A Bayesian Approach for Automatic Algorithm Selection

Haipeng Guo
Department of Computing and Information Sciences
Kansas State University, Manhattan, KS 66506
hpguo@cis.ksu.edu

Abstract

This paper introduces a self-training *automatic algorithm selection system* based on experimental methods and probabilistic learning and reasoning techniques. The system aims to select the most appropriate algorithm according to the characteristics of the input problem instance. The general methodology is described, the system framework is presented, and key research problems are identified.

1 Introduction

Given a computational problem, there usually exists many different algorithms to solve it exactly or approximately. Different algorithms often perform better on different classes of problem instances. The *algorithm selection problem* asks the following question: *"which algorithm should we select to solve the input instance?"*

The automatic algorithm selection problem arises in various situations. It is important both theoretically and practically. Theoretically, we computer scientists always like to seek a better understanding to the problem instance hardness and algorithm performance so that we can deliver better and faster algorithms for the given computational task. In practice, it helps us gain more efficient computations to solve the problem. Algorithm selection is particularly crucial for applications with real-time constrains.

There are two possible ways to attack the problem: analytically and experimentally. Furthermore, there are three analytical algorithm comparison and selection methods. The first one is to apply worst-case analysis to all candidate algorithms, compare their complexities, and select the best one. This approach has been widely studied in classical theoretical computer science, however, it does not always work well. Some algorithm may have a bad worst-case complexity, but they perform averagely very well in practice. For example, Quicksort has a worst-case complexity of $O(n^2)$, however, such pathological behavior is never observed in practice. Also, for many complex, approximate, randomized, and heuristic algorithms formal analysis is often infeasible. The second approach is to analyze the algorithm's average-case complexity. The drawback of this method is that it often requires a strong assumption on the distribution probabilities of input instances, which is hard to make in most cases. The third analytical approach considers instance classes instead of single instances or all instances as a whole [Mannila, 1985]. It defines a subproblem of the original problem by some intuitive criteria of instance easiness (or hardness) and then studies the worst-case complexity of the subproblem. Algorithms optimal to the instance hardness measures are designed and analyzed on these subproblems, with resource requirements increasing smoothly when moving to larger subproblems. The drawback of this approach is that it is feasible only to simple problems. It is hard to be applied to NP-hard optimization problems which are more complex yet important in practice. Another drawback is although it provides a way to design adaptive algorithms that are optimal for some measures, it is usually impossible to design an algorithm that are optimal for all measures. For algorithms adaptive to different measures there also exists an algorithm selection problem. Furthermore, all these analytical approaches are generally not suitable to making predictions about the empirical hardness of problem instances and the run time performance of the algorithms.

In this paper, I describe an experimental methodology of building automatic algorithm selection system using machine learning techniques in which the results of theoretical analysis serve mainly as a source of domain knowledge to guide the experiment design. Such a methodology is more feasible because of the following reasons. First, any theoretical results will still need to be implemented and verified. Second, in practice the human expert of algorithm selection seldom depends solely on theoretical analysis without observing the algorithms' run time behaviors. Third, the analytical method has its inherent limitations as we pointed out above. Finally, the progresses recently made in the field of experimental algorithmics, machine learning and Artificial Intelligence (AI) have provided some useful techniques to help solve the automatic algorithm selection problem.

2 Background and Related Works

The algorithm selection problem is originally formulated in [Rice, 1976]. Later on it has been mainly applied to the selection of problem-solving method in scientific computing [Houstis *et al.*, 2000], specifically to the performance evaluation of numerical softwares. An abstract model for the problem is also given in [Rice, 1976] as reproduced in Figure 1, where x is the input instance in the problem space and w is

