

In Situ Reconfiguration of Heuristic Search on a Problem Instance Basis

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.

Our objective is to design a problem solving method which dynamically adapts its search configuration in order to speed up finding a solution.

Previous approaches have used performance data gathered on past problem solving episodes to predict performance of different problem solving configurations on future problem instances. These approaches prediction quality depend on the future problem distribution being known.

The main novelty of our approach is that we reconfigure the search based solely on performance data gathered while solving the current problem instance.

As means to this end we use a formula based on these design decisions and problem characteristics which predicts how long the problem will run until a solution is found.

the best method to solve that problem. One approach around this dilemma is instead of determining which method is best, on a problem-by-problem basis we empirically determine which method is best, on average, for the given problem domain. However, this approach has two problems, one is the cost of determining which method is best on average for the domain. The other problem is that this approach still gives worse results than if we could select the best method for each problem.

These problems have lead researchers to explore ways of learning, in a domain, appropriate mappings from problem instances to which method to use in solving that problem. These learning approaches may take care of the second problem, but they are probably just as expensive (and perhaps more so) as empirically determining the best "average" method.

We are proposing a different approach to deciding which method is best for a given problem. Our approach is to use parametrized complexity formulas for the different alternative methods that (when given the appropriate parameter values for the problem instance) allow us to predict which method will be the cheapest to use in solving this problem. Our approach requires three things: (1) the parameterized complexity formulas; (2) a technique for obtaining the problem-specific values for these parameters; and (3) an operational criteria for when these values should be revised.

This paper will outline a specific context for exploring this approach, namely, we will confine ourselves to selecting the best heuristic (from a set of heuristics) for use in solving a given problem with heuristic search. We will use a standard complexity formula for heuristic search effort. This paper will describe our technique for obtaining the values for the parameters in the formula and suggest the basis for when these values should be updated.

Introduction

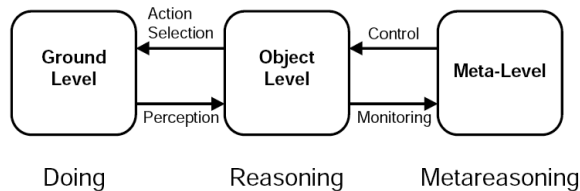


Figure 1: Metareasoning in situated agents

Although, agents can often be described as shown in figure 1 (Cox, 2008), this paper will only look at the agent's interactions between its object level and meta-level, focusing primarily on the agent's Metareasoning. Metareasoning is defined as reasoning about reasoning (Russell, 2003). In particular, an agent may need to reason about the control of their object level reasoning. This metareasoning may lead to less work being done at the object level, however, a pitfall is that this metareasoning is not cost-free and can actually make the agent perform worse. To be effective, the savings at the object must exceed the extra overhead at the meta-level.

It is an oft stated truism that no one problem solving method is best for all problems. However, given a specific arbitrary problem, it is not always obvious how to select

Problem Description

Experiments (Holte, 2005) suggest that choosing the best heuristic on a problem by problem basis can result in substantial speed ups over just using one heuristic for all problems in the domain. Holte observes that for 78 out of 100 instances, heuristics other than the default one do

¹Unfortunately the domain specific default heuristic was not demonstrated to be the best on average for all problems in the test set. We intend to run a set of experiments that performs this specific comparison.

better. The problem is that as Holte observes: There was no obvious rule to decide which abstraction to use on a given problem instance. Holte's abstractions are heuristics and as he mentions, the effectiveness of a heuristic depends on the problem instance being solved. This is exactly the problem we are attacking.

In order to achieve this objective we need a parametrized formula that predicts heuristic performance on a problem by problem basis. Not all parameters can be determined before we attempt to solve the problem. However, for some we can approximate their value as we attempt to solve the problem. Those parameters which can be gathered from the problem description are called *a priori*, while those parameters gathered while solving the problem are called *in situ* parameters.

Our Approach

Our approach is to move away from looking for a way to select the best heuristic for an arbitrary problem suite from an arbitrary set of heuristics. Instead our approach uses a parametrized formula that predicts how long it will take to expand a heuristic search tree (*HST*) to a specific F-value depth using a specific heuristic. These parameters include data like effective branching factor, effective depth reduction, CPU time to expand a node, etc.

In general, automatically generating such parametrized formulas 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.

An alternative to estimating and comparing search effort for the different configurations is to try to find features in the problem instance description that allow the problem instance to be matched to the best available problem solver configuration for that problem. Finding these features is not trivial and their existence is not guaranteed. We discuss this in more detail in the related research section.

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 the next iteration's performance. For iterative search algorithms, different heuristics can be the best choice for different iterations of the same problem.

In order to reconfigure the Heuristic Search implementation for the next iteration we need to estimate which of the available configurations will minimize the time formula. The gathered *in situ* parameters are used to predict performance on the next iteration.

Sampled prediction quality depends on how frequently they are sampled. There is a trade off between prediction quality and computational effort. Thus it might not be economical to refine our predictions if the associated sampling cost is too high.

Sampling and Heuristic Search Configuration

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 of our approach is to dynamically adapt the Heuristic Search configuration while solving the current problem 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 parameters need to be re-sampled.

While it would be desirable to predict the performance of each possible Heuristic Search configuration, the associated time costs can be too large. Paraphrasing ((Russell,2003),page 603), we choose which information we gather following the principle: Sampling has value to the extent that it is likely to cause the system to use a different heuristic, and to the extent that the new heuristic will cause the search to be significantly cheaper than using the previous heuristic.

The most important parameter for each search configuration is which heuristic is used. Each heuristic has an evaluation cost and culls some nodes. We predict this parameter's future values based on previous iteration results. Prediction quality might degrade considerably as we do subsequent iterations. We need to weigh the need of more sampling (when prediction quality is not enough to make a good informed decision) against the overhead of sampling.

An advantage of iterative search on exponential search spaces is that the cost of gathering updated values of sampled parameters for the current iteration is much cheaper than acquiring them on the next iteration. This makes our approach of gathering statistics on previous iterations quite efficient.

The introspective phase keeps predictions on each possible search configurations. Each of these predictions has an associated confidence level. As the number of iterations increases, confidence may decrease significantly.

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.

Search Effort Formula

$$t_{t,F} = N_{e,F} * t_e + N_{d,F} * t_n \quad Eq 1.$$

Eq 1. predicts the total time($t_{t,F}$) it takes to perform an IDA* iteration bounded by an F-limit using parameters:

- $N_{g,F}$ is the number of nodes generated for iteration F. Nodes generated can be modelled as a uniform search tree (*UST*) whose depth has been reduced from F to F-r by the heuristic culling(Korf,2001).

$$N_{g,F} = \frac{b^{F-r} - 1}{b - 1} \quad Eq 2.$$

r is the effective depth reduction and b is the effective branching factor of the brute force search(*BFST*) tree generated by using no heuristic. Both b and r are sampled in situ parameters.

- $N_{d,F}$ is the number of nodes whose expansion has been delayed until a future iteration because their F value is bigger than the current iteration depth bound.

$$N_{d,f} = b^{F-r} \quad Eq 3.$$

- $N_{e,F}$ is the number of nodes expanded for iteration F. Nodes expanded is all nodes generated minus the leaf nodes.

$$N_{e,F} = N_{g,F} - N_{d,F} \quad Eq 4.$$

- N_B is the number of nodes generated by the *BFST*. It is modelled as if all nodes in the tree had the same branching factor. We can use Eq 5 to calculate the effective branching factor b.

$$N_B = \frac{b^F - 1}{b - 1} \quad Eq 5.$$

- t_e is the time costs associated to expanding a node. it includes heuristic evaluation time t_h , goal evaluation time t_g and node creation time t_c .

$$t_e = t_h + t_g + (t_c * b) \quad Eq 6.$$

- t_h is the heuristic evaluation time. depends on the heuristic. For closed formulas like Manhattan distance the cost is constant but for abstraction based heuristics the cost is potentially as big as the base level search. t_g and t_c is sampled a priori parameters as we do not expect their cost to change significantly from node to node.

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, a depth or simply a constant growth rate once the search space is big enough (Korf,2001). Equations 1 to 5 are based on the view that the effect of a heuristic is

reducing the depth of the *BFST*. There are other alternative models but for the purposes of this paper Korf's model suits our needs.

Meatareasoning costs

For each iteration we need to estimate which available Heuristic Configuration will be best. This involves keeping track of the following in situ parameters:

- Effective Branching factor: This can vary from iteration to iteration. The root node has its own branching factor which does not have to be the average effective branching factor. It needs to be re-sampled for each iteration until it converges to the effective branching factor of the *BFST*.
- Effective Depth Reduction: The heuristic culling effect is being modeled as a reduction of the effective depth and can change as the heuristic efficiency can vary from iteration to iteration. The more frequently this is re-sampled, the better its accuracy.
- t_i : This depends on the heuristic complexity. For closed formulas like the Manhattan distance its cost is constant from iteration to iteration. If the heuristic expands a search space then its evaluation effort can be as complex as any other search and can be modeled with its own effective branching factor and effective depth.

We need a confidence level mechanism. We are still working on this. A confidence level is critical for deciding when we need to update any prediction. We are reviewing different statistical mechanisms which could accomplish this.

Related Research

Dynamic Planners

Most research on problem solvers that can dynamically reconfigure themselves are based on learning a priori features which match best available configurations to problem instances. In the *BUS* metaplanner(Howe,99) 6 different planners are available. A large sample of problem instances are used to train *BUS* to match problem features with a planner. These features correlate planners to problem instances and depend on the number of initial instances, predicates, available actions, etc. The utility function used for learning is based on the comparison of expected performance vs actual performance. We call this learning technique *a posteriori* as data is gathered after solving a number of problem instances and is used to predict future performance on new problem instances.

The resulting metaplanner can solve a suite of problems faster than any of the individual planners and, not surprisingly, can solve more problems than any of the individual planners. The main drawback of this approach was that even though there is a statistical connection between the a priori features learnt and planner performance, these connections are not strong enough to be adequately predictive. Furthermore the training set of

problems were uneven in their distribution of feature values. Using the right training problem suite distribution is a concern for any posteriori learning technique.

These two problems are avoided by our approach. First problem type distribution is not a factor as our predictions are not based on past performance data for a set of training problem instances. Secondly our prediction mechanism is based on a standard heuristic search space size formula which predicts the search effort quite accurately with a moderate amount of sampling.

Another problem was the large amount of training instances needed. This needs to be accounted as an extra overhead.

A similar approach is taken by COMPOSER(Gratch, 1996). His research focuses on how to combine transformations to increase the problem solver's expected utility. The focus of his research is to deal with the utility problem, mainly how positive transformations can have negative interactions. This is a very interesting problem but it is not the focus of this report. We are selecting among available heuristics but for the time being we are not combining them. For our purposes COMPOSER transformations are akin to heuristics. COMPOSER also uses a set of training examples. It learns which transformations to use and in which order to combine them. Our previous criticism regarding problem distribution would apply.

COMPOSER uses an efficient bounded error mechanism to minimize the size of the training suite, thus reducing the overhead compared to the Howe approach.

Heuristic Search effort

Most research on Heuristic Search effort has been focused on estimating the size of search spaces expanded by Heuristic Search. Most 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 implementation. For both cases the size of the search space expanded tends to be the deciding factor on which search method is fastest as other variables 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. He notes how the same heuristic can improve performance up to 50 times on a particular problem instance compared to its average effectiveness. This motivates our decision to reconfigure the search method based on performance data gathered for the current problem instance.

Summary

This paper proposed a new approach to selecting which problem solving method to use for a given problem. It outlined the specific context we are using to explore this approach, namely, we have confined ourselves to selecting the best heuristic (from a set of heuristics) for use in solving a given problem with heuristic search. We are using a standard complexity formula for predicting heuristic search effort. This paper described our technique for obtaining the values for the parameters in the formula and suggested the basis for when these values should be updated.

References

- Cox, M. T., and Raja, A. 2007. *Metareasoning: A Manifesto*, Technical Report, BBN TM-2028, BBN Technologies. www.mcox.org/Metareasoning/Manifesto
- Gratch,J., DeJong,G. 1996 . *A Decision-theoretic Approach to Adaptive Problem Solving*. Artificial Intelligence 88(1-2): 101-142.
- Holte,R., Grajkowski,J. and Tanner,B. 2005. *Hierarchical Heuristic Search Revisited*.SARA, LNAI 3607, pp. 121 133.
- Howe, E., Dahlman, C., Hansen., Scheetz, M. and von Mayrhauser, A. 1999. *Exploiting Competitive Planner Performance*. Proceedings of the European Conference in Planning, pp 60 72, .
- Korf,R.1985.*Depth first iterative deepening: An optimal admissible tree search*. Artificial Intelligence. pp. 95-109.
- 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. 2003. *Artificial Intelligence: A Modern Approach*, Second Edition, Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey.