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 outperforms the CPLEX special-purpose automated tuning tool.

1 Introduction

Current state-of-the-art mixed integer programming (MIP) solvers are highly parameterized. Their 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 in the first place. 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 to only 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 several methods for the automated configuration of various complex algorithms [20, 19, 18, 15].

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 automated configuration of CPLEX 10.1.1, considering 5 types of MIP instances [19].

The main contribution of this paper is a thorough study of the applicability of one of our black-box techniques to the MIP domain. We go beyond previous work by configuring three different MIP solvers (GUROBI, LPSOLVE, and the most recent CPLEX version 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 tuning tool. We 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 52 while the tuning tool returned the CPLEX defaults. For GUROBI, speedups were consistent but small (up to a factor of 2.3), and for LPSOLVE we obtained speedups up to a factor of 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 chose to study (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 on the same problem instances regularly differ by orders of magnitude. 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

The algorithm configuration problem we consider in this work involves an algorithm to be configured (a *target algorithm*) with a set of parameters that affect its performance, a set of problem instances of interest (*e.g.*, 100 vehicle routing problems), and a performance metric to be optimized (*e.g.*, average runtime; optimality gap). The target algorithm's parameters 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 a particular heuristic is chosen). 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).

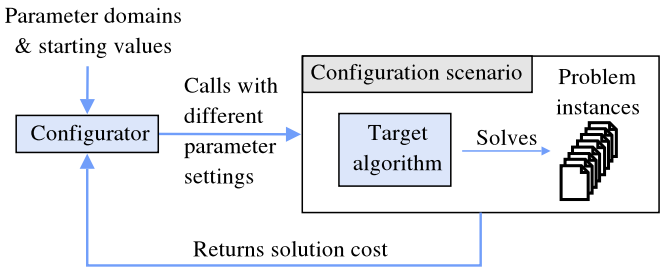


Fig. 1. A configuration procedure (short: configurator) executes the target algorithm with specified parameter settings on one or more problem instances, observes algorithm performance, and uses this information to decide which subsequent target algorithm runs to perform. A configuration scenario includes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

We refer to instances of this algorithm configuration problem as *configuration scenarios*, and we address these using automatic methods that we call *configuration procedures*; this is illustrated in Figure 1. Observe that we treat algorithm configuration as a black-box optimization problem: a configuration procedure 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 [7, 13, 17] and model-free [6, 1] literatures. Some relatively recent model-free approaches permit both larger numbers of parameters and categorical domains, in particular Composer [12], F-Race [9, 8], GGA [3], and our own ParamILS [20, 19]. 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 [12] successfully applied the Composer system to optimize the five parameters of LR-26, an algorithm for scheduling communication between a collection of ground-based antennas and spacecraft in deep space. Adenso-Diaz and Laguna [1] demonstrated that their Calibra system was able to optimize the parameters of six unrelated metaheuristic algorithms, matching or surpassing the performance achieved manually by their developers. F-Race and its extensions have been used to optimize numerous algorithms, 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 [8].

Our group successfully used various versions of PARAMILS to configure algorithms for a wide variety of problem domains. So far, the focus of that work has been on the configuration of solvers for the propositional satisfiability problem (SAT); we optimized both tree search [16] and local search solvers [21], 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 details, see Ref. 15).

2.3 Configuration Procedure Used: FOCUSEDILS

The configuration procedure used in this work is an instantiation of the PARAMILS framework [20, 19]. However, we do not mean to argue for the use of PARAMILS in particular, but rather aim to provide a lower bound on the performance improvements that can be achieved by applying general-purpose automated configuration tools to MIP solvers; future tools may achieve even better performance.

PARAMILS performs an iterated local search (ILS) in parameter configuration space; configurations are evaluated by running the target algorithm with them. The search is initialized at the best out of ten random parameter configurations and the target algorithm's default configuration. Next, PARAMILS performs a first-improvement local search that ends in a local optimum. It then iterates three phases: (1) a random perturbation to escape the local optimum; (2) another local search phase resulting in a new local optimum; and (3) an acceptance criterion that typically accepts the new local optimum if it is better than the previous one. The PARAMILS instantiation we used here is FOCUSEDILS version 2.4, which aggressively rejects poor configurations and focuses its efforts on the evaluation of good configurations. Specifically, it starts with performing only a single target algorithm run for each configuration considered, and performs additional runs for good configurations as the search progresses. This process guarantees that—given enough time and a training set that is perfectly representative of unseen test instances—FOCUSEDILS will identify the best configuration in the given design space [20, 19]. (Further details of PARAMILS and FOCUSEDILS can be found in our previous publications [20, 19].)