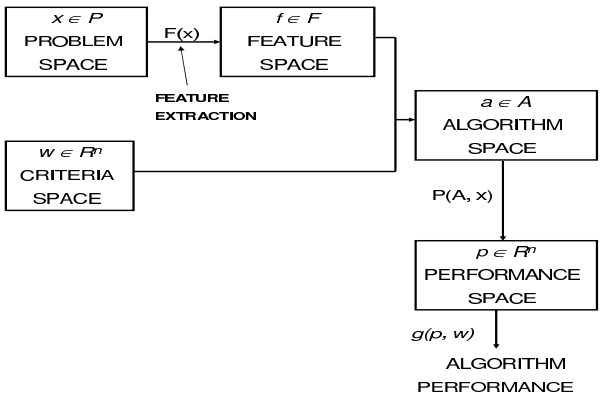

Figure 1: The Abstract Model of Algorithm Selection Problem

the performance criteria. The input problem instance is represented by the feature(s) f in the feature space by a feature extraction procedure. The task is to build a selection mapping S that provides a good (measured by w) algorithm A to solve x subject to the constrains that the performance of A is optimized.

Algorithm selection can be *static* or *dynamic*. Static algorithm selection system makes the selection and commits to the selected algorithm, while the dynamic algorithm selection system may change its selection dynamically by monitoring the running of the algorithm. One special kind of dynamic algorithm selection is *recursive algorithm selection* [Lagoudakis and Littman, 2000; 2001] in which a decision of algorithm selection needs to be made every time a recursive call is made. For example, a sorting algorithm often needs to recursively sort smaller instances. At each recursive call, you need to make the decision about which sorting algorithm to choose. The goal becomes to optimize a sequence of algorithm selection decisions dynamically. In this paper I consider the static algorithm selection problem. Hopefully, the methodology can be applied to dynamic situations with only slight modifications.

Automatic algorithm selection is an interdisciplinary problem invovling many fields such as complexity theory [Garey and Johnson, 1979], computability theory [Homer and Selman, 2001], experimental algorithmics [Johnson, 1996], artificial intelligence [Russell and Norvig, 1995], machine learning [Mitchell, 1997], and so on. Complexity theory provides basic analytical tools and results for algorithm comparison and selection. But the automatizing of the analysis of algorithm complexity is generally infeasible. Computability theory, specifically the Rice's theorem, points out the nonexistence of a program that takes as input only the description of the problem instance and the description of the algorithms and return the best algorithm to solve it. It means that the general automatic algorithm selection problem is undecidable. But this does not rule out the possibility of inducing an algorithm selection system from the experimental data collected by running different algorithms on instances with different features.

In recent years researchers in experimental algorithmics, combinatorial optimization, and artificial intelligence have studied the *empirical hardness* of NP-hard instances. They have managed to find simple relationships between the characteristics of the problem instances and the hardness of the problems independent of any algorithms. One of the major findings is the *phase-transition* [Cheeseman *et al.*, 1991; Mitchell *et al.*, 1992] in NP-hard problems. Their approach is to vary some parameter values of the input instances looking for a hard-easy-hard transition corresponding to a phase transition in the resources that required to solve the problem. It is easy to see that this approach can also be applied to study the relationships between the features of the input instances and the algorithm-dependent hardness of the problem, which is key to solve the algorithm selection problem.

In practice, the selection of algorithm is done by hand by some algorithm selection experts who have a good theoretical understanding to the computational complexity of various algorithms and are very familiar with their run time behaviors on different input instances. To automate this process is to build an expert system that has the capability to perform the same task of algorithm selection. Like any intelligence systems, the algorithm selection expert system should have at least three core components: knowledge representation, learning, and inference. The knowledge and information in automatic algorithm selection contains inherent uncertain factors: the uncertainty in the input problem space, the working mechanism of the algorithms (especially these complex and randomized algorithms), the influence of different implementations, the uncertainty of run time environments, etc. Researchers in the community of Uncertainty in Artificial Intelligence (UAI) have advocated the use of probabilistic graphic models such as *Bayesian networks* [Pearl, 1988] in intelligent systems which reason under uncertainty. In this paper I propose to use Bayesian networks as the core model of an automatic algorithm selection system.

Bayesian networks (BNs), also known as Bayesian belief networks, causal networks, or probabilistic networks, are currently the dominant uncertainty knowledge representation technique in AI [Pearl, 1988; Neapolitan, 1990; Russell and Norvig, 1995]. BNs are Directed Acyclic Graphs (DAGs) where nodes represent random variables, and edges represent conditional dependence between random variables. Each node in the network has a conditional probability table, or CPT. Each column in the CPT contains the conditional probability of each node value for a possible combination of values for its parent nodes. The topology of the network can be thought of as an abstract knowledge base representing the general structure of the causal process in the domain. The numbers in the network - these probabilities - are interpreted as beliefs, i.e., probability is a measure of belief in a proposition given particular evidence. A Bayesian network provides a complete description of the domain. It encodes joint probability distributions (JPD) in a compact manner. BNs

can be learned from data using Bayesian network learning algorithms such as K2 [Cooper and Herskovits, 1992]. The learned network represents the dependency relationships between these domain variables. It then can be used to answer queries about the domain by inference.

3 The Self-training Automatic Algorithm Selection System

In this section I describe a self-training automatic algorithm selection system based mainly on Bayesian methods [Horvitz *et al.*, 2001; Guo, 2002]. Knowledge of dependencies between the characteristics of input problem instances and the performance of the candidate algorithms can be considered as some sort of uncertain knowledge. The uncertainty knowledge can be encoded by a Bayesian network, "the algorithm selection expert network", which is automatically learned from some training data with the guidance of domain knowledge. To learn the expert network, we need to have a representative training data that contains the knowledge we are seeking. In order to achieve this, we develop a controllable random problem instances generator that can randomly generate input instances with the specified characteristic. By controlling the characteristic parameters values, random instances can be generated for both training and testing. Since the space of all possible problems is very large and at the same time many extreme characteristics are rarely encountered in real applications, it's reasonable and necessary to consider only a subset of it, the set of "real world problems" (RWP). This can be done by first extracting the real world distributions of all characteristic parameters from a collection of real world samples, then generating "random" problem instances from them. Once we can generate synthetic problem instances, we can then select the candidate algorithms we are interested in and run them on these instances to generate the training data. These data records the characteristics parameter values of the input instances, the algorithms being used, and the run time performance of the algorithms. They contain the uncertain knowledge of how well each algorithm matches each class of problems statistically. The expert network for algorithm selection can be learned from the training data using Bayesian networks learning algorithm. After we have this expert network in hand, we can use it to select the best algorithm for a given input instance and even predict the run time performance of the algorithm on it.

To summarize, the proposed methodology consists of the following steps.

1. Identify a list of feasible instance characteristics using domain knowledge. They need to be representative and easy to compute comparing to the cost of the problem solving itself.
2. Identify a list of candidate algorithms for solving the problem.
3. Generate a representative set of test instances with different characteristic values settings uniformly at random (or using the RWP characteristic).
4. Run the candidate algorithms on these elaborately designed instances and collect the performance data.

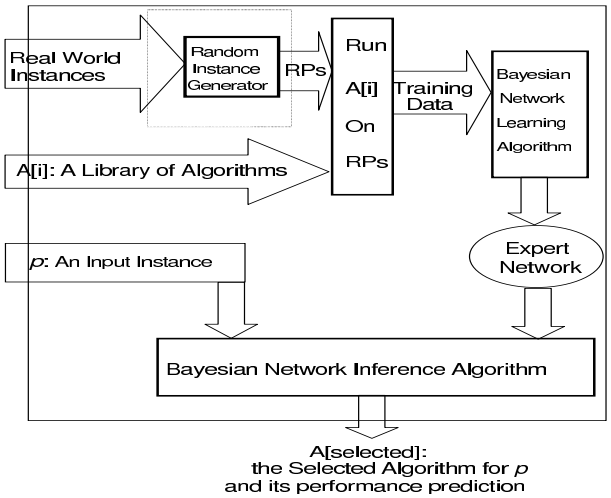

Figure 2: Overview of the Automatic Algorithm Selection System

5. Apply Bayesian network learning techniques to induce a predictive model (a Bayesian network) out of the experimental data.
6. For a new instance, analyze its characteristic and use the learned Bayesian network to infer the most appropriate algorithm to solve it.

Figure 2 shows the overview of the system. The system should be easily extended in the future to include new characteristic parameters and new inference algorithms. It can start a self-training cycle once new features and/or new algorithms are plugged in.

4 Experimental Results

In this section I report some results of applying the proposed methodology to algorithm selection for sorting and the Most Probable Explanation (MPE) problem.

For sorting, the algorithms I considered included insertion sort, shellsort, heapsort, mergesort and quicksort. The instance characteristics included the size of the input permutation and three presortedness measures: the number of inversions (INV), the number of runs (RUN), and length of the longest ascending subsequence (LAS) [Knuth, 1981]. Because sorting can be solved in $O(n \log n)$, the meta-level reasoning becomes too expensive if using a Bayesian network. Our results showed that although the learned Bayesian network provided the best classification accuracy of selecting the best sorting algorithm, the gain in overall computational time was negative. Instead, a simple decision tree provided the best overall performance as a meta-level reasoner. Reasoning using decision trees can be done quickly because it just checks a set of if-then rules. The classification accuracy and reasoning time of Bayesian network and decision tree are list in table 1.

Table 1: Reasoning Time (microseconds) and Classification Accuracy (%) in Sorting Algorithm Selection

	C4.5 DecisionTree	BayesNet
accuracy (%)	90.03	90.61
time	43.35	14,845.35

In the NP-hard MPE problem, solution quality rather than time is more important in evaluating the system's overall performance because the reasoning time can basically be ignored comparing to the actual computational time. The MPE problem, also called belief revision, computes the most probable explanation given the observed evidence in a Bayesian network. For the MPE problem, the algorithms I considered included MCMC Gibbs sampling, importance forward sampling, multi-start hillclimbing, tabu search, and ant colony optimization. MPE instance characteristics being used included number of nodes of the network, network topological type, network connectedness, CPT skewness, evidence percentage, and evidence distribution. The learned Bayesian network is shown in Figure 3. Its classification accuracy was 76.08%. Our test result showed that the MPE algorithm selection system provided the best overall performance for solving the MPE problem compared to all candidate MPE algorithms alone.

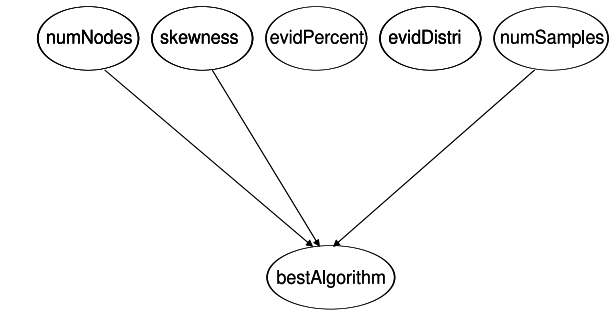

Figure 3: The Learned BN for Approximate MPE Algorithm Selection

5 Discussions and Summary

In this paper I have presented a Bayesian approach for building automatic algorithm selection system. The core of the system is a Bayesian network which represents the uncertain knowledge of the mapping relationship between algorithms performance and problem features. It has the capability of self-training and facilitates efficient learning and inference of the model. When new algorithms or new problem features are plugged in, the system can use idle time to generate random instances and training data to train itself, discover new knowledge, and improve its algorithm selection capability automatically. After inducing the model from training data, the system can select the best algorithm for a given input instance x as solving a Bayesian network inference problem as follows: it first computes the characteristic vector of input x, then uses the characteristic values as observed evidence of the corresponding nodes of the Bayesian network, and finally computes the most probable algorithm given the evidence. The methodology relies heavily on experimental methods and probabilistic learning and reasoning techniques. The general model should apply to other automatic computing system as well.

The key research issues of this scheme include the random generation problem [Sinclair, 1997] to guarantee the representativeness of the training data, the use of analytical domain knowledge to guide the design of the algorithmics experiments, the measuring of the algorithm performance, and the problem of learning the model and reasoning on it.

