Using Sampling to Dynamically Reconfigure Problem-Solvers

Santiago Franco, Mike Barley
The University of Auckland, Department of Computer Science
Private Bag 92019,Auckland, New Zealand
santiago.franco@gmail.com
mbar098@cs.auckland.ac.nz

Abstract

The time it takes a program to solve a particular problem depends heavily upon the choice of problem solving method, the data representation, heuristics etc. The specific choices can have a dramatic impact on performance.
This research aims to produce a formula based on these design decisions and problem characteristics which predicts how long the problem will run until a solution is found.
As means to this end we are working on a prototype problem solving method which dynamically adapts its search configuration in order to speed up finding a solution.

Problem Description

The ultimate goal of this research is to design a problem solver which can dynamically reconfigure itself to speed up its problem solving. In order to achieve this objective we need: (1) a formula that relates problem parameters and design decisions to the CPU time that the configuration will need in order to solve that problem; (2) a method for deriving the parameter values from the problem; and (3) a way of finding the set of design decision choices that minimizes the problem-solving CPU time. The challenge is to mathematically model the impact of the different design choices of methods, data representations, and heuristics upon the time taken solve a given problem.

In general, automatically generating such a formula for an arbitrary family of problem solvers is beyond the current state of the art. Therefore we have focused on one such family of problem solvers, namely, heuristic search problem solvers.

We have identified a number of design decisions and problem parameters that affect the CPU time needed to solve a problem, and have formulated a rudimentary formula that relates these decisions and parameters to the CPU time. We have devised a strategy for incrementally acquiring approximations of the problem parameters needed by the formula, and are in the process of implementing a prototype that dynamically reconfigures itself to reduce its expected CPU time to solve the problem.

Note that we talk about incrementally acquiring approximations of the problem parameters , this is because not all the problem parameters can be acquired a priori. Some of them will only be known with certainty after the problem has been solved. However, for some we can approximate their value as we attempt to solve the problem. We will call the first type *a priori*, the second *a posteriori*. Those parameters gathered while solving the problem we call *sampled parameters*.

Our formula calculates the total CPU time by multiplying the expected number of search nodes by the expected time cost of creating a search node. Roughly, the expected number of search nodes is the average node branching factor raised to the minimum solution length depth.

Unfortunately, the values for these problem parameters cannot be determined by simply examining the statement of the problem. In particular, we seldom know the minimum solution path length for arbitrary problems.

One problem in predicting how long the search will take using a specific Heuristic Search configuration is that one needs to predict how many nodes the informed search method (using that heuristic) will explore in searching for a solution (as well as the cost of exploring a node). The size of the search space will usually be exponential with respect to the length of the solution found. Since we normally do not know the length of the solution before we solve the problem, we can not predict how many nodes will be explored a priori.

Knowing the exact length of the solution is not necessary if the Heuristic Space is expanded in stages , e.g., IDA*. We can then use statistics gathered on the previous iterations to predict next iteration performance. For iterative search different heuristics can be the best choice for different iterations of the same problem.

Summarizing: Heuristic Search performance is a function of heuristic selection, problem instance, search method and current iteration.

In order to reconfigure the Heuristic Search implementation for the next iteration we need to calculate which of the available configurations minimize the time formula. We gather sampled parameters which are used to predict performance on the exponentially bigger next iteration. Sampled parameters prediction quality depends on how frequently they are sampled. There is a trade off between prediction quality and computational effort.

Related Research

Most of related research has been centered on estimating the size of search spaces expanded by Heuristic Search. Most of existing literature concentrates on studying either different search implementations with the same candidate

heuristics, e.g. IDA* vs A*, or heuristic selection given a search configuration. For both cases the size of the search space expanded tends to be the deciding factor on which search method is fastest as other variants are the same.

