Hydra-MIP: Automated Algorithm Configuration and Selection for Mixed Integer Programming

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown

Department of Computer Science
University of British Columbia, Canada
{xulin730, hutter, hoos, kevinlb}@cs.ubc.ca

Abstract. State-of-the-art mixed integer programming (MIP) solvers are highly parameterized. For heterogeneous and *a priori* unknown instance distributions, no single parameter configuration generally achieves consistently strong performance, and hence it is useful to select from a portfolio of different configurations. HYDRA is a recent method for using automated algorithm configuration to derive multiple configurations of a single parameterized algorithm for use with portfolio-based selection. This paper shows that, leveraging two key innovations, HYDRA can achieve strong performance for MIP. First, we describe a new algorithm selection approach based on classification with a non-uniform loss function, which significantly improves the performance of algorithm selection for MIP (and SAT). Second, by modifying HYDRA's method for selecting candidate configurations, we obtain better performance as a function of training time.

1 Introduction

Mixed integer programming (MIP) is a general approach for representing constrained optimization problems with integer-valued and continuous variables. Because MIP serves as a unifying framework for NP-complete problems and combines the expressive power of integrality constraints with the efficiency of continuous optimization, it is widely used both in academia and industry. MIP used to be studied mainly in operations research, but has recently become an important tool in AI, with applications ranging from auction theory [19] to computational sustainability [8]. Furthermore, several recent advances in MIP solving have been achieved with AI techniques [7, 13].

One key advantage of the MIP representation is that highly optimized solvers can be developed in a problem-independent way. IBM ILOG's CPLEX solver[1] is particularly well known for achieving strong practical performance; it is used by over 1 300 corporations (including one-third of the Global 500) and researchers at more than 1 000

Proceedings of the 18^{th} RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2011).
In conjunction with IJCAI 2011, Barcelona, Spain, July 17-18, 2011.

[1] http://ibm.com/software/integration/optimization/cplex-optimization-studio/

universities [16]. Here, we propose improvements to CPLEX that have the potential to directly impact this massive user base.

State-of-the-art MIP solvers typically expose many parameters to end users; for example, CPLEX 12.1 comes with a 221-page parameter reference manual describing 135 parameters. The CPLEX manual warns that “integer programming problems are more sensitive to specific parameter settings, so you may need to experiment with them.” How should such solver parameters be set by a user aiming to solve a given set of instances? Obviously—despite the advice to “experiment”—effective manual exploration of such a huge space is infeasible; instead, an automated approach is needed.

Conceptually, the most straightforward option is to search the space of algorithm parameters to find a (single) configuration that minimizes a given performance metric (*e.g.*, average runtime). Indeed, CPLEX itself includes a self-tuning tool that takes this approach. A variety of problem-independent algorithm configuration procedures have also been proposed in the AI community, including I/F-Race [3], ParamILS [15, 14], and GGA [2]. Of these, only PARAMILS has been demonstrated to be able to effectively configure CPLEX on a variety of MIP benchmarks, with speedups up to several orders of magnitude, and overall performance substantially better than that of the CPLEX self-tuning tool [13].

While automated algorithm configuration is often very effective, particularly when optimizing performance on homogeneous sets of benchmark instances, it is no panacea. In fact, it is characteristic of NP-hard problems that no single solver performs well on all inputs (see, *e.g.*, [30]); a procedure that performs well on one part of an instance distribution often performs poorly on another. An alternative approach is to choose a *portfolio* of different algorithms (or parameter configurations), and to select between them on a per-instance basis. This *algorithm selection problem* [24] can be solved by gathering cheaply computable features from the problem instance and then evaluating a learned model to select the best algorithm [20, 9, 6]. The well-known SATZILLA [30] method uses a regression model to predict the runtime of each algorithm and selects the algorithm predicted to perform best. Its performance in recent SAT competitions illustrates the potential of portfolio-based selection: it is the best known method for solving many types of SAT instances, and almost always outperforms all of its constituent algorithms.

Portfolio-based algorithm selection also has a crucial drawback: it requires a strong and sufficiently uncorrelated portfolio of solvers. While the literature has produced many different approaches for solving SAT, there are few strong MIP solvers, and the ones that do exist have similar architectures. However, algorithm configuration and portfolio-based algorithm selection can be combined to yield *automatic portfolio construction* methods applicable to domains in which only a single, highly-parameterized algorithm exists.

Two such approaches have been proposed in the literature. HYDRA [28] is an iterative procedure. It begins by identifying a single configuration with the best overall performance, and then iteratively adds algorithms to the portfolio by applying an algorithm configurator with a customized, dynamic performance metric. At runtime, algorithms are selected from the portfolio as in SATZILLA. ISAC [17] first divides instance sets into clusters based on instance features using the G-means clustering algorithm, then applies an algorithm configurator to find a good configuration for each cluster. At runtime, ISAC computes the distance in feature space to each cluster centroid and selects the configuration for the closest cluster. We note two theoretical reasons to prefer HYDRA to ISAC. First, ISAC's clustering is solely based on distance in feature space, completely ignoring the importance of each feature to runtime. Thus, ISAC's performance can change dramatically if additional features are added (even if they are uninformative). Second, no amount of training time allows ISAC to recover from a misleading initial clustering or an algorithm configuration run that yields poor results. In contrast, HYDRA can recover from poor algorithm configuration runs in later iterations.

In this work, we show that HYDRA can be used to build strong portfolios of CPLEX configurations, dramatically improving CPLEX's performance for a variety of MIP benchmarks, as compared to ISAC, algorithm configuration alone, and CPLEX's default configuration. This achievement leverages two modifications to the original HYDRA approach, presented in Section 2. Section 3 describes the features and CPLEX parameters we identified for use with HYDRA, along with the benchmark sets upon which we evaluated it. Section 4 evaluates HYDRA-MIP and presents evidence that our improvements to HYDRA are also useful beyond MIP. Section 5 concludes and describes future work.

2 Improvements to Hydra

It is difficult to directly apply the original HYDRA method to the MIP domain, for two reasons. First, the data sets we face in MIP tend to be highly heterogeneous; preliminary prediction experiments (not reported here for brevity) showed that HYDRA's linear regression models were not robust for such heterogeneous inputs, sometimes yielding extreme mispredictions of more than ten orders of magnitude. Second, individual HYDRA iterations can take days to run—even on a large computer cluster—making it difficult for the method to converge within a reasonable amount of time. (We say that HYDRA has converged when substantial increases in running time stop leading to significant performance gains.)

In this section, we describe improvements to HYDRA that address both of these issues. First, we modify the model-building method used by the algorithm selector, using a classification procedure based on decision forests with a non-uniform loss function. Second, we modify HYDRA to add multiple solvers in each iteration and to reduce the cost of evaluating these candidate solvers, speeding up convergence. We denote the original method as $\texttt{Hydra}_{\texttt{LR},1}$ ("LR" stands for linear regression and "1" indicates the number of configurations added to the portfolio per iteration), the new method including only our first improvement as $\texttt{Hydra}_{\texttt{DF},1}$ ("DF" stands for decision forests), and the full new method as $\texttt{Hydra}_{\texttt{DF},k}$.

2.1 Decision forests for algorithm selection

There are many existing techniques for algorithm selection, based on either regression [30, 26] or classification[10, 9, 25, 23]. SATzilla [30] uses linear basis function regression to predict the runtime of each of a set of K algorithms, and picks the one with the best predicted performance. Although this approach has led to state-of-the-art performance for SAT, it does not directly minimize the cost of running the portfolio on a set of instances, but rather minimizes the prediction error separately in each of K predictive models. This has the advantage of penalizing costly errors (picking a slow algorithm over a fast one) more than less costly ones (picking a fast algorithm over a slightly faster one), but cannot be expected to perform well when training data is sparse. Stern et al [26] applied the recent Bayesian recommender system *Matchbox* to algorithm selection; similar to SATzilla, this approach is cost-sensitive and uses a regression model that predicts the performance of each algorithm. CPHYDRA[23] uses case-based reasoning to determine a schedule of constraint satisfaction solvers (instead of picking a single solver). Its k-nearest neighbor approach is simple and effective, but determines similarity solely based on instance features (ignoring instance hardness). Finally, ISAC uses a cost-agnostic clustering approach for algorithm selection. Our new selection procedure uses an explicit cost-sensitive loss function—punishing misclassifications in direct proportion to their impact on portfolio performance—without predicting runtime. Such an approach has never before been applied to algorithm selection: all existing classification approaches use a simple 0–1 loss function that penalizes all misclassification equally (*e.g.*, [25, 9, 10]). Specifically, this paper describes a cost-sensitive classification approach based on decision forests (DFs). Particularly for heterogeneous benchmark sets, DFs offer the promise of effectively partitioning the feature space into qualitatively different parts. In contrast to clustering methods, DFs take runtime into account when determining that partitioning.

We constructed cost-sensitive DFs as collections of T cost-sensitive decision trees [27]. Following [4], given n training data points with k features each, for each tree we construct a bootstrap sample of n training data points sampled uniformly at random with repetitions; during tree construction, we sample a random subset of $\log_2(k) + 1$ features at each internal node to be considered for splitting the data at that node. Predictions are based on majority votes across all T trees. For a set of m algorithms $\{s_1, \ldots, s_m\}$, an $n \times k$ matrix holding the values of k features for each of n training instances, and an $n \times m$ matrix P holding the performance of the m algorithms on the n instances, we construct our selector based on $m \cdot (m-1)/2$ pairwise cost-sensitive decision forests, determining the labels and costs as follows. For any pair of algorithms (i, j), we train a cost-sensitive decision forest $DF(i, j)$ on the following weighted training data: we label an instance q as i if $P(q, i)$ is better than $P(q, j)$, and as j otherwise; the weight for that instance is $|P(q, i) - P(q, j)|$. For test instances, we apply each $DF(i, j)$ to vote for either i or j and select the algorithm with the most votes as the best algorithm for that instance. Ties are broken by only counting the votes from those decision forests that involve algorithms which received equal votes; further ties are broken randomly.

We made one further change to the mechanism gleaned from SATZILLA. Originally, a subset of candidate solvers was chosen by determining the subset for which portfolio performance is maximized, taking into account model mispredictions. Likewise, a similar procedure was used to determine presolver policies. These internal optimizations were performed based on the same instance set used to train the models. However, this can be problematic if the model overfits the training data; therefore, in this work, we use 10-fold cross validation instead.

2.2 Speeding up convergence

HYDRA uses an automated algorithm configurator as a subroutine, which is called in every iteration to find a configuration that augments the current portfolio as well as possible. Since algorithm configuration is a hard problem, configuration procedures are incomplete and typically randomized. Because a single run of a randomized configuration procedure might not yield a high-performing parameter configuration, it is common practice to perform multiple runs in parallel and to use the configuration that performs best on the training set [12, 14, 28, 13].

Here, we make two modifications to HYDRA to speed up its convergence. First, in each iteration, we add k promising configurations to the portfolio, rather than just the single best. If algorithm configuration runs were inexpensive, this modification to HYDRA would not help: additional configurations could always be found in later iterations, if they indeed complemented the portfolio at that point. However, when each iteration must repeatedly solve many difficult MIP instances, it may be impossible to perform more than a small number of HYDRA iterations within any reasonable amount of time, even when using a computer cluster. In such a case, when many good (and rather different) configurations are found in an iteration, it can be wasteful to retain only one of these.

Our second change to HYDRA concerns the way that the 'best' configurations returned by different algorithm configuration runs are identified. $\texttt{Hydra}_{\texttt{DF},1}$ determines the 'best' of the configurations found in a number of independent configurator runs by evaluating each configuration on the full training set and selecting the one with best performance. This evaluation phase can be very costly: *e.g.*, if we use a cutoff time of 300 seconds per run during training and have 1 000 instances, then computing the training performance of each candidate configuration can take nearly four CPU days. Therefore, in $\texttt{Hydra}_{\texttt{DF},k}$, we select the configuration for which the configuration procedure's internal estimate of the average performance improvement over the existing portfolio is largest. This alternative is computationally cheap: it does not require any evaluations of configurations beyond those already performed by the configurator. However, it is also potentially risky: different configurator runs typically use the training instances in a different order and evaluate configurations using different numbers of instances. It is thus possible that the configurator's internal estimate of improvement for a parameter configuration is high, but that it turns out to not help for instances the configurator has not yet used.

Fortunately, adding k parameter configurations to the portfolio in each iteration mitigates this problem: if each of the k selected configurations has independent probability p of yielding a poor configuration, the probability of all k configurations being poor is only p^k.

3 MIP: Features, Data Sets, and Parameters

While the improvements to HYDRA presented above were motivated by MIP, they can nevertheless be applied to any domain. In this section, we describe all domain-*specific* elements of HYDRA-MIP: the MIP instance features upon which our models depend, the CPLEX parameters we configured, and the data sets upon which we evaluated our methods.

3.1 Features of MIP Instances

We constructed a large set of 139 MIP features, drawing on 97 existing features [21, 11, 17] and also including 42 new probing features. Specifically, existing work used features based on problem size, graph representations, proportion of different variable types (*e.g.*, discrete *vs* continuous), constraint types, coefficients of the objective function, the linear constraint matrix and the right hand side of the constraints. We extended those features by adding more descriptive statistics when applicable, such as medians, variation coefficients, and interquantile distances of vector-based features. For the first time, we also introduce a set of MIP probing features based on short runs of CPLEX using default settings. These contain 20 single probing features and 22 vector-based features. The single probing features are as follows. *Presolving features* (6 in total) are CPU times for presolving and relaxation, # of constraints, variables, nonzero entries in the constraint matrix, and clique table inequalities after presolving. *Probing cut usage features* (8 in total) are the number of each of 7 different cut types, and total cuts applied. *Probing result features* (6 in total) are MIP gap achieved, # of nodes visited), # of feasible solutions found, # of iterations completed, # of times CPLEX found a new incumbent by primal heuristics, and # of solutions or incumbents found. Our 22 vector-based features contain descriptive statistics (averages, medians, variation coefficients, and interquantile distances, *i.e.*, q90-q10) for the following 6 quantities reported by CPLEX over time: (a) improvement of objective function; (b) number of integer-infeasible variables at current node; (c) improvement of best integer solution; (d) improvement of upper bound; (e) improvement of gap; (f) nodes left to be explored (average and variation coefficient only).

3.2 CPLEX Parameters

Out of CPLEX 12.1's 135 parameters, we selected a subset of 74 parameters to be optimized. These are the same parameters considerd in [13], minus two parameters governing the time spent for probing and solution polishing. (These led to problems when the captime used during parameter optimization was different from that used at test time.) We were careful to keep all parameters fixed that change the problem formulation (*e.g.*, parameters such as the optimality gap below which a solution is considered optimal). The 74 parameters we selected affect all aspects of CPLEX. They include 12 preprocessing parameters; 17 MIP strategy parameters; 11 parameters controlling how aggressively to use which types of cuts; 8 MIP "limits" parameters; 10 simplex parameters; 6 barrier optimization parameters ; and 10 further parameters. Most parameters have an "automatic" option as one of their values. We allowed this value, but also included other values (all other values for categorical parameters, and a range of values for numerical parameters). Exploiting the fact that 4 parameters were conditional on others taking certain values, they gave rise to $4.75 \cdot 10^{45}$ distinct parameter configurations.

3.3 MIP Benchmark Sets

Our goal was to obtain a MIP solver that works well on heterogenous data. Thus, we selected four heterogeneous sets of MIP benchmark instances, composed of many well studied MIP instances. They range from a relatively simple combination of two homogenous subsets (CL∪REG) to heterogenous sets using instances from many sources (*e.g.*, MIX). While previous work in automated portfolio construction for MIP [17] has only considered very easy instances (ISAC(new) with a mean CPLEX default runtime below 4 seconds), our three new benchmarks sets are much more realistic, with CPLEX default runtimes ranging from seconds to hours.

CL∪REG is a mixture of two homogeneous subset, CL and REG. CL instances come from computational sustainability; they are based on real data used for the construction of a wildlife corridor for endangered grizzly bears in the Northern Rockies [8] and encoded as mixed integer linear programming (MILP) problems. We randomly selected 1000 CL instances from the set used in [13], 500 for training and 500 for testing. REG instances are MILP-encoded instances of the winner determination problem in combinatorial auctions. We generated 500 training and 500 test instances using the regions generator from the Combinatorial Auction Test Suite [22], with the number of bids selected uniformly at random from between 750 and 1250, and a fixed bids/goods ratio of 3.91 (following [21]).

CL∪REG∪RCW is the union of CL∪REG and another set of MILP-encoded instances from computational sustainability, RCW. These instances model the spread of the endangered red-cockaded woodpecker, conditional on decisions about certain parcels of land to be protected. We generated 990 RCW instances (10 random instances for each combination of 9 maps and 11 budgets), using the generator from [1] with the same parameter setting, except a smaller sample size of 5. We split these instances 50:50 into training and test sets.

ISAC(new) is a subset of the MIP data set from [17]; we could not use the entire set, since the authors had irretrievably lost their test set. We thus divided their 276 training instances into a new training set of 184 and a test set of 92 instances. Due to the small size of the data set, we did this in a stratified fashion, first ordering the instances based on CPLEX default runtime and then picking every third instance for the test set.

MIX subsets of the sets studied in [13]. It includes all instances from MASS (100 instances), MIK (120 instances), CLS (100 instances), and a subset of CL (120 instances) and REG200 (120 instances). (Please see [13] for the description of each underlying set.) We preserved the training-test split from [13], resulting in 280 training and 280 test instances.

4 Experimental Results

In this section, we examined HYDRA-MIP's performance on our MIP datasets. We began by describing the experimental setup, and then evaluated each of our improvements to $\text{Hydra}_{LR,1}$.

4.1 Experimental setup

For algorithm configuration we used PARAMILS version 2.3.4 with its default instantiation of FOCUSEDILS with adaptive capping [14]. We always executed 25 parallel configuration runs with different random seeds with a 2-day cutoff. (Running times were always measured using CPU time.) During configuration, the captime for each CPLEX run was set to 300 seconds, and the performance metric was penalized average runtime (PAR-10, where PAR-k of a set of r runs is the mean over the r runtimes, counting timed-out runs as having taken k times the cutoff time). For testing, we used a cutoff time of 3 600 seconds. In our feature computation, we used a 5-second cutoff for computing probing features. We omitted these probing features (only) for the very easy ISAC(new) benchmark set. We used the Matlab version R2010a implementation of cost-sensitive decision trees; our decision forests consisted of 99 such trees. All of our experiments were carried out on a cluster of 55 dual 3.2GHz Intel Xeon PCs with 2MB cache and 2GB RAM, running OpenSuSE Linux 11.1.

In our experiments, the total running time for the various HYDRA procedures was often dominated by the time required for running the configurator and therefore turned out to be roughly proportional to the number of HYDRA iterations performed. Each

DataSet	Model	Train (cross valid.)		Test		SF: LR/DF (Test)	
		Time	PAR (Solved)	Time	PAR (Solved)	Time	PAR
CL	LR	**39.7**	**39.7 (100%)**	39.4	39.4 **(100%)**	**1.00×**	**1.00×**
UREG	DF	**39.7**	**39.7 (100%)**	**39.3**	**39.3 (100%)**		
CLU	LR	105.1	105.1 **(100%)**	102.6	102.6 **(100%)**	**1.04×**	**1.04×**
REGURCW	DF	**98.8**	**98.8 (100%)**	**98.8**	**98.8 (100%)**		
	LR	2.68	2.68 **(100%)**	2.36	2.36 **(100%)**	**1.18×**	**1.18×**
ISAC(new)	DF	**2.19**	**2.19 (100%)**	**2.00**	**2.00 (100%)**		
	LR	52	52 **(100%)**	56	172 **(99.6%)**	**1.17×**	**1.05×**
MIX	DF	**48**	**48 (100%)**	**48**	**164 (99.6%)**		

Table 1. *MIPzilla performance (average runtime and PAR in seconds, and percentage solved), varying predictive models. Column SF gives the speedup factor achieved by cost-sensitive decision forests (DF) over linear regression (LR) on the test set.*

iteration required 50 CPU days for algorithm configuration, as well as validation time to (1) select the best configuration in each iteration (only for Hydra$_{\mathrm{LR},1}$ and Hydra$_{\mathrm{DF},1}$); and (2) gather performance data for the selected configurations. Since Hydra$_{\mathrm{DF},4}$ selects 4 solvers in each iteration, it has to gather performance data for 3 additional solvers per iteration (using the same captime as used at test time, 3 600 seconds), which roughly offsets its savings due to ignoring the validation step. Using the format (Hydra$_{\mathrm{DF},1}$, Hydra$_{\mathrm{DF},4}$), the overall runtime requirements in CPU days were as follows: (366,356) for CLUREG; (485, 422) for CLUREGURCW; (256,263) for ISAC(new); and (274,269) for MIX. Thus, the computational cost for each iteration of Hydra$_{\mathrm{LR},1}$ and Hydra$_{\mathrm{DF},1}$ was similar.

4.2 Algorithm selection with decision forests

To assess the impact of our improved algorithm selection procedure, we evaluated it in the context of SATzilla-style portfolios of different CPLEX configurations, dubbed MIPzilla. As component solvers, we always used the CPLEX default plus CPLEX configurations optimized for the various subsets of our four benchmarks. Specifically, for ISAC(new) we used the six configurations found by GGA in [17]. For CLUREG, CLUREGURCW, and MIX we used one configuration optimized for each of the benchmark instance sets that were combined to create the distribution (*e.g.*, CL and REG for CLUREG). We took all such optimized configurations from [13], and manually optimized the remaining configurations using PARAMILS.

In Table 1, we presented performance results for MIPzilla on our four MIP benchmark sets, contrasting the original linear regression (LR) models with our new cost-sensitive decision forests (DF). Overall, MIPzilla was never worse with DF than with LR, and sometimes substantially better. For relatively simple data sets, such as CLUREG and CLUREGURCW, the difference between the models was quite small. For

DataSet	Model	Train (cross valid.)		Test		SF: LR/DF (Test)	
		Time	PAR (Solved)	Time	PAR (Solved)	Time	PAR
	LR	172	332 (**99.5%**)	177	458 (99.1%)	**1.08×**	**1.13×**
RAND	DF	**147**	**308 (99.5%)**	**164**	**405 (99.3%)**		
	LR	518	2224 (94.7%)	549	2858 (92.9%)	**1.16×**	**1.26×**
HAND	DF	**363**	**1327 (97.0%)**	**475**	**2268 (94.4%)**		
	LR	459	2195 (94.6%)	545	3085 (92.1%)	**1.12×**	**1.34×**
INDU	DF	**382**	**1635 (96.1%)**	**487**	**2300 (94.4%)**		

Table 2. SATZILLA *performance (average runtime and PAR in seconds, and percentage solved), varying predictive models. Column SF gives the speedup factor achieved by cost-sensitive decision forests (DF) over linear regression (LR) on the test set.*

more heterogeneous data sets, MIPzilla performed much better with DF than with LR: *e.g.*, 18% and 17% better in terms of final portfolio runtime in the case of ISAC(new) and MIX. Overall, our new cost-sensitive classification-based algorithm selection was clearly preferable to the previous mechanism based on linear regression. In further experiments, we also evaluated alternate approaches based on random regression forests (trained separately for each algorithm as in the linear regression approach), decision forests without costs, and support vector machines (SVMs) both with and without costs. We found that the cost-sensitive variants always outperformed the cost-free ones. In these more extensive experiments, we observed that cost-sensitive DF always performed very well and linear regression performed inconsistently, with especially poor performance on heterogenous data sets.

Our improvements to the algorithm selection procedure, although motivated by the application to MIP, were in fact problem independent. We therefore conducted an additional experiment to evaluate the effectiveness of SATZILLA based on our new cost-sensitive decision forests, compared to the original version using linear regression models. We used the same data used for building SATzilla2009 [29]. The number of training/test instances were 1211/806 (RAND category with 17 candidate solvers), 672/447 (HAND category with 13 candidate solvers) and 570/379 (INDU category with 10 candidate solvers). Table 2 shows that by using our new cost-sensitive decision forest, we improved SATZILLA's performance 29% (in average over three categories) in terms of PAR over the previous (competition-winning) version of SATZILLA; for the important industrial category, we observed PAR improvements of 34%. Because there exists no highly parameterized SAT solver with strong performance across problem classes (analogous to CPLEX for MIP), we did not investigate HYDRA for SAT. [2] However, we noted that this paper's findings suggest that there is merit in constructing such highly parameterized solvers for SAT and other NP-hard problems.

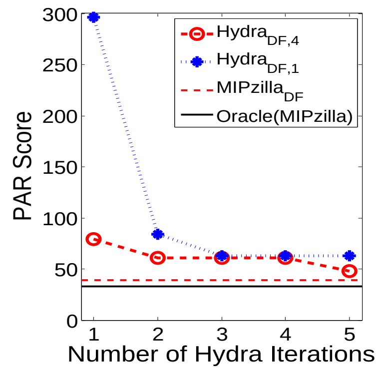
[2] The closest to a SAT equivalent of what CPLEX is for MIP would be MiniSAT [5], but it does not expose many parameters and does not perform well for random instances. The highly parameterized SATenstein solver [18] cannot be expected to perform well across the board for SAT; in particular, local search is not the best method for highly structured instances.

DataSet	Solver	Train (cross valid.)		Test	
		Time	PAR (Solved)	Time	PAR (Solved)
CL UREG	Default	424	1687 (96.7%)	424	1493 (96.7%)
	ParamILS	145	339 (99.4%)	134	296 (99.5%)
	$\text{Hydra}_{DF,1}$	64	97 (99.9%)	63	63 (**100%**)
	$\text{Hydra}_{DF,4}$	**42**	**42** (**100%**)	**48**	**48** (**100%**)
	MIPzilla	**40**	**40** (**100%**)	**39**	**39** (**100%**)
	Oracle (MIPzilla)	**33**	**33** (**100%**)	**33**	**33** (**100%**)
CL UREG URCW	Default	405	1532 (96.5%)	406	1424 (96.9%)
	ParamILS	148	148 (**100%**)	151	151 (**100%**)
	$\text{Hydra}_{DF,1}$	**89**	**89** (**100%**)	**95**	**95** (**100%**)
	$\text{Hydra}_{DF,4}$	106	106 (**100%**)	112	112 (**100%**)
	MIPzilla	**99**	**99** (**100%**)	**99**	**99** (**100%**)
	Oracle (MIPzilla)	**89**	**89** (**100%**)	**89**	**89** (**100%**)
ISAC (new)	Default	3.98	3.98 (**100%**)	3.77	3.77 (**100%**)
	ParamILS	2.06	2.06 (**100%**)	2.13	2.13 (**100%**)
	$\text{Hydra}_{LR,1}$	1.67	1.67 (**100%**)	1.52	1.52 (**100%**)
	$\text{Hydra}_{DF,1}$	1.2	1.2 (**100%**)	1.42	1.42 (**100%**)
	$\text{Hydra}_{DF,4}$	**1.05**	**1.05** (**100%**)	**1.17**	**1.17** (**100%**)
	MIPzilla	2.19	2.19 (**100%**)	2.00	2.00 (**100%**)
	Oracle (MIPzilla)	1.83	1.83 (**100%**)	1.81	1.81 (**100%**)
MIX	Default	182	992 (97.5%)	156	387 (99.3%)
	ParamILS	139	717 (98.2%)	126	357 (99.3%)
	$\text{Hydra}_{LR,1}$	74	74 (**100%**)	90	205 (**99.6%**)
	$\text{Hydra}_{DF,1}$	60	60 (**100%**)	65	181 (**99.6%**)
	$\text{Hydra}_{DF,4}$	**53**	**53** (**100%**)	**62**	**177** (**99.6%**)
	MIPzilla	**48**	**48** (**100%**)	**48**	**164** (**99.6%**)
	Oracle (MIPzilla)	**34**	**34** (**100%**)	**39**	**155** (**99.6%**)

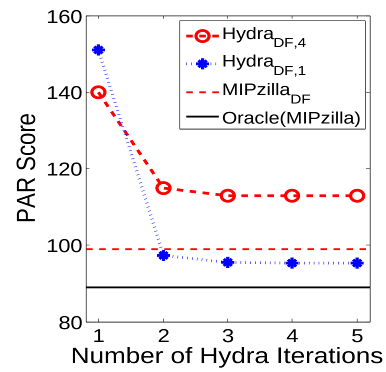
Table 3. *Performance (average runtime and PAR in seconds, and percentage solved) of $\text{Hydra}_{DF,4}$, $\text{Hydra}_{DF,1}$ and $\text{Hydra}_{LR,1}$ after 5 iterations.*

4.3 Evaluating HYDRA-MIP

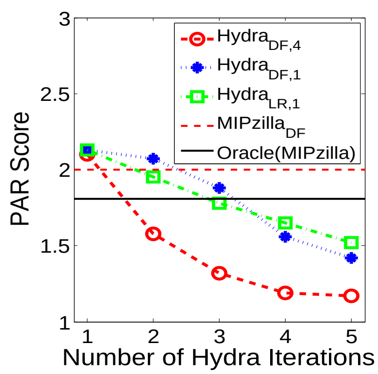
Next, we evaluated our full $\text{Hydra}_{DF,4}$ approach for MIP; on all four MIP benchmarks, we compared it to $\text{Hydra}_{DF,1}$, to the best configuration found by PARAMILS, and to the CPLEX default. For ISAC(new) and MIX we also assessed $\text{Hydra}_{LR,1}$. We did not do so for CLUREG and CLUREGURCW because, based on the results in Table 1, we expected the DF and LR models to perform almost identically. Table 3 presents these results. First, comparing $\text{Hydra}_{DF,4}$ to PARAMILS alone and to the CPLEX default, we observed that $\text{Hydra}_{DF,4}$ achieved dramatically better performance, yielding between 2.52-fold and 8.83-fold speedups over the CPLEX default and between 1.35-fold and 2.79-fold speedups over the configuration optimized with PARAMILS in terms of average runtime. Note that (due probably to the heterogeneity of the data sets) the built-in CPLEX self-tuning tool was unable to find any configurations better than the default for any of



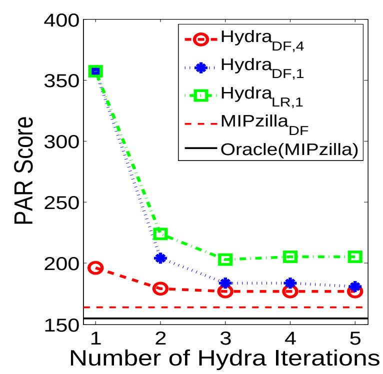
(a) `CLUREG`



(b) `CLUREGURCW`



(c) `ISAC(new)`



(d) `MIX`

Fig. 1. *Performance per iteration for* $\texttt{Hydra}_{DF,4}$*,* $\texttt{Hydra}_{DF,1}$ *and* $\texttt{Hydra}_{LR,1}$*, evaluated on test data.*

our four data sets. Compared to $\texttt{Hydra}_{LR,1}$, $\texttt{Hydra}_{DF,4}$ yielded a 1.3-fold speedup for `ISAC(new)` and a 1.5-fold speedup for `MIX`. $\texttt{Hydra}_{DF,4}$ also typically performed better than our intermediate procedure $\texttt{Hydra}_{DF,1}$, with speedup factors up to 1.21 (`ISAC(new)`). However, somewhat surprisingly, it actually performed worse for one distribution, `CLUREGURCW`. We analyzed this case further and found that in $\texttt{Hydra}_{DF,4}$, after iteration three PARAMILS did not find any configurations that would further improve the portfolio, even with a perfect algorithm selector. This poor PARAMILS performance could be explained by the fact that HYDRA's dynamic performance metric only rewarded configurations that made progress on solving some instances better; almost certainly starting in a poor region of configuration space, PARAMILS did not find configurations that made progress on *any* instances over the already strong portfolio, and thus lacked guidance towards better regions of configuration space. We believed that this problem could be addressed by means of better configuration procedures in the future.

Figure 1 shows the test performance the different HYDRA versions achieved as a function of their number of iterations, as well as the performance of the MIPzilla portfolios we built manually. When building these MIPzilla portfolios for CL∪REG, CL∪REG∪RCW, and MIX, we exploited ground truth knowledge about the constituent subsets of instances, using a configuration optimized specifically for each of these subsets. As a result, these portfolios yielded very strong performance. Although our various HYDRA versions did not have access to this ground truth knowledge, they still roughly matched MIPzilla's performance (indeed, $\text{Hydra}_{\text{DF},1}$ outperformed MIPzilla on CL∪REG). For ISAC(new), our baseline MIPzilla portfolio used CPLEX configurations obtained by ISAC [17]; all HYDRA versions clearly outperformed MIPzilla in this case, which suggests that its constituent configurations are suboptimal. For ISAC(new), we observed that for (only) the first three iterations, $\text{Hydra}_{\text{LR},1}$ outperformed $\text{Hydra}_{\text{DF},1}$. We believed that this occurred because in later iterations the portfolio had stronger solvers, making the predictive models more important. We also observed that $\text{Hydra}_{\text{DF},4}$ consistently converged more quickly than $\text{Hydra}_{\text{DF},1}$ and $\text{Hydra}_{\text{LR},1}$. While $\text{Hydra}_{\text{DF},4}$ stagnated after three iterations for data set CL∪REG∪RCW (see our discussion above), it achieved the best performance at every given point in time for the three other data sets. For ISAC(new), $\text{Hydra}_{\text{DF},1}$ did not converge after 5 iterations, while $\text{Hydra}_{\text{DF},4}$ converged after 4 iterations and achieved better performance. For the other three data sets, $\text{Hydra}_{\text{DF},4}$ converged after two iterations. The performance of $\text{Hydra}_{\text{DF},4}$ after the first iteration (i.e., with 4 candidate solvers available to the portfolio) was already very close to the performance of the best portfolios for MIX and CL∪REG.

4.4 Comparing to ISAC

We spent a tremendous amount of effort attempting to compare $\text{Hydra}_{\text{DF},4}$ with ISAC [17], since ISAC is also a method for automatic portfolio construction and was previously applied to a distribution of MIP instances. ISAC's authors supplied us with their their training instances and the CPLEX configurations their method identified, but are generally unable to make their code available to other researchers and, as mentioned previously, were unable to recover their test data. We therefore compared $\text{Hydra}_{\text{DF},4}$'s and ISAC's relative speedups over the CPLEX default (thereby controlling for different machine architectures) on their training data. We note that $\text{Hydra}_{\text{DF},4}$ was given only 2/3 as much training data as ISAC (due to the need to recover a test set from [17]'s original training set); the methods were evaluated using only the original ISAC training set; the data set is very small, and hence high-variance; and all instances were quite easy even for the CPLEX default. In the end, $\text{Hydra}_{\text{DF},4}$ achieved a 3.6-fold speedup over the CPLEX default, as compared to the 2.1-fold speedup reported in [17].

As shown in Figure 1, all versions of HYDRA performed much better than a MIPzilla portfolio built from the configurations obtained from ISAC's authors for the ISAC(new) dataset. In fact, even a perfect oracle of these configurations only achieved an average runtime of 1.82 seconds, which is a factor of 1.67 slower than $\texttt{Hydra}_{\texttt{DF},4}$.

5 Conclusion

In this paper, we showed how to extend HYDRA to achieve strong performance for heterogeneous MIP distributions, outperforming CPLEX's default, PARAMILS alone, ISAC and the original HYDRA approach. This was done using a cost-sensitive classification model for algorithm selection (which also lead to performance improvements in SATZILLA), along with improvements to HYDRA's convergence speed. In future work, we plan to investigate more robust selection criteria for adding multiple solvers in each iteration of $\texttt{Hydra}_{\texttt{DF},k}$ that consider both performance improvement and performance correlation. Thus, we may be able to avoid the stagnation we observed on CL∪REG∪RCW. We expect that $\texttt{Hydra}_{\texttt{DF},k}$ can be further strengthened by using improved algorithm configurators, such as model-based procedures. Overall, the availability of effective procedures for constructing portfolio-based algorithm selectors, such as our new HYDRA, should encourage the development of highly parametrized algorithms for other prominent NP-hard problems in AI, such as planning and CSP.

References

1. K. Ahmadizadeh, C. Dilkina, B.and Gomes, and A. Sabharwal. An empirical study of optimization for maximizing diffusion in networks. In *CP*, 2010.
2. C. Ansotegui, M. Sellmann, and K. Tierney. A gender-based genetic algorithm for the automatic configuration of solvers. In *CP*, pages 142–157, 2009.
3. M. Birattari, Z. Yuan, P. Balaprakash, and T. Stüzle. *Empirical Methods for the Analysis of Optimization Algorithms*, chapter F-race and iterated F-race: an overview. 2010.
4. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
5. N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th Intl. Conf. on Theory and Applications of Satisfiability Testing, LNCS*, volume 2919, pages 502–518, 2004.
6. C. Gebruers, B. Hnich, D. Bridge, and E. Freuder. Using CBR to select solution strategies in constraint programming. In *ICCBR*, pages 222–236, 2005.
7. A. Gilpin and T. Sandholm. Information-theoretic approaches to branching in search. *Discrete Optimization*, 2010. doi:10.1016/j.disopt.2010.07.001.
8. C. P. Gomes, W. van Hoeve, and A. Sabharwal. Connections in networks: A hybrid approach. In *CPAIOR*, 2008.
9. A. Guerri and M. Milano. Learning techniques for automatic algorithm portfolio selection. In *ECAI*, pages 475–479, 2004.
10. E. Horvitz, Y. Ruan, C. P. Gomes, H. Kautz, B. Selman, and D. M. Chickering. A Bayesian approach to tackling hard computational problems. In *UAI*, pages 235–244, 2001.
11. F. Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University Of British Columbia, Computer Science, 2009.

12. F. Hutter, D. Babić, H. H. Hoos, and A. J. Hu. Boosting Verification by Automatic Tuning of Decision Procedures. In *FMCAD*, pages 27–34, Washington, DC, USA, 2007. IEEE Computer Society.
13. F. Hutter, H. H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In *CPAIOR*, 2010.
14. F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.
15. F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *AAAI*, pages 1152–1157, 2007.
16. IBM. IBM ILOG CPLEX Optimizer – Data Sheet. Available online: `ftp://public.dhe.ibm.com/common/ssi/ecm/en/wsd14044usen/WSD14044USEN.PDF`, 2011.
17. S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. ISAC - instance specific algorithm configuration. In *ECAI*, 2010.
18. A. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown. SATenstein: Automatically building local search SAT solvers from components. pages 517–524, 2009.
19. D. Lehmann, R. Müller, and T. Sandholm. The winner determination problem. In *Combinatorial Auctions*, chapter 12, pages 297–318. 2006.
20. K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. A portfolio approach to algorithm selection. In *IJCAI*, pages 1542–1543, 2003.
21. K. Leyton-Brown, E. Nudelman, and Y. Shoham. Empirical hardness models: Methodology and a case study on combinatorial auctions. *Journal of the ACM*, 56(4):1–52, 2009.
22. K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM-EC*, pages 66–76, 2000.
23. E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.
24. J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.
25. H. Samulowitz and R. Memisevic. Learning to solve QBF. In *AAAI*, pages 255–260, 2007.
26. D. Stern, R. Herbrich, T. Graepel, H. Samulowitz, L. Pulina, and A. Tacchella. Collaborative expert portfolio management. In *AAAI*, pages 210–216, 2010.
27. K. M. Ting. An instance-weighting method to induce cost-sensitive trees. *IEEE Trans. Knowl. Data Eng.*, 14(3):659–665, 2002.
28. L. Xu, H. H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *AAAI*, pages 210–216, 2010.
29. L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. SATzilla2009: an Automatic Algorithm Portfolio for SAT. Solver description, SAT competition 2009, 2009.
30. L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, June 2008.