Chapter 19

Empirical Hardness Models for Combinatorial Auctions

Kevin Leyton-Brown, Eugene Nudelman and Yoav Shoham

In this chapter we consider the empirical hardness of the winner determination problem. We identify distribution-nonspecific features of data instances and then use statistical regression techniques to learn, evaluate and interpret a function from these features to the predicted hardness of an instance, focusing mostly on ILOG's CPLEX solver. We also describe two applications of these models: building an algorithm portfolio that selects among different WDP algorithms, and inducing test distributions that are harder for this algorithm portfolio.

19.1 Introduction

Figure 14 in Chapter 18 showed that the runtimes of WDP algorithms can vary by many orders of magnitude across different problems of the same size, and even across different instances drawn from the same distribution. (In particular, this figure showed CPLEX runtimes for WDP instances of the same size varying from about a hundredth of a second to about a day.) This raises a puzzling question—what characteristics of the instances are responsible for this enormous variation in empirical hardness? Since an understanding of the amount of time an auction will take to clear is a requirement in many combinatorial auction application areas, an answer to this question would greatly benefit the practical deployment of WDP algorithms.

It is not altogether surprising to observe significant variation in runtime for a WDP algorithm, as such variation has been observed in a wide variety of algorithms for solving other $\mathcal{NP}$-hard problems. Indeed, in recent years a growing number of computer scientists have studied the *empirical* hardness of individual instances or distributions of various $\mathcal{NP}$-hard problems, and in many cases have 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 Cheeseman et al. (1991) and Achlioptas et al. (2000)—is to vary some parameter of the input looking for a easy-hard-easy 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 (Selman et al. 1996); it has also been applied to other decision problems such as quasigroup completion (Gomes and Selman 1997). Another approach rests on a notion of backbone (Monasson et al. 1998; Achlioptas et al. 2000), which is the set of solution invariants.

For optimization problems, experimental researchers have looked at reductions to decision problems, or related the backbone of an optimization problem to its empirical hardness (Slaney and Walsh 2001). It is also possible to take an analytic approach, although this approach typically requires strong assumptions about the algorithm and/or the instance distribution (e.g., that the branching factor is constant and node-independent and that edge costs are uniform throughout the search tree) (Zhang 1999; Korf and Reid 1998).

Some optimization problems do not invite study by existing experimental *or* theoretical approaches. Existing experimental techniques have trouble

when problems have high-dimensional parameter spaces, as it is impractical to manually explore the space of all relations among 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; moreover, they tend to become unwieldy when applied to complex algorithms, or to problems with variable and interdependent edge costs and branching factors. Furthermore, standard techniques 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.

The empirical properties of the combinatorial auction winner determination problem are difficult to study for all of the reasons discussed above. Instances are characterized by a large number of apparently relevant features. Many different input distributions exist, each with its own large set of parameters. There is significant variation in edge costs throughout the search tree. Finally, it is desirable to predict the empirical hardness of individual problem instances. Thus, a new approach is called for.

19.1.1 Methodology

Instead of using any of the approaches mentioned above, we suggested an experimental methodology for constructing hardness landscapes for a given algorithm (Leyton-Brown et al. 2002). Such models are thus capable of predicting the running time of a given algorithm on new, previously-unseen problem instances. They are built as follows:

1. One or more algorithms are chosen.

2. A problem instance distribution is selected, and the distribution is sampled to generate a set of problem instances.

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 fast-to-compute, distribution-independent features is selected.

5. For each problem instance the running time of each optimization algorithm is measured, and all features are computed.

6. Redundant or uninformative features are eliminated.

7. A function of the features is learned to predict each algorithm's running time, and prediction error is analyzed.

The application of machine learning to the prediction of running time has received some recent study (see, eg., (Horvitz et al. 2001; Ruan et al. 2002; Lagoudakis and Littman 2000; Lagoudakis and Littman 2001)); however, there is no other work of which we are aware that uses a machine learning approach in order to understand the empirical hardness of an $\mathcal{NP}$-hard problem. There have been some reports in the literature about the relation of particular features or parameters of input distributions to the hardness of WDP instances (e.g., (Sandholm 2002)), but to our knowledge no systematic study has been attempted. Finally, as is common with hard problems, empirical evaluations have focused on scaling behavior of algorithms on different distributions rather than on structural differences at a fixed size.

As described above, our main motivation for proposing this methodology has been the problem of *understanding* the characteristics of data instances which are predictive of long running times. However, empirical hardness models have other more practical uses, making them important for CA practitioners as well as for academic researchers. These applications of empirical hardness models include:

- predicting how long an auction will take to clear;
- tuning benchmark distributions for hardness;

- constructing algorithm portfolios;
- designing package bidding rules to reduce the chances of long clearing times;
- efficiently scheduling auction clearing times;
- improving the design of WDP algorithms.

19.2 Building hardness models for WDP

19.2.1 Optimization algorithm

In recent years, researchers working on the WDP have converged towards 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 academic special purpose software. In this chapter we selected CPLEX 7.1 as our WDP algorithm, [1] although we do consider some special-purpose software in Section 19.4. A survey describing the architecture of special-purpose WDP algorithms is given by Sandholm (Chapter 14).

19.2.2 Instance distribution

There are a variety of widely-used benchmark generators for CAs (see Chapter 18). To avoid bias, we used all Legacy and CATS generators that are able to generate instances with arbitrary numbers of goods and undominated bids, and created the same number of instances with each generator. Our instance distribution can thus be understood as sampling uniformly from instances created by the following generators:[2]

- Uniform (L2)
- Constant (L3)
- Decay (L4)
- Exponential (L6)
- Binomial (L7)
- Regions (CATS)
- Arbitrary (CATS)
- Matching (CATS)
- Scheduling (CATS)

Most of these generators has one or more parameters which must be assigned values before instances can be generated. As described in Section 4.2 of Chapter 18, for each parameter of each instance generator we established a "reasonable" range, and then before creating an instance sampled uniformly at random from this range. This helped us to explore more of the parameter space.

19.2.3 Problem size

Some sources of empirical hardness in $\mathcal{NP}$-hard problem instances are well-understood. For the WDP—an $\mathcal{NP}$-hard problem, as discussed in Lehmann, Müller and Sandholm (Chapter 12)—it is known that instances generally become harder as the problem gets larger: i.e., as the number of bids and goods increases. Furthermore, as argued in Chapter 18, the removal of dominated bids can have a significant effect. 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. We therefore defined problem size as the pair (*number of goods, number of non-dominated bids*).

19.2.4 Features

As described above, we must characterize each problem instance with a set of features. There is no known automatic way of constructing such a feature set: researchers must use domain knowledge to identify properties of the instance that appear likely to provide useful information. We do restrict the sorts of features we will use in two ways, however. First, we only consider features that can be generated from *any* problem instance, without knowledge of how that instance was constructed. (For example, we do not use parameters of the specific distribution used to generate an instance.) Second, we restrict ourselves to those features that are computable in low-order polynomial time, since the computation of the features should scale well as compared to solving the optimization problem.

We determined 35 features which we thought could be relevant to the empirical hardness of WDP, ranging in their computational complexity from linear to cubic time. After having generated feature values for all our problem instances, we examined our data to identify redundant features. After eliminating these, we were left with 25 features, which are summarized in Figure 19.1. We describe our features in more detail below, and also mention some of the redundant features that were eliminated.

There are two natural graphs associated with each instance; examples of

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 cliquiness." For each node calculate the number of edges among its neighbors 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**: $stdev(p_i/|S_i|)$

25. **Deviation of price per square root of the number of goods**: $stdev(p_i/\sqrt{|S_i|})$.

Figure 19.1: Four Groups of Features

these graphs appear in Figure 19.2. 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) has an edge between each pair of bids that cannot appear together in the same allocation (thus it is the constraint graph for the associated CSP). As is true for all CSPs, the BG captures a lot of useful

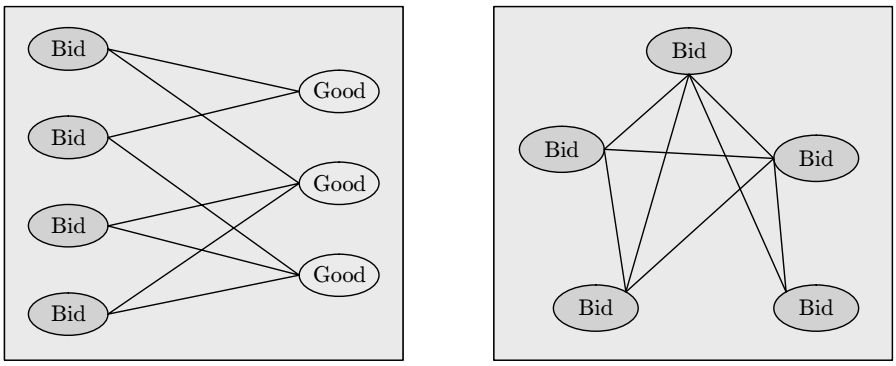

Figure 19.2: Examples of the Graph Types Used in Calculating Features 1–19: Bid-Good Graph (left); Bid Graph (right)

information about the problem instance. Our second group of features are concerned with structural properties of the BG.[3] We originally measured the first, second and third quartiles 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 as 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 virtually every instance consisted of a single component.[4]

The third group of features is calculated from the solution vector of the linear programming relaxation of the WDP. Recall that WDP can be formulated as an integer program, as described in Chapter 12. In our notation S_i stands for the set of goods in bid i, p_i for the corresponding

price, and a variable x_i is set to 1 if and only if bid i is part of an optimal allocation.

We calculate the *integer slack* vector by replacing each component x_i with $|0.5 - x_i|$. These features appeared promising 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 that explicitly considers the prices associated with bids. Observe that the scale of the prices has no effect on hardness; however, the spread is crucial, since it impacts pruning. We note that feature 25 was shown to be an optimal bid-ordering heuristic for certain greedy WDP approximation schemes in (Gonen and Lehmann 2000).

19.2.5 Running experiments

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 minimal preprocessing. 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 interrupted CPLEX runs once they 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 the runtime data described here. We also computed 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.

19.2.6 Learning models

Since we wanted to learn a continuous-valued model of the features, we used statistical regression techniques. (A large literature addresses the statistical techniques we used; for an introduction see, e.g., (Hastie et al. 2001).) We used the logarithm of CPLEX running time as our response variable (dependent variable). In a sense, this equalizes the effort that the regression

algorithm spends on fitting easy and hard instances—taking the log essentially corresponds to penalizing the relative prediction error rather than absolute error. Without this transformation, a 100-second prediction error would be penalized equally on an instance that took 0.01 seconds to run as on an instance that took 10,000 seconds. Our use of log runtime as the response variable also allows us to ask the question of how accurately our methods would be able to reconstruct the gross hardness figure (Fig. 14) for unseen instances, without any knowledge of the distribution from which each instance was drawn.

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. Regression was performed using the open-source R package (see `www.r-project.org`).

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.

Overall, we found that even linear models have a surprising ability to predict the amount of time CPLEX will take to solve novel WDP instances: in our experiments most instances were predicted very accurately, and few instances were dramatically mispredicted. Overall, our results show that our linear model would be able to do a good job of classifying instances into the bins shown in Figure 14 in Chapter 18, even without knowledge of 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).

Our experimental results with linear models are summarized in Table 19.1 and Figures 19.3 and 19.4. In Table 19.1 we report both RMSE and

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

Table 19.1: Linear Regression: Test Set Error and Adjusted R^2

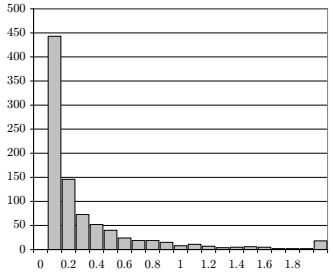

Figure 19.3: Linear Regression: Test Set Root Mean Squared Error

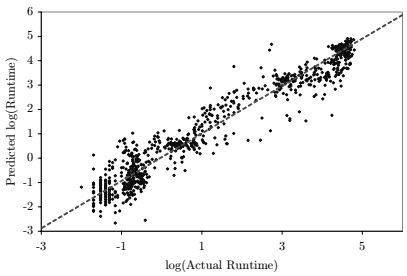

Figure 19.4: Linear Regression: Error Scatterplot

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 an adjustment penalizing 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. Figure 19.3 shows a histogram of the RMS error, with bin width 0.1. Figure 19.4 shows a scatterplot of predicted log runtime vs. actual log runtime.

Nonlinear models

Although our linear model was quite effective, we expected nonlinear interactions between our features to 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 meant that we chose our model from the space of all second-degree polynomials in our 25-dimensional feature space, rather than from the space of all hyperplanes in that space as in Section 19.2.6. 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 Table 19.2. As above, Figures 19.5 and 19.6 show a histogram of root mean squared error and a scatterplot of predicted log runtime vs. actual log runtime. Comparing these figures to Figures 19.3 and 19.4 confirms our judgment that the quadratic model is substantially better overall.

Data point	Mean Abs. Err.	RMSE	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

Table 19.2: Quadratic Regression: Test Set Errors and Adjusted R^2

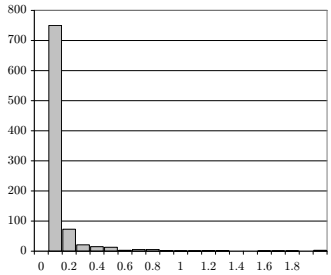

Figure 19.5: Quadratic Regression: Test Set Root Mean Squared Error

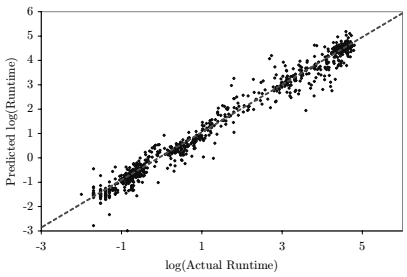

Figure 19.6: Quadratic Regression: Error Scatterplot

19.3 Analyzing hardness models

The results summarized above demonstrate that it is possible to learn a model of our features that very accurately predicts the log of CPLEX running time on novel WDP 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. In some other cases, however, we are interested in *understanding* what makes an instance empirically hard. In this section we discuss the interpretation of our models.

19.3.1 Cost of omission

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. Indeed, this will often be the case. However, this simplistic analysis technique ignores the effects of correlation between features. For example, 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[5] to estimate good values for the shrinkage parameters—and found no significant improvement on either accuracy or on interpretability of the model. Thus we do not discuss these results further.

Another family of techniques allows interpretation *without* the consideration of coefficient magnitudes. These techniques attempt to select good subsets of the features, with the number of features in the subset given

as a parameter. Small models are desirable for our goal of analyzing hardness models 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. (Intuitively, when subsets get small enough then the optimal model will not be able to afford to spend its feature choices on a highly-correlated set of features.) If the number of original features is relatively small, it is possible to determine the optimal subset by exhaustively enumerating all feature subsets of the desired size and evaluating the quality of each corresponding model. However, most of the time such an exhaustive enumeration is infeasible and some incomplete optimization approach must be used instead.

In order to choose the size of the subset to analyze, we plotted subset size (from 1 to the total number of variables) versus the RMSE of the best model involving a subset of that size. We then analyzed 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 very important to note that our technique identifies a set of features which is *sufficient* to achieve a particular level of accuracy,

not a set of features which is *necessary* for this degree of performance. It is entirely possible that many different subsets will achieve nearly the same RMSE—when many correlations exist between features, as in our WDP dataset, this is quite likely. Thus, we must be careful not to draw overly general conclusions from the particular variables appearing in the best subset of a given size, and even more careful about reasoning about the absence of a particular feature. The strength of our approach is in providing a conceptual picture of the sorts of features that are important for predicting empirical hardness; the substitution of one feature for another covariant feature is irrelevant when 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.

19.3.2 Experimental results

Figure 19.7 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

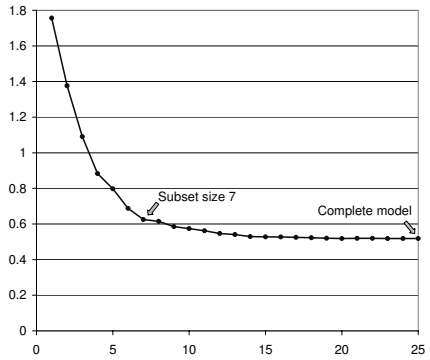

Figure 19.7: Linear Regression: Subset size vs. RMSE.

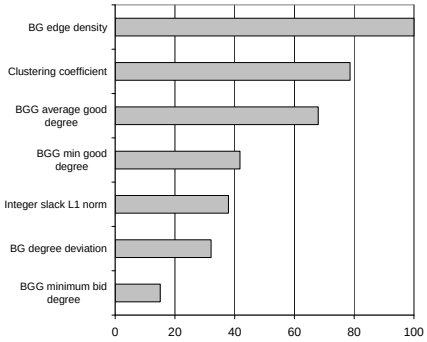

Figure 19.8: Linear Regression: Cost of omission for subset size 7.

did not cause a large decrease in RMSE, which suggested that the features in the eight-feature model were more highly correlated. Figure 19.8 shows the seven features in this model and their respective costs of omission (scaled to 100).

The most overarching conclusion we can draw from this data 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 the problem's constraints 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.

We now consider second-order models, where we had 350 features and

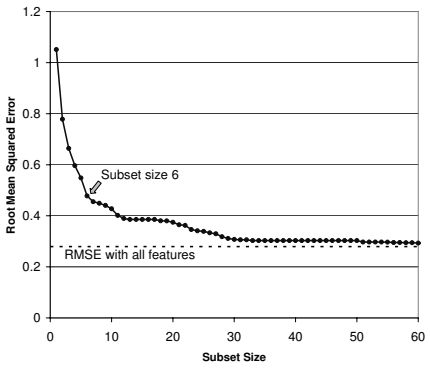

Figure 19.9: Quadratic Regression: Subset size vs. RMSE.

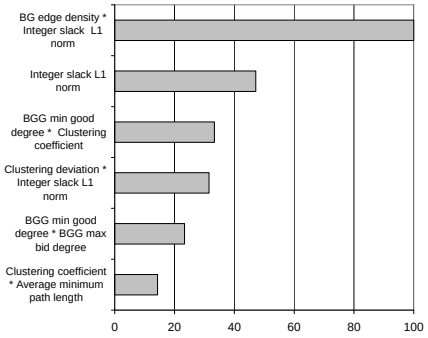

Figure 19.10: Quadratic Regression: Cost of omission for subset size 6.

thus exhaustive exploration of feature subsets was impossible. Instead, we used three different greedy subset selection methods (forward selection; backward selection; sequential replacement) and at each size chose the best subset among the three. Figure 19.9 describes the best subsets containing between 1 and 60 features for second-order models. Due to our use of greedy subset selection techniques, the subsets shown in Figure 19.9 are likely not the RMSE-minimizing subsets of the given sizes; nevertheless, we can still conclude that subsets of these sizes are *sufficient* to achieve the accuracies shown here. We observe that allowing interactions between features dramatically improved the accuracy of our very small-subset models; indeed, our 5-feature quadratic model outperformed our 25-feature linear model.

Figure 19.10 shows the costs of omission for the variables from the best six-feature subset. As in the case of our linear model, we observe that the

most critical features are structural: edge density of BG, the clustering coefficient and node degrees. 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 kept as the only first-order feature.

We can look at the features that were important to our quadratic and linear models in order to gain understanding about how our models work. The importance of the ℓ_1 norm is quite intuitive: the easiest problems can be completely solved by LP, yielding an ℓ_1 norm of 0; the norm is close to 0 for problems that are almost completely solved by LP (and hence usually do not require much search to resolve), and larger for more difficult problems. The BG edge density feature describes the overall constrainedness of the problem. Generally, we would expect that very highly constrained problems would be easy, since more constraints imply a smaller search space; however, our experimental results show that CPLEX takes a long time on such problems. It seems that either CPLEX's calculation of the LP bound at each node becomes much more expensive when the number of constraints in the LP increases substantially, or the accuracy of the LP relaxation decreases (along with the number of nodes that can be pruned); in either case this cost overwhelms the savings that come from searching in a smaller space. Some

other important features are intuitively similar to BG edge density. For example, the node degree statistics describe the max, min, average and standard deviation of the number of constraints in which each variable is involved; they indicate how quickly the search space can be expected to narrow as variables are given values (i.e., as bids are assigned to or excluded from the allocation). Similarly, the clustering coefficient features measure the extent to which variables that conflict with a given variable also conflict with each other, another indication of the speed with which the search space will narrow as variables are assigned. Finally, we can now understand the importance of the feature which was by far the most important in our 6-feature quadratic model: the product of the BG edge density and the integer slack ℓ_1 norm. Note that this feature takes a large value only when both BG edge density and ℓ_1 norm are large; the explanations above show that problems are easy for CPLEX whenever either of these features has a small value. Since BG edge density and ℓ_1 norm are relatively uncorrelated on our data, their product gives a powerful prediction of an instance's hardness.

It is also interesting to notice which features were consistently *excluded* by subset selection. In particular, it is striking that no price features were important in either our first- or second-order models (except implicitly, as

part of LP relaxation features). 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 (indeed, it led us to eliminate the "uniform random" (L1) distribution in which average price per good varied most dramatically across bids). Another group of features that were generally not chosen for small subsets were path length features: graph radius, diameter, average minimum path length, etc. It seems that statistics derived from neighbor relations in constraint graphs are much more meaningful for predicting hardness than other graph-theoretic statistics derived from notions of proximity or connectedness.

19.4 Using hardness models to build algorithm portfolios

When algorithms exhibit high runtime variance, one is faced with the problem of deciding which algorithm to use for solving a given instance. In 1976 Rice dubbed this the "algorithm selection problem" (Rice 1976). Though Rice offered few concrete techniques, all subsequent work on algorithm selection (e.g., (Gomes and Selman 2001; Lagoudakis and Littman

2000; Lagoudakis and Littman 2001; Lobjois and Lemaître 1998)) can be seen as falling into his framework. Despite this literature, however, the overwhelmingly most common approach to algorithm selection remains measuring different algorithms' performance on a given problem distribution, and then always selecting the algorithm with the lowest average runtime. This approach, which we dub "winner-take-all", has driven recent advances in algorithm design and refinement, but has resulted in the neglect of many algorithms that, while uncompetitive on average, offer excellent performance on particular problem instances. Our consideration of the algorithm selection literature, and our dissatisfaction with the winner-take-all approach, has led us to ask the following two questions. First, what general techniques can we use to perform per-instance (rather than per-distribution) algorithm selection? Second, once we have rejected the notion of winner-take-all algorithm evaluation, how should we evaluate novel algorithms? We address the first question here, and the second question in Section 19.5.

Given our existing technique for predicting runtime, we propose the following simple approach for the construction of algorithm portfolios:

1. Train a model for each algorithm, as described above.

2. Given an instance:

(a) Compute feature values

(b) Predict each algorithm's running time using runtime models

(c) Run the algorithm predicted to be fastest

19.4.1 Experimental results

In order to build an algorithm portfolio, we needed more algorithms for solving the WDP. In addition to CPLEX we considered two special-purpose algorithms from the combinatorial auctions literature for which a public implementation was available: GL (Gonen-Lehmann) (Gonen and Lehmann 2001), a simple branch-and-bound algorithm with CPLEX's LP solver as its heuristic and CASS (Fujishima et al. 1999), a more complex branch-and-bound algorithm with a non-LP heuristic. We used the methodology described in Section 19.1.1 to build regression models for GL and CASS; for the results in this section all models were learned using simple linear regression, without a log transformation on the response variable.[6]

Figure 19.11 compares the average runtimes of our three algorithms (CPLEX, CASS, GL) to that of the portfolio (note the change of scale on the graph, and the repeated CPLEX bar). Note that CPLEX would be chosen under winner-take-all algorithm selection. The "optimal" bar shows the performance of an ideal portfolio where algorithm selection is performed

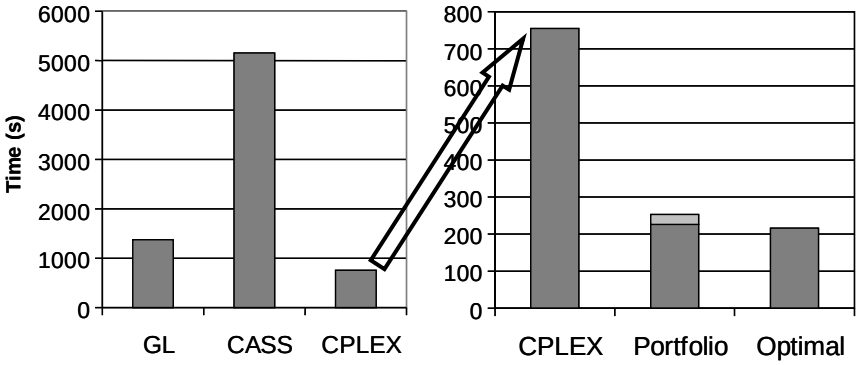

Figure 19.11: Algorithm and Portfolio Runtimes

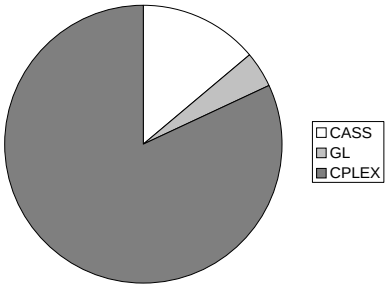

Figure 19.12: Optimal

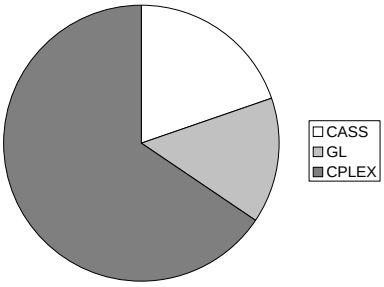

Figure 19.13: Selected

perfectly and with no overhead. The portfolio bar shows the time taken to compute features (light portion) and the time taken to run the selected algorithm (dark portion). Despite the fact that CASS and GL are much slower than CPLEX on average, the portfolio outperforms CPLEX by roughly a factor of 3. Moreover, neglecting the cost of computing features, our portfolio's selections take only 5% longer to run than the optimal selections.

Figures 19.12 and 19.13 show the frequency with which each algorithm is

selected in the ideal portfolio and in our portfolio. They illustrate the quality of our algorithm selection and the relative value of the three algorithms. Observe that our portfolio does not always make the right choice (in particular, it selects GL much more often than it should). However, most of the mistakes made by our models occur when both algorithms have very similar running times; these mistakes are not very costly, explaining why our portfolio's choices have a running time so close to optimal.

Observe that our variable importance analysis from Section 19.3.2 gives us some insight about why an algorithm like CASS is able to provide such large gains over algorithms like CPLEX and GL on a significant fraction of instances.[7] While holding the size of the data we have demonstrated the Unlike CASS, both GL and CPLEX use an LP relaxation heuristic. It is possible that when the number of constraints (and thus the bid graph edge density feature) increases, such heuristics become less accurate, or larger LP input size incurs substantially higher per-node costs. On the other hand, additional constraints reduce feasible search space size. Like many search algorithms, CASS often benefits whenever the search space becomes smaller; thus, CASS can achieve better overall performance on problems with a very large number of constraints.

We can also compare the performance of our portfolio to an alternative

portfolio that task-swaps among its constituent algorithms, described for example in Gomes and Selman (2001). Portfolios built using this approach always take time equal to the number of algorithms in the portfolio times the runtime of the optimal portfolio. Thus, on this dataset running the alternative portfolio would have been only very slightly faster than running CPLEX alone. If we observe that GL is rarely chosen by the optimal portfolio and that it contributes little over CPLEX when it is chosen, we can conclude that GL should be dropped from the task-swapping portfolio. Even if we do so, however, the alternate portfolio still takes nearly twice as long to run as the portfolio built using our techniques.

19.5 Using hardness models to induce hard distributions

Once we have recognized the value of selecting among existing WDP algorithms using a portfolio approach, it is necessary to reexamine the data we use to design and evaluate our algorithms. When the purpose of designing new algorithms is to reduce the time that our portfolio will take to solve problems, we should aim to produce new algorithms that *complement* that existing portfolio. First, it is essential to choose a distribution D that reflects the problems that will be encountered in practice. Given a portfolio,

the greatest opportunity for improvement is on instances that are hard for that portfolio, very common in D, or both. More precisely, the importance of a region of problem space is proportional to the amount of time the current portfolio spends working on instances in that region. (For previous work on generating hard test data on a different problem domain, see e.g., Selman et al. (1996).)

19.5.1 Inducing harder distributions

Let H_f be a model of portfolio runtime based on instance features, constructed as the minimum of the models that constitute the portfolio. By normalizing, we can reinterpret this model as a density function h_f. By the argument above, we should generate instances from the product of this distribution and our original distribution, D. However, it is problematic to sample from $D \cdot h_f$: D may be non-analytic (an instance generator), while h_f depends on features and so can only be evaluated after an instance has been created.

One way to sample from $D \cdot h_f$ is rejection sampling (Doucet et al. 2001): generate problems from D and keep them with probability proportional to h_f. Furthermore, if there exists a second distribution that is able to guide the sampling process towards hard instances, rejection sampling can use it to

reduce the expected number of rejections before an accepted sample. This is indeed the case for parameterized instance generators: e.g., all of the CATS (and legacy) distributions have some tunable parameters $\overrightarrow{p}$, and although the hardness of instances generated with the same parameter values can vary widely, $\overrightarrow{p}$ is (weakly) predictive of hardness. We can generate instances from $D \cdot h_f$ in the following way:[8]

1. Create a hardness model H_p with features $\overrightarrow{p}$, and normalize it to create a pdf, h_p.

2. Generate a large number of instances from $D \cdot h_p$.

3. Construct a distribution over instances by assigning each instance s probability proportional to $\frac{H_f(s)}{h_p(s)}$, and select an instance by sampling from this distribution.

Observe that if h_p turns out to be helpful, hard instances from $D \cdot h_f$ will be encountered quickly. Even in the worst case where h_p directs the search away from hard instances, observe that we still sample from the correct distribution because the weights are divided by $h_p(s)$ in step 3.

In our case, D is factored as $D_g \cdot D_{p_i}$, where D_g is a uniform distribution over the CATS and legacy instance generators in our dataset, each having a different parameter space, and D_{p_i} is a distribution over the parameters of

the chosen instance generator i. In this case it is difficult to learn a single H_p. A good solution is to factor h_p as $h_g \cdot h_{p_i}$, where h_g is a hardness model using only the choice of instance generator as a feature, and h_{p_i} is a hardness model in instance generator i's parameter space. Likewise, instead of using a single feature-space hardness model H_f, we can train a separate model for each generator $H_{f,i}$ and normalize each to a pdf $h_{f,i}$.[9] The goal is now to generate instances from the distribution $D_g \cdot D_{p_i} \cdot h_{f,i}$, which can be done as follows:

1. For every instance generator i, create a hardness model H_{p_i} with features $\overrightarrow{p_i}$, and normalize it to create a pdf, h_{p_i}.

2. Construct a distribution over instance generators h_g, where the probability of each generator i is proportional to the average hardness of instances generated by i.

3. Generate a large number of instances from $(D_g \cdot h_g) \cdot (D_{p_i} \cdot h_{p_i})$

   (a) select a generator i by sampling from $D_g \cdot h_g$

   (b) select parameters for the generator by sampling from $D_{p_i} \cdot h_{p_i}$

   (c) run generator i with the chosen parameters to generate an instance.

4. Construct a distribution over instances by assigning each instance s

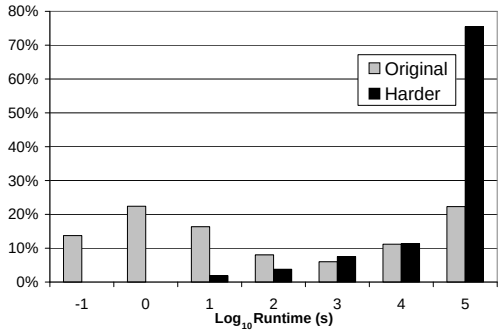

Figure 19.14: Inducing Harder Distributions

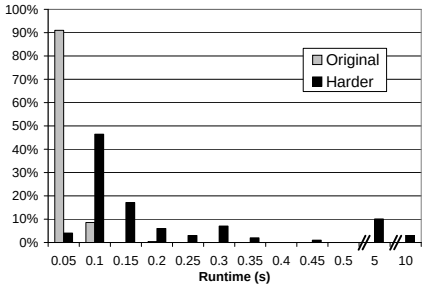

Figure 19.15: Matching

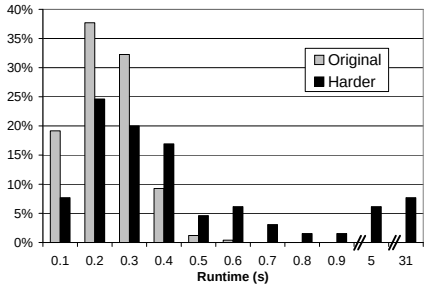

Figure 19.16: Scheduling

from generator i probability proportional to $\frac{H_{f,i}(s)}{h_g(s) \cdot h_{p_i}(s)}$, and select an instance by sampling from this distribution.

19.5.2 Experimental results

Due to the wide spread of runtimes in our composite distribution D (7 orders of magnitude) and the high accuracy of our model h_f, it is quite easy for our technique to generate harder instances. These results are presented in Figure 19.14. Because our runtime data was capped, there is no way to know if the hardest instances in the new distribution are harder than the hardest

instances in the original distribution; note, however, that very few easy instances are generated. Instances in the induced distribution came predominantly from the CATS "arbitrary" distribution, with most of the rest from "L3".

To demonstrate that our technique also works in more challenging settings, we sought a different distribution with small runtime variance. As described in Chapter 18, there has been ongoing discussion in the WDP literature about whether those CATS distributions that are relatively easy could be configured to be harder. We consider two easy distributions with low variance from CATS, *matching* and *scheduling*, and show that they can indeed be made much harder than originally proposed. Figures 19.15 and 19.16 show the histograms of the runtimes of the ideal portfolio before and after our technique was applied. In fact, for these two distributions we generated instances that were (respectively) 100 and 50 times harder than anything we had previously seen! Moreover, the *average* runtime for the new distributions was greater than the observed *maximum* running time on the original distribution.

19.6 Conclusion

In this chapter we showed how to build models of the empirical hardness of WDP, and discussed various applications for these models. First, we identified structural, distribution-independent features of WDP instances and showed that they contain enough information to predict CPLEX running time with high accuracy. Next, we showed that these models can be effective for straightforward prediction of running time, gaining deeper insight into empirical hardness through the analysis of learned models, the construction of algorithm portfolios, and tuning distributions for hardness.

Acknowledgments

This chapter is based on work first presented in Leyton-Brown et al. (2002), Leyton-Brown et al. (2003b) and Leyton-Brown et al. (2003a). We would therefore like to acknowledge the contributions of Galen Andrew and James McFadden, who were coauthors on Leyton-Brown et al. (2003b) and Leyton-Brown et al. (2003a).

Notes

[1]We must note that CPLEX is constantly being improved. Unfortunately,

it is not easy to rerun 3 CPU-years worth of experiments. The results presented here are specific to version 7.1, and might change in future versions. We emphasize, however, that both the techniques and features that we introduce here are quite general, and can be applied to any WDP solver. Furthermore, limited experiments with CPLEX 8.0 suggest that the qualitative runtime distribution and our models' accuracy remain very similar to the results presented here, at least for our WDP benchmark distributions.

[2]The attentive reader will notice the omission of the CATS Paths generator from this list. Indeed, we did initially include instances from this generator in our experiments. However, the definition of this generator changed substantially from version 1.0 of the CATS software (Leyton-Brown et al. 2000) to version 2.0 (Chapter 18). To avoid confusion we dropped these instances, though we note that the change did not make a significant difference to our experimental results.

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

[4]It would have been 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.

[5]Cross-validation is a standard machine learning technique which provides an unbiased estimate of test set error using only the training set. First, the training set is split into k different subsets. Validation set errors are then computed by performing learning in turn on each of $k-1$ of those subsets and evaluating resulting model on the remaining subset. The average of these k validation set errors is then used as an approximation of model's performance on test data.

[6]We argued above that applying a log transform to the response variable leads to regression models that minimize relative rather than absolute error. This is useful when building models with the goal of understanding why instances vary in empirical hardness. In this section, on the other hand, we care about building portfolios that will outperform their constituent algorithms in terms of average runtime. This implies that we are concerned with absolute error, and so we do not perform a log transform in this case.

[7]Observe that, in order to maintain continuity with other parts of the chapter such as this variable importance analysis, we have described the construction of algorithm portfolios optimized for fixed-size inputs. We have observed (both with combinatorial auctions and in other domains like SAT) that it is possible to build accurate runtime models with variable-size data and hence that our portfolio approach is not restricted in any way to fixed-size inputs. It is therefore worth emphasizing that our methodology for building both empirical hardness models and algorithm portfolios can be applied directly to the construction of models for variable-size data.

[8]In true rejection sampling step 2 would generate a single instance that would be then accepted or rejected in step 3. Our technique approximates this process, but doesn't require us to normalize H_f and guarantees that we will output an instance after generating a constant number of samples.

[9]However, the experimental results presented in Figures 19.14–19.16 use hardness models H_f trained on the whole dataset rather than using models trained on individual distributions. Learning new models would probably yield even better results.

References

Achlioptas, Dimitris, Carla P. Gomes, Henry A. Kautz and Bart Selman (2000). Generating satisfiable problem instances. *AAAI*.

Cheeseman, Peter, Bob Kanefsky and William M. Taylor (1991). Where the Really Hard Problems Are. *IJCAI-91*.

Doucet, Arnaud, Nando de Freitas and Neil Gordon (ed.) (2001). *Sequential Monte Carlo methods in practice*. Springer-Verlag.

Fujishima, Yuzo, Kevin Leyton-Brown and Yoav Shoham (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. *IJCAI*.

Gomes, Carla P. and Bart Selman (1997). Problem structure in the presence of perturbations. *AAAI/IAAI*.

Gomes, Carla P. and Bart Selman (2001). Algorithm portfolios. *Artificial Intelligence*, *126*(1-2), 43–62.

Gonen, Rica and Daniel Lehmann (2000). Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. *ACM Conference on Electronic Commerce*.

Gonen, Rica and Daniel Lehmann (2001). *Linear programming helps solving large multi-unit combinatorial auctions* (Technical Report TR-2001-8). Leibniz Center for Research in Computer Science.

Hastie, Trevor, Robert Tibshirani and Jerome Friedman (2001). *Elements of statistical learning*. Springer.

Horvitz, Eric, Yongshao Ruan, Carla P. Gomes, Henry A. Kautz, Bart Selman and David M. Chickering (2001). A Bayesian approach to tackling hard computational problems. *UAI*.

Korf, Richard E. and Michael Reid (1998). Complexity analysis of admissible heuristic search. *AAAI-98*.

Lagoudakis, Michail and Michael Littman (2000). Algorithm selection using reinforcement learning. *ICML*.

Lagoudakis, Michail and Michael Littman (2001). Learning to select branching rules in the DPLL procedure for satisfiability. *LICS/SAT*.

Leyton-Brown, Kevin, Eugene Nudelman, Galen Andrew, James McFadden and Yoav Shoham (2003a). Boosting as a metaphor for algorithm design. *Constraint Programming*.

Leyton-Brown, Kevin, Eugene Nudelman, Galen Andrew, James McFadden and Yoav Shoham (2003b). A portfolio approach to algorithm selection. *IJCAI-03*.

Leyton-Brown, Kevin, Eugene Nudelman and Yoav Shoham (2002). Learning the empirical hardness of optimization problems: The case of combinatorial auctions. *CP*.

Leyton-Brown, Kevin, Mark Pearson and Yoav Shoham (2000). Towards a universal test suite for combinatorial auction algorithms. *ACM EC*.

Lobjois, Lionel and Michel Lemaître (1998). Branch and bound algorithm selection by performance prediction. *AAAI*.

Monasson, Rémi, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman and

Lidror Troyansky (1998). Determining computational complexity for characteristic 'phase transitions'. *Nature*, *400*.

Rice, John R. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Ruan, Yongshao, Eric Horvitz and Henry Kautz (2002). Restart policies with dependence among runs: A dynamic programming approach. *CP*.

Sandholm, Tuomas (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, *135*, 1–54.

Selman, Bart, David G. Mitchell and Hector J. Levesque (1996). Generating hard satisfiability problems. *Artificial Intelligence*, *81*(1-2), 17–29.

Slaney, John and Toby Walsh (2001). Backbones in optimization and approximation. *IJCAI-01*.

Zhang, Weixiong (1999). *State-space search: Algorithms, complexity, extensions, and applications*. Springer.