Automated Configuration of Mixed Integer Programming Solvers

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

University of British Columbia, 2366 Main Mall, Vancouver BC, V6T 1Z4, Canada
{hutter,hoos,kevinlb}@cs.ubc.ca

Abstract. State-of-the-art solvers for mixed integer programming (MIP) problems are highly parameterized, and finding parameter settings that achieve high performance for specific types of MIP instances is challenging. We study the application of an automated algorithm configuration procedure to different MIP solvers, instance types and optimization objectives. We show that this fully-automated process yields substantial improvements to the performance of three MIP solvers: CPLEX, GUROBI, and LPSOLVE. Although our method can be used "out of the box" without any domain knowledge specific to MIP, we show that it nevertheless outperforms the only special-purpose automated tuning tool for MIP of which we are aware, which is part of CPLEX.

1 Introduction

Current state-of-the-art mixed integer programming (MIP) solvers are highly parameterized. These parameters give users control over a wide range of design choices, including: which preprocessing techniques to apply; what balance to strike between branching and cutting; which types of cuts to apply; and the details of the underlying linear (or quadratic) programming solver. Solver developers typically take great care to identify default parameter settings that are robust and achieve good performance across a variety of problem types. However, the best combinations of parameter settings differ across problem types, which is of course the reason that such design choices are exposed as parameters. Thus, when a user is interested only in good performance for a given family of problem instances, as is the case in many application situations, it is often possible to substantially outperform the default configuration of the solver.

When the number of parameters is large, finding a solver configuration that leads to good empirical performance is a challenging optimization problem. (For example, this is the case for CPLEX: in version 12, its 221-page parameter reference manual describes 135 parameters that affect the search process.) MIP solvers exist precisely because humans are not good at solving high-dimensional optimization problems. Nevertheless, parameter optimization is usually performed manually. Doing so is tedious and laborious, requires considerable expertise, and often leads to results far from optimal.

There has been recent interest in automating the process of parameter optimization for MIP. The idea is to require the user only to specify a set of problem instances of interest and a performance metric, and then to trade machine time for human time to automatically identify a parameter configuration that achieves good performance. Notably, IBM ILOG CPLEX—the most widely used commercial MIP solver—introduced an automated tuning tool in version 11. In our own recent work, we proposed methods for the automated configuration of complex, black-box algorithms [22, 21, 20, 17].

While we mostly focused on solvers for propositional satisfiability (based on both local and tree search), we also conducted preliminary experiments that showed the promise of our methods for MIP. Specifically, we studied the configuration of CPLEX 10.1.1, considering 5 types of MIP instances [21].

The main contribution of this paper is a thorough study of the applicability of our black-box techniques to the MIP domain. We go beyond all previous work by configuring three different MIP solvers (GUROBI, LPSOLVE, and the more recent CPLEX versions 11.2 & 12.1), by considering a wider range of instance distributions, by considering multiple configuration objectives (notably, performing the first study on automatically minimizing the optimality gap), and by comparing our method to CPLEX's automated configuration tool (new in version 11). Our results show that our approach consistently sped up all three MIP solvers and also clearly outperformed the CPLEX tuning tool. For example, for a set of real-life instances from computational sustainability, our approach sped up CPLEX by a factor of 23 while the tuning tool returned the CPLEX defaults. For GUROBI, speedups were consistent but small (up to a factor of 3), and for LPSOLVE they reached a factor up to 153.

The remainder of this paper is organized as follows. In the next section, we describe automated algorithm configuration, including existing tools and applications. Then, we describe the MIP solvers we configure (Section 3) and discuss the setup of our experiments (Section 4). Next, we report results for optimizing both the runtime of the MIP solvers (Section 5) and the optimality gap they achieve within a fixed time (Section 6). We then compare our approach to the CPLEX tuning tool (Section 7) and conclude with some general observations and an outlook on future work (Section 8).

2 Automated Algorithm Configuration

Whether manual or automated, effective algorithm configuration is central to the development of state-of-the-art algorithms. This is particularly true when dealing with $\mathcal{NP}$-hard problems, where the runtimes of weak and strong algorithms regularly differ by orders of magnitude on the same problem instances. Existing theoretical techniques are typically not powerful enough to determine whether one parameter configuration will outperform another, and therefore algorithm designers have to rely on empirical approaches.

2.1 The Algorithm Configuration Problem

An algorithm to be configured (a *target algorithm*) has a set of parameters, which can be *numerical* (*e.g.*, level of a real-valued threshold); *ordinal* (*e.g.*, low, medium, high); *categorical* (*e.g.*, choice of heuristic), Boolean (*e.g.*, algorithm component active/inactive); and even *conditional* (*e.g.*, a threshold that affects the algorithm's behaviour only when its corresponding component is active). In some cases, a value for one parameter can be incompatible with a value for another parameter; for example, some types of preprocessing are incompatible with the use of certain data structures. Thus, some parts of parameter configuration space are *forbidden*; they can be described succinctly in the form of forbidden partial instantiations of parameters (*i.e.*, constraints). In addition to the parameter space, the problem is also defined by a set of instances of interest (*e.g.*, 100 vehicle routing problems) and the metric to be optimized (*e.g.*, average runtime; optimality gap). We call an instance of the problem a *configuration scenario.*

We focus on addressing configuration scenarios using automatic methods that we call *configuration procedures*. We illustrate this process in Figure 1. Observe that we treat algorithm configuration as a black-box optimization problem: a configuration procedure

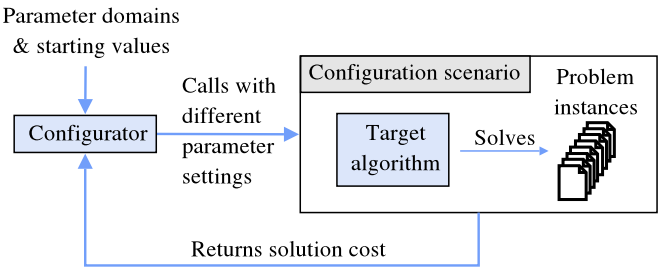


Fig. 1. A configuration procedure (short: configurator) executes the target algorithm with specified parameter settings on one or more problem instances, observes algorithm performance, and uses this information to decide which subsequent target algorithm runs to perform. A configuration scenario includes the target algorithm to be configured and a collection of instances.

executes the target algorithm on a problem instance and receives feedback about the algorithm's performance without any access to the algorithm's internal state. (Because the CPLEX tuning tool is proprietary, we do not know whether it operates similarly.)

2.2 Configuration Procedures and Existing Applications

A variety of black-box, automated configuration procedures have been proposed in the CP and AI literatures. There are two major families: *model-based* approaches that learn a response surface over the parameter space, and *model-free* approaches that do not. Much existing work is restricted to scenarios having only relatively small numbers of numerical (often continuous) parameters, both in the model-based [8, 15, 19] and model-free [12, 6, 1] literatures. Some relatively recent model-free approaches permit both larger numbers of parameters and categorical domains, in particular Composer [14], F-Race [10, 7], GGA [3], and our own ParamILS [22, 21]. As mentioned above, the automated tuning tool introduced in CPLEX version 11 can also be seen as a special-purpose algorithm configuration procedure; we believe it to be model free.

Blackbox configuration procedures have been applied to optimize a variety of parametric algorithms. Gratch and Chien [14] successfully applied the Composer system to optimize the LR-26 algorithm for scheduling communication between a collection of ground-based antennas and spacecraft in deep space. LR-26 is a heuristic scheduling approach with 5 parameters whose optimization significantly improved performance. Adenso-Diaz and Laguna [1] demonstrated that their Calibra system was able to optimize the parameters of six unrelated metaheuristics from the literature, matching or surpassing the performance achieved manually by their developers. F-Race and its extensions have been used to optimize numerous metaheuristics, including iterated local search for the quadratic assignment problem; ant colony optimization for the travelling salesperson problem; and the best performing algorithm submitted to the 2003 timetabling competition [9].

Our group has been successful in using various versions of PARAMILS to configure algorithms for a wide variety of problem domains. The focus of that work has so been far on the configuration of solvers for the propositional satisfiability problem (SAT); we optimized both tree search [18] and local search solvers [23], in both cases substantially advancing the state of the art for the types of instances studied. We also successfully configured algorithms for the most probable explanation problem in Bayesian networks; global continuous optimization; protein folding; and algorithm configuration itself [for

Algorithm	Parameter type	# params of type	# values considered	Total # configs
CPLEX 11.2 (12.1)	Categorical	44 (45)	3–7	$4.75 \cdot 10^{46}$ ($1.90 \cdot 10^{47}$)
	Boolean	6	2	
	Integer	18	5–7	
	Continuous	7	5–8	
GUROBI	Categorical	16	3–5	$3.84 \cdot 10^{14}$
	Boolean	4	2	
	Integer	3	5	
	Continuous	2	5	
LPSOLVE	Categorical	7	3–8	$1.22 \cdot 10^{15}$
	Boolean	40	2	

Table 1. Target algorithms and characteristics of their parameter configuration spaces.

details, see 17]. As described above, we also conducted preliminary experiments on the application of PARAMILS to optimizing CPLEX [21].

3 MIP Solvers

We now discuss the three MIP solvers we configured and their configuration spaces. Table 1 gives an overview.

IBM ILOG CPLEX is the most-widely used commercial optimization tool for solving MIPs. As stated on the CPLEX website (`http://www.ilog.com/products/cplex/`), currently over 1 300 corporations and government agencies use CPLEX, along with researchers at over 1 000 universities. Nevertheless, CPLEX is massively parameterized and end users often have to experiment with these parameters:

> "Integer programming problems are more sensitive to specific parameter settings, so you may need to experiment with them" (ILOG CPLEX 12.1 user manual, page 235)

Thus, the automated configuration of CPLEX is very promising and has the potential to directly impact a large user base.

For the experiments in this paper, we used two different versions of CPLEX. We primarily used version 11.2.[1] However, when our instances were small enough (as they were for two of our datasets), we instead used the teaching edition of CPLEX 12.1, which is restricted to problems having at most 500 variables and 500 constraints.

We defined the parameter configuration space of CPLEX as follows. Using the CPLEX "parameters reference manual", we identified 75 parameters that can be modified in order to optimize performance. 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 75 parameters we selected affect all aspects of CPLEX. There are 12 preprocessing parameters (mostly categorical); 17 MIP strategy parameters (mostly categorical); 10 categorical parameters deciding about how aggressively to use which types of cuts; 9 numerical MIP "limits" parameters; 10 simplex parameters (half of them categorical); 6 barrier optimization parameters (mostly categorical); and 11

[1] Unfortunately, our license renewal subscription expired before the release of CPLEX version 12, and so we do not yet have licenses for 12.1. We intend to rerun all experiments using 12.1 for the final version of this paper, if accepted.

further parameters. Most parameters have an "automatic" option as one of their values. We allowed this value but also allowed other values (all other values for categorical parameters; and a range of values for numerical parameters). Exploiting the fact that 4 parameters are conditional on others taking certain values, these 75 parameters gave rise to $4.75 \cdot 10^{46}$ distinct parameter configurations.

We used two further variants of this configuration space. For mixed integer quadratically-constrained problems (MIQCP), there were 1 new binary and 1 new categorical parameter with 3 values. However, 3 categorical parameters with 4, 6, and 7 values were lost, and for one categorical parameter with 4 values only 2 values remained. This led to a total of $8.49 \cdot 10^{44}$ possible configurations. For CPLEX 12.1, we used the same parameters as for CPLEX 11.2, further adding a 4-valued categorical parameter controlling multi-commodity flow (MCF) cuts. This yielded a configuration space of size $1.90 \cdot 10^{47}$.

GUROBI is a recent commercial MIP solver;[2] it is competitive with CPLEX on some types of MIP instances [25]. We used v.2.0.1 with an unlimited free academic license.

GUROBI's parameter configuration space is smaller than CPLEX's. Using the online description of GUROBI's parameters,[3] we identified 25 parameters for configuration. These consisted of 12 mostly-categorical parameters that determine how aggressively to use each type of cuts, 6 mostly-categorical simplex parameters, 3 MIP parameters, and 4 other mostly-Boolean parameters. After disallowing some problematic parts of configuration space (see Section 4.3), we considered $3.84 \cdot 10^{14}$ distinct configurations.

LPSOLVE is one of the most prominent open-source MIP solvers. We determined 47 parameters based on the information at `http://lpsolve.sourceforge.net/`. These parameters are rather different from those of GUROBI and CPLEX: 7 parameters are categorical, and the rest are Boolean switches indicating whether various solver modules should be employed. 14 parameters concern presolving; 9 concern pivoting; 14 concern the Branch & Bound strategy; and 10 concern other functions. Taking into account one conditional parameter and disallowing problematic parts of configuration space (see Section 4.3), we considered $1.22 \cdot 10^{15}$ distinct parameter configurations.

4 Experimental Setup

We now describe our experimental setup: benchmark sets, details of the configuration procedure we used, and our computational environment.

4.1 Benchmark Sets

We collected a wide range of MIP benchmarks from public benchmark libraries and other researchers, and split each of them 50:50 into disjoint training and test sets; we detail them in the following.

MJA This set comprises 343 machine-job assignment instances encoded as mixed integer quadratically constrained programming (MIQCP) problems [2]. We obtained it from the Berkeley Computational Optimization Lab (BCOL).[4] On average, these instances contain 2 769 variables and 2 255 constraints (with standard deviations 2 133 and 1 592, respectively).

[2] `http://www.gurobi.com/`

[3] `http://www.gurobi.com/html/doc/refman/node378.html#sec:Parameters`

[4] `http://www.ieor.berkeley.edu/~atamturk/bcol/`, where this set is called `conic.sch`.

CLS This set comprises 100 capacitated lot-sizing instances encoded as mixed integer linear programming (MILP) problems [5]; again, we obtained it from BCOL. Each instance contains 181 variables and 180 constraints.

MIK This set of 120 MILP-encoded mixed-integer knapsack instances [4] was also obtained from BCOL. On average, these instances contain 384 variables and 151 constraints (with standard deviations 309 and 127, respectively).

Regions200 This set comprises 2 000 instances of the combinatorial auction winner determination problem, encoded as MILP instances. They were generated using the `regions` generator from the Combinatorial Auction Test Suite [24], with the *goods* parameter set to 200 and the *bids* parameter set to 1 000. On average, the resulting MILP instances contain 1 002 variables and 385 inequalities (with standard deviations 1.7 and 3.4, respectively).

Regions70 This set contains 2 000 instances similar to those in Regions200 but much smaller; we created this set specifically to be used with the size-restricted teaching edition of CPLEX 12.1. On average, the resulting MILP instances contain 352 variables and 135 inequalities (with standard deviations 1.7 and 2.1, respectively).

COR-LAT This set contains 2 000 real-life MILP instances used for the construction of a wildlife corridor for grizzly bears in the Northern Rockies region [13, the instances were provided by Bistra Dilkina]. All instances had 466 variables; on average they had 486 constraints (with standard deviation 25.2).

MASS This set contains 100 integer programming instances modelling multi-activity shift scheduling [11]. On average, the resulting MILP instances contain 81 994 variables and 24 637 inequalities (with standard deviations 9 725 and 5 391, respectively)

4.2 Configuration Procedure

As our configuration procedure we used FOCUSEDILS version 2.4 [21]. This instantiation of the PARAMILS framework aggressively rejects poor configurations and focuses its efforts on the evaluation of good configurations by performing additional runs using them. It also supports *adaptive capping*, a technique for speeding up configuration tasks in which runtime is minimized. Specifically, when running PARAMILS, the user specifies a so-called *captime* κ, the maximal amount of time after which PARAMILS will terminate each run of the target algorithm. Adaptive capping dynamically sets the captime for individual target algorithm runs, thus permitting substantial savings in computation time.

As in our previous work, for each configuration task we perform 10 runs of FOCUSEDILS and use the result of the run with best *training* performance. This is sound since no knowledge of the test set is required in order to make the selection; the only drawback is a 10-fold slowdown of the overall procedure. If none of the 10 FOCUSEDILS runs encounters a successful algorithm run, then our procedure returns the algorithm default.

4.3 Avoiding Problematic Parts of Parameter Configuration Space

Occasionally, we encountered problems running GUROBI and LPSOLVE with certain combinations of parameters on particular problem instances. These problems included segmentation faults as well as several more subtle failure modes in which incorrect results could be returned by a solver. To deal with them, we took the following measures in our experimental protocol. (CPLEX did not show these problems on any of the instances studied here.)

First, we established reference solutions for all MIP instances using CPLEX 11.2 and GUROBI, both run with their default parameter configurations for up to one CPU hour per instance. (In some cases, neither of the two solvers could find a solution within this time, in which case we skipped the following verification stage.) We then conducted preliminary experiments (not reported here) to identify partial parameter configurations for which the solvers produced incorrect results as compared to these reference solutions, or a segmentation fault. Any such partial configuration was then removed from the configuration space considered in our experiments using PARAMILS's mechanism of forbidden partial parameter instantiations.

Furthermore, in our later experiments, whenever a target algorithm run started by PARAMILS disagreed with the respective reference solution, or when the MIP solver produced a segmentation fault, we considered the empirical cost of that run to be ∞, thereby driving the local search process underlying PARAMILS away from the problematic parameter configuration. This allowed PARAMILS to gracefully handle target algorithm failures that we had not observed in our preliminary experiments. Whenever a target algorithm run reported to have failed due to an internal error (such as 'out of memory'), we counted it as having timed out with its specified captime.

4.4 Computational Environment

We carried out the configuration of LPSOLVE on the 840-node Westgrid Glacier cluster, each with two 3.06 GHz Intel Xeon 32-bit processors and 2–4GB RAM, running OpenSuSE Linux 10.1. All other configuration experiments, as well as all evaluation was performed on a cluster of 55 dual 3.2GHz Intel Xeon PCs with 2MB cache and 2GB RAM, running OpenSuSE Linux 10.1; runtimes were measured as CPU time on these reference machines. We measured the computational requirements of CPLEX 11.2 with an external CPU timer since it sometimes returned erratic (often negative) runtimes.

5 Minimization of Runtime Required to Prove Optimality

In our first set of experiments, we studied the extent to which automated configuration can improve the time performance of CPLEX 11.2, GUROBI, and LPSOLVE for solving the seven types of instances discussed in Section 4.1. This led to $3 \cdot 6 + 1 = 19$ configuration scenarios (the MJA instances are quadratically constrained and could only be solved with CPLEX).

For most configuration scenarios, we allowed a total configuration time budget of 2 CPU days for each of our 10 PARAMILS runs, with a captime of $\kappa = 300$ seconds for each MIP solver run. In order to save computation time, we only used 5 CPU hours and a per-run captime of $\kappa = 30$ seconds for configuring CPLEX and GUROBI on the easiest benchmark sets (CATS100, MIK, and MJA). In order to penalize timeouts, during configuration we used the penalized average runtime criterion [dubbed "PAR-10" in 21], counting each timeout as $10 \cdot \kappa$. For evaluation, we report timeouts separately.

For each configuration scenario, we compared the performance of the parameter configuration identified using PARAMILS against the default configuration, using a test set of instances disjoint from the training set used during configuration. We note that this default configuration is typically determined using substantial time and effort; for example, the CPLEX 12.1 user manual states (on p. 478):

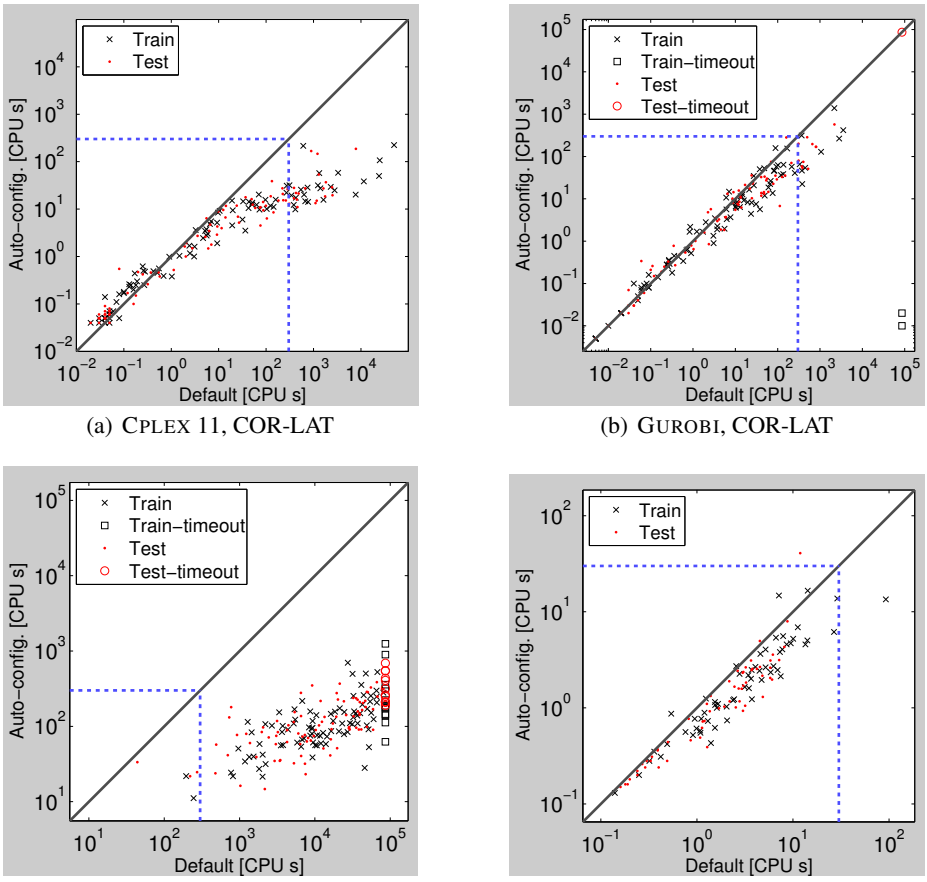
Algorithm	Scenarios	% test inst. solved in 24h		mean runtime for solved [s]		Speedup
		default	PARAMILS	default	PARAMILS	factor
CPLEX	MJA	100%	100%	2.61	**1.52**	1.71×
	MIK	100%	100%	3.97	**1.36**	2.92×
	CATS100	100%	100%	1.54	**0.35**	4.43×
	CATS200	100%	100%	57.5	**10.0**	5.75×
	CLS	100%	100%	70.5	**8.26**	8.54×
	MASS	100%	100%	477	**247**	1.94×
	COR-LAT	100%	100%	300	**12.9**	23.3×
GUROBI	MIK	100%	100%	2.70	**2.17**	1.24×
	CATS100	100%	100%	2.08	**1.04**	2.01×
	CATS200	100%	100%	56.6	**36.4**	1.56×
	CLS	100%	100%	58.9	**47.2**	1.25×
	MASS	100%	100%	493	**281**	1.75×
	COR-LAT	99%	99%	92.1	**30.4**	3.03×
LPSOLVE	MIK	37%	37%	21 670	21 670	1×
	CATS100	100%	100%	9.52	**1.71**	5.56×
	CATS200	88%	**100%**	19 000	**124**	153×
	CLS	14%	**58%**	39 300	**1 440**	27.4×
	MASS	17%	17%	8 661	8 661	1×
	COR-LAT	50%	92%	7 916	**229**	34.6×

Table 2. Results for minimizing the runtime to find an optimal solution and prove its optimality. All results are for test sets disjoint from the training sets used for the automated configuration. We give the number of timeouts after 24 CPU hours as well as the mean runtime for those instances that were solved; bold-faced entries indicate better performance of the configurations found by PARAMILS than for the default configuration. Each configuration experiment used 10 PARAMILS runs of 2 CPU days each.

> "A great deal of algorithmic development effort has been devoted to establishing default ILOG CPLEX parameter settings that achieve good performance on a wide variety of MIP models."

Table 2 describes our configuration results. For each of the benchmark sets, PARAMILS improved CPLEX's performance. Specifically, we achieved speedups ranging from 1.7-fold to 23-fold. For GUROBI, the speedups were again consistent but less pronounced: 1.24-fold to 3-fold. For the open-source solver LPSOLVE, the speedups were most substantial, but there were also 2 cases in which PARAMILS did not improve over LPSOLVE's default, namely the MIK and MASS benchmarks. This occurred because, within the 300-second captime we used during configuration, none of the thousands of LPSOLVE runs performed by PARAMILS solved a single benchmark instance for either of the two benchmark sets. It is interesting to observe that LPSOLVE exhibited such poor performance for the MIK benchmark, which was one of the easiest benchmark sets for both of the commercial solvers.

Figure 2 shows the speedups for 4 configuration scenarios. Figures 2 (a) to (c) show the scenario with the largest speedup for each of the solvers. In all cases, PARAMILS' configurations scaled better to hard instances than the algorithm defaults, which in some cases timed out on the hardest instances. PARAMILS' *worst* performance was for the 2 LPSOLVE scenarios, for which it simply returned the default configuration; in Figure 2(d), we instead plot results for the more interesting second-worst case, the configuration



(a) CPLEX 11, COR-LAT (b) GUROBI, COR-LAT

(c) LPSOLVE, CATS 200 (d) GUROBI, MIK

Fig. 2. Results for configuration of MIP solvers to reduce the time for finding an optimal solution and proving its optimality.

of GUROBI on MIK. Observe that here, performance was actually rather good for most instances, and that the poor speedup in test performance was due to a single hard test instance. Better generalization performance would be achieved if more training instances were available.

6 Minimization of Optimality Gap

Sometimes, we may be interested in optimizing a performance measure other than runtime. For example, constraints on the time available for solving a given MIP instance might preclude running the solver to completion, and in such cases, we may be interested in minimizing the optimality gap (also known as MIP gap) achieved within a fixed amount of time, T.

To investigate the efficacy of our automated configuration approach in this context, we applied it to CPLEX 11.2, GUROBI and LPSOLVE on the 5 benchmark distributions with the longest average runtimes, with the objective of minimizing the average relative

Scenario	Algorithm	% feasible solutions found		mean gap when feasible	
		default	PARAMILS	default	PARAMILS
CPLEX	MIK	100%	100%	0.05%	**0.01**%
	CLS	100%	100%	0.29%	**0.19**%
	Regions200	100%	100%	3.19%	**0.87**%
	COR-LAT	72%	**90**%	2.92%	4.46%
	MASS	10%	**26**%	0.33%	4.71%
GUROBI	MIK	100%	100%	0.02%	**0.01**%
	CLS	100%	100%	0.53%	**0.44**%
	Regions200	100%	100%	3.11%	**2.37**%
	COR-LAT	83%	**91**%	2.72%	2.05%
	MASS	32%	32%	76.4%	**52.2**%
LPSOLVE	MIK	100%	100%	652%	**14.3**%
	CLS	100%	100%	29.6%	**7.39**%
	Regions200	100%	100%	11.0%	**6.30**%
	COR-LAT	36%	**87**%	2.49%	1.86%
	MASS	0%	0%	-	-

Table 3. Results for configuration of MIP solvers to reduce the relative optimality gap reached within 10 CPU seconds. For each combination of solver and benchmark, we performed 10 independent PARAMILS runs of 5 hours each on the respective training benchmark set, selected the run with the best training performance, and reported results on the respective test sets. We give the number of test instances for which no feasible solution was found within 10 seconds, the mean relative gap for the remaining test instances, and the reduction factor (gap using default/gap using optimized configuration) on the test set. Bold face indicates the better configuration (recall that our lexicographic objective function cares first about the number of instances with feasible solutions, and then considers the mean gap among feasible instances only to break ties).

optimality gap achieved within $T = 10$ CPU seconds. To deal with runs that did not find feasible solutions, we optimized a lexicographic objective function, which counts the fraction of instances for which feasible solutions were found and breaks ties based on the mean relative gap for those instances. For each of the 15 configuration scenarios, we performed 10 PARAMILS runs, each with a time budget of 5 CPU hours.

Table 3 shows the results of this experiment. For all but one of the 15 configuration scenarios, the automatically-found parameter configurations performed substantially better than the algorithm defaults. In 4 cases, feasible solutions were found for more instances, and in 10 of the 11 remaining scenarios, the relative gaps were smaller, often (namely, in 5 out of the 10 cases) by a factor of at least 2.

For the one configuration scenario where we did not achieve an improvement, LPSOLVE on MASS, the default configuration of LPSOLVE could not find a feasible solution for *any* of the training instances, and the same turned out to be the case for the thousands of configurations considered by PARAMILS. This provides further evidence that the MASS instances may simply be too hard for LPSOLVE.

More detailed examination of these results (data not shown) reveals that for some scenarios (*e.g.*, for LPSOLVE on MIK), the relative gap shrank for all test (and training) instances. There were also cases (*e.g.*, for CPLEX on Regions200 and for GUROBI on CLS), where substantially more instances were solved to optimality (using a standard tolerance of 0.01%).

7 Comparison to CPLEX Tuning Tool

The CPLEX tuning tool is a built-in CPLEX function available in versions 11 and above.[5] It allows the user to minimize CPLEX's runtime on a given set of instances. As in our approach, the user also specifies a per-run captime, the default for which is $\kappa = 10\,000$ seconds, and an overall time budget. The user can further decide whether to minimize mean or maximal runtime across the set of instances. (We note that these two criteria are more similar than they might appear since the mean is usually dominated by the runtimes of the hardest instances.) By default, the objective for tuning is minimal mean runtime and the time budget is set to infinity, allowing the CPLEX tuning tool to perform all the runs it deems necessary.

Since CPLEX is proprietary, we do not know the inner workings of the tuning tool; however, we can make some inferences from its outputs. In our experiments, it always started by running the default parameter configuration on each instance in the benchmark set. Then, it tested a set of named parameter configurations, such as 'no_cuts', 'easy', and 'more_gomory_cuts'. The set of configurations tested depended on the benchmark set given.

PARAMILS differs from the CPLEX tuning tool in at least three crucial ways. First, it searches in the vast space of all possible configurations, while the CPLEX tuning tool focuses on a small set of handpicked candidates. Second, PARAMILS is a randomized procedure that can be run for any amount of time, and that can find different solutions when multiple copies are run in parallel; it reports better configurations as it finds them. The CPLEX tuning tool is deterministic, and runs for a fixed amount of time (dependent on the instance set given) unless the time budget intervenes earlier; it does not return a configuration until it terminates. Third, because PARAMILS does not rely on domain-specific knowledge, it can be applied out of the box to the configuration of other MIP solvers and, indeed, arbitrary parameterized algorithms. In contrast, the few configurations in the CPLEX tuning tool appear to have been selected based on substantial domain insights, and the fact that different parameter configurations are tried for different types of instances leads us to believe that it relies upon MIP-specific instance characteristics. While in principle, this could be an advantage, in its current form it appears to be rather restrictive.

We compared the performance of the configurations found by the CPLEX tuning tool to that of configurations found with PARAMILS. We used the tuning tool's default settings to optimize mean runtime on our training sets, and tested performance on our test sets. We ran the tool from CPLEX 11.2 on all benchmark sets, and from CPLEX 12.1 (teaching edition) for the two sets (CLS and Regions70) in which instances were small enough to permit its use. We ran PARAMILS for the same amount of time the CPLEX tuning tool took in each case. This time is often rather small and—in contrast to PARAMILS—the tool cannot use additional time in order to improve performance. We have already demonstrated substantial speedups with comparably large time budgets for configuration in Section 5; here, we were interested in the quality of configurations found given a shorter time budget. In particular, when the tuning tool required time t for

[5] Incidentally, our first work on the configuration of CPLEX predates the CPLEX tuning tool. This work, involving Hutter, Hoos, Leyton-Brown, and Stützle, was presented and published as a technical report at a doctoral symposium in Sept. 2007 [16]. At that time no other mechanism for automatically configuring CPLEX was available; CPLEX 11 was released Nov. 2007.

a benchmark set, we ran two versions of our configuration procedures: one version using 10 parallel runs of PARAMILS, each with time budget $t/10$; and one version using 10 runs of PARAMILS, each with time budget t. The first version used the same total CPU time as the CPLEX tuning tool; the second version takes the same amount of wall clock time but runs in parallel on 10 processors.

Table 3 reports the results of the comparison. First, we note that the CPLEX tuning tool simply returned the algorithm defaults in 5 of the 9 cases; in 1 case it crashed, running out of memory. In the remaining 3 cases, it returned configurations that only differed in up to three parameters from the default: 'easy' (perform only 1 cutting plane pass; apply the periodic heuristic every 50 nodes; and branch based on pseudo-reduced costs), 'no_cuts' (perform no cuts and branch based on pseudo-reduced costs), and 'aggressive_heuristic' (apply the periodic heuristic every 3 nodes and apply the RINS heuristic every 20 nodes). 2 of these 3 cases were on Regions70, where the tuning tools of both CPLEX 11.2 and CPLEX 12.1 yielded minor improvements over the default. Only on benchmark set MASS did the tuning tool actually lead to a noticeable improvement over the defaults.

PARAMILS outperformed the tuning tool for almost all configuration scenarios, sometimes substantially so. Given the same total time budget as the CPLEX tuning tool required, PARAMILS(t/10) performed better for 6 of the 9 scenarios, with a mean runtime up to 6 times smaller. For two scenarios, it performed slightly worse and for one (MIK) it performed worse by a factor of $1/0.37 = 2.7$. For that scenario, PARAMILS simply did not have enough time to find good configurations; it also only had time to perform runs with 18 instances. Given the same wall-clock time budget as the tuning tool but 10 processors, PARAMILS(t) performed better on 8 of the 9 scenarios, with a mean runtime up to 10 times smaller. The only case where the CPLEX tuning tool performed better (given the restricted time budget used here) was for benchmark set MASS, and there the difference was small.

We examine the performance difference between the two methods on a per-instance basis in Figure 3. Specifically, we consider the two CPLEX 12.1 scenarios and the two CPLEX 11.2 scenarios with the largest and the smallest speedups respectively. Figures 3 (a) through (c) show the consistent speedups achieved by PARAMILS, in (b) and (c) with a trend towards greater improvements for harder instances. Figure 3(d) shows the only case where the tuning tool yielded a configuration with better test performance than the one selected by PARAMILS. Note, however, that PARAMILS achieved better performance on most instances, and that its higher mean runtime was due to a single outlier. Furthermore, on the *training* instances—the only instances to which both the CPLEX tuning tool and PARAMILS had access—PARAMILS achieved much lower mean runtime, by a factor of 5.95. This difference was again mostly due to a single outlying instance (the '$\times$ symbol on the far right in Figure 3(d)). We note that the presence of such outliers suggests that reliable generalization on this benchmark set would occur only given much more training data. Nevertheless, we are encouraged that PARAMILS found a way to solve the training-set outlier (in 1 748 seconds) while the tuning tool accepted a time-out after 10 000 seconds.

8 Conclusions and Future Work

In this study we have demonstrated that by using automated algorithm configuration, substantial performance improvements can be obtained for three widely used MIP solvers on

Scenario	CPLEX tuning tool		CPLEX mean runtime on test set [s]			
	Tuning Time t	Name of result	Default	Tuned	PARAMILS(t/10)	PARAMILS(t)
Regions70	796	'easy'	0.181	0.175	0.11 (**1.60**×)	0.08 (**2.13**×)
CLS	104 673	'defaults'	48.4	48.4	14.4 (**3.37**×)	12.6 (**3.84**×)
Regions70	1 511	'no_cuts'	0.18	0.17	0.19 (0.90×)	0.12 (**1.43**×)
CLS	≥ 47 622	Error: out of memory	70.5	70.5	11.6 (**6.09**×)	8.70 (**8.10**×)
Regions200	65 116	'defaults'	57.5	57.5	13.7 (**4.19**×)	10.4 (**5.55**×)
MIK	13 431	'defaults'	3.97	3.97	10.8 (0.37×)	2.91 (**1.36**×)
MJA	1 987	'defaults'	2.61	2.61	2.15 (**1.21**×)	2.22 (**1.17**×)
MASS	97 107	'aggressive_heuristic'	477	428	431 (0.99×)	464 (0.92×)
COR-LAT	96 674	'defaults'	300	300	91.9 (**3.27**×)	29.1 (**10.3**×)

Table 4. Comparison of our approach against the CPLEX tuning tool, for CPLEX version 12.1 (upper part of the table) and version 11.2 (lower part of the table). For each benchmark set, we give the time t required by the CPLEX tuning tool and the CPLEX name of the configuration it judged best. We give the mean runtime of the default configuration; the configuration the tuning tool selected; the configuration selected using 10 PARAMILS runs each allowed time $t/10$; and the configuration selected using 10 PARAMILS runs each allowed time t. For the latter two, in parentheses we give the speedup factor over the tuning tool's performance. For CLS, the CPLEX11.2 tuning tool ran out of memory after 47 622 s, and we chose $t = 104\,673$ as the time required by CPLEX 12.1's tuning tool on the same set. Boldface indicates improved performance.

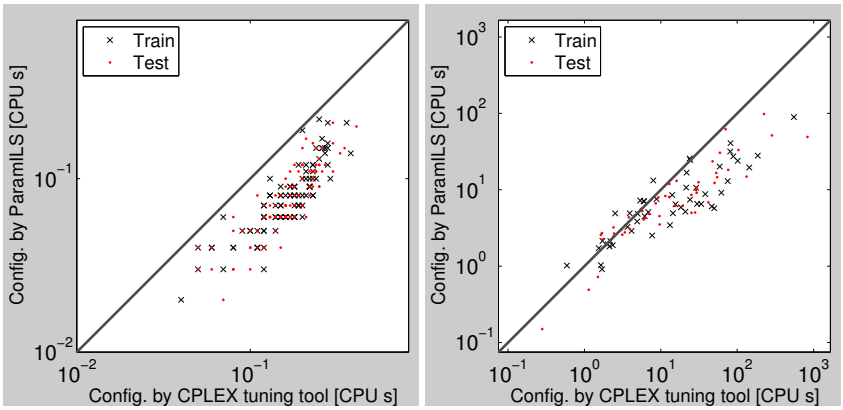
a broad range of benchmark sets, in terms of minimizing run-time for proving optimality, and of minimizing the optimality gap. This is particularly noteworthy considering the effort that has gone into optimizing the default configurations for commercial MIP solvers such as CPLEX and GUROBI.