The proposed algorithm selection system can be extended from the following two aspects. First, in the framework described in last section the learned Bayesian network, as a meta-level reasoner, is mainly used as a classifier to classify an input instance to the best algorithm to solve it according to the instance's descriptive characteristics. In theory any other classifier should work as well, for example, a decision tree or a naive Bayes classifier. It is natural to try all these models, compare their performance and select the best. These can be seen as meta-meta-level reasoning. Second, it is easy to extend the system from automatic algorithm selection to *autonomic algorithm selection*. The system can be designed in such a way that it is able to sense and adapt to changes in the underlying distribution of the input instances so as to gain a better classification accuracy. Again, this requires a meta-meta-level reasoner to keep monitoring the environment changes and being aware of the differences between its meta-level reasoning model and the changed environment. As suggested in [Lagoudakis and Littman, 2000], one simple way to do this is to allow most recent data to overshadow old data.

Some future research directions may include (1) the comparison of different models in terms of their capability of knowledge representation, learning, and inference for algorithm selection; (2) the study of characteristics of real world instances to improve the representativeness of randomly generated instances and training data; (3) apply the methodology to other self-managing computing systems.

Acknowledgments

I would like to thank the anonymous reviewer's valuable comments and advices. I also want to thank my advisor Dr. William H. Hsu for his continue support to my Ph.D. study at K-State.

References

[Cheeseman *et al.*, 1991] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.

[Cooper and Herskovits, 1992] G.F. Cooper and E. Herskovits. A bayesian method for the induction of probabilis-

tic networks from data. *Machine Learning*, 9(4):309–347, 1992.

[Garey and Johnson, 1979] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NPCompleteness*. Freeman, 1979.

[Guo, 2002] H. Guo. A bayesian metareasoner for algorithm selection for real-time bayesian network inference problems. In *AAAI02 Doctoral Consortium Abstract*, 2002.

[Homer and Selman, 2001] S. Homer and A.L. Selman. *Computability and Complexity Theory*. Springer Verlag New York, 2001.

[Horvitz *et al.*, 2001] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and D.M. Chickering. A bayesian approach to tackling hard computational problems. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, August 2001.

[Houstis *et al.*, 2000] E.N. Houstis, A.C. Catlin, J.R. Rice, V.S. Verykios, N. Ramakrishnan, and C.E. Houstis. PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. *TOMS*, 26(2):227–253, 2000.

[Johnson, 1996] D. Johnson. A theoretician's guide to the experimental analysis of algorithms, 1996.

[Knuth, 1981] D. E. Knuth. *The art of computer programming: Sorting and Searching*, volume 3. Addison-Wesley, 1981.

[Lagoudakis and Littman, 2000] M.G. Lagoudakis and M.L. Littman. Algorithm selection using reinforcement learning. In *Proc. 17th International Conf. on Machine Learning*, pages 511–518. Morgan Kaufmann, San Francisco, CA, 2000.

[Lagoudakis and Littman, 2001] M. G. Lagoudakis and M.L. Littman. Selecting the right algorithm. In *Proceedings of the 2001 AAAI Fall Symposium Series: Using Uncertainty within Computation*, Boston, MA, 2001.

[Mannila, 1985] H. Mannila. *Instance Complexity for Sorting and NO-complete problems*. PhD thesis, Department of Computer Science, University of Helsiki, 1985.

[Mitchell *et al.*, 1992] D.G. Mitchell, B. Selman, and H.J. Levesque. Hard and easy distributions for SAT problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, Menlo Park, California, 1992. AAAI Press.

[Mitchell, 1997] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[Neapolitan, 1990] R.E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. John Wiley and Sons, New York, 1990.

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann, San Mateo, CA, 1988.

[Rice, 1976] J.R. Rice. *The algorithm selection problem*, volume 15, pages 65–118. 1976.

[Russell and Norvig, 1995] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[Sinclair, 1997] A. Sinclair. *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Springer-Verlag, 1997.