CENTRO DE CIENCIAS EXATAS E TECNOLOGICAS - CCE

DEPARTAMENTO DE INFORMATICA

THESSES:

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

1. clarify the problem or opportunity
2. analyze the problem or opportunity
3. develop alternative solutions
4. evaluate potential outcomes of each alternative
5. decide
6. act

Many important problems can be represented as a state space problem. In the last few decades, Artificial Intelligence

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 [**???**] have been developed to automatically generate heuristics from domain and problem specifications. The component that generate heuristics have the name of *heuristic generators* [**?**]. 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 [**?**].

Solving domain-independent planning problems can be very hard. Most of the benchmark problems are NP-Complete [**?**].

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 for 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* [**?**], 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 the 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 are 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) [**?**]), delete relaxations (e.g., h^{max} [**?**]), and landmarks (e.g., LM-cut [**?**]).

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 [**?**]. Another approach mention the possibility of combining patterns for smaller problems with the object of producing heuristics for larger problems [**?**], e.g., 8-puzzle and/or 15-puzzle to create a heuristic to work with 24-puzzle.

[**?**] experimented with heuristic search algorithms guided by heuristic functions which considered the maximum value of a set of heuristics (max), and heuristic functions which considered the value of a heuristic selected randomly from a set of heuristics (random). Holte et.,found that maxing over multiple PDBs, the per node evaluation cost is equal to the sum of the individual PDB evaluation costs. Holte et al., also found that the running time of search algorithms expanding smaller search trees could be larger than the running time of search algorithms expanding larger search trees. [**?**] use random 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. This is because, for large PDB we have less abstract pattern. Consequently, the smaller the search tree.

The idea of not using all heuristics to evaluate every node has appeared several times recently. See for instance *selmax* **?**, Lazy A* (LA*), and Rational Lazy A* (RLA*) [**?**]. These algorithms decide which heuristics from a set 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 is often lower than both max or random.

The previous works mentioned above showed that reducing the search tree size does not necessarily reduces the search running time. This is a form of *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, [**?**] 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* [**?**]. 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

This project is concerned with cost-optimal state-space planning using the A* algorithm [**?**]. 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 employing these functions. The cardinality of the subset of heuristics found is also part of the research.

4.1 Specific Objectives

We expect to achieve the following specific objectives in this project.

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 aiming at reducing the search running time.

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 exponential to polynomial.

The term search in planning is related to the process of finding solutions. Search algorithms seeks a path from start to goal.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 conditions. In the common case, we aim at the best of such paths, often in terms of path length or path cost. 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 planner we use to solve the domain-independent problems is the planning system Fast-Downward [**?**]. 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 domain-independent planning benchmarks domains.

As part of the metodology, we will study the state of the art in Maximazing submodular set functions [**?**].

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										✓