim et al., 2011] have been developed to automatically generate heuristics from domain and problem specifications. The component that generate heuristics have the name of *heuristic generators* [Barley et al., 2014]. Many of these *heuristic generators* work by creating abstractions of the original problem spaces. Usually there are a number of different ways to abstract the problem space, and the generators search for a good abstraction to create their heuristic. These searches can be guided by predictions about the impact that different choices will have on the size of the problem's search space [Haslum et al., 2007].

Solving domain-independent problems is a very hard task when we do planning. Almost everything is Pspace-complete or worse. For example the simple language of propositional STRIPS is Pspace-complete [Bylander, 1994]. Also, most benchmark problems for planning are **NP-**complete or even tractable [Helmert, 2006].

Sometimes we will find instances that takes too much time to find the solution, and the measure of that time could be hours, days or even years. Users of search algorithms usually do not know *a priori* how long it will take to solve a problem instance − some instances could be solved quickly while others could take long time.

In domain-independent planning one is required to solve a problem without a priori information about the problem. Moreover, the process of solving the problem is fully automated.

Domain-independent planners can potentially be applied in various areas such as manufacturing, space systems, software engineers, robotics, education, and entertainment. On the whole, an strategy which allows us to solve a larger number of problems using domain-independent planning is a very important task in AI.

The proposal of this project is to develop an approach of selecting heuristics functions from a large set of heuristics with the objective to optimize the search of the solution applied by the domain-independent planning.

2 Related Work

Search Algorithms, such as A* [Hart and Raphael, 1968], are fundamental in many AI applications. A* uses the $f(s) = g(s) + h(s)$ cost function to guide its search. Here, $g(s)$ is the cost of the path from the start state to state s, and $h(s)$ is the estimated cost-to-go from s to a goal; $h(.)$ is known as the heuristic function.

A heuristic h is *admissible* if $h(s) \leq dist(s, goal)$ for every state **s** and goal state *goal*, where *dist(n, m)* is the cost of at least-cost path from n to m. If $h(n)$ is admissible, *i.e.,* always returns a lower bound estimate of the optimal cost. Search algorithms that use an admissible heuristic guaranteed to find an optimal path from the start state to a goal state if one exists.

In addition to being admissible, the heuristic also could be *consistent*. A heuristic h is *consistent* if for every pair of states, m and n, $h(m) - h(n) \leq dist(m,n)$

There are different approaches for creating heuristics: abstractions (e.g., pattern databsses (PDBs) [Culberson and Schaeffer, 1996]), delete relaxations (e.g., h^{max} [Bonet and Geffner, 2001]), and landmarks(e.g., LM-cut [Helmert and Domshlak, 2009]).

Works related to PDBs shows that, on average, for a single PDB the larger the PDB (i.e., the less abstract the pattern), the smaller the search tree [Holte and Hernádvölgyi, 1999]. Another approach mention the possibility of combining patterns for smaller problems with the object of produce heuristics for larger problems [Culberson and Schaeffer, 1998], e.g., 8-puzzle and/or 15-puzzle to create a heuristic to work with 24-puzzle.

[Holte et al., 2004] looked at combining smaller PDBs for the same problem and they created experiments. The experiments they have done with the set of heuristics are maximing and random. Maximing is when evaluating all the individual heuristics (PDBs) and returning their maximum value and Random is when selecting PDBs at random from a set of PDBs. Maxing over many small PDBs often produce smaller search trees than a larger PDB with size (in number of entries) equal the sum of the sizes of the smaller PDBs. Holte found that in smaller search three, the search times can be larger. This because when maxing over multiple PDBs, the per node evaluation cost is the sum of the indiviaul PDB evaluation costs. [Zahavi et al., 2007] use ranndom combiner and showed that using a large set of PDBs can reduce the search tree size more than using max over a smaller subset of those PDBs.

The idea of not using all heuristics to evaluate every node has appeared several times recently *selmax* Domshlak et al. [2011], Lazy A* (LA*), and Rational Lazy A* (RLA*) [Tolpin et al., 2013]. These algorithms decide which of a set of heuristics should be used to evaluate a given node. They can be explained most easily as choosing between a cheap and less accurate heuristic, h1, and an accurate but expensive heuristic, h2. *selmax* chooses which to use by estimating whether more time will be saved by using h1 and possibly not expanding it. *selmax* learns online a classifier that predicts when it should use h2 at a node. *selmax* may expand more nodes than max but its overall problem solving time should be lower than either max or random.

Althought, the idea of use multi-heuristics and get a subset of heuristics that enhance the search is a good approach, in the work on maxing and randomizing over multiple Pattern Database (PDBs), we often encounter the problem that simply reducing the search tree does not necessarily decrease the search time. This is a form of the *utility* problem. In the 1980's there was work on learning search control rules for planners that has many parallels with the current work on multiple PDBs. The goal of both is to add new sources of knowledge to speedup the search for solutions. In the 1980's, learning methods automated the acquisition of search control rules. However, [Minton, 1990] found that sometimes adding more search control rules would reduce the search space but could also increase the search time. Minton's approach to this problem was to estimate the utility of a new search control rule and only add those rules whose benefits outweighed their costs.

Recent work on combining multiple PDBs has encountered the same problem. Adding another PDB to max often reduces the size of the search tree, but it increases the per node evaluation costs. Sometimes the benefits of having a smaller search tree are outweighed by the additional per node costs. Before adding another PDB to the combination, its utility must be estimated. Estimating a PDB's utility involves not only reasoning about its impact on search tree size but also per node evaluation costs. These two impacts must be evaluated in light of how they both affect search time.

The system most similar to our proposal of selecting heuristics is RIDA* [Barley et al., 2014]. RIDA* also selects a subset from a pool of heuristics to guide the A* search. In RIDA* this is done by starting with an empty subset and trying all combinations of size one before trying the combination of size two and so on. RIDA* stops after evaluating a fixed number of subsets. While RIDA* is able to evaluate a set of heuristics with tens of elements, we want that our approach could evaluate a set of heuristics with thousands of elements.

3 Problem

The idea we want to develop is based in two steps: First, we make selection of the better heuristics from a large set of heuristics. Then, we evaluate each node with the subset of heuristics from the large pool in order to find quickly an optimal path to a given problem. In this way, we assume that we can solve problems quickly and without spend too much time.

3.1 Formalizing the Problem

This project is concerned with cost-optimal state-space planning using the A* algorithm [Hart and Raphael, 1968]. We asume that a pool, ζ, of hundreds or even thousands of heuristics is available, and that the final heuristic we assume to guide A*, h_{max}, will be defined as the maximum over a subset ζ' of those heuristics ($h_{max(s,\zeta')} = \max_{h \in \zeta'} h(s)$). The choice of the subset ζ' can greatly affect the efficiency of A*. For a given size N and planning task $\triangledown$, a subset containing N heuristics from ζ is optimal if no other subset containing N heuristics from ζ results in A* expanding fewer nodes when solving $\triangledown$.

4 Objective

The general objective of this project is to develop an approach for selecting a subset of heuristic functions with the goal of reducing the running time of search algorithms emoploying these functions. The cardinality of the subset of heuristics found is other part of the research.

4.1 Specific Objectives

What we expect to achieve as a specific objectives are:

1. Develop a method to find a subset of heuristics ζ from a large pool that optimize in number of nodes expanded the process of search.

2. We are going to try different strategies to obtain the size N that fits the best the subset of heuristics.

3. We know that Stratification Sampling (SS) does not make even reasonable predictions for the number of nodes expanded by A*. Nevertheless, even though SS produces poor predictions for the number of nodes expanded by A*, we would like to verify whether these predictions can be helpful in selecting a subset of heuristics to guide the A* search.

5 Methodology

Heuristic functions estimate the distance-to-go from a given node to the goal. This information can be exploited by search algorithms to assess whether one state is more promising than the rest. Heuristics can help reducing the number of alternatives from an exponential number to a polynomial number.

The term search in planning is related to the process of finding solutions. Every search algorithm searches for the completation of a given task. The process of problem solving can often be modeled as a search in a state space starting from some given initial state with rules describing how transform one state into another. The rules have to be applied over and over again to eventually satisfy some goal condition. In the common case, we aim at the best of such paths, often in terms of path length or path cost. Search has been an important part of Artificial Intelligence since its very beginning, as the core technique for problem solving. Many important applications of search algorithms have emerged since then, including ones in action and route planning, robotics, software and hadware, theorem proving and computational biology.

The problems resolved by the domain-independent planning are the standard benchmark for classical planning systems. And the planner we use to resolve those domains problems is the planning system Fast-Downward [Helmert, 2006]. Fast-Downward is a classical planning system based on heuristic search. It can deal with general deterministic planning problems encoded in the propositional fragment of PDDL 2.2. We will use Fast-Downward as our planning system to develop our new approach and test it in the benchmarks domains it contains.

During the research we want to develop a method using optimization for the selection of a subset of heuristics from a large loop of hundreds or even thousand of heuristics and to do this we will study further the state of the art in Maximazing submodular set functions [Nemhauser et al., 1978].

6 Budget

Table 1: Total Cost of the Master

Cost Specification	Description	Cost ($R)	Source of Income
1.- Personal	Stipend to conduct the research (Scholarship amount X 24)	36,000	CAPES
Subtotal 1		**36,000**	
2.- Bibliographic Material	Books, Technical Journal, etc	400	Own Resources
Subtotal 2		**400**	
3.- Materials	Sheets of papers	50	Own Resources
	Printing	100	Own Resources
	USB	20	Own Resources
Subtotal 3		**170**	
Total = Subtotal 1 + Subtotal 2 + Subtotal 3		**36,570**	

7 Timetable

Table 2: Schedule from June to March

Activities	June	July	August	September	October	November	December	January	February	March
Present the project to the PPG	✓									
Implementing the *heuristic generator*	✓									
Implementing the prediction strategy A* or SS		✓								
Test which strategy of prediction fits the best the subset of heuristics		✓	✓							
Testing with the domain-independent problems of Fast-Downward			✓	✓						
Analyze the results and prepare a paper to submit				✓						
Organize and Analyze the results of SS compared with IDA* and Selection of heuristics functions					✓	✓				
Writing the theses						✓	✓	✓	✓	
Dissertation theses										✓

References

Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. *AAAI*, 7:1007–1012, 2007.

Stefan Edelkamp. Automated creation of pattern database search heuristics. In *Model Checking and Artificial Intelligence*, pages 35–50. Springer Berlin Heidelberg, 2007.

Raz Nissim, Jörg Hoffmann, and Malte Helmert. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, 2011.

Michael W. Barley, Santiago Franco, and Patricia J. Riddle. Overcoming the utility problem in heuristic generation: Why time matters. *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*, 2014.

Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1):165–204, 1994.

Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26:191–246, 2006.

N. J.; Hart, P. E.; Nilsson and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC*, 4(2):100–107, 1968.

Joseph C Culberson and Jonathan Schaeffer. Searching with pattern databases. In *Advances in Artifical Intelligence*, pages 402–416. Springer Berlin Heidelberg, 1996.

Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33, 2001.

Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: what's the difference anyway? In *ICAPS*, pages 162–169, 2009.

Robert C Holte and István T Hernádvölgyi. A space-time tradeoff for memory-based heuristics. In *AAAI/IAAI*, pages 704–709, 1999.

Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3): 318–334, 1998.

Robert C Holte, Jack Newton, Ariel Felner, Ram Meshulam, and David Furcy. Multiple pattern databases. In *ICAPS*, pages 122–131, 2004.

Uzi Zahavi, Ariel Felner, Jonathan Schaeffer, and Nathan R Sturtevant. Inconsistent heuristics. *Proceedings of the national conference on artificial intelligence*, 22(2):1211, 2007.

Carmel Domshlak, Malte Helmert, Erez Karpas, and Shaul Markovitch. The selmax planner: Online learning for speeding up optimal planning. *Seventh international planning competition (IPC 2011), deterministic part*, pages 108–112, 2011.

David Tolpin, Tal Beja, Solomon Eyal Shimony, Ariel Felner, and Erez Karpas. Towards rational deployment of multiple heuristics in a*. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 674–680. AAAI Press, 2013.

Steven Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2–3):363 – 391, 1990.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.