Learning the Empirical Hardness of Optimization Problems: The case of combinatorial auctions

Kevin Leyton-Brown, Eugene Nudelman and Yoav Shoham*

Computer Science Department, Stanford University, Stanford CA 94305
{kevinlb;eugnud;shoham}@cs.stanford.edu

Abstract. We propose a new approach for understanding the algorithm-specific empirical hardness of $\mathcal{NP}$-Hard problems. In this work we focus on the empirical hardness of the winner determination problem—an optimization problem arising in combinatorial auctions—when solved by ILOG's CPLEX software. We consider nine widely-used problem distributions and sample randomly from a continuum of parameter settings for each distribution. We identify a large number of distribution-nonspecific features of data instances and use statistical regression techniques to learn, evaluate and interpret a function from these features to the predicted hardness of an instance.

1 Introduction

It is no secret that particular instances of $\mathcal{NP}$-Hard problems can be quite easy to solve in practice. In recent years researchers in the constraint programming and artificial intelligence communities have studied the *empirical* hardness of individual instances or distributions of $\mathcal{NP}$-Hard problems, and have often managed to find simple mathematical relationships between features of the problem instances and the hardness of the problem. The majority of this work has focused on decision problems: that is, problems that ask a yes/no question of the form, "Does there exist a solution meeting the given constraints?". The most successful approach for understanding the empirical hardness of such problems—taken for example in [3, 1]—is to vary some parameter of the input looking for a hard-easy-hard transition corresponding to a phase transition in the solvability of the problem. This approach uncovered the famous result that 3-SAT instances are hardest when the ratio of clauses to variables is about 4.3; it has also been applied to other decision problems such as quasigroup completion [7]. Another approach rests on a notion of backbone [17, 1], which is the set of solution invariants.

1.1 Empirical Hardness of Optimization Problems

Some researchers have also examined the empirical hardness of optimization problems, which ask a real-numbered question of the form, "What is the best solution meeting the given constraints?". These problems are clearly different from decision problems, since

* This work was supported by DARPA grant F30602-00-2-0598, the Intelligent Information Systems Institute, Cornell, and a Stanford Graduate Fellowship. Thanks to Rámon Béjar, Carla Gomes, Henry Kautz, Bart Selman, Lyle Ungar and Ioannis Vetsikas for helpful discussions.

they always have solutions and hence cannot give rise to phase transitions in solvability. One way of finding hardness transitions related to optimization problems is to transform them into decision problems of the form, "Does there exist a solution with the value of the objective function $\geq x$?" This approach has yielded promising results when applied to MAX-SAT and TSP. Other experimentally-oriented work includes the extension of the concept of backbone to optimization problems [24], although it is often difficult to define for arbitrary problems and can be costly to compute.

A second approach is to attack the problem analytically rather than experimentally. For example, Zhang performed average case theoretical analysis of particular classes of search algorithms [25]. Though his results rely on independence assumptions about the branching factor and heuristic performance at each node of the search tree that do not generally hold, the approach has made theoretical contributions—describing a polynomial/exponential-time transition in average-case complexity—and shed light on real-world problems. Korf and Reid predict the average number of nodes expanded by a simple heuristic search algorithm such as A* on a particular problem class by making use of the distribution of heuristic values in the problem space [14]. As above, strong assumptions are required: e.g., that the branching factor is constant and node-independent, and that edge costs are uniform throughout the tree.

Both experimental and theoretical approaches have sets of problems to which they are not well suited. Existing experimental techniques have trouble when problems have high-dimensional parameter spaces, as it is impractical to manually explore the space of all relations between parameters in search of a phase transition or some other predictor of an instance's hardness. This trouble is compounded when many different data distributions exist for a problem, each with its own set of parameters. Theoretical approaches are also difficult when the input distribution is complex or is otherwise hard to characterize, but they also have other weaknesses. They tend to become intractable when applied to complex algorithms, or to problems with variable and interdependent edge costs and branching factors. Furthermore, they are generally unsuited to making predictions about the empirical hardness of individual problem instances, instead concentrating on average (or worst-case) performance on a class of instances.

1.2 Our Methodology

Some optimization problems do not invite study by existing experimental *or* theoretical approaches: problems characterized by a large number of apparently relevant features, the existence of many, highly parameterized distributions, significant variation in edge costs throughout the search tree and the desirability of predicting the empirical hardness of individual problem instances.[1] We propose a novel experimental approach for predicting the running time of a given algorithm on individual instances of such a problem, drawn from one of many different distributions. Our methodology follows:

1. An optimization algorithm is selected.
2. A set of problem instance distributions is selected. For each parameter of each distribution a range of acceptable values is established.

[1] We observe that these characteristics are not unique to optimization problems, but in this paper we limit our discussion to optimization.

3. Problem size is defined and a size is chosen. Problem size will be held constant to focus on unknown sources of hardness.
4. A set of polytime-computable, distribution-independent features is selected.
5. To generate instances, a distribution is chosen at random and then the range of acceptable values for each parameter is sampled. This step is repeated until the desired number of problem instances have been generated.
6. For each problem instance the running time of the optimization algorithm is determined, and all features are computed.
7. Redundant or uninformative features are eliminated.
8. A function of the features is learned to predict running time (or some other measure of empirical hardness), and prediction error is analyzed.

The application of machine learning to the prediction of running time has received some recent study (see, eg., [12]); however, there is no other work of which we are aware that uses such an approach primarily in order to understand the empirical hardness of an $\mathcal{NP}$-Hard problem. Although our methodology applies to the broad class of problems described above, for concreteness we concentrate on one widely-studied problem that exemplifies the class. The winner determination problem (WDP) is a constraint programming optimization problem associated with combinatorial auctions. It has often been observed that WDP algorithms vary by many orders of magnitude in their running times for different problems of the same size—even for different instances drawn from the same distribution. However, little is known about what causes WDP instances to differ in their empirical hardness. Understanding what characteristics of data instances are predictive of long running times would be useful for predicting how long an auction will take to clear, tuning data distributions for hardness, constructing algorithm portfolios, designing package bidding rules to reduce the chances of long clearing times and possibly for improving the design of WDP algorithms.

1.3 Overview

In section 2 we give an introduction to combinatorial auctions in general, and previous work on the WDP in particular. We then survey nine of the most widely-studied combinatorial auction data distributions in section 3. (Because of space limitations we give the range of acceptable values we chose for each distribution on our website.) In section 4 we discuss holding the problem size constant in order to concentrate on unknown sources of hardness; in section 5 we describe our 25 distribution-independent features. Finally, our experimental results and analysis are presented in section 6.

2 Combinatorial Auctions

Combinatorial auctions have received considerable attention in computer science over the past several years because they provide a general framework for allocation and decision-making problems among self-interested agents: agents may bid for bundles of goods, with the guarantee that these bundles will be allocated "all-or-nothing". These auctions are particularly useful in cases where agents consider some goods to be *complementary*, which means that an agent's valuation for some bundle exceeds the sum of its valuation for the goods contained in the bundle. They may also allow agents to specify that they consider some goods to be *substitutable*, e.g., agents can state XOR constraints between bids, indicating that at most one of these bids may be satisfied.

2.1 Combinatorial Auction Winner Determination

The winner determination problem (WDP) is choosing the subset of bids that maximizes the seller's revenue, subject to the constraint that each good can be allocated at most once. A WDP instance consists of a set of bids: bid i is made up of a bundle of goods denoted b_i and a price offer p_i. If a set of bids are joined by an XOR constraint, the constraint is represented through the addition of a "dummy good" which is included in b_i. (A dummy good is an artificial good that exists only to enforce an XOR constraint; see [6].) The WDP may be formulated as an integer program, where indicator variables x_i encode the inclusion or exclusion of each bid i from the allocation:

$$\begin{aligned} \text{maximize:} \quad & \sum_i x_i p_i & \\ \text{subject to:} \quad & \sum_{i|g \in b_i} x_i \leq 1 & \forall g \\ & x_i \in \{0, 1\} & \forall i \end{aligned}$$

Although this problem is equivalent to weighted set-packing and is therefore $\mathcal{NP}$-Hard, there are cases where it is important to solve the WDP to optimality. There has been much discussion of the application of the Vickrey-Clarke-Groves mechanism to combinatorial auctions (e.g., a good survey of combinatorial auction research is [4]). This economic mechanism requires that a provably-optimal solution to the WDP be provided. Parkes and Ungar [19], among others, have proposed ascending auction mechanisms that also require provably-optimal solutions to the WDP. Also, it is known that the WDP is inapproximable within any constant factor: cf. [21].

Much recent work has concerned algorithms for solving the WDP to optimality. A very influential early paper was [20], but it focused on tractable subcases of the problem and addressed computational approaches to the general WDP only briefly. The first algorithms designed specifically for the general WDP were published at IJCAI-99 [6, 21]; these authors improved and extended upon their algorithms in [16, 22]. Other influential algorithmic work on the general WDP is [18, 8].

More recently, researchers have converged towards solving the WDP with branch-and-bound search, using a linear-programming relaxation of the problem as a heuristic. There has thus been increasing interest in the use of ILOG's CPLEX software to solve the WDP, particularly since the mixed integer programming module in that package improved substantially in version 6 (released 2000), and again in version 7 (released 2001). In version 7.1 this off-the-shelf software has reached the point where it is competitive with the best special purpose software, Sandholm's CABOB [22]. Indeed, CABOB makes use of CPLEX's linear programming package as a subroutine and uses branch-and-bound search. The combinatorial auction research community has thus seen convergence towards branch-and-bound search in general, and CPLEX in particular, as the preferred approach to optimally solving the WDP. In this paper we selected CPLEX 7.1 as our WDP algorithm.

3 Artificial Data Distributions

3.1 Legacy Data Distributions

The wealth of research into algorithms for solving the WDP has created a need for many instances on which to test these algorithms. Since to date computational hurdles have made real-world data scarce, researchers have generally turned to artificial distributions for the evaluation of WDP algorithms. Along with the first wave of algorithms for the WDP, seven distributions were proposed in [21, 6, 11] that have been widely used by other researchers including many of those cited above. Each of these distributions may be seen as an answer to two questions: what number of goods to request in a bundle, and what price to offer for a bundle. Given a required *number* of goods, all distributions select *which* goods to include in a bid uniformly at random without replacement. The following methods are used to select the number of goods in a bundle:

- **Uniform:** Uniformly distributed on $[1, num\_goods]$
- **Normal:** Normally distributed with $\mu = \mu_g$ and $\sigma = \sigma_g$
- **Constant:** Fixed at $constant\_goods$
- **Decay:** Starting with 1, increment the size of the bundle until $rand(0, 1) > \alpha$
- **Binomial:** Request n goods with probability $p^n(1-p)^{num\_goods-n}C(num\_goods, n)$
- **Exponential:** Request n goods with probability $Ce^{-n/q}$

The following methods are used to select a price offer:

- **Fixed Random:** Uniform on $[low, hi]$
- **Linear Random:** Uniform on $[low \cdot n, hi \cdot n]$
- **Normal:** Draw from a normal distribution with $\mu = \mu_p$ and $\sigma = \sigma_p$

The seven legacy distributions follow:

- **[L1]** *Sandholm*: Uniform; Fixed Random: $low = 0$, $hi = 1$
- **[L2]** *Sandholm*: Uniform; Linear Random: $low = 0$, $hi = 1$
- **[L3]** *Sandholm*: Constant: $constant\_goods = 3$; Fixed Random: $low = 0$, $hi = 1$
- **[L4]** *Sandholm*: Decay: $\alpha = 0.55$; Linear Random: $low = 0$, $hi = 1$
- **[L5]** *Boutilier et al.*: Normal: $\mu_g = 4$, $\sigma_g = 1$; Normal: $\mu_p = 16$, $\sigma_p = 3$
- **[L6]** *Fujishima et al.*: Exponential: $q = 5$; Linear Random: $low = 0.5$, $hi = 1.5$
- **[L7]** *Fujishima et al.*: Binomial: $p = 0.2$; Linear Random: $low = 0.5$, $hi = 1.5$

3.2 CATS Distributions

Subsequent research has exposed a variety of problems with these legacy distributions: see, for example, [2, 4, 15]. Broadly, these concerns may be divided into challenges to the realism of these distributions as a model for bidding in CA's and claims that these distributions are not empirically hard. We attempted to address the first group of problems in previous work [15], proposing the Combinatorial Auction Test Suite

(CATS). Indeed, the CATS distributions have been widely used, by many of the authors cited above and also for example by [10, 23, 13]. In this paper we consider four CATS distributions: **regions, arbitrary, matching,** and **scheduling**.[2] Although the available space does not permit the enumeration of each distribution's parameters, we give a high-level description of each distribution. *Regions* models an auction of real estate, or more generally of any goods over which two-dimensional adjacency is the basis of complementarity; bids request goods that are adjacent in a planar graph. *Arbitrary* is similar, but relaxes the planarity assumption and models arbitrary complementarities between discrete goods such as electronics parts or collectables. *Matching* applies to FAA airline take-off and landing rights auctions; each bid requests one take-off and landing slot bundle, and each bidder submits an XOR'ed set of bids for acceptable bundles. *Scheduling* models a distributed job-shop scheduling domain, with bidders requesting an XOR'ed set of resource time-slots that will satisfy their specific deadlines.

By modeling bidders explicitly and creating bid amounts, sets of goods and sets of substitutable bids from models of bidder valuations and models of problem domains, we aimed for the CATS distributions to serve as a step towards a realistic set of test distributions. However, we made no efforts to tune the distributions to provide hard instances. In practice, many researchers have remarked that some CATS problems are comparatively easy. In section 6 we show experimentally that some CATS distributions are always very easy for CPLEX, while others can be extremely hard.

4 The Issue of Problem Size

Some sources of empirical hardness in $\mathcal{NP}$-Hard problem instances are well-understood. Our goal is to understand what *other* features of instances are predictive of hardness so we hold these parameters constant, concentrating on variations in other features.

For the WDP, it is well known that problems become harder as the number of goods and bids increases.[3] For this reason, researchers have traditionally reported the performance of their WDP algorithms in terms of the number of bids and goods of the input instances. While it is easy to fix the number of goods, holding the number of bids constant is not as straightforward as it might appear. Most special-purpose algorithms make use of a polynomial-time preprocessing step which removes bids that are strictly dominated by one other bid. More precisely, bid i is dominated by bid j if the goods requested by i are a (non-strict) superset of the goods requested by j, and the price offer of i is smaller than or equal to the price offer of j. (This is similar in flavor to the use of arc-consistency as a preprocessing step for a CSP or weighted CSP problem.) It is thus possible for the size of problems given as input to the WDP algorithm to vary even if all generated instances had the same number of bids.

[2] We have not included the **paths** distribution because we are updating and extending it; we will include results concerning this distribution in the full version of this paper.

[3] An exception is that problems generally become easier when the number of bids grows *very* large in distributions favoring small bundles, because each small bundle is sampled much more often than each large bundle, giving rise to a new distribution for which the optimal allocation tends to involve only small bundles. We consider much smaller numbers of bids here.

It is not obvious whether domination ought to remove many bids, or whether the relationship between the average number of non-dominated bids and total bids ought to vary substantially from one distribution to another. No other work of which we are aware characterizes this relationship, so we set out to measure it. Figure 4 shows the number of non-dominated bids as a function of the total number of bids generated. In these experiments, with each line representing an average over 20 runs, bids were generated for an auction with 64 goods, and the program stopped after 2000 non-dominated bids had been made. We observe that some of the "legacy" distributions are considerably more likely than others to generate non-dominated bids; we do not show the CATS distributions in this graph as all four generated virtually no dominated bids.

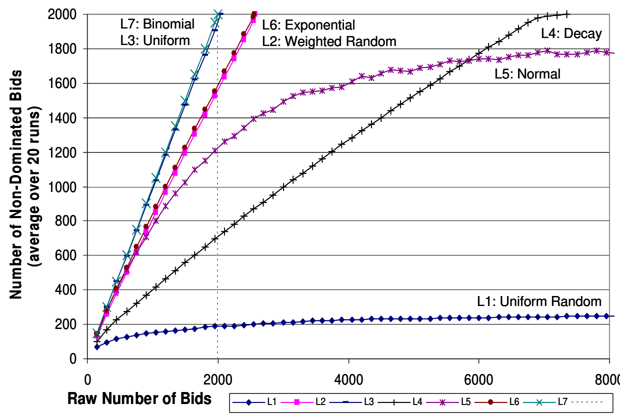


Fig. 1. Non-Dominated Bids vs. Raw Bids

Of course, many other polynomial-time preprocessing steps are possible, e.g., a check for bids that are dominated by a pair of other bids. Indeed, CPLEX employs many, much more complex preprocessing steps before initiating its own branch-and-bound search. However, our own experience with algorithms for the WDP has suggested that other polynomial-time preprocessing steps offer much poorer performance in terms of the number of bids discarded in a given amount of time. The results above certainly suggest that the results of strict domination checking should not be disregarded, since distributions differ substantially in the ratio between the number of non-dominated bids and the raw number of bids. We thus defined the problem size as the pair (*number of goods, number of non-dominated bids*). We re-implemented the CATS software so that it generated instances for all CATS and legacy distributions with a specified number of *non-dominated* bids: the software iteratively generated bids and removed dominated bids until the specified target was reached. Our focus on the number of non-dominated bids forced us not to consider bids from the distributions L1 and L5 in the remainder of the paper, because they often failed to generate the specified number of non-dominated bids even after millions of bids were created. (We note that this helps explain why L1 and L5 have been found empirically easy by other researchers.)

Bid-Good Graph Features:

1-3. **Bid nodes degree statistics**: max and min degree of the bid nodes, and standard deviations.
4-7. **Good nodes degree statistics**: average, maximum, minimum degree of the good nodes, and their standard deviations.

Bid Graph Features:

8. **Edge Density**: number of edges in the BG divided by the number of edges in a complete graph with the same number of nodes.
9-11. **Node degree statistics**: the max and min node degrees in the BG, and their standard deviation.
12-13. **Clustering Coefficient and Deviation**. A measure of "local cliqueness." For each node calculate the number of edges divided by $k(k-1)/2$, where k is the number of neighbors. We record average (the clustering coefficient) and standard deviation.
14. **Average minimum path length**: the average minimum path length, over all pairs of bids.
15. **Ratio of the clustering coefficient to the average minimum path length**: One of the measures of the smallness of the BG.
16-19. **Node eccentricity statistics**: The eccentricity of a node is the length of a shortest path to a node furthest from it. We calculate the maximum eccentricity of BG (graph diameter), the minimum eccentricity of BG (graph radius), average eccentricity, and standard deviation of eccentricity.

LP-Based Features:

20-22. L_1, L_2, L_∞ norms of the integer slack vector.

Price-Based Features:

23. **Standard deviation of prices among all bids**: $stdev(p_i)$
24. **Deviation of price per number of goods**: $stdev(p_i/|b_i|)$
25. **Deviation of price per square root of the number of goods**: $stdev(p_i/\sqrt{|b_i|})$.

Fig. 2. Four Groups of Features

5 Features

As described above, we characterize each problem instance with a set of features. There is no automatic way of constructing a feature set: researchers must use their domain knowledge to identify properties of the instance that appear likely to provide useful information about empirical hardness. We only consider features that can be generated from *any* problem instance, without knowledge of how that instance was constructed. Furthermore we restrict ourselves to those features that are computable in polynomial time, since the computation of the features should scale well as compared to solving the optimization problem. Although features must be manually selected, there are statistical techniques for identifying useless features. Identifying such features is important because highly correlated features can unnecessarily increase the dimensionality of the hypothesis space: this can degrade the performance of some regression algorithms and also makes the resulting formula harder to interpret.

For our WDP case study, we determined 35 features which we thought could be relevant to the empirical hardness of the optimization, ranging in their computational complexity from linear to cubic time. After feature selection we were left with 25 features: these are summarized in Fig. 2. We describe our features in more detail below, and also mention some of the features that were eliminated by feature selection.

There are two natural graphs associated with each instance. First, is the *bid-good graph* (BGG): a bipartite graph having a node for each bid, a node for each good and an edge between a bid and a good node for each good in the given bid. We measure a variety of BGG's properties: extremal and average degrees and their standard deviations for each group of nodes. The average number of goods per bid was perfectly correlated with another feature, and so did not survive our feature selection.

The *bid graph* (BG) represents conflicts among bids (thus it is the constraint graph for the associated CSP). As is true for all CSPs, the BG captures a lot of useful information about the problem instance. Our second group of features are concerned with structural properties of the BG.[4] We originally measured the first and third quartiles and the median of the BG node degrees, but they turned out to be highly correlated with edge density. We also measured the average number of conflicts per bid, but since the number of bids was held constant this feature was always proportional to edge density. We considered using the number of connected components of the BG to measure whether the problem is decomposable into simpler instances, but found that nearly every instance consisted of a single component. Finally, it would be desirable to include some measure of the size of the (unpruned) search space. For some problems branching factor and search depth are used; for WDP neither is easily estimated. A related measure is the number of maximal independent sets of BG, which corresponds to the number of feasible solutions. However, this counting problem is hard, and to our knowledge does not have a polynomial-time approximation.

The third group of features is calculated from the solution vector of the LP relaxation of the WDP. We calculate the *integer slack* vector by replacing each component x_i with $min(|x_i|, |1 - x_i|)$. These features appeared particularly useful both because the slack gives insight into the quality of CPLEX's initial solution and because CPLEX uses LP as its search heuristic. Originally we also included median integer slack, but excluded the feature when we found that it was always zero.

Our last group of features is the only one to explicitly consider the prices associated with bids. While the scale of the prices has no effect on hardness the spread is crucial, since it impacts pruning. We note that feature 25 was shown to be an optimal bid-ordering heuristic for branch-and-bound search on the WDP in [8].

6 Experimental Results

We generated three separate data sets of different problem sizes, to ensure that our results were not artifacts of one particular choice of problem size. The first data set contained runs on instances of 1000 bids and 256 goods each, with a total of 4500 instances (500 instances per distribution). The second data set with 1000 bids and 144 goods had a total of 2080 instances; the third data set with 2000 bids and 64 goods contained 1964 instances. Where we present results for only a single data set, the first data set was always used. All of our runtime data was collected by running CPLEX 7.1 with preprocessing turned off. We used a cluster of 4 machines, each of which had 8 Pentium III Xeon 550 MHz processors and 4G RAM and was running Linux 2.2.12. Since many of the instances turned out to be exceptionally hard, we stopped CPLEX after it had expanded 130,000 nodes (reaching this point took between 2 hours and 22 hours, averaging 9 hours). Overall, solution times varied from as little as 0.01 seconds to as much as 22 hours. We estimate that we consumed approximately 3 years of CPU time collecting this data. We also computed our 35 features for each instance. (Recall that feature selection took place after all instances had been generated.) Each feature in each data set was normalized to have a mean of 0 and a standard deviation of 1. Regression was performed using the open-source R package (see `www.r-project.org`).

[4] We thank Rámon Béjar for providing code for calculating the clustering coefficient.

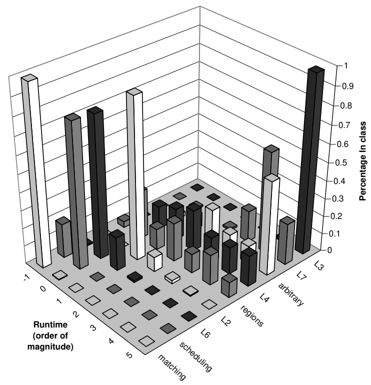


Fig. 3. Gross Hardness

6.1 Gross Hardness

Before attempting to characterize the hardness of our problems with machine learning techniques, we examined the gross hardness of each distribution. To our knowledge no previously published results show the *distribution* of hard and easy problems across different data distributions. Figure 3 shows the results of 500 runs for each distribution on problems with 256 goods and 1000 non-dominated bids, indicating the number of instances with the same order-of-magnitude runtime—i.e., $\lfloor \log_{10}(runtime) \rfloor$—for each of our nine distributions. Each instance of each distribution had different parameters, each of which was sampled from a range of acceptable values.

This figure illustrates that some data distributions were always easy for CPLEX, while others were nearly always hard. It is interesting that most distributions had instances that varied in hardness by several orders of magnitude, despite the fact that all instances had the same problem size. In the next sections we will describe our attempts to predict the order-of-magnitude hardness of data instances—effectively, to induce the hardness information in Fig. 3 without running CPLEX or identifying the distribution from which an instance was drawn.

6.2 Homing in on Sources of Hardness

Since we wanted to learn a continuous-valued model of the features, we used statistical regression techniques.[5] We chose the logarithm of CPLEX running time as our response variable—the value to be predicted—rather than absolute running time, for consistency with our original motivation of automatically reconstructing the gross hardness figure. We wanted the model to be penalized according to whether the predicted and actual values had the same order of magnitude: if we had tried to predict absolute running times then the model would have been penalized very little for dramatically mispredicting the running time of very easy instances, and would have been penalized heavily for slightly mispredicting the running time of the hardest instances. We performed regression on a training set consisting of 80% of each of our datasets, and then tested our model on the remaining 20% to evaluate its ability to generalize to new data.

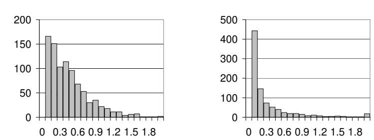
[5] A large literature addresses the statistical techniques we used; for an introduction see, e.g., [9].

Linear Regression One of the simplest and most widely-studied regression techniques is linear regression. This technique works by finding a hyperplane in the feature space that minimizes root mean squared error (RMSE), which is defined as the square root of the average squared difference between the predicted value and the true value of the response variable. Minimizing RMSE is reasonable because it conforms to the intuition that, holding mean absolute error constant, models that mispredict all instances equally should be preferred to models that vary in their mispredictions. Although we go on to consider nonlinear regression, it is useful to consider the results of linear regression for two reasons. First, one of our main goals was to understand the factors that influence hardness, and insights gained from a linear model are useful even if other, more accurate models can be found. Second, our linear regression model serves as a baseline to which we can compare the performance of more complex regression techniques.

In Fig. 4(a) we report both RMSE and mean absolute error, since the latter is often more intuitive. A third measure, adjusted R^2, is the fraction of the original variance in the response variable that is explained by the model, with a penalty for more complex models. Despite this penalty, adjusted R^2 is a measure of fit to the training set and cannot entirely correct for overfitting; nevertheless, it can be an informative measure when presented along with test set error. Overall, this table shows that our linear model would be able to do a good job of classifying instances into the bins shown in Fig. 3, despite the fact that it is not given the distribution from which each instance was drawn: 93% of the time the log running times of the data instances in our test set were predicted to the correct order of magnitude (i.e., with an absolute error of less than 1.0). Figures 4(b) and 4(c) show histograms of mean absolute and RMS error, with bin width 0.1. These histograms show that most instances are predicted very accurately, and few instances are dramatically mispredicted.

Data Point	Mean Abs Err	RMSE	Adj. R^2
1000 Bids/256 Goods	0.399	0.543	0.938
1000 Bids/144 Goods	0.437	0.579	0.909
2000 Bids/64 Goods	0.254	0.368	0.912

(a) Errors and Adjusted R^2



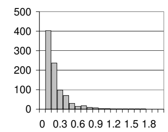
(b) MAE (c) RMSE

Fig. 4. Experimental results for linear regression

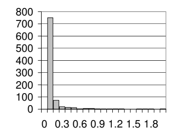
Nonlinear Models Although our linear model was quite effective, we expected that nonlinear interactions between our features would be important, and therefore looked to nonlinear models. A simple way of performing nonlinear regression is to compute new features based on nonlinear interactions between the original features, and then to perform linear regression on the union of both sets of features. We added all products of pairs of features to our linear model, including squares of individual features, which gave us a total of 350 features. This allowed us to fit a 2nd degree polynomial instead of our previous linear model. For all three of our datasets this model gave considerably better error measurements on the test set and also explained nearly all the variance in the training set, as shown in Fig. 5(a).

Data Point	Mean Abs Err	RMSE	Adj. R^2
1000 Bids/256 Goods	0.183	0.297	0.987
1000 Bids/144 Goods	0.272	0.475	0.974
2000 Bids/64 Goods	0.163	0.272	0.981

(a) Errors and Adjusted R^2



(b) MAE



(c) RMSE

Fig. 5. Experimental results for 2nd degree polynomial regression

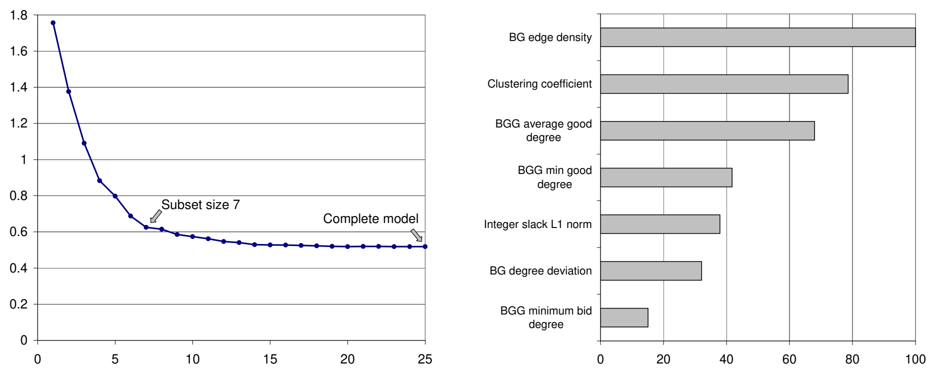
We also explored another nonlinear regression technique, Multivariate Adaptive Regression Splines (MARS) [5]. MARS models are linear combinations of the products of one or more basis functions, where basis functions are the positive parts of linear functions of single features. The RMSE on our MARS models differed from the RMSE on our second-order model only in the second decimal place; as MARS models can be unstable and difficult to interpret, we focus on the second-order model here.

Analysis The results summarized above demonstrate that it is possible to learn a model of our features that accurately predicts the log of CPLEX running time on novel instances. For some applications (e.g., predicting the time it will take for an auction to clear; building an algorithm portfolio) accurate prediction is all that is required. For other applications it is necessary to *understand* what makes an instance empirically hard. In this section we set out to interpret our models.

It is tempting to interpret a model by comparing the coefficients assigned to the different features; since all features have the same mean and standard deviations, more important features should tend to have larger coefficients. The reason that this does not work is that most features are at least somewhat correlated. Two perfectly correlated but entirely unimportant features can have large coefficients with opposite signs in a linear model; in practice, since imperfect correlation and correlations among larger sets of variables are common, it is difficult to untangle the effects of correlation and importance in explaining a given coefficient's magnitude. One solution is to force the model to have smaller coefficients and/or to contain fewer variables. Requiring smaller coefficients reduces interactions between correlated variables; two popular techniques are ridge regression and lasso regression. We evaluated these techniques using cross validation and found no significant effect on errors or on interpretability of the model, so we do not present these results here.

Another family of techniques allows interpretation without the consideration of coefficient magnitudes. These techniques select "good" subsets of the features, essentially performing exhaustive enumeration when possible and various greedy search techniques otherwise. Small models are desirable for analysis because they are easier to interpret directly and because a small, optimal subset will tend to contain fewer highly covariant features than a larger model. We plotted subset size (from 1 to the total number of variables) versus the RMSE of the best model built from a subset of each size. We then chose the smallest subset size at which there was little incremental benefit gained by moving to the next larger subset size. We examined the features in the model, and also measured each variable's cost of omission—the (normalized) difference between the RMSE of the model on the original subset and a model omitting the given variable.

It is important to note that because of correlation between features many different subsets may achieve nearly the same RMSE, and that very little can be inferred from what *particular* variables are included in the best subset of a given size. This is of little concern, however, when subset selection is used only to gain a conceptual understanding of the features that are important for predicting empirical hardness; in this case the substitution of one feature for another covariant feature is irrelevant because the inclusion of either feature in the model has the same intuitive meaning. It is also worth noting that subset selection and cost of omission were both evaluated using the test set, but that all model selection was evaluated using cross validation, and all analysis was performed after our models had been learned.



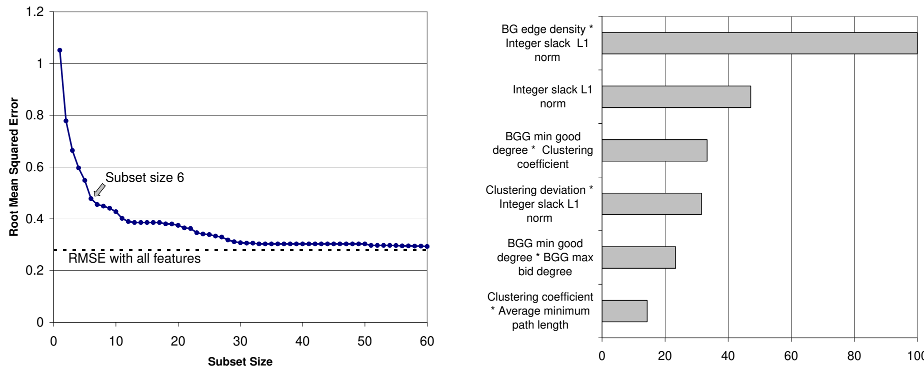
(a) Subset size vs. RMSE.

(b) Cost of omission for subset size 7.

Fig. 6. Subset Selection in Linear Models.

Figure 6(a) shows the RMSE of the best subset containing between 1 and 25 features for linear models; since we had only 25 features in total we selected the best subsets by exhaustive comparison. We chose to examine the model with seven features because it was the first for which adding another feature did not cause a large decrease in RMSE. Figure 6(b) shows the seven features in this model and their respective costs of omission (scaled to 100). The most striking conclusion is that structural features are the most important. Edge density of BG is essentially a measure of the constrainedness of the problem, so it is not surprising to find that this feature is the most costly to omit. Clustering coefficient, the second feature, is a measure of average cliquiness of BG; this feature gives an indication of how local constraints in the problem are. All but one of the remaining features concern node degrees in BG or BGG; the final feature is the L_1 norm of the linear programming slack vector. The importance of this feature is quite intuitive: the L_1 norm is close to 0 for problems that are almost completely solved by LP, and larger for more difficult problems.

Figure 7(a) shows the best subsets containing between 1 and 60 features for second-order models. In this case we had 350 features, making exhaustive exploration of features impossible; we instead used four different greedy subset selection methods and at each size chose the best subset among the four. The subsets shown in Fig. 7(a) are likely not the RMSE-minimizing subsets of the given sizes, but since our goal was only



(a) Subset size vs. RMSE.

(b) Cost of omission for subset size 6.

Fig. 7. Subset Selection in Second-Order Models.

to understand what sorts of features are important this likely lack of optimality is not a serious problem. We observe that the interactions between features dramatically improved RMSE on very small subsets. Figure 7(b) shows the costs of omission for the variables from the best subset of six features. Again we observe that edge density of BG is a critically important feature, as are the clustering coefficient and node degrees. We observe that overall many second-order features were selected. The L_1 norm becomes more important than in the linear model when it is allowed to interact with other features; in the second-order model it is also sufficiently important to be included as the only first-order feature. It is striking that no price features were important in either our first- or second-order models. Although price-based features do appear in larger models, they seem not to be as critically important as structural or LP-based features. This may be partially explained by the fact that the removal of dominated bids eliminates the bids that deviate most substantially on price, and indeed caused us to eliminate the distribution (L1) in which average price per good varied most dramatically across bids.

7 Conclusion and Future Directions

This paper makes two main contributions. First, we performed an extensive experimental investigation into the hardness of the WDP. We contrasted widely-used WDP distributions with respect to their empirical hardness, and showed that the traditional definition of problem size can be misleading. We identified structural, distribution-independent features of WDP instances and showed that, somewhat surprisingly, they contain enough information to predict CPLEX running time with high accuracy.

Second and more importantly, we proposed a new, general methodology for understanding the empirical hardness of complex, high-dimensional $\mathcal{NP}$-Hard problems. We believe that our methodology, based on using machine learning techniques to identify hard regions of the feature space, is applicable to a wide variety of hard problems; we intend to test this hypothesis in future work. Furthermore, we intend to explore some of the applications described earlier: straightforward prediction of running time, effective algorithm selection in algorithm portfolios, tuning distributions for hardness and gaining deeper insight into empirical hardness through the analysis of learned models.

References

[1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable instances. In *AAAI-00*, 2000.

[2] A. Anderson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *ICMAS*, 2000.

[3] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the Really Hard Problems Are. In *IJCAI-91*, 1991.

[4] S. de Vries and R. Vohra. Combinatorial auctions: A brief survey. Unpublished, 2000.

[5] J. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19, 1991.

[6] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI-99*, 1999.

[7] Carla P. Gomes and Bart Selman. Problem structure in the presence of perturbations. In *AAAI/IAAI*, 1997.

[8] R. Gonen and D. Lehmann. Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *ACM Conference on Electronic Commerce*, 2000.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *Elements of Statistical Learning*. Springer, 2001.

[10] R. C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Canadian Conference on AI*, 2001.

[11] H.H. Hoos and C. Boutilier. Solving combinatorial auctions using stochastic local search. In *AAAI-00*, 2000.

[12] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and M. Chickering. A bayesian approach to tackling hard computational problems, 2001.

[13] R. Kastner, C. Hsieh, M. Potkonjak, and M. Sarrafzadeh. On the sensitivity of incremental algorithms for combinatorial auctions, 2002. UCLA CS Tech. Report 020000.

[14] R. Korf and M. Reid. Complexity analysis of admissible heuristic search. *AAAI-98*, 1998.

[15] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, 2000.

[16] K. Leyton-Brown, Yoav Shoham, and Moshe Tennenholtz. An algorithm for multi-unit combinatorial auctions. In *Proceedings of AAAI-00*, 2000.

[17] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, , and L. Troyansky. Determining computational complexity for characteristic 'phase transitions'. *Nature*, 400, 1998.

[18] N. Nisan. Bidding and allocation in combinatorial auctions. In *ACM Conference on Electronic Commerce*, 2000.

[19] D. C. Parkes. iBundle: An efficient ascending price bundle auction. In *ACM Conference on Electronic Commerce*, 1999.

[20] M.H. Rothkopf, A. Pekec, and R.M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1998.

[21] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI-99*, 1999.

[22] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Cabob: A fast optimal algorithm for combinatorial auctions. In *IJCAI-01*, 2001.

[23] Dale Schuurmans, Finnegan Southey, and Robert C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *IJCAI-01*, 2001.

[24] J. Slaney and T. Walsh. Backbones in optimization and approximation. In *IJCAI-01*, 2001.

[25] W. Zhang. *State-Space Search: Algorithms, Complexity, Extensions, and Applications*. Springer, 1999.