Hardness-Aware Restart Policies

Yongshao Ruan
University of Washington
Seattle WA 98195
ruan@cs.washington.edu

Eric Horvitz
Microsoft Research
Redmond WA 98052
horvitz@microsoft.com

Henry Kautz
University of Washington
Seattle WA 98195
kautz@cs.washington.edu

Abstract

Recent work has demonstrated that it is possible to boost the efficiency of combinatorial search procedures via the use of principled restart policies. We present a coupling of machine learning and dynamic programming that extends prior efforts by endowing restart policies with knowledge of the hardness of the specific instance being solved. This ability allows a restart policy to take into consideration an updated probability distribution over hardness as a previously unseen instance is being solved. We discuss the methods, highlighting their importance for real-world applications of combinatorial search. Finally, we present the empirical results.

1 Introduction

A key observation made over the last decade is that procedures for solving difficult combinatorial search problems show great variance in running time [1, 2, 3, 4]. The large variance in run time has been a significant obstacle to creating real-time reasoning systems in numerous challenging domains.

In pursuit of fast and robust search algorithms, investigators have targeted the uncertainty in run time directly with techniques including restarts and algorithm portfolios [5, 6, 7, 8, 9]. We describe new methods that complement recent efforts on developing sound probabilistic restart policies [10, 11, 12]. The earlier work has centered on the use of machine learning to inform a reasoner about the probability distribution over the time required to solve a particular run of an instance drawn from a source distribution or *ensemble*. The efforts in predictive modeling to date have not addressed the challenges of handling the great variation in the hardness of different instances—as captured for example, by the median run time required to solve an instance, derived by solving the instance multiple times.

Developing a means for recognizing the hardness of specific instances is important for the real-world scenario of committing to the solution of a particular instance that has never been seen before. We attack this problem directly by developing new machinery that allows a solver to continue to *update beliefs about the overall hardness of a specific instance* that must be solved, conditioning on information gathered with repeated restarts.

We shall first summarize prior work on restart policies. Then, we present new methods for clustering an ensemble of instances into sub-ensembles to decrease hardness variation within sub-ensembles. The creation of instance sub-ensembles enables a system to learn and reason about the probability distribution over the hardness of the specific instance at hand. We next review the dynamic-programming approach to identifying ideal restart policies [12], now taking into consideration the likelihood of different hardnesses. We describe experiments we performed to test the methods. Finally, we discuss research opportunities.

2 Research on Restart Policies

The run time of backtracking heuristic search algorithms is notoriously unpredictable. Gomes *et al.* [7] demonstrated the effectiveness of randomized restarts on a variety of problems in scheduling, theorem-proving, and planning. In this approach, randomness is added to the branching heuristic of a systematic search algorithm; if the search algorithm does not find a solution within a given number of backtracks (referred to as the *cutoff*), the algorithm is restarted with a new random seed. Luby *et al.* [13] described restart policies for any stochastic process for two scenarios where runtime itself is the only observable: (i) when each run is a random sample from a known distribution, one can calculate a fixed optimal cutoff; (ii) when there is no knowledge of the distribution, a *universal schedule* mixing short and longer cutoffs comes within a log factor of the minimal run time.

Horvitz *et al.* [10] showed that it is possible to do better than Luby's fixed optimal policy by making observations of a variety of features related to the nature and progress of problem solving during an early portion of the run (referred to as the *observation horizon*) and learning, and then using, a Bayesian model to predict the length of each run. Under the assumption that each run is an independent random sample of one runtime distribution (RTD), [11] used observations to discriminate the potentially short runs from the long ones and then adopted different restart cutoffs for the two types of runs.

Ruan *et al.* [12] considered the case where there are k known distributions, and each run is a sample from *one* of the distributions—but the solver is not told *which* distribution. The paper showed how offline dynamic programming can be used to generate the optimal restart policy, and how the policy can be coupled with real-time observations to control restarting.

All of these scenarios can be taken to be simplified versions of the real-world situation where each distribution corresponds to a heterogeneous ensemble of instances, and the same problem instance is used for each run. The analysis of the real-world scenario requires machinery that relates the RTD of an ensemble to the RTD's of its individual instances under a randomized solver.

This paper addresses the challenge of creating techniques for tackling the real-world scenario. The method has three steps as follows:

1. *Collecting Data.* Training instances are randomly sampled from an ensemble of instances and each training instance is solved many times with a randomized solver to obtain its RTD. At the start of each run, we also collect a set of features which will be used for learning a decision which classifies instances into sub-ensembles, based on the observed features.
2. *Partitioning Ensembles.* Next, we partition the training instances into sub-ensembles so that instances in a sub-ensembles have similar RTD's. We employ machine learning to to build a predictive model that provides a probability distribution over the parent sub-ensemble for an instance.
3. *Constructing Restart Policies.* Then, we compose an optimal restart policy harnessing an offline dynamic programming approach [12]. As we shall see, such policies require more than a simple combination of optimal cutoffs for all RTD's.

We also undertook a set of experiments. After building the partitions and composing ideal *hardness-aware* restart polices, we sample a new set of instances from the ensemble and use them to test the efficacy of the constructed restart policies.

3 Variability in Problem Hardness

Many problem solving scenarios involve solving instances drawn from a distribution of problems of mixed hardness and where *every* sampled problem must be solved. It has proven extremely difficult in practice to define realistic problem ensembles where instances do not vary widely in hardness [14]. Different instances have widely varying RTD's (and thus, widely varying optimal fixed cutoffs) even in the well-known random 3-SAT problem ensembles with fixed clause-to-variable ratios [3].

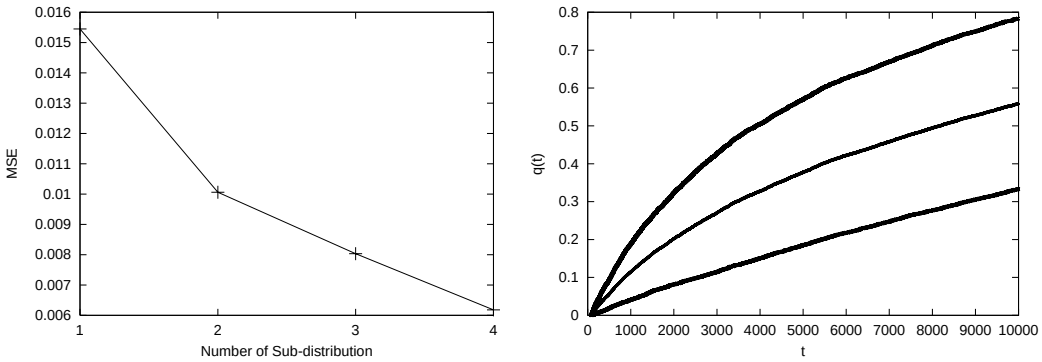

Figure 1: **Left:** Reduction of variability (MSE) within sub-ensembles with the number of sub-ensembles for the 3-colorable graph-coloring problem (GCP). The x-axis represents the number of sub-ensembles used, and the y-axis represents the average mean-squared error (MSE) measurement of the variability within each cluster. **Right:** Differences between RTD's of sub-ensembles and whole ensemble. The x-axis is the length of a run t, and the y-axis $q(t)$ is the probability of a run finishing in less than t steps. The middle curve is the RTD for the whole ensemble; upper curve is the RTD for the easy sub-ensemble and the lower curve is the RTD for the hard sub-ensemble.

Ideally, if all instances in an ensemble have the same run-time distribution, *i.e.*, there is no variability within the ensemble, the problem of finding the optimal restart policy for a newly given an instance from the ensemble collapses to the problem of finding Luby's optimal cutoff [13] for the run-time distribution of the whole ensemble. In practice, it is typical to see high variability in the intrinsic hardness of instances within an ensemble. Prior work on *quasigroup with holes* (QWH) demonstrated that variability in hardness of QWH instances with the same size of square and same number of holes is an order of magnitude or more [15]. The high variability in hardness remains for this problem even when we impose finer-grained properties on the ensemble. For example, we see great variation in the time for solving QWH instances even when we consider only those instances that show a balanced hole pattern [14]. Similar observation have been made on other domains, *e.g.*, combinatorial auctions [16].

The high variability in the hardness of instances makes it infeasible to adopt a fixed-cutoff restart policy. If the cutoff is set too low, some hard instances may have little chance being solved. On the other hand, if the cutoff is set too high, we may waste too much time on multiple unfruitful runs. Another candidate restart policy is Luby's *universal policy*. But as our experimental results will show, this policy is relatively inefficient.

We approach the problem of the high variability in run time by decomposing a given ensemble into a mixture of ensembles. We seek to divide the mixture into several sub-ensembles such that the variability in hardness within each sub-ensemble is significantly reduced and the RTD of the sub-ensemble is a reasonable approximation of the RTD of instances within the sub-ensemble.

4 Partition of a Problem Ensemble by Hardness

We can divide an ensemble into sub-ensembles by hardness in several different ways. We have focused on two approaches: (1) segmenting the ensemble into sub-ensembles by median run times of instances, and (2) by clustering instances by Euclidean distances between run-time distributions.

4.1 Median-Based Partition of Ensembles

With the median-based approach, we decompose a mixture of ensembles into sub-ensembles with methods that consider the median run times of instances observed after multiple runs.

We employed a straightforward approach to segmenting the instances: Instances whose

median run times are within certain ranges are clustered into the same sub-ensemble and ranges are divided such that sub-ensembles have equal numbers of instances. We used mean-squared error (MSE) to measure the variability within an ensemble or sub-ensemble.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(median(i) - median(e))^2$$

where $median(i)$ and $median(e)$ are the median run times of instance i and the ensemble respectively, and m is the number of instances in the ensemble.

4.2 Distribution-Based Partition of Ensembles

In another approach to creating sub-distributions, we take into consideration a measure of Euclidean distance among the run time distributions of different instances. We define Euclidean distance between run-time distributions of instance i and j as

$$DistQ(i,j) = \sqrt{\frac{1}{n}\sum_{t<T}(q_i(t) - q_j(t))^2}$$

where $q_i(t) = \sum_{t' \leq t} p_i(t')$ is the cumulative function of $p_i(t)$, the probability that a run will stop exactly at t for instance i, and n is the number of data points such that $t < T$.

Another potentially useful measure of difference between probability distributions for run time is the Kullback-Leibler (K-L) distance [17], based on a relative entropy formulation. The K-L distance is an asymmetric measure that is not formerly a distance (*e.g.*, it does not observe triangular equality). However, the definition of K-L distance requires that it is never the case that one probability distribution has a zero where the other does not, which is not always satisfied by our experimental data.

In our experiments, we used mean-squared error (MSE) as a measure of the variability within an ensemble or sub-ensemble and we employ a k-means clustering algorithm [18] to minimize the variability within each sub-ensemble.:

$$MSE = \frac{1}{m}\sum_{i=1}^{m} DistQ(i,e)^2$$

where $DistQ(i,e)$ is the Euclidean distance between run-time distributions of instance i and the ensemble, and m is the number of instances in the ensemble.

4.3 Sample Results of Partitioning

We wish to partition given ensembles into sub-ensembles by hardness so as to produce sub-ensembles with small variability in run time. As an example of how variability, represented as mean-squared error, within sub-ensembles changes with the number of sub-ensembles, we display, on the left side of Fig. 1 the results of the distribution-based partition method for the 3-colorable graph coloring problem (GCP). The graph shows that the variability within sub-ensembles decreases rapidly with an increasing number of sub-ensembles. We found in our experiments that partitioning an ensemble into a modest number of sub-ensembles significantly reduced the variance of run time.

The right side of Fig. 1 displays an example of the differences between the run-time distributions of two sub-ensembles and the whole ensemble for the 3-colorable Graph Coloring problem. The upper curve is the RTD for the easy sub-ensemble, the middle curve is the RTD for the whole ensemble, and the lower curve is the RTD for the hard sub-ensemble.

5 Generating Restart Policies with Dynamic Programming

The dynamic programming approach described [12] is particularly interesting to our study as we can apply it in a straightforward manner to construct restart policies for sub-

ensembles. We shall now review the dynamic programming solution within the framework of sub-ensembles partitioned by hardness.

We denote the run-time distributions of sub-ensembles as $D_1, ..., D_n$. Our goal is to find a policy $(t_1, t_2, ...)$, where t_i is the cutoff for ith run, such that the total number of steps to a solution is minimized. After each unsuccessful run, the solver's beliefs about the likelihood of the instance being generated by each sub-ensemble distribution are updated.

Let d_i be the prior probability of a run being chosen from distribution D_i, $p_i(t)$ as the probability that a run which has been selected from D_i will stop exactly at t, and $q_i(t) = \sum_{t' \leq t} p_i(t')$ as the cumulative function of $p_i(t)$, where $i = 1, ..., n$. We shall assume that p_i is non-trivial in the sense that $p_i(\inf) < 1$. Each state is a tuple of $(d_1, ..., d_n)$ and the set of actions for all states is the set of cutoffs. Given an action t, *i.e.*, cutoff = t, and state $S = (d_1, ..., d_n)$, the next possible state is either the termination state where the problem is solved, or the a non-terminal state $S' = (d'_1, ..., d'_n)$, where $d'_1, ..., d'_n$ are the updated probabilities. We denote the termination state as $S_0 = (0, .., 0)$. The termination state is a cost-free state; once the solver reaches that state it remains there at no further cost. The objective of restart control policy is to reach the termination state with minimal expected cost.

The optimal expected solution time from state $S = (d_1, ..., d_n)$ is the optimized sum of the immediate cost $R(S, t)$ and the optimal expected solution time of the two possible future states, which is given by the following Bellman equation:

$$E^*(S) = \min_t \{R(S, t) + P(S'|S, t)E^*(S')\}$$

As shown in [12], a Markov decision process (MDP) can be used to derive an optimal restart policy for this case.

Methods presented in [12] also highlighted methods for folding in a consideration of evidence observed over the course of runs, and continuing to update its beliefs about the hardness of instances. We use these methods here to update beliefs about the instance being drawn from each sub-ensemble D_i. In particular, we explore the case where an evidential feature F, reflecting the solver state or solver progress, is observed during a run. F is a function of the initial trace of the solver as calculated by a decision tree over low-level variables. F may be binary valued or multi-valued [12]. To encode F as part of a state, a state S can be denoted as $(d_1, ..., d_n, F)$.

In state $S_n = (d_1, ..., d_n, F_n)$, with cutoff setting to t, if a solution is found, the solver will be in the termination state S_0, or if no solution is found, in one of the states $S_{n+1} = (d'_1, ..., d'_n, F_{n+1})$ for all possible values of F_{n+1}. The transition probability from S_n to any other states is 0. Similarly, we can define the optimal expected solution time from state $S_n = (d_1, ..., d_n, F_n)$ as the following equation:

$$E^*(S_n) = \min_t \{R(S_n, t) + \sum_{F_{n+1}} P(S_{n+1}|S_n, t)E^*(S_{n+1})\}$$

Similarly, the restart policy for the case with run-time observations can be computed with the use of dynamic programming. As mentioned above, the specific series of cutoffs the policy generates depends upon the features observed during each run of an instance. Therefore the policy takes the general form of a tree rather than a list.

6 Experiments

We performed a set of empirical studies to explore our approach to finding optimal restart policies with and without observations. The benchmark domains we considered were the quasigroup domain (QCP and QWH) [5, 15, 14], graph coloring problems [19], and logistics planning problems [20].

Restart Policy	QCP (CSP)		QWH (SATZ)		GCP (SATZ)		Planning (SATZ)	
	ERT	%	ERT	%	ERT	%	ERT	%
DistQ(4), no predictor	95,846	50	74,191	39	21,172	68	13,502	44
Median(4), no predictor	93,797	51	68,720	43	19,542	70	13,238	45
Median(2), predictor	78,959	59	52,636	57	18,475	72	8,772	64
Median(3), predictor	82,346	57	58,894	52	18,814	71	8,910	63
Luby's optimal, No sub-ensemble	∞	-	∞	-	∞	-	∞	-
Luby's optimal, RTD of testing instances	33,869	83	30,084	76	15,083	77	4,005	84
Luby's universal	191,060	0	120,363	0	64,793	0	23,723	0

Table 1: Results of comparative experiments of policies for sub-ensembles, with and without observation, with Luby *et al.*'s universal restart policy. ERT is the expected run time (choice points) and improvements are measured over Luby *et al.*'s universal policy. $Median(n)$ ($DistQ(n)$) uses median run time (distance between run-time distributions) to cluster instances into n sub-ensembles.

6.1 Benchmark Domains and Solvers

Our first benchmark domain was the Quasigroup Completion Problem (QCP) [5]. For our studies, we generated totally 1,000 instances, of which 600 are satisfiable and the rest are unsatisfiable. All the instances are of order 30 with 337 unassigned variables or *holes*.

The second problem domain we explored is solving propositional satisfiability (SAT) encodings of the graph coloring problem (GCP). The instances used in our studies are generated using Culberson's flat graph generator [19]. The challenge is to determine whether an instance is 3-colorable. We generated 1,000 satisfiable instances. The instances are generated in such a way that all instances are 3-colorable but not 2-colorable.

We also explored a planning problem in the logistics domain [20]. We generated instances with 5 cities, 15 packages, 2 planes and 1 truck per city. We generate totally 1000 satisfiable instances. To decrease the variance among instances, all of the instances can be solved with 12 parallel steps but cannot be solved with 11 steps.

The randomized backtracking solvers for the problems encoded as SAT was Satz-Rand [7], a randomized version of the Satz system of Li and Anbulagan [21]. For the QCP problems, we experimented a specialized randomized CSP solver built using the ILOG constraint programming library.

6.2 Learning Models to Predict Hardness

We pursued the construction from data of decision trees we refer to as *distribution-predictors*. Such models serve to provide probabilities that an instance is derived from each of the different sub-ensembles under consideration, based on observations. To generate the predictive models, we implemented the methods introduced by Horvitz *et al.*.

For distribution predictors for instances solved by the Satz-rand solver, we found that the most predictive features for the domains explored are LambdaPos, a measure of interaction among binary clauses, and number of backtracks; for the CSP solver, the most predictive base features are the average number of colors available for filling squares and the average number of holes (See [10] for details about the features).

6.3 Comparative Analysis of Policies

In the experiments, each ensemble contains 1,000 instances with 1,000 runs each, of which 800 instances are used as training dataset and the other 200 are used as testing dataset. To identify ensembles, we used both $DistQ$ and median run times to clusters instances into a small number of sub-ensembles (not exceeding four ensembles).

We constructed the restart policies for sub-ensembles offline by using policy iteration for dynamic programming. In the procedure, we transform the continuous and infinite state space into a discrete state space and then apply finite-state dynamic programming methods.

For all of the experiments, we discretized the search space uniformly into 100 segments, taking into consideration the tradeoff between computational efficiency and accuracy. In another method, implemented to increase computational efficiency, we tested one cutoff for each 100 steps, instead of exhausting all possible cutoffs. The construction of policies via dynamic programming with policy iteration required from several minutes to hours depending on the number of feature values on a Pentium-800 machine with one gigabyte of memory.

To characterize the improvements gained with the dynamic dependent restart policies, we ran comparative experiments with Luby's optimal fixed restart policy for the known distribution of the whole training ensemble (*i.e.*, without the partitioning of the ensemble into sub-ensembles), and Luby's universal restart policy. Except for the case for the universal restart policy[1], we constructed restart policies, including all dynamic dependent restart policies for sub-ensembles and fixed optimal restart policies for the whole ensembles, from training data. We tested the policies on hold-out cases that had not been used for training. We also compared the results with the ideal cases where we have knowledge of the RTD's of testing instances. In this ideal case, the optimal expected run times of testing instances can be obtained by applying Luby's fixed optimal restart policy to the RTD for the particular instance being solved. (In practice, of course, the actual RTD of the instance being solved is not known.) All the comparison results are shown in Table 1.

For the problem domains studied, we found that the optimal fixed cutoff restart policy of Luby *et al.* for the training ensemble could not solve all the testing instances. The policy fails because the high variability of hardness within the ensembles leads to a situation where a low fixed cutoff indicated by the policy has little chance of solving hard instances.

We found that the hardness-aware restart policies, taking observations into consideration, are more efficient than the restart policies overlooking observations. We attributed the improvement in solution time, ranging from about 6% to 34% for the domains, to effectively harnessing the predictive models for identifying the proper sub-ensemble.

We found that for the restart policy with observations, the expected run times for a partitioning into two sub-ensembles are slightly better than those for three sub-ensembles. We believe that the slight degradation associated with the finer-grained partitioning is based in an induced tradeoff. Because we held the number of training cases constant, we could generate more accurate predictive models for the situation of fewer sub-ensembles; the gains in the predictive accuracy of models for the smaller partitioning overcame the higher variabilities within the sub-ensembles.

7 Summary and Directions

We introduced a method for constructing restart policies that leverage a continually updated probability distribution over sub-ensembles of different hardness *during* problem solving. The methods address the high variability in hardness of instances seen in attempts to solve real world problems. To illustrate the value of the methods, we performed several experiments that compare the new policies with static restart procedures. We developed learning predictors that can provide probability distributions over the hardness of a new instance.

In our ongoing work, we are working to enhance the efficiency of the restart policies through building more powerful predictive models. We are pursuing more powerful models for predicting both the hardness of an instance and the likely execution time of individual runs of an instance, *conditioned* on different sub-ensembles.

References

[1] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In D. Johnson and M. Trick, editors, *Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26*, pages 521–532. AMS, 1993.

[1] Luby 's universal restart policy does *not* change from distribution to distribution.

[2] I. Gent and T. Walsh. Easy Problems are Sometimes Hard. *Artificial Intelligence*, 70:335–345, 1993.

[3] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264:1297–1301, 1994.

[4] T. Hogg, B. Huberman, and C. Williams (Eds.). Phase Transitions and Complexity (Special Issue). *Artificial Intelligence*, 81(1–2), 1996.

[5] Carla P. Gomes and Bart Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 221–227, New Providence, RI, 1997. AAAI Press.

[6] Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed Distributions in Combinatorial Search. In Gert Smolka, editor, *Principles and practice of Constraint Programming (CP97) Lecture Notes in Computer Science*, pages 121–135, Linz, Austria., 1997. Springer-Verlag.

[7] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–438, New Providence, RI, 1998. AAAI Press.

[8] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100, 2000.

[9] AAAI-2000 Workshop on Leveraging Probability and Uncertainty in Computation, 2000.

[10] Eric Horvitz, Yongshao Ruan, Carla Gomes, Henry Kautz, Bart Selman, and Max Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings the 17th Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, pages 235–244, Seattle, USA, 2001.

[11] Henry Kautz, Eric Horvitz, Yongshao Ruan, Bart Selman, and Carla Gomes. Dynamic randomized restarts: Optimal restart policies with observation. AAAI2002, 2002.

[12] Yongshao Ruan, Eric Horvitz, and Henry Kautz. Restart policies with dependence among runs: A dynamic programming approach. In *Principles and Practice of Constraint Programming - CP 2002*, 2002.

[13] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Process. Letters*, 29:173–180, 1993.

[14] Henry Kautz, Yongshao Ruan, D. Achlioptas, Carla P. Gomes, Bart Selman, and Mark Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 351–358, 2001.

[15] Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, and Bart Selman. Generating satisfiable problem instances. In *AAAI/IAAI*, pages 256–261, 2000.

[16] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Principles and Practice of Constraint Programming - CP 2002*, 2002.

[17] S. Kullback. *Information Theory and Statistics*. 1968.

[18] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley symposium on mathematics, statistics and probability*, volume 1, pages 366–371, 1967.

[19] Joseph C. Culberson and Feng Luo. Exploring the k-colorable landscape with iterated greedy. In David S. Johnson and Michael A. Trick, editors, *Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Vol. 36*, pages 245–284, 1996.

[20] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1188–1194, Portland, OR, 1996. AAAI Press.

[21] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 366–371. AAAI Pess, 1997.