Holte(Holte,2005) compares hierarchical abstract IDA*,in which heuristics are calculated on demand, vs disjoint pattern databases(Korf,2002), on which the whole abstract space is searched and stored before solving any problem instance. Holte notes that comparative heuristic performance depends on how many instances with the same goal are to be solved and how his abstract search method is configured (how many abstract levels are cached). Furthermore the performance of his Heuristic Search configuration is dependent as well on the problem instance being solved.

Our Approach

The objective of this research is to automate choosing which heuristic search configuration will be the fastest for the current problem for the next iteration. The novelty on our approach is to dynamically adapt the Heuristic Search configuration to minimize search time until a solution is found.

Every time we perform an iteration and the solution is not found the solving method starts an introspective phase. For the next iteration we need to decide what is the preferred Heuristic Search configuration and which sampled parameters need to be updated.

While it would be desirable to predict the performance of each possible Heuristic Search configuration the time costs associated can be too large. We choose which information we gather following the principle:

> Information has value to the extent that it is likely to cause a change of plan, and to the extent that the new plan will be significantly better than the previous plan (Russell,1995).

For example we might consider whether to add duplicate checking or inverse operator checking to the Heuristic Search configuration for next iteration. Duplicate check is more expensive than inverse operator checking but its pruning potential is much bigger if the search space is big enough. As we have formulas for predicting the search space size, we can alter the current preferred search configuration from inverse operator checking to duplicate checking.

An advantage of iterative search on exponential search spaces is that the cost of gathering updated values of sampled parameters will be exponentially cheaper than acquiring them on the next iteration.

The introspective phase needs to keep predictions on each possible search configurations. Each sampled parameter has an associated accuracy. As the number of iterations increases, the sampled parameters may loose accuracy. This, in turn, may affect our ability to determine which configuration is best. When this happens, we need to update the values of the sampled parameters.

Following is a brief description of some of the parameters used in predicting future performance.

$$t_n = N_{\exp}(n)*(t_p(n-1)+t_h(n-1)+t_g(n-1)+t_{\exp}(n-1)) + N_{cull}(n)*(t_p(n-1)+t_h(n-1))+N_p(n)*t_p$$

The formula above predicts the time(t_n) it takes to perform the next IDA* iteration using sampled parameters.

- N_{exp} is the number of nodes expanded on the search. Every node expanded has been tested for whether it should be expanded.
- N_{cull} are those nodes whose expansion is delayed till a future iteration because their F value is bigger than the current iteration depth bound.
- N_p will never be expanded because of search control rules like cyclic paths, inverse operator, duplicate check, etc.

There are several models predicting the search space size. In general they try to map the Heuristic Search Space to a Uniform Search Tree with an effective branching factor,depth or simply a constant growth rate once the search space is big enough(Korf,2001). There are different formulas using this parameters which allow us to estimate the nodes expanded, culled and pruned for next iteration. Keeping tabs on these is quite inexpensive.

- t_h: depends on the cost of the heuristic. For closed formulas like Manhattan the cost is constant. For abstraction based heuristics the time cost is as complex as any other planning search and is a function of abstract search method, abstract space complexity and how/whether the results are cached.
- t_p: test is constant as long as an efficient hashing function can be built. Otherwise, the cost is linearly dependent of the size of the pruning database to check against. Normally a hashing function can be build for any state of a domain. We are considering full duplicate checking, cyclic path detection and inverse operator checking as pruning functions.
- t_g:goal evaluation is in most cases constant for the domain and can be precomputed for all problem instances.
- t_{exp}: node expansion costs are usually constant and can be precomputed for most domains.

References

Holte,R., Grajkowski,J. and Tanner,B. 2005. *Hierarchical Heuristic Search Revisited*.SARA, LNAI 3607, pp. 121 133.

Korf, R., Felner, A. 2002. *Disjoint pattern database heuristics*. Artificial Intelligence 134, pp. 9 22

Korf R., Reid, M. & Edelkamp, S. 2001. *Time complexity of iterative-deepening-A**. Artificial Intelligence 129: pp. 199-218.

Russell, S., Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey.