Universidade Federal de Viçosa

Campus Viçosa

Centro de Ciencias Exactas e Tecnologicas (CCE)

Mestrado em Ciencia da Computacão

On Selecting of Heuristics Functions for Domain−Independent Planning

AUTHOR: Marvin Abisrror Zarate

ADVISOR: PhD. Levi Henrique Santana de Lelis

CO−ADVISOR: PhD. Santiago Franco

Viçosa (MG), 01 of December of 2015

This thesis is dedicated to my Mother.

Acknowledgements

I would like to express my sincere gratitude to my advisor PhD. Levi Henrique Santana de Lelis, for the continuous support and guidance during the thesis process. His valuable advice, patience and encouragement have been of great importance for this work.

Besides my advisor, I would like to thank to my co−advisor: PhD. Santiago Franco for his insightful feedback, interest and tough questions.

To the professors of the DTI, particularly the Master Degree program with all its members, played an invaluable role in my graduate education.

Last, but not least, I would like to thank my Mother.

Abstract

In this dissertation we present a greedy method based on the theory of supermodular optimization for selecting a subset of heuristics functions from a large set of heuristics with the objective of reducing the running time of the search algorithms.

Holte et al. [2006] showed that search can be faster if several smaller pattern databases are used instead of one large pattern database. We introduce a greedy method for selecting a subset of the most promising heuristicss from a large set of heuristics functions to guide the A* search algorithm. If the heuristics are consistent, our method selects a subset which is guaranteed to be near optimal with respect to the resulting A* search tree size. In addition to being consistent, if all heuristics have the same evaluation time, our subset is guaranteed to be near optimal with respect to the resulting A* running time. We implemented our method in Fast Downward and showed empirically that it produces heuristics which outperform the state of the art heuristics in the International Planning Competition benchmarks.

Table of Contents

Dedication I

Acknowledgements II

Abstract III

List of Tables VI

List of Figures VII

1. Introduction 1

1.1. Background 1

1.2. Problem Statement and Motivation 1

1.3. Aim and Objectives 3

1.3.1. Aim 3

1.3.2. Objectives 3

1.4. Scope, Limitations, and Delimitations 3

1.5. Justification 3

1.6. Hypothesis 3

1.7. Contribution of the Thesis 4

1.8. Organization of the Thesis 4

2. Literature Review 5

2.1. Heuristics 6

2.2. Problem Formulation 7

2.2.1. Out of place (O.P) 8

2.2.2. Manhatham Distance (M.D) 8
2.3. Heuristic Generators 9
2.3.1. Pattern Database (PDB) 9
2.3.2. Neural Network 9
2.3.3. Genetic Algorithm 9
2.4. Take advantage of Heuristics 9
2.5. Number of heuristics to create. 10
2.6. Heuristic Subset 10
2.7. — 11
2.7.1. — 11

3. Greedy Random Heuristic Selection (GRHS) 12
3.1. – 12
3.1.1. Overview 12

4. Stratified Sampling (SS) 13

5. Experiments 14

6. Conclusion 15
6.1. Summary of Contributions 15

References 16

List of Tables

List of Figures

2.1. The left tile−puzzle is the initial distribution of tiles and the right tile−puzzle is the goal distribution of tiles. Each one represent a State. 6
2.2. Heuristic Search: I: Initial State, s: Some Sate, G: Goal State 6
2.3. Out of place heuristic 8
2.4. Manhatham distance heuristic 8
2.5. One heuristic of size M 10
2.6. Two heuristics of size M/2 10
2.7. N heuristics of size M/N 10

CHAPTER 1

Introduction

1.1 Background

Exists many problems of Artificial Intelligent (AI), such as: Finding the shortest path from one point to another in a game map, solve the games of PACMAN, 8−tile−puzzle, Rubick's cube, etc. The level of difficulty to solve the problems mentioned are linked with the size of the search space generated.

State−space search algorithms have been used to solve the problems mentioned above. And in this dissertation we study the approach to solve problems in order to reduce the size of the search tree generated and the running time of the search algorithm.

1.2 Problem Statement and Motivation

Every problem of Artificial Intelligent can be cast as a state space problem. The state space is a set of states where each state represent a possible solution to the problem and each state is linked with other states if exists a function that goes from one state to another. In the search space there are many solutions that represent the same state, each of this solutions are called node. So, many nodes can be represented as one state. To find the solution of the problem is required the use of search algorithms such as: Depth First Search

(DFS), which looks the solution of the problem traversing the search space exploring the nodes in each branch before backtracking up to find the solution. Another search algorithm is Breadth First Search (BFS), which looks for the solution exploring the neighbor nodes first, before moving to the next level of neighbors. The mentioned algorithms have the characteristic that when they do the search, they generate a larger search space, basically for two main reasons: a) Consider the total number of states to be analyzed in order to determinate if the solution is found. b) There is no guide to get to the solution. The search space that these algorithms generate are called Brute force search tree (BFST).

There are other types of algorithms called heuristic informed search, which are algorithms that requires the use of heuristics. The heuristic is the estimation of the distance for one node in the search tree to get to the near solution. The heuristic informed search generates a smaller search tree in comparison to the BFST, because the heuristic guides the search exploring the nodes that are in the solution path and prunes the nodes which are not. Also, the use of heuristics reduce the running time of the search algorithm.

There are different approaches to create heuristics, such as: Pattern Databases (PDBs), Neural Network, and Genetic Algorithm. These systems that create heuristics receive the name of Heuristics Generators. And one of the approaches that have showed most successfull results in heuristic generation is the PDBs, which is memory-based heuristic functions obtained by abstracting away certain problem variables, so that the remaining problem ("pattern") is small enough to be solved optimally for every state by blind exhaustive search. The results stored in a table, represent a PDB for the original problem. The abstraction of the search space gives an admissible heuristic function, mapping states to lower bounds.

Exists many ways to take advantage of all the heuristics that can be created, for example: [Holte et al., 2006] showed that search can be faster if several smaller pattern databases are used instead of one large pattern database. In addition [Domshlak et al., 2010] and [Tolpin et al., 2013] results showed that evaluating the heuristic lazily, only when they are essensial to a decision to be made in the search process is worthy in comparison to take the maximum of the set of heuristics. Then, using all the heuristics do not guarantees to solve the major number of problems in a limit time.

1.3 Aim and Objectives

1.3.1 Aim

The objective of this dissertation is to develop meta-reasoning approaches for selecting heuristics functions from a large set of heuristics with the goal of reducing the running time of the search algorithm employing these functions.

1.3.2 Objectives

- Develop an approach for selecting a subset of heuristic functions with the goal of reducing the running time of the search algorithms employing these functions.
- Develop approaches to obtain the cardinality of the subsets of heuristics found.
- Develop a method to find a subset of heuristics from a large pool of heuristics that optimize the number of nodes expanded in the process of search.
- Use Stratified Sampling (SS) algorithm for predicting the search tree size of Iterative-Deepening A* (IDA*). We use SS as our utility function.

1.4 Scope, Limitations, and Delimitations

TODO

1.5 Justification

TODO

1.6 Hypothesis

This thesis will intend to prove the hypotheses listed below:

- **H1:** The verification that our objective function of selection is related with two properties: Monotonicity and Submodularity .
- **H2:** Reducing the size of the search tree generated helps to solve more problems.

1.7 Contribution of the Thesis

The main contributions of this Thesis are:

- Provide a prediction method to estimate the size of the search tree generated.
- Provide a meta−reasoning approach based on the size of the search tree generated.
- Provide a meta−reasoning approach based on the evaluation cost of each heuristic.

1.8 Organization of the Thesis

The Thesis is organized as follows:

1. In Chapter 1, the introduction to the thesis is provided which also includes our motivation and defines its scope.
2. In Chapter 2, we review the State of the Art.
3. In Chapter 3, we introduce our meta−reasoning approach.
4. In Chapter 4, we introduce.
5. In Chapter 5, we .
6. We conclude in Chapter 6 by discussing further improvements and future work.

In the next chapter, the domain 8−tile−puzzle is used to understand the concepts that will be helpful for the other chapters.

CHAPTER 2

Literature Review

A SAS^{+}*planning task* [Bäckström and Nebel, 1995] is a 4 tuple $\nabla = \{V, O, I, G\}$. V is a set of *state variables.* Each variable $v \in V$ is associated with a finite domain of possible D_v. A state is an assignment of a value to every $v \in V$. The set of possible states, denoted V, is therefore $D_{v_1} \times ... \times D_{v_2}$. O is a set of operators, where each operator $o \in O$ is triple $\{pre_o, post_o, cost_o\}$ specifying the preconditions, postconditions (effects), and non-negative cost of o. pre_o *and* $post_o$ are assignments of values to subsets os variables, V_{pre_o} *and* V_{post_o}, respectively. Operator o is applicable to state s if s and pre_o agree on the assignment of values to variables in V_{pre_o}. The effect of o, when applied to s, is to set the variables in V_{post_o} to the values specified in $post_o$ and to set all other variables to the value they have in s. G is the goal condition, an assignment of values to a subset of variables, V_G. A state is a goal state if it and G agree on the assignment of values to the variable in V_G. I is the initial state, and the planning task, ∇, is to find an optimal (least-cost) sequence of operators leading from I to a goal state. We denote the optimal solution cost of ∇ as C^*

The state space problem illustrated in the figure 2.1 is a game that consists of a frame of numbered square tiles in random order with one tile missing. The puzzle also exists in other sizes, particularly the smaller 8−puzzle. If the size is 3×3 tiles, the puzzle is called the 8−puzzle or 9−puzzle, and if 4 × 4 tiles, the puzzle is called the 15−puzzle or 16−puzzle named, respectively, for the number of tiles and the number of spaces. The object of the puzzle is to place the tiles in order by making sliding moves that use the empty space.

The legal operators are to slide any tile that is horizontally or vertically adjacent to the blank into the blank position. The problem is to rearrange the tiles from some random initial configuration into a particular desired goal configuration. The 8−puzzle contains 181,440 reachable states, the 15−puzzle contains about 10^{13} reachable states, and the 24−puzzle contains almost 10^{25} states.

Initial

4	1	2
8		3
5	7	6

Goal

1	2	3
4	5	6
7	8	

Figure 2.1: The left tile−puzzle is the initial distribution of tiles and the right tile−puzzle is the goal distribution of tiles. Each one represent a State.

Instead of using an algorithm of Brute force search that will analyze all the possible solutions. We can obtain heuristics from the problem of the slide tile puzzle that will help us to solve the problem.

2.1 Heuristics

State−space algorithms, such as A* [Hart and Raphael, 1968], are important 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 s, and $h(s)$ is the estimated cost−to−go from s to a gial; $h(.)$ is known as the heuristic function. The heuristic is the mathematical concept that represent to the estimate distance from the node s to the nearest goal state.

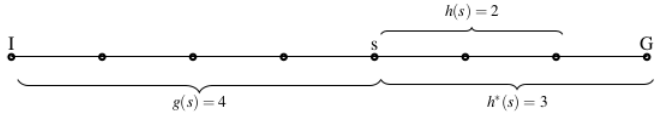

Figure 2.2: Heuristic Search: I: Initial State, s: Some Sate, G: Goal State

In the figure 2.2 the optimal distance from the Initial State I to the state s is 4 and represented by $g(s)$. The $h^*(s)$ represent the optimal distance from s to the Goal State G. And the $h(s)$ is the estimation distance from s to G.

A heuristic function $h(s)$ estimates the cost of a solution path from s to a goal state. A heuristic is admissible if $h(s) \leq h^*(s)$ for all $s \in V$, where $h^*(s)$ is the optimal cost of s. A heuristic is consisten iff $h(s) \leq c(s,t) + h(t)$ for all states s and t, where $c(s,t)$ is the cost of the cheapest path from s to t. For example, the heuristic function provided by a pattern database (PDB) heuristic [Culberson and Schaeffer, 1998] is admissible and consistent.

Given a set of admissible and consistent heuristics $\zeta = \{h_1, h_2, \ldots, h_M\}$, the heuristic $h_{max}(s,\zeta) = \max_{h \in \zeta} h(s)$ is also admissible and consistent. When describing our method we assume all heuristics to be consistent. We define $f_{max}(s,\zeta) = g(s) + h_{max}(s,\zeta)$, where $g(s)$ is the cost of the path expanded from I to s. $g(s)$ is minimal when A* using a consistent heuristic expands s. We call an A* search tree the tree defined by the states expanded by A* using a consistent heuristic while solving a problem ∇.

2.2 Problem Formulation

When solving ∇ using the consistent heuristic function $h_{max}(\zeta^{'})$ for $\zeta^{'} \subseteq \zeta$, A* expands in the worst case $J(\zeta^{'}, \nabla)$ nodes, where

$$J(\zeta^{'}, \nabla) = |\{s \in V | f_{max}(s, \zeta^{'} \leq C^*)\}| \tag{2.1}$$

$$J(\zeta^{'}, \nabla) = |\{s \in V | h_{max}(s, \zeta^{'} \leq C^*) - g(s)\}| \tag{2.2}$$

We present a greedy algorithm for approximately solving the following optimization problem,

$$\begin{aligned} &\textbf{minimize}_{\zeta' \in 2^{|\zeta|}} J(\zeta^{'}, \nabla) \\ &\textbf{subject to} |\zeta^{'} = N| \end{aligned} \tag{2.3}$$

Where N could be determined by a hard constraint such as the maximum number of PDBs one can store in memory.

The heuristics can be obtained from each state of the problem. For example, for the problem of the 8−tile−puzzle figure 2.1 we can get two heuristics.

2.2.1 Out of place (O.P)

Counts the number of objects out of place.

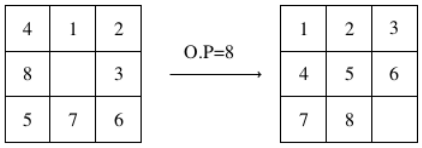

Figure 2.3: Out of place heuristic

The tiles numbered with 4, 1, 2, 3, 6, 7, 5, 8, and 4 are out of place then each object count as 1 and the sum would be 8.

2.2.2 Manhatham Distance (M.D)

Counts the minimum number of operations to get to the goal state.

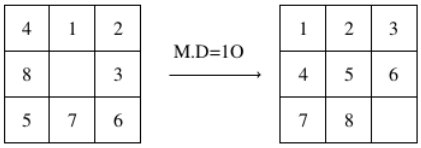

Figure 2.4: Manhatham distance heuristic

The tile 4 count 1 to get to the goal position. The tile 1 count 1 to get to the goal position. The tile 2 count 1 to get to the goal position. The tile 3 count 1 to get to the goal position. The tile 6 count 1 to get to the goal position. The tile 7 count 1 to get to the goal position. The tile 5 count 1 to get to the goal position. The tile 8 count 1 to get to the goal position. Then the sum would be 10.

In order to solve the problem, we get the heuristics, which are information from the problem to solve the problem. Exists systems that can create heuristics for each problem. Those systems are called Heuristic Generators.

2.3 Heuristic Generators

Heuristic Generators works by creating abstractions of the original problem spaces. There are different ways to abstract the problem space such as:

2.3.1 Pattern Database (PDB)

It's obtained by abstracting away certain problem variables, so that the remaining problem ("pattern") is small enough to be solved optimally for every state by blind exhaustive search. The results stored in a table, represent a PDB for the original problem. The abstraction of the search space gives an admissible heuristic function, mapping states to lower bounds.

2.3.2 Neural Network

2.3.3 Genetic Algorithm

2.4 Take advantage of Heuristics

The heuristics generators can create hundreds or even thousand of heuristics. In fact, exists different ways to take advantage of those heuristics. For example: If we want to use all the heuristics created by the heuristic generator. It would not be a good idea to use all of them because the main problem involved would be the time to evaluate each heuristic in the search tree, it could take too much time.

One way to take advantage of heuristics would be to take the maximum of the set of heuristics. For example, using three different heuristics $h1, h2$ and $\max(h1, h2)$. Heuristic

$h1$ and $h2$ are based on domain abstractions and the $\max(h1, h2)$ is the maximum heuristic value of $h1$ and $h2$.

Exists different approaches to take advantage from a large set of heuristics. In this dissertation we use the meta−reasoning based on the minimum evaluation time.

2.5 Number of heuristics to create.

Something here

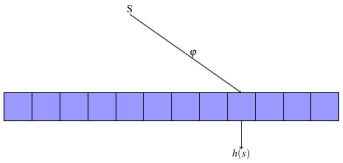

Figure 2.5: One heuristic of size M

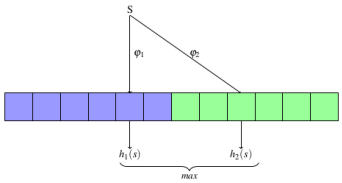

Figure 2.6: Two heuristics of size M/2

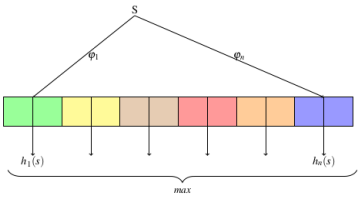

Figure 2.7: N heuristics of size M/N

2.6 Heuristic Subset

The heuristics generators systems can create many heuristics. Let's suppose $|\zeta| = 1000$ heuristics were created considering the time and memory avaiable and we want to select the best N = 100 heuristics. This would be:

$$\binom{1000}{100} = 10^{138} possibilities$$

So, try to select heuristics from a large set of heuristics are going to be treated as an optimization problem. In order to select a subset of heuristics, our objective function should guarantee two properties: Monotonicity and Submodularity.

2.7 —

2.7.1 —

In the next chapter, ...

CHAPTER 3

Greedy Random Heuristic Selection (GRHS)

The purpose of this section is to introduce the meta−reasoning proposed for specifying ...

3.1 −

3.1.1 Overview

In the next chapter, ...

CHAPTER 4

Stratified Sampling (SS)

In this chapter, we present the Stratified Sampling.

CHAPTER 5

Experiments

In this chapter, we verify the the use-cases studied in the previous chapter using our meta.−reasoning approach introduced in Chapter 3. The main purpose of this chapter is to prove the hypotheses introduced in chapter 1:

CHAPTER 6

Conclusion

6.1 Summary of Contributions

In this thesis,

References

Robert C Holte, Ariel Felner, Jack Newton, Ram Meshulam, and David Furcy. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence*, 170 (16):1123–1136, 2006.

Carmel Domshlak, Erez Karpas, and Shaul Markovitch. To max or not to max: Online learning for speeding up optimal planning. In *AAAI*, 2010.

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.

Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

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. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.