Ann Oper Res (2007) 156: 61–82
DOI 10.1007/s10479-007-0229-6

A machine learning approach to algorithm selection for $\mathcal{NP}$-hard optimization problems: a case study on the MPE problem

Haipeng Guo · William H. Hsu

Published online: 7 August 2007

Abstract Given one instance of an $\mathcal{NP}$-hard optimization problem, can we tell in advance whether it is exactly solvable or not? If it is not, can we predict which approximate algorithm is the best to solve it? Since the behavior of most approximate, randomized, and heuristic search algorithms for $\mathcal{NP}$-hard problems is usually very difficult to characterize analytically, researchers have turned to experimental methods in order to answer these questions. In this paper we present a machine learning-based approach to address the above questions. Models induced from algorithmic performance data can represent the knowledge of how algorithmic performance depends on some easy-to-compute problem instance characteristics. Using these models, we can estimate approximately whether an input instance is exactly solvable or not. Furthermore, when it is classified as exactly unsolvable, we can select the best approximate algorithm for it among a list of candidates. In this paper we use the MPE (most probable explanation) problem in probabilistic inference as a case study to validate the proposed methodology. Our experimental results show that the machine learning-based algorithm selection system can integrate both exact and inexact algorithms and provide the best overall performance comparing to any single candidate algorithm.

1 Introduction

Given one instance of an $\mathcal{NP}$-hard optimization problem, the algorithm selection problem asks which exact or approximate algorithm is the best to solve it. It is important for both theoretical and practical reasons. Theoretically, computer scientists always want to seek a better understanding of problem (instance) hardness and algorithm performance. Practically, it can provide more efficient computations. This is especially helpful for some real-time applications that are under the pressure of some hard or soft computational deadlines.

H. Guo (✉)
Computer Science and Techology Program, BNU-HKBU United International College, ZhuHai, China
e-mail: hpguo@uic.edu.hk

W.H. Hsu
Department of Computing and Information Sciences, Kansas State University, Manhattan, USA
e-mail: bhsu@cis.ksu.edu

In order to solve the algorithm selection problem, we need to have a good understanding of the performance of various algorithms on different types of problem instances. Traditionally, there are two complementary approaches to study algorithms: the analytical approach and the experimental approach. Analytical approaches use purely mathematical methods to analyze an algorithm's asymptotic behavior, often applying to a worst case or an average case scenario. Experimental approaches draw conclusions from algorithm performance data collected from specially designed experiments.

Theoretical analysis has been the dominant method to the study of algorithms. It should still be our first choice whenever possible. But it has several drawbacks when applying to algorithm selection for $\mathcal{NP}$-hard optimization problems. First, many worst case results do not apply to typical instances in practice. Meanwhile, average case analysis is often very difficult because it requires a reasonable estimate of all instances' probabilities. Second, worst case analysis treats all instances of the same size collectively as a whole although these instances can be very different in terms of features other than size. Third, many newly-developed approximate, randomized, and heuristic search algorithms for $\mathcal{NP}$-hard optimization problems, such as genetic algorithms, simulated annealing, and stochastic local search, simply resist formal analysis. They usually have a large number of parameters and create an extremely complex state space that is very difficult to analyze with existing mathematical methods. Last but not the least, the analytical approach is generally not suitable to predict an algorithm's performance on a single problem instance. To distinguish the varying resource requirements of instances that have the same size but are different in other features, we need to move from problem complexity to instance complexity. There are at least three ways to deal with this analytically. The first one uses resource-bounded Kolmogorov complexity (Orponen et al. 1994) where the instance complexity of a string x, with respect to a set A, is defined as the length of the shortest program of some time-bounded Turing machine that correctly decides if x is in A. The goal here is to identify the hard instances that make a language hard. It provides simple characterizations for many fundamental complexity-theoretic properties. The second method defines instance complexity by restricting the allowed algorithms. For example, "Instance p_1 is harder than p_2 if it takes more time for a specified algorithm A to solve p_1". The third method considers instance classes instead of single instances (Mannila 1985). It defines a subproblem of the original problem by some intuitive criteria of instance hardness (or easiness)[1] and then studies the worst-case complexity of the subproblem. Algorithms are designed and analyzed on these subproblems, with resource requirements increasing smoothly when moving to larger subproblems. The resulting algorithm is called optimal to the instance hardness measures used. Although the first method using Kolmogorov complexity has produced some interesting results, these results are mainly along the line of complexity classes. They do not help much on practical instance-based algorithm selection. The second method is not very attractive because it depends on a particular algorithm. The third method has made some progresses in designing adaptive sorting algorithms that are optimal to many measures of presortedness. However, because of its analytical nature, this method is easy for simple problems like sorting, but hard for arbitrary $\mathcal{NP}$-hard optimization problems. Also, it is usually very difficult to design one adaptive algorithm that is optimal for many instance hardness measures.

Because of the above drawbacks of the analytical approach, experimental algorithmics has drawn significant attention in the last decade (McGeoch 1986; Hooker 1994; Johnson 2002; Moret 2002). The experimental approach has the advantage of focusing on

[1] For example in sorting, the presortedness of permutation, be it inversions, runs, or the length of the longest ascending subsequence, can serve as instance hardness measure.

typical instances. It also allows us to conduct instance-based algorithm comparison and selection. This is especially useful to the selection of $\mathcal{NP}$-hard optimization algorithms that are hard to analyze theoretically. With the progresses made in AI and machine learning, more advanced data analysis techniques have been applied to experimental algorithmics. In this paper, we propose a machine learning-based approach to build a practical algorithm selection system for $\mathcal{NP}$-hard optimization problems. We use the MPE problem in Bayesian network inference as a case study to validate the proposed approach. The rest of this paper is organized as follows. In Sect. 2 we describe our approach and review related works. In Sect. 3 we introduce Bayesian networks and the MPE problem. In Sect. 4 we discuss experimental setups and training data collection. Section 5 presents results of model induction and system evaluation. Finally in Sect. 6, we conclude with some discussions and future directions.

2 A machine learning-based approach to algorithm selection for $\mathcal{NP}$-hard optimization problems and the related works

Our approach is fairly straightforward: first collect experimental data about problem characteristics and algorithm performance, and then apply machine learning to induce predictive algorithm selection models from the training data. We are partly motivated by the observation that some easy-to-compute instance features can be used as good indicators of some algorithm's performance on the specific class of instances. This knowledge can help us to select the best algorithm for these instances. In $\mathcal{NP}$-hard problem-solving, researchers have long noticed that algorithms exploiting special problem instance features can perform on the particular class of instances better than the worst-case scenario. In light of this, two of the main directions of this work are to study different instance features in terms of their goodness as a predictive measure for some algorithm's performance, and to utilize this knowledge to build a practical algorithm selection system.

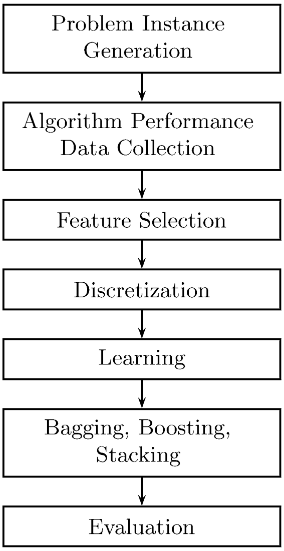
2.1 The proposed machine learning approach

The questions raised in the beginning of this paper can be formulated as two machine learning tasks. The first one is to learn the concept of "exactly solvable". The second one is to learn "which approximate algorithm is the best". Both are classification tasks. In the first task, there are two classes, "yes" or "no". In the second task, the number of classes is equal to the number of candidate algorithms.

Both tasks can be handled in a similar manner by the procedure shown in Fig. 1. The first two steps, instance generation and algorithm performance data collection, prepare the training data. The next two steps, feature selection and discretization, preprocess the data. Next in the learning step, machine learning algorithms are applied to induce the algorithm selection classifiers. Also, some meta-learning methods—such as bagging, boosting, and stacking (Witten and Frank 1999)—can be used to improve learning. Finally, the best learned model is chosen and evaluated on test data.

Each of the above steps has some important research problems to solve. In data preparation, we need to identify a list of candidate algorithms and a set of good problem features that can differentiate the algorithm space. Here domain knowledge plays an important role. In instance generation, we want to generate problem instances with controlled parameter values uniformly at random, which is a challenging problem itself. In data preprocessing, we want to preselect the most relevant feature subset to reduce the complexity of learning.

Fig. 1 A machine learning-based approach to algorithm selection


Problem Instance Generation
Algorithm Performance Data Collection
Feature Selection
Discretization
Learning
Bagging, Boosting, Stacking
Evaluation

In the step of learning, we need to compare multiple models and decide which is the best. In evaluation, the task is to verify that the learned models can indeed select the right algorithm and provide a better overall performance.

2.2 Related works

Related works can be roughly classified into three categories: experimental algorithmics, algorithm selection, and the application of machine learning to study of algorithms.

During the last decade experimental algorithmics has drawn significant attention in the algorithm community (Hooker 1994; Sanders 2002; Ramakrishnan and Valdes-perez 2000; Moret 2002; Johnson 2002). The experimental method is especially useful to study algorithms for $\mathcal{NP}$-hard optimization problems: Gent and Walsh (1993) presents an empirical analysis of search in GSAT; Hoos and Stutzle (2000) gives an experimental study to stochastic local search for SAT; Rardin and Uzsoy (2001) discusses methodological issues on experimental evaluations of heuristics. Most of these researchers only use some simple statistical methods to analyze experimental data.

The algorithm selection problem was first formulated in (Rice 1976). The goal is to select one among a set of algorithms that solves the input instance the best in some sense. An abstract model of algorithm selection is shown in Fig. 2, where x is the input instance in the problem space and w is the performance criteria. The input problem instance is represented as the feature(s) f in the feature space by a feature extraction procedure $F(x)$. The task is to build a selection mapping between the problem space S and the algorithm space A that provides a good (measured by w) algorithm to solve x subject to the constrains that algorithm performance is optimized. Works in this direction include the followings: Lucks and Gladwell (1992) discusses automated selection of mathematical softwares; Houstis et al. (2000) discusses a knowledge-based system to select scientific algorithms; Fink (1998) describes a statistical technique for the automated selection of problem-solving methods; Lobjois and Lema (1998) studies the selection of branch and bound algorithms; Lagoudakis et al. (2001) applies reinforcement learning to dynamic selection of

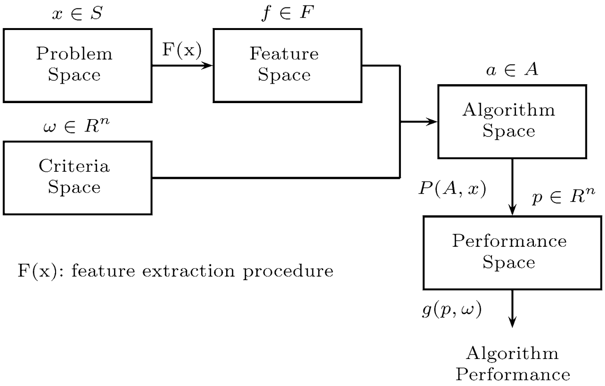


Fig. 2 The abstract model of algorithm selection

sorting algorithms in which the algorithm selection problem is modeled as a Markov Decision Process. His work demonstrates that learning and optimization methods can be effectively used to cope with uncertainty in computation. He also applied machine learning to select branching rules in the DPLL procedure for SAT (Lagoudakis and Littman 2001; Leyton-Brown et al. 2003a) presents a portfolio approach to algorithm selection. Comparing to these works, our research is the first that applies machine learning to algorithm selection to integrate both exact and approximate algorithms for $\mathcal{NP}$-hard optimization problems.

Our work has been developed from the study of uncertain reasoning under bounded resources, which is crucial for many real-time AI applications. Examples of these include online diagnosis, crisis monitoring, and real-time decision support systems. In these tasks the correctness of a computation depends not only on its accuracy but also on its timeliness. Some mission-critical applications require a hard computation deadline to be strictly enforced where the utility drops to zero instantly if the answer to the query is not returned and a control is not produced. Other soft real-time domains only admit a soft deadline where the utility degrades gradually after the deadline is passed.

In the past 20 years Bayesian networks have become the most popular models for uncertain reasoning. Researchers have broadly developed two types of methods to address real-time inference in Bayesian networks. The first method is to use anytime algorithms (Zilberstein 1993), or flexible computation (Horvitz 1990). These are iterative refinement algorithms that can be interrupted at "any" time and still produce results of some guaranteed quality. Most stochastic simulation, heuristic search, and partial evaluation inference algorithms belong to this category. The second method is to combine multiple different inference algorithms where each of these may be more or less appropriate for different characteristics of the problems. The architecture unifying various algorithms often contains a key meta-reasoning component that partitions resources between meta-reasoning and reasoning in order to gain a better overall performance of problem solving. Works in this category include intelligent reformulation (Breese and Horvitz 1990), algorithm portfolio (Gomes and Selman 1997), cooperative inference (Santos et al. 1995), etc. This paper is concerned with a specific type of meta-reasoning, namely *algorithm selection*, for the real-time MPE problem with a soft deadline. We use a learning-based approach to induce an MPE algorithm selection model from the training data. The learning needs to be done only once and it takes only about thirty minutes. Then the learned models are available to anyone as an MPE algorithm selection meta-reasoner. For an input MPE instance, it can inspect the instance's features and select the best algorithm in only a few seconds, and achieve the best overall performance of reasoning.

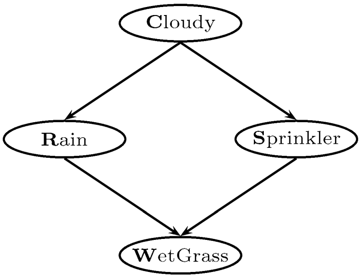
Our work is partly motivated by Santos et al. (1995) and Horvitz et al. (2001). In Santos et al. (1995), all algorithms to be included into the system must have both anytime and anywhere characteristics. In our approach, candidate algorithms do not have to have the anywhere property. Horvitz et al. (2001) applies a Bayesian method to induce a meta-reasoning model to help decision-making in hard problem solving. We compare various machine learning techniques to learn an algorithm selection meta-reasoner that can integrate various algorithms and gain a better overall performance of reasoning. Another motivation of this work is to automate and mimic human expert's algorithm selection process. The difficulty of automatic algorithm selection is largely due to the uncertainty in the input problem space and the lack of understanding of the algorithms' working mechanism. This is especially true for $\mathcal{NP}$-hard problems and complex, randomized algorithms. From the viewpoint of expert systems and machine learning, the automatic algorithm selection system acts as an intelligent meta-reasoner that can learn the uncertain knowledge of algorithm selection from its past experiences of problem solving, and can use this learned knowledge (models) to reason on algorithm selection for the input instance in order to make the right decision.

A similar method has been independently developed in Leyton-Brown et al. (2002, 2003a, 2003b). The differences of their work and this work are analyzed in the following. First, their work has been developed from studying the empirical hardness of the WDP problem, while ours has been developed from the investigation of using algorithm selection as a meta-reasoning strategy for the real time MPE problem. Second, they consider only exact algorithm and focus on how to predict the running time of the algorithm. We consider both exact and inexact algorithms and emphasize on how to integrate these algorithms to build a practical algorithm selection system. Third, they investigate regression models, while we mainly study classifiers. In Leyton-Brown et al. (2003b), the authors argue that regression models are more suitable for algorithm selection than classifiers because of the following reason:

> … Any standard classification algorithm (e.g., a decision tree) could be used to learn which algorithm to choose given features of the instance and labelled training examples. The problem is that such classification algorithms use the wrong metric: they penalize misclassifications equally regardless of their cost. We want to minimize a portfolio's average runtime, not its accuracy in choosing the optimal algorithm. Thus we should penalize misclassifications more when the difference between the running times of the chosen and fastest algorithm is large than when it is small…

The above argument is not very convincing for the following reasons. First, classifiers can also penalize misclassifications differently if being used properly. Second, when misclassification happens in selection of competing approximate algorithms, often times the performance difference between the incorrectly selected algorithm and the actual best algorithm is small. We will revisit this point later in Sect. 5.5. Also, although regression models might be a better choice than classifiers for running time prediction of exact algorithms, it is still open whether this is also true to algorithm selection for both exact and inexact algorithms. In the MPE problem, the most popular exact algorithm not only has an exponential time complexity, but also an exponential space complexity. Often times when the exact algorithm fails, it stops and throws out an "out of memory" error. We need to capture this using the first classifier, and then start another selection of approximate algorithms using the second classifier. Also when comparing two inexact algorithms, one needs to consider both the running time and the solution quality. These may cause some trouble for a regression model of running-time prediction. In summary, Leyton-Brown's work and this work belong to the

P(C=T)	P(C=F)
0.5	0.5



C	P(R=T)	P(R=F)
T	0.8	0.2
F	0.2	0.8

C	P(S=T)	P(S=F)
T	0.1	0.9
F	0.5	0.5

R S	P(W=T)	P(W=F)
T T	0.99	0.01
T F	0.9	0.1
F T	0.9	0.1
F F	0.0	1.0

Fig. 3 The Sprinkler network

same research trend of applying machine learning to experimental algorithmics for $\mathcal{NP}$-hard problems. In the future it is necessary to conduct a thorough experimental comparison between regression models and classifiers for algorithm selection. Other related works include (Gomes and Selman 1997; Jitnah and Nicholson 1998; Hoos and Stutzle 1998; Ruan et al. 2004).

3 Bayesian networks and the MPE problem

A Bayesian network (BN) is a Directed Acyclic Graph (DAG) where nodes represent random variables and edges represent conditional dependencies between variables (Pearl 1988). Each node has a Conditional Probability Table (CPT) that describes the conditional probability distribution of that node given its parents' states. The distributions can be discrete or continuous. For simplicity we only consider discrete ones. BNs represent joint probability distributions in a compact manner. Let $\{X_1, \ldots, X_n\}$ be the random variables in a network. Every entry in the joint distribution $P(X_1, \ldots, X_n)$ can be calculated using the following chain rule:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\pi(X_i)), \tag{1}$$

where $\pi(X_i)$ denotes the parents of node X_i. Figure 3 shows a simple BN with 4 nodes, the Sprinkler network (Russell and Norvig 2003).

Let (G, P) be a Bayesian network where G is a DAG and P is a set of CPTs, one for each node in G. Evidence $\mathbf{E}$ is a set of instantiated nodes with observed value $\mathbf{e}$. An explanation is a complete assignment of all nodes' values consistent with the observed evidence. Each explanation's probability can be computed in linear time using (1).

MPE is an explanation with the highest probability. It provides the most likely state of the world given the observed evidence. The MPE problem has a number of applications in diagnosis, abduction and explanation. Unfortunately, it is NP-Complete (Littman 1999). Approximating the MPE is also $\mathcal{NP}$-hard (Abdelbar and Hedetniemi 1998). So approximate algorithms must be applied to large, densely connected networks.

Clique-tree propagation is the most popular exact algorithm to compute the MPE (Lauritzen and Spiegelhalter 1988; Jensen et al. 1990; Shafer and Shenoy 1990). It exploits the factorization of the joint distribution that the Bayesian networks provide to reduce the computation needed in probabilistic inference. Clique-tree propagation is efficient for sparse networks but can be very slow if the network is dense and complex. In fact, both its worst-case time complexity and space complexity are exponential in the induced width (also known as the largest clique size) of the underlying undirected graph.

Approximate MPE algorithms trade accuracy for efficiency so that they can at least find a near-optimal explanation in a reasonable amount of time on intractable networks. There are two basic categories of approximate algorithms: stochastic sampling algorithms and search-based algorithms. Stochastic sampling algorithms can be divided into importance sampling algorithms (Fung and Chang 1989) and Markov Chain Monte Carlo (MCMC) methods (Pearl 1988). They differ from each other in whether samples are independent from each other or not. Both algorithms can be applied on a large range of network sizes. But when the network is large and evidence very unlikely, the most probable explanation will also be very unlikely. Thus the probability of it being hit by any sampling schemes will be rather low. This is the main weakness of sampling algorithms. Search algorithms have been studied extensively in combinatorial optimization. Researchers have applied various optimization algorithms to solve MPE, for example, the best first search (Shimony and Charniak 1999), linear programming (Santos 1991), stochastic local search (Kask and Dechter 1999), genetic algorithms (Mengshoel 1999), etc. More recently, Park (2002) tried to convert MPE to MAX-SAT, and then use an MAX-SAT solver to solve it indirectly. Other search algorithms often use some heuristics to guide search in order to avoid getting stuck into local optimal. The most popular heuristics include Stochastic HillClimbing, Simulated Annealing (Kirkpatrick et al. 1983), Tabu Search (Glover and Laguna 1997), etc. Each of these may work well on some but poorly on other MPE instances. Under real-time constraints, it would be very helpful if we could know in advance which algorithm is the best for what instances.

4 Experimental setup and data preparation

In the following we present experimental results of applying our proposed method to algorithm selection for the MPE problem. This section discusses experimental setup and data preparation.

4.1 The algorithm space

Our candidate MPE algorithms include one exact algorithm: Clique-Tree Propagation (CTP) (Lauritzen and Spiegelhalter 1988); two sampling algorithms: Gibbs Sampling (Pearl 1988) and Forward Sampling (also called Likelihood Weighting) (Fung and Chang 1989); two local search-based algorithms: Multi-start HillClimbing and Tabu Search (Glover and Laguna 1997); and one hybrid algorithm combining both sampling and search: Ant Colony Optimization (ACO). These algorithms are chosen because currently they are among the most commonly used MPE algorithms. A classification of these algorithms is shown in Fig. 4. Because of the lack of space, we refer interested readers to (Guo 2003) for detailed descriptions.

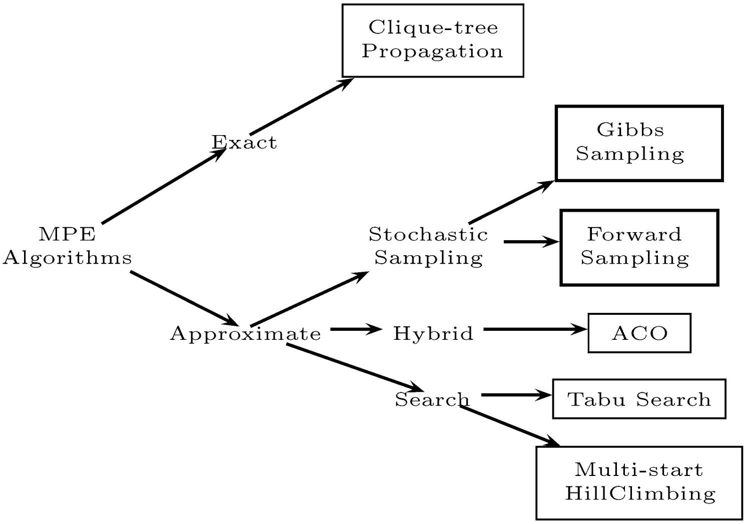


Fig. 4 Candidate MPE algorithms

4.2 The instance feature space

An MPE instance $\langle G, P, E\rangle$ consists of three components: the network structure G, the CPTs P, and the evidence E. Correspondingly, we will consider three different kinds of instance features: network characteristics, CPT characteristics, and evidence characteristics.

Network characteristics include *topological type* and *network connectedness*. We distinguish three topological types: polytrees, two-level networks (Noisy-OR), and multiply connected networks. Network connectedness, or *conn*, is simply calculated as $conn = \frac{n\_arcs}{n\_nodes}$. These two characteristics have a direct influence on the exact inference algorithm's performance. In contrast, none of them affects sampling algorithms.

CPT characteristics include *CPT size* and *CPT skewness*. Since we only consider binary nodes, the maximum number of parents of a node, *max_parents*, can be used to bound the CPT size. The skewness of the CPTs is computed as follows (Jitnah and Nicholson 1998): for a vector (a column of the CPT table), $v = (v_1, v_2, \dots, v_m)$, of conditional probabilities,

$$skew(v) = \frac{\sum_{i=1}^{m} |\frac{1}{m} - v_i|}{1 - \frac{1}{m} + \sum_{i=2}^{m} \frac{1}{m}}, \tag{2}$$

where the denominator scales the skewness from 0 to 1. The skewness of a CPT is the average of the skewness of all columns, whereas the skewness of the network is the average of the skewnesses of all CPTs. We will see that CPT skewness has the most significant influence on the relative performance of sampling and search-based algorithms.

Evidence characteristics include the *proportion* and the *distribution type* of evidence nodes. Evidence proportion is simply the number of evidence nodes, *n_evid*, divided by *n_nodes*: $\frac{n\_evid}{n\_nodes}$. Usually, more evidence nodes implies less likely evidence. Hence, the MPE will also be quite unlikely and the probability of it being hit by any sampling scheme can not be very high. The distribution of evidence nodes in the network also affects the hardness of MPE instances. If most evidence nodes are "cause" nodes, the problem is called

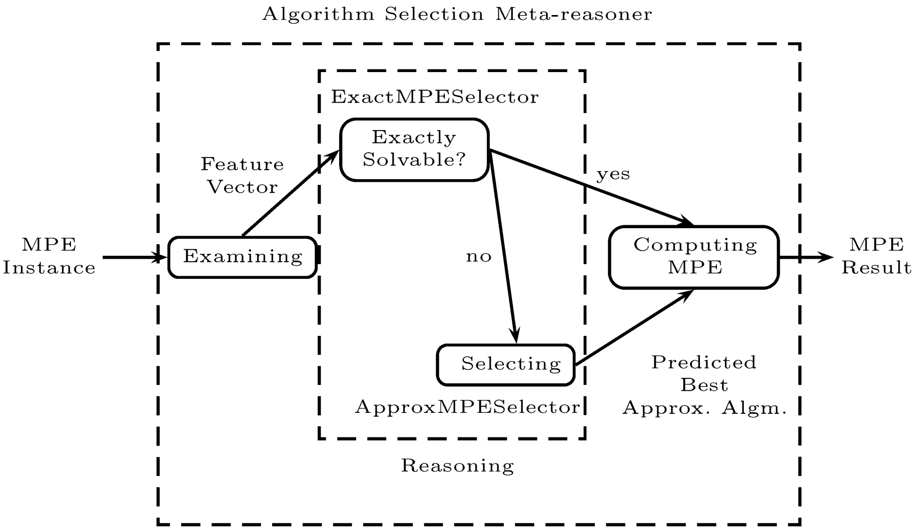


Fig. 5 The algorithm selection meta-reasoner

predictive reasoning. If most evidence nodes are "effect" nodes, it is called *diagnostic reasoning*. It has been proven that predictive reasoning is easier than diagnostic reasoning (Shimony and Domshlak 2003). In our experiments, we will consider three different types of evidence distributions: strictly predictive, strictly diagnostic, and randomly distributed. An inference problem is called "strictly predictive" if all evidence nodes have no non-evidence parents; it is called "strictly diagnostic" if all evidence nodes have no non-evidence children.

We are aware that there might exist some other features that could work as well or even better. These particular features were chosen according to the domain knowledge, previous literature (Jitnah and Nicholson 1998; Ide and Cozman 2002; Shimony and Domshlak 2003), and our initial experimental results. Another practical reason is because they can all be calculated in polynomial time.

4.3 The algorithm selection meta-reasoner to be learned

Our first goal is to identify the class of MPE instances for which the exact inference algorithm is applicable. When the exact algorithm is not applicable (most probably due to an out-of-memory error in practice), we need to look at various approximate algorithms. So our second goal is to learn the predictive model that can determine which approximate algorithm is the best. Therefore, the algorithm selection meta-reasoner to be learned will consist of two classifiers as shown in Fig. 5: the *ExactMPESelector* for exact algorithm selection, and the *ApproxMPESelector* for approximate algorithm selection.

4.4 Data preparation

In data preparation, we need to first generate MPE instances with different characteristics. The random generation of MPE instances with controlled parameter values is based on a Markov chain method (Ide and Cozman 2002). It is reasonable and necessary to consider only a subset of all possible MPE instances, i.e., the set of "Real World Problems"

(RWP). In order to simulate RWP BNs, we first extract the ranges of all characteristic parameter values from a collection of 13 real world samples, call it D_{RWBN}, and then generate networks and MPE instances based on the extracted distributions. These networks are quite different from each other and they are representatives of what BNs are used for in the real world. The ranges of their characteristic values are as follows: $30 \leq n\_nodes \leq 1{,}000$; $conn \in [1.0, 2.0]$; $maxParents < 10$; $0.25 < skewness < 0.87$. These characteristics information will be used to guide the generation of our training datasets.

The first training dataset, D_{MPE1}, will be used to learn *ExactMPESelector*. It is generated as follows: we first randomly generate networks with varying connectedness from 1.0 to 2.0 and maximum number of parents varying from 3 to 10. The number of nodes used are $\{30, 50, 80, 100, 120, 150, 200\}$. We then run exact algorithm CTP on these networks and record the performance. To perform inference, CTP first compiles the network into a clique tree, and then passes messages(probability functions) between nodes of the clique tree. We first record the maximum clique size if the compilation phase is successful. Otherwise, if it fails with an out-of-memory error, we label the network as "no" instance. If the compilation can be completed but takes longer than 5 minutes, we also label the instance as "no". According to our experimental results on a Pentium III 1 GHz machine with 512 MB of RAM, the largest clique size that the exact algorithm could handle was around 22. When the largest clique size was less than or equal to 20, the construction of the clique tree could be completed within 5 minutes. When it was 21 or 22, it took quiet a long time to construct the clique tree and sometimes failed. So we decided to use a conservative cutoff (5 minutes) to avoid a "no" instance being misclassified as "yes". D_{MPE1} has four numeric attributes: *n_node*, *topology*, *connectedness*, and *maxParents*. The target class, *ifUseExactAlgorithm*, takes boolean values representing whether an exact algorithm is applicable or not. The final D_{MPEI} contains a total of 1,893 instances.

The second training dataset for learning *ApproxMPESelector*, D_{MPE2}, only contains two-level and multiply networks. We generate a set of networks with different characteristic values and then run all approximate algorithms on them with different evidence settings. We allow each algorithm to run for a fixed number of samples (for random sampling algorithms) or a fixed number of search points (for search algorithms), i.e. the number of times that the search space is visited or (1) is called. In our implementations of all search and sampling algorithms, the same subroutine is used to evaluate each search point or sample so that all algorithms can roughly spend the same amount of time generating each sample or evaluating each search point. We label the instance using the best algorithm that returns the best MPE value. If the returned MPE probabilities of two algorithms are equal, we select the one uses less number of samples to find the MPE as the better one. The total number of samples or search points used was 300, 1000, or 3,000. The resulting training dataset contains 5,184 instances generated from 192 networks.[2] It has 8 predictive attributes: *n_node*, *topology*, *connectedness*, *maxParents*, *skewness*, *evidPercent*, *evidDistri*, and *n_samples*. The target class is the best algorithm for this instance. The format of D_{MPE2} is shown in Table 1. The statistics of D_{MPE2} are listed in Table 2. We can see that Gibbs sampling has never been the winner while ant colony optimization algorithm is the best for nearly half of the instances.

[2] $5184 = 192 \times 3 \times 3 \times 3$. These 3s are for three different evidence proportions, evidence distribution types, and number of samples allowed on each network.

Table 1 Format of training dataset D_{MPE2}

#nodes	topology	conn	maxParents	skewness	evid%	evidDist	#samples	bestAlgo
50	multiply	4.56	9	0.1	10	predictive	300	multi_hc
50	multiply	4.56	9	0.1	10	random	300	aco
...	...	...	...	...	...	...	...	...
100	multiply	3.80	8	0.5	30	diagnostic	3,000	aco
100	multiply	3.80	8	0.5	30	random	3,000	aco

Table 2 Statistics of attribute values in D_{MPE2}

	#nodes	conn	maxParents	skewness	evid%	#samples
Minimum	50	1.19	3	0.09	10	300
Maximum	100	4.88	10	0.90	30	3,000
Mean	75	2.49	5.64	0.50	20	1,433
StdDev	25	1.26	2.43	0.33	10	1,144

label	topology		evidDist		
	multiply	twolevel	predictive	diagnostic	random
count	3,240	1,944	1,728	1,728	1,728

	bestAlgorithm				
	gibbs_sampling	forward_sampling	multiHC	tabu	aco
count	0	862	1,077	578	2,667
percentage	0%	16.62%	20.78%	11.15%	51.45%

5 Experimental results: model induction and evaluation

We now apply various machine learning algorithms to induce the predictive algorithm selection models. We consider three different kinds of models: decision tree learning (C4.5), naive Bayes classifier, and Bayesian network learning (K2) (Cooper and Herskovits 1992). We also consider three meta-learning methods: bagging, boosting and stacking, which all use C4.5 as the base learner. So totally we have six different learning schemes. Before learning, we also conduct necessary data preprocessing such as feature selection and/or discretization.

5.1 Experiment 1: learning ExactMPESelector

In experiment 1, we run all 6 learning schemes on D_{MPE1} to learn the exact algorithm selector *ExactMPESelector*. Table 3 shows the classification accuracies of each learned model. The first two columns show the classification accuracy (c.a.) and its standard deviation (s.d.) of each learned model on the training data. It is computed by 10 ten-fold cross validations.

Table 3 Experiment 1: learning *ExactMPESelector*

	C45	NaiveBayes	BayesNet	Bagging	Boosting	Stacking
c.a. (%)	94.80	82.79	90.06	94.75	94.81	94.56
s.d. (%)	0.27	0.36	0.24	0.25	0.23	0.45
c.a. (%)	98.8	88.8	92.2	98.8	98.8	98.8

In a k-fold cross validation, the training data is randomly divided into k mutually exclusive subsets of approximately equal size. In each subset, the class is represented in approximately the same proportions as in the whole data set. The learning algorithm is executed and the learned model tested k times. For each iteration, one subset is held out as test set and the remaining $k-1$ subsets are used for training. Finally, the k estimates are averaged to yield the overall classification accuracy.[3] We also compute the classification accuracy of each learned model on a separated test dataset of 1,000 instances. The results are shown in the third column of Table 3. This dataset is generated from 500 different networks independent of the training data. The random generation method is analog to procedure described in Sect. 4.4, but different distributions of feature values are used.

We choose the best model out of these 6, i.e. the one that has both high classification accuracy and efficient reasoning mechanism. We can see that C4.5, bagging, boosting, and stacking all have a high classification accuracy. We also notice that NaiveBayes has the worst performance, which verifies that the features in D_{MPE1} are not independent of each other. Since C4.5 is much simpler and more efficient on reasoning, we use the decision tree learned by C4.5 as the best model for *ExactMPESelector*.

The learned decision tree is shown in Fig. 6. We can see that the basic rule for exact algorithm selection is that exact clique-tree propagation algorithm is applicable if the network is small or sparse.

5.2 Experiment 2: algorithm space reduction & feature selection

In the following experiments, we will look at approximate MPE algorithm selection. The training dataset used is D_{MPE2}. Each instance of D_{MPE2} has 9 attributes. The first 8 are predictive attributes and the last one is the target class attribute which labels the best approximate algorithm for this instance.

Because Gibbs sampling has never been the best algorithm in the training dataset D_{MPE2} (see Table 2), we can remove it from the candidate algorithms. We then apply a GA-wrapped C4.5 feature selection classifier to see which feature subset is the best (Witten and Frank 1999). The wrapper uses C4.5 as the evaluation classifier to evaluate the fitness of feature subsets. A simple genetic algorithm is used to search the attribute space. Both the population size and number of generations are 20. The crossover probability is 0.6 and the mutation probability is 0.033. The features selected by the GA are: {*n_node, skewness, evidPercent, evidDistri, n_samples*}. In the following experiments, we will use this selected feature subset to learn *ApproxMPESelector*.

[3] Often times, a single stratified k-fold cross-validation might not produce a reliable estimate, so we typically run cross-validation many times and average the results. In this experiment we run the ten-fold cross validation for 10 times.

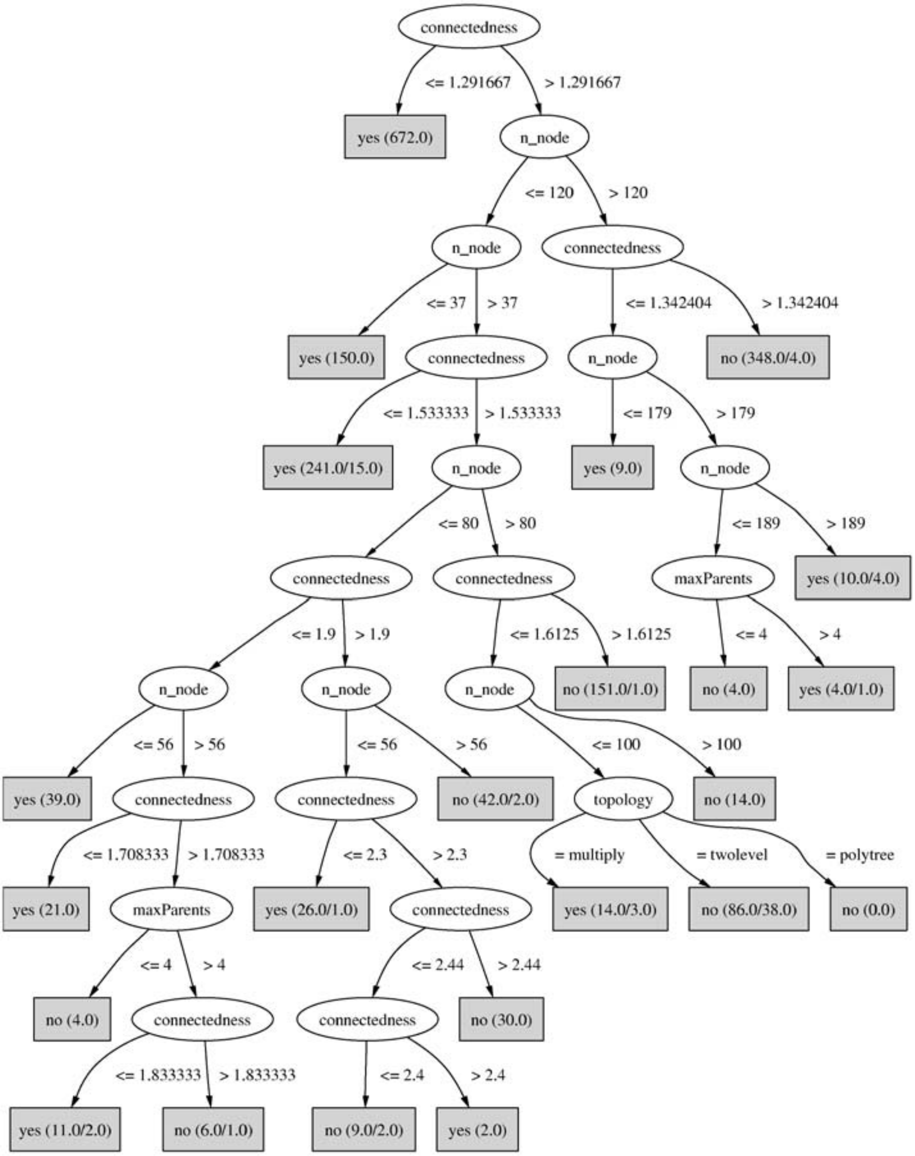


Fig. 6 ExactMPESelector: the learned decision tree for exact MPE algorithm selection

5.3 Experiment 3: learning ApproxMPESelector

In experiment 3, we run the same 6 learning schemes on the selected subset of D_{MPE2} to see which learns the best model. The experimental results are shown in Table 4. Again, the first two columns show the classification accuracy and its standard deviation of each learned model on the training dataset, and the third column shows the classification accuracy on an independently generated testing dataset of 1,000 instances. We can see that the model induced by C4.5 has the highest classification accuracy of 77.75%, Naive Bayes classifier

Table 4 Experiment 3: learning *ApproxMPESelector*

	C4.5	NaiveBayes	BayesNet	Bagging	Boosting	Stacking
c.a. (%)	77.75	72.77	76.08	75.44	77.16	77.36
s.d. (%)	0.23	0.03	0.01	0.27	0.26	0.32
c.a. (%)	74.2	70.8	65.6	68.6	68.8	72.6

has the worst performance, and the classification accuracy of Bayesian networks learning is 76.08%. According to these results, we choose the decision tree as the best model for *ApproxMPESelector*.

The learned decision tree is shown in Fig. 7 in which "sk" represents "skewness", "ns" represents "number of samples", "nn" represents "number of nodes", "ed" represents "evidence distribution", and "ep" represents "evidence percentage". We can see that skewness is an important feature because it differentiate the algorithm space most significantly. In general, on unskewed networks search-based algorithms outperform ACO and sampling-based algorithms; on medium skewed networks ACO basically dominates; and on highly skewed networks ACO generally outperforms other algorithms, but forward sampling is competitive.

This interesting finding reveals ACO's nature as a combination of sampling and search. The sampling part is that each individual ant can use CPTs as heuristic functions to explore new trails. The search part is that a colony of ants can exchange information through pheromone trails so as to cooperatively "learn" the best solution. Basically, if we set the pheromone weight α to 0, then ACO becomes forward sampling, because it only uses CPTs as the heuristic functions when generating ant trails (samples). With the use of pheromone trails ($\alpha \neq 0$), ACO manages to outperform forward sampling on both skewed and medium networks. As the skewness decreases, the number of local optima increases and the instance becomes more difficult for sampling algorithms, while simple search heuristic like random restart will have more chances to explore new areas in the solution space. That is why search algorithms outperform ACO on unskewed networks. This result implies that as a combination of sampling and search, ACO's search aspect is weaker than its sampling aspect. It also suggests a possible way to improve ACO. If we can detect that the skewness is low in advance, then we can change ants' strategy to favor exploration more than exploitation in order to gain a better performance. For more detailed descriptions of ACO for the MPE problem, see (Guo et al. 2004).

5.4 Experiment 4: influences of each individual feature

Experiment 4 studies the influences of each individual feature on the relative performance of these algorithms. We partition the training dataset by each feature's values and record the number of times of each algorithm being the best at each feature value level (Table 5). The results are summarized as follows:

1. *Number of nodes n_nodes* affects the relative performance of two search algorithms more significantly. When *n_nodes* increases from 50 to 100, Multi-start HillClimbing becomes the best algorithm more frequently and the chances for tabu search being the best drops. This can be explained by the constant size of the tabu list used. When network becomes larger while the tabu list remains the same, the tabu list's influence becomes weaker. This makes it loses to multi-start HillClimbing more times.

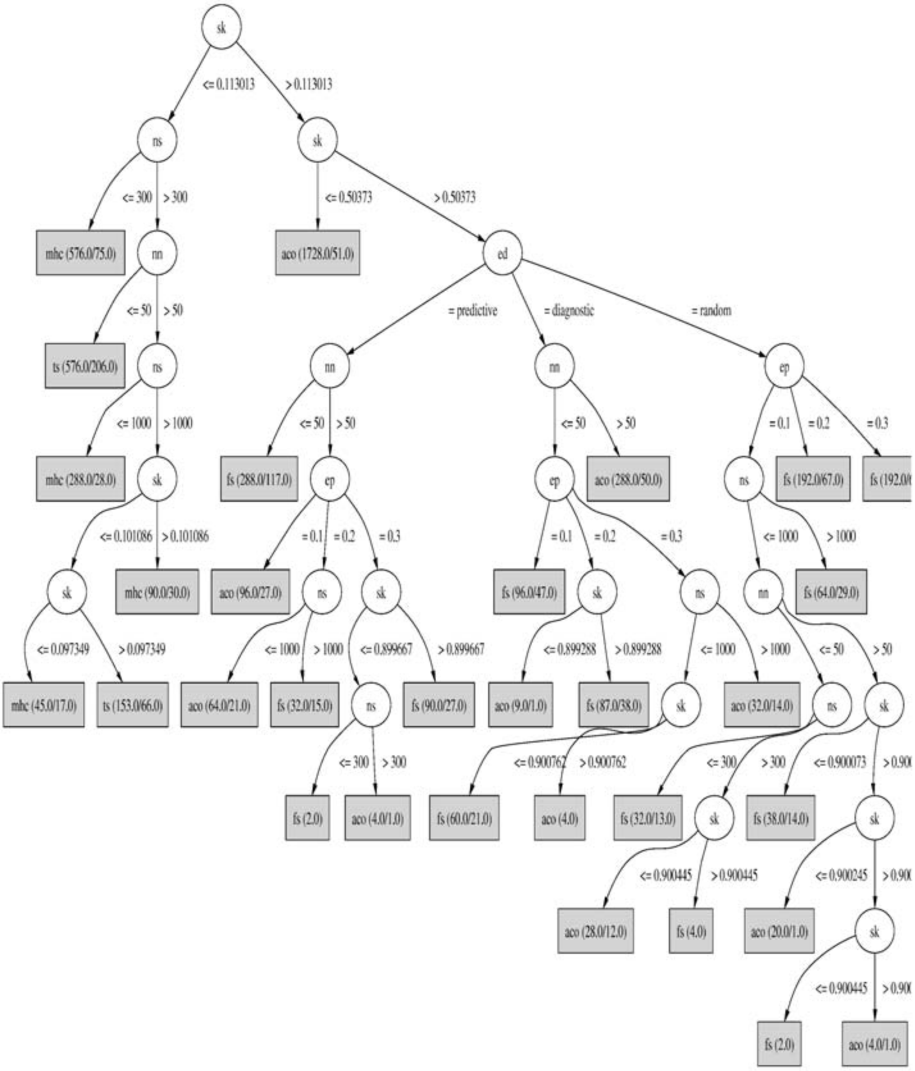


Fig. 7 ApproxMPESelector: the learned decision tree for approximate MPE algorithm selection

2. *Number of samples* Again, the relative performances of two search algorithms are affected significantly, but forward sampling and ACO's are not. When the given number of samples increases from 300 to 1,000 to 3,000, Tabu search becomes the best algorithm more often and Multi-start HillClimbing loses its top rank. It seems that Tabu search can utilize available search points better than Multi-start HillClimbing.
3. *CPT skewness* Skewness has the most significant influence on the relative performance of these algorithms as shown in Table 5. When the skewness is low, search algorithms (Multi-start HillClimbing & Tabu search) perform much better than sampling algorithms. Most times Multi-start Hillclimbing wins, and forward sampling never wins. When the skewness is around 0.5, ACO outperforms all other algorithms almost all the time. When

the skewness increases to 0.9, forward sampling and ACO are the winners and perform equally well. We also notice that forward sampling works well only for highly skewed networks and ACO works for both highly-skewed networks and medium-skewed networks.

4. *Evidence proportion* The result shows that changing evidence percentage does not affect two search algorithms' relative performance, but it affects forward sampling and ACO. ACO is outperformed by forward sampling as the percentage of evidence nodes increases from 10% to 30%. This can be explained as follows. When the evidence proportion increases, the likelihood of the evidence becomes smaller. Thus the probability of the MPE being hit by any sampling schemes is low. ACO performs better than forward sampling become it has a search component. We should also note that evidence percentage's influence is much weaker than that of skewness.
5. *Evidence distribution* The relative performance of Multi-start HillClimbing is not affected. Tabu search is only slightly affected. It shows that diagnostic inference is relatively hard for forward sampling but easy for ACO. In contrast, random distributed evidence is relatively hard for ACO but easy for forward sampling.

5.5 Experiment 5: evaluating the MPE algorithm selection system

In this experiment, we evaluate the learned models on both synthetic and real world networks. The two synthetic test datasets, $D_{MPETest1}$ and $D_{MPETest2}$, are randomly generated from different networks independent of the training datasets. The random generation method used is the same as that of the training datasets, but the distributions of the parameter values are different. Larger networks, different connectedness and skewness values, and different number of samples have been used.

System evaluation on synthetic networks $D_{MPETest1}$ contains 1,000 instances for *exactMPESelector* randomly generated from 500 different synthetic networks. The learned exact algorithm selector *exactMPESelector*'s classification accuracy on $D_{MPETest1}$ is 99.8%. It only misclassifies two "yes" instances as "no".

$D_{MPETest2}$ contains 1,000 approximate MPE instances generated from another set of 500 synthetic networks. The learned approximate MPE algorithm selector *ApproxMPESelector*'s classification accuracy on $D_{MPETest2}$ is 69.4%. The confusion matrix[4] is as follows:

$$\left(\begin{array}{llllll} a & b & c & d & <-- & \textit{classified as} \\ 56 & 0 & 0 & 90 & | & a = \textit{forward\_sampling} \\ 0 & 94 & 20 & 1 & | & b = \textit{multi\_hc} \\ 6 & 6 & 36 & 7 & | & c = \textit{tabu} \\ 172 & 0 & 4 & 508 & | & d = \textit{aco} \end{array}\right).$$

From the confusion matrix we can see that the majority of misclassifications happens between *forward_sampling* and *aco*. By carefully examining the experimental results

[4] In the confusion matrix, all of the columns represent the predicted classes, and thus a piece of data belongs to the column if it is classified as belonging to this class. The rows represent the actual classes, and a piece of data is thus represented in a particular row if it belongs to the corresponding class. A perfect classification results in a matrix with 0's everywhere but on the diagonal. A cell which is not on the diagonal but has a high count signifies that the class of the row is somewhat confused with the class of the column by the classification system.

Table 5 Partitioning D_{MPE2} by each feature's value

n_Nodes	Number of times of being best algorithm							
	forward_sampling		multiHC		tabu		aco	
50	496	9.57%	380	7.33%	439	8.47%	1,277	24.63%
100	366	7.06%	697	13.45%	139	2.68%	1,390	26.81%

n_Samples	Number of times of being best algorithm							
	forward_sampling		multiHC		tabu		aco	
300	274	5.28%	505	9.74%	7	0.14%	942	18.17%
1000	286	5.52%	372	7.18%	174	3.36%	896	17.28%
3000	302	5.83%	200	3.86%	397	7.66%	829	15.99%

skewness	Number of times of being best algorithm							
	forward_sampling		multiHC		tabu		aco	
0.1	0	0.00%	1059	20.43%	512	9.88%	157	3.03%
0.5	4	0.08%	9	0.17%	38	0.73%	1677	32.35%
0.9	858	16.55%	9	0.17%	28	0.54%	833	16.07%

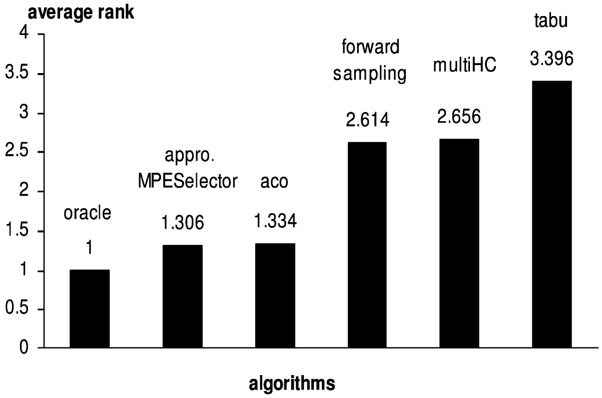
evidPercentage	Number of times of being best algorithm							
	forward_sampling		multiHC		tabu		aco	
10%	248	4.78%	358	6.91%	165	3.18%	957	18.46%
20%	281	5.42%	348	6.71%	208	4.01%	891	17.19%
30%	333	6.42%	371	7.16%	205	3.95%	819	15.80%

evidDistribution	Number of times of being best algorithm							
	forward_sampling		multiHC		tabu		aco	
predictive	303	5.85%	356	6.87%	195	3.76%	874	16.86%
random	357	6.89%	357	6.87%	219	4.25%	795	15.34%
diagnostic	202	3.90%	364	7.02%	164	3.16%	998	19.25%

we find that most of the misclassification happen on highly skewed networks where *forward_sampling* and *aco* are competing. For all of these misclassified instances, the normalized average MPE error ratio between the predict best algorithm and the actual best algorithm is 1.98%. It implies that the actual influence of these misclassifications is minor.

In order to show that *ApproxMPESelector* outperforms any single approximate algorithm, we conduct a ranking test between all approximate algorithms and *ApproxMPESelector* as if it is another MPE algorithm. We first collect the ranking information of all algorithms for all instances in $D_{MPETest2}$, and then compute the average rank of each single

Fig. 8 The average rank of each algorithm along with the learned approximate algorithm selector and an oracle that could make perfect selection of the best algorithm



algorithm and *ApproxMPESelector*. We make the distribution of the test data different from the distribution of the training data because we want to see how the learned model performs on a different set of instances. The average rank of an algorithm is the sum of its ranks on all instance divided by the number of all instances. A perfect algorithm selector with 100% classification accuracy should have an average rank of 1. If an algorithm out of k candidates is always the worst, then its average rank should be k. Figure 8 shows the average ranks of all algorithms along with *ApproxMPESelector* and an oracle that could make the perfect selection. In Fig. 8, $k = 5$. We can see that *ApproxMPESelector* ranks higher than any single algorithm. In Fig. 8 *ApproxMPESelector* is only slightly better than ACO, it is because there are more skewed instances in $D_{MPETest2}$. We can expect that if we increase the number of unskewed instances *ApproxMPESelector* would rank much higher than the second best algorithm.

System evaluation on real world networks Here we test the system on 13 real world networks. All networks can be correctly classified by *exactMPESelector*. There are 11 "yes" networks. We also run *approxMPESelector* on these 11 networks. All predicted best approximate algorithms agree with the actual best algorithms. These two "no" networks are *link* and *munin1* and *ApproMPESelector* selects ACO as the best approximate algorithm for both. This also agrees with the actual experimental results.

In summary, the test results on both synthetic and real world networks illustrate that the proposed machine learning-based approach can be used to solve the MPE algorithm selection problem. As a meta-reasoner, the learned models can make reasonable decision on selecting both exact and best approximate MPE algorithms for the input instance. The algorithm selection system can integrate exact and approximate algorithms to provide the best overall performance of probabilistic inference.

6 Concluding remarks

We have reported an approach combining experimental algorithmics and machine learning to build a practical algorithm selection system for $\mathcal{NP}$-hard optimization problems. The system consists of two learned classifiers as predictive models. The first classifier decides if exact algorithm is applicable to solve the given instance. The second one determines which approximate algorithm is the best. Our experimental results on the MPE problem show that the learned algorithm selection meta-reasoner can use some polynomial time computable instance characteristics to select the best algorithm for $\mathcal{NP}$-hard problems. The system gives the best overall performance comparing to any single algorithm. The learning procedure needs to be done only once and it takes only a few minutes. Then the learned model is available to anyone as an MPE algorithm selection meta-reasoner. The time of computing features and selecting the best algorithm are negligible comparing to the actual problem solving time.

Our results on the MPE problem also reveal that CPT skewness is the most important feature for approximate MPE algorithm selection. In general, search-based algorithms work better on unskewed networks and sampling algorithms work better on skewed networks. Other features, such as *n_nodes*, *n_samples*, *evidPercent* and *evidDistri*, all affect these algorithms' relative performance to some degree, although not as strong as *skewness* does.

In the proposed method, one important and difficult task is to identify the set of candidate features. This depends largely on domain knowledges and expert experiences. Another limitation of our method is that the size of training data grows exponentially in the number of features used. Actually this is true for any experimental methods. The fact that our training data are generated from a specific set of real world instances may also limit the learned system's applicable range.

Future work can be done in at least the following three directions. First, as mentioned before, it is worthwhile to compare regression models with classifiers on the problem addressed in this paper. Second, the proposed approach could be applied to algorithm selection of other $\mathcal{NP}$-hard problems to build more efficient computation systems. Third, some newly developed heuristic search algorithm for the MPE problem, such as stochastic local search (Hutter 2005), can be included to build a better *approxMPESelector*. These algorithms might perform better on low skewness networks than Tabu search and Multi-Restart HillClimbing.

References

Abdelbar, A. M., & Hedetniemi, S. M. (1998). Approximating MAPs for belief networks is $\mathcal{NP}$-hard and other theorems. *Artificial Intelligence*, *102*, 21–38.

Breese, J. S., & Horvitz, E. (1990). Ideal reformulation of belief networks. In *UAI90* (pp. 129–144).

Cooper, G., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*(4), 309–347.

Fink, E. (1998). How to solve it automatically: selection among problem-solving methods. In *Proceedings of the fourth international conference on artificial intelligence planning systems* (pp. 128–136).

Fung, R., & Chang, K. C. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. *Uncertainty in Artificial Intelligence*, *5*, 209–219.

Gent, I. P., & Walsh, T. (1993). An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research*, *1*, 47–59.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic.

Gomes, C. P., & Selman, B. (1997). Algorithm portfolio design: theory vs. practice. In *UAI97* (pp. 190–197).

Guo, H. (2003). *Algorithm selection for sorting and probabilistic inference: a machine learning-based approach*. PhD thesis, Kansas State University.

Guo, H., Boddhireddy, P., & Hsu, W. (2004). Using Ant algorithm to solve MPE. In *The 17th Australian joint conference on artificial intelligence*, Dec. 2004, Cairns, Australia.

Hooker, J. (1994). Needed: an empirical science of algorithms. *Operations Research*, *42*, 201–212.

Hoos, H., & Stutzle, T. (1998). Evaluating Las Vegas algorithms—pitfalls and remedies. In *UAI98*.

Hoos, H., & Stutzle, T. (2000). Local search algorithms for SAT: an empirical evaluation. *Journal of Automated Reasoning*, *24*(4), 421–481.

Horvitz, E. (1990). *Computation and action under bounded resources*. PhD thesis, Stanford University.

Horvitz, E., Ruan, Y., Kautz, H., Selman, B., & Chickering, D. M. (2001). A Bayesian approach to tackling hard computational problems. In *UAI01* (pp. 235–244).

Houstis, E. N., Catlin, A. C., Rice, J. R., Verykios, V. S., Ramakrishnan, N., & Houstis, C. (2000). PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*, *26*(2), 227–253.

Hutter, F. (2005). *Stochastic local search for solving the most probable explanation problem in Bayesian networks*. M.S. thesis, Intellectics Group, Darmstadt University of Technology.

Ide, J. S., & Cozman, F. G. (2002). Random generation of Bayesian networks. In *Brazilian symposium on artificial intelligence*, Pernambuco, Brazil.

Jensen, F. V., Olesen, K. G., & Anderson, K. (1990). An algebra of Bayesian belief universes for knowledge-based systems. *Networks*, *20*, 637–659.

Jitnah, N., & Nicholson, A. E. (1998). Belief network algorithms: a study of performance based on domain characterization. In *Learning and reasoning with complex representations* (Vol. 1359, pp. 169–188). New York: Springer.

Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. In M. H. Goldwasser, D. S. Johnson & C. C. McGeoch (Eds.), *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges* (pp. 215–250).

Kask, K., & Dechter, R. (1999). Stochastic local search for Bayesian networks. In *Workshop on AI and statistics 99* (pp. 113–122).

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Lagoudakis, M., & Littman, M. (2001). Learning to select branching rules in the DPLL procedure for satisfiability. *Electronic notes in discrete mathematics (ENDM): Vol. 9. LICS 2001 workshop on theory and applications of satisfiability testing (SAT 2001)*, Boston, MA, June 2001.

Lagoudakis, M., Littman, M., & Parr, R. (2001). Selection the right algorithm. In *Proceedings of the 2001 AAAI fall symposium series: using uncertainty within computation*, Cape Cod, MA, November 2001.

Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society Series B*, *50*, 157–224.

Leyton-Brown, K., Nudelman, E., & Shoham, Y. (2002). Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In *Constraint programming 2002 (CP-02)*.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., & Shoham, Y. (2003a). A portfolio approach to algorithm selection. In *IJCAI*.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., & Shoham, Y. (2003b). *Boosting as a metaphor for algorithm design*. Preprint.

Littman, M. (1999). Initial experiments in stochastic search for Bayesian networks. In *Proceedings of the sixteenth national conference on artificial intelligence* (pp. 667–672).

Lobjois, L., & Lema, M. (1998). Branch and bound algorithm selection by performance prediction. In *Proceedings of the fifteenth national/tenth conference on AI/innovative applications of AI* (pp. 353–358).

Lucks, M., & Gladwell, I. (1992). Automated selection of mathematical software. *ACM Transactions on Mathematical Software*, *18*(1), 11–34.

Mannila, H. (1985). *Instance complexity for sorting and NP-complete problems*. PhD thesis, Department of Computer Science, University of Helsinki.

McGeoch, C. C. (1986). *Experimental analysis of algorithms*. PhD thesis, Carnegie-Mellon University.

Mengshoel, O. J. (1999). *Efficient Bayesian network inference: genetic algorithms, stochastic local search, and abstraction*. Computer Science Department, University of Illinois at Urbana-Champaign.

Moret, B. M. E. (2002). Towards a discipline of experimental algorithmics. In *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*. *DIMACS monographs* (Vol. 59, pp. 197–213).

Orponen, P., Ko, K., Schoning, U., & Watanabe, O. (1994). Instance complexity. *Journal of the ACM*, *41*(1), 96–121.

Park, J. D. (2002). Using weighted MAX-SAT engines to solve MPE. In *Proceedings of the 18th national conference on artificial intelligence (AAAI)* (pp. 682–687).

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Mateo: Morgan Kaufmann.

Ramakrishnan, N., & Valdes-perez, R. E. (2000). Note on generalization in experimental algorithmics. *ACM Transactions on Mathematical Software*, *26*(4), 568–580.

Rardin, R. L., & Uzsoy, R. (2001). Experimental evaluation of heuristic optimization algorithms: a tutorial. *Journal of Heuristics*, *7*(3), 261–304.

Rice, J. R. (1976). The algorithm selection problem. In M. V. Zelkowitz (Ed.), *Advances in computers* (Vol. 15, pp. 65–118).

Ruan, Y., Kautz, H., & Horvitz, E. (2004). The backdoor key: a path to understanding problem hardness. In *Nineteenth national conference on artificial intelligence*, San Jose, CA, 2004.

Russell, S., & Norvig, P. (2003). *Artificial intelligence: a modern approach*. Englewood Cliffs: Prentice-Hall.

Sanders, P. (2002). Presenting data from experiments in algorithmics. In *Experimental algorithmics: from algorithm design to robust and efficient software* (pp. 181–196). New York: Springer.

Santos, E. (1991). On the generation of alternative explanations with implications for belief revision. In *UAI 91* (pp. 339–347).

Santos, E., Shimony, S. E., & Williams, E. (1995). *On a distributed anytime architecture for probabilistic reasoning* (Technique report AFIT/EN/TR94-06). Department of Electrical and Computer Engineering, Air Force Institute of Technology.

Shafer, G., & Shenoy, P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence*, *2*, 327–352.

Shimony, S. E., & Charniak, E. (1999). A new algorithm for finding MAP assignments to belief network. In *UAI 99* (pp. 185–193).

Shimony, S. E., & Domshlak, C. (2003). Complexity of probabilistic reasoning in directed-path singly connected Bayes networks. *Artificial Intelligence*, *151*, 213–225.

Witten, I. H., & Frank, E. (1999). *Data mining: practical machine learning tools and techniques with Java implementations*. Los Altos: Morgan Kaufmann.

Zilberstein, S. (1993). *Operational rationality through compilation of anytime algorithms*. PhD thesis, University of California at Berkeley.