We have observed that the success of our fully automated approach depends on two factors that also play an important role when configuring these (and other) solvers manually: the availability of sufficiently large benchmark sets and the use of suitably chosen limits on the runtime of the solver during configuration (captimes). Not surprisingly, when using relatively small benchmark sets, performance improvements on training instances sometimes do not fully translate to test instances (see, *e.g.*, LPSOLVE or CPLEX on MASS, or GUROBI on MIK); we note that this effect can be avoided by using bigger benchmark sets (in our experience, about 1000 instances are typically sufficient). The choice of captimes is largely a tradeoff between the risk of poor performance for difficult instances and the risk of wasting time during the configuration process. In the future, we plan to investigate strategies for automating the choice of captimes during configuration.

In future work, we also plan to study *why* certain parameter configurations work better than others. The algorithm configuration approach we have used here, PARAMILS, can identify very good (possibly optimal) configurations, but it does not yield information on the importance of each parameter, interactions between parameters, or the interaction between parameters and characteristics of the problem instances at hand. Partly to address those issues, we are actively developing an alternative algorithm configuration approach that is based on response surface models [19, 20, 17].

Acknowledgements

We thank Louis-Martin Rousseau and Bistra Dilkina for making available their sets of MIP instances. Furthermore, we thank Gurobi Optimization for making a full version of their software freely available for academic purposes and for providing assistance for running it on our cluster;



(a) CPLEX 12.1; Regions70 (b) CPLEX 12.1; CLS

(c) CPLEX 11.2; COR-LAT (d) CPLEX 11.2; MASS

Fig. 3. Comparison of configurations returned by the CPLEX tuning tool and by our approach. For our approach, we used 10 parallel PARAMILS runs each allowed the time budget t required by the CPLEX tuning tool.

IBM for making a teaching edition of CPLEX available to us; and Westgrid for support in using their compute cluster. FH acknowledges support from a postdoctoral research fellowship by the Canadian Bureau for International Education. HH and KLB gratefully acknowledge support from NSERC to their respective discovery grants, and from the MITACS NCE for seed project funding.

References

[1] Adenso-Diaz, B. and Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, 54(1):99–114.

[2] Aktürk, S. M., Atamtürk, A., and Gürel, S. (2007). A strong conic quadratic reformulation for machine-job assignment with controllable processing times. Research Report BCOL.07.01, University of California-Berkeley.

[3] Ansotegui, C., Sellmann, M., and Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of solvers. In *Proc. of CP-09*, pages 142–157.

[4] Atamtürk, A. (2003). On the facets of the mixed–integer knapsack polyhedron. *Mathematical Programming*, 98:145–175.

[5] Atamtürk, A. and Muñoz, J. C. (2004). A study of the lot-sizing polytope. *Mathematical Programming*, 99:443–465.
[6] Audet, C. and Orban, D. (2006). Finding optimal algorithmic parameters using the mesh adaptive direct search algorithm. *SIAM Journal on Optimization*, 17(3):642–664.
[7] Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In *Proc. of MH-07*, pages 108–122.
[8] Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation: The New Experimentalism*. Natural Computing Series. Springer Verlag, Berlin.
[9] Birattari, M. (2004). *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.
[10] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proc. of GECCO-02*, pages 11–18.
[11] Cote, M., Gendron, B., and Rousseau, L. (2010). Grammar-based integer programing models for multi-activity shift scheduling. Technical Report CIRRELT-2010-01, Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport.
[12] Coy, S. P., Golden, B. L., Runger, G. C., and Wasil, E. A. (2001). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1):77–97.
[13] Gomes, C. P., van Hoeve, W.-J., and Sabharwal, A. (2008). Connections in networks: A hybrid approach. In *5th CPAIOR*, volume 5015 of *LNCS*, pages 303–307.
[14] Gratch, J. and Chien, S. A. (1996). Adaptive problem-solving for large-scale scheduling problems: A case study. *JAIR*, 4:365–396.
[15] Huang, D., Allen, T. T., Notz, W. I., and Zeng, N. (2006). Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466.
[16] Hutter, F. (2007). On the potential of automatic algorithm configuration. In *SLS-DS2007: Doctoral Symposium on Engineering Stochastic Local Search Algorithms*, pages 36–40. Technical report TR/IRIDIA/2007-014, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium.
[17] Hutter, F. (2009). *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University Of British Columbia, Department of Computer Science, Vancouver, Canada.
[18] Hutter, F., Babić, D., Hoos, H. H., and Hu, A. J. (2007a). Boosting Verification by Automatic Tuning of Decision Procedures. In *Proc. of FMCAD'07*, pages 27–34, Washington, DC, USA. IEEE Computer Society.
[19] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Murphy, K. P. (2009a). An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proc. of GECCO-09*, pages 271–278.
[20] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Murphy, K. P. (2010). Time-bounded sequential parameter optimization. In *Proc. of LION-4*, LNCS. Springer Verlag. To appear.
[21] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009b). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.
[22] Hutter, F., Hoos, H. H., and Stützle, T. (2007b). Automatic algorithm configuration based on local search. In *Proc. of AAAI-07*, pages 1152–1157.
[23] KhudaBukhsh, A., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2009). SATenstein: Automatically building local search SAT solvers from components. In *Proc. of IJCAI-09*, pages 517–524.
[24] Leyton-Brown, K., Pearson, M., and Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In *Proc. of EC'00*, pages 66–76, New York, NY, USA. ACM.
[25] Mittelmann, H. (2010). Mixed integer linear programming benchmark (serial codes). `http://plato.asu.edu/ftp/milpf.html`. Version last visited on January 26, 2010.