In practice, we are typically forced to work with finite sets of benchmark instances, and performance on a small training set is often not very representative for performance on other, unseen instances of similar origin. PARAMILS (and any other configuration tool) can only optimize performance on the training set it is given; it cannot guarantee that this leads to improved performance on a separate set of test instances. In particular, with very small training sets, a so-called *over-tuning* effect can occur: given more time, automated configuration tools find configurations with better training but worse test performance [8, 20].

Since target algorithm runs with some parameter configurations may take a very long (potentially infinite) time, PARAMILS requires the user to specify a so-called *captime* κ_{max}, the maximal amount of time after which PARAMILS will terminate a run of the target algorithm as unsuccessful. FOCUSEDILS version 2.4 also supports *adaptive capping*, a speedup technique that sets the captimes $\kappa \leq \kappa_{max}$ for individual target algorithm runs, thus permitting substantial savings in computation time.

FOCUSEDILS is a randomized algorithm that tends to be quite sensitive to the ordering of its training benchmark instances. For challenging configuration tasks some of its runs often perform much better than others. For this reason, in previous work we adopted the strategy to perform 10 independent parallel runs of FOCUSEDILS and use the result of the run with best *training* performance [16, 19]. This is sound since no knowledge of the test set is required in order to make the selection; the only drawback

Algorithm	Parameter type	# parameters of this type	# values considered	Total # configurations
CPLEX MILP (MIQCP)	Boolean	6 (7)	2	$1.90 \cdot 10^{47}$ ($3.40 \cdot 10^{45}$)
	Categorical	45 (43)	3–7	
	Integer	18	5–7	
	Continuous	7	5–8	
GUROBI	Boolean	4	2	$3.84 \cdot 10^{14}$
	Categorical	16	3–5	
	Integer	3	5	
	Continuous	2	5	
LPSOLVE	Boolean	40	2	$1.22 \cdot 10^{15}$
	Categorical	7	3–8	

Table 1. Target algorithms and characteristics of their parameter configuration spaces. For details, see `http://www.cs.ubc.ca/labs/beta/Projects/MIP-Config/`.

is a 10-fold increase in overall computation time. If none of the 10 FOCUSEDILS runs encounters any successful algorithm run, then our procedure returns the algorithm default.

3 MIP Solvers

We now discuss the three MIP solvers we chose to study and their respective parameter 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. 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.

We used CPLEX 12.1 (the most recent version) and defined its parameter configuration space as follows. Using the CPLEX 12 "parameters reference manual", we identified 76 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 76 parameters we selected affect all aspects of CPLEX. They include 12 preprocessing parameters (mostly categorical); 17 MIP strategy parameters (mostly categorical); 11 categorical parameters deciding 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 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, these 76 parameters gave rise to $1.90 \cdot 10^{47}$ distinct parameter configurations. For mixed integer quadratically-constrained problems (MIQCP), there were some additional parameters (1 binary and 1 categorical parameter with 3 values). However, 3 categorical parameters with 4, 6, and 7 values were no longer applicable, and for one categorical

parameter with 4 values only 2 values remained. This led to a total of $3.40 \cdot 10^{45}$ possible configurations.

GUROBI is a recent commercial MIP solver that is competitive with CPLEX on some types of MIP instances [23]. We used version 2.0.1 and defined its configuration space as follows. Using the online description of GUROBI's parameters,[1] we identified 26 parameters for configuration. These consisted of 12 mostly-categorical parameters that determine how aggressively to use each type of cuts, 7 mostly-categorical simplex parameters, 3 MIP parameters, and 4 other mostly-Boolean parameters. After disallowing some problematic parts of configuration space (see Section 4.2), we considered 25 of these 26 parameters, which led to a configuration space of size $3.84 \cdot 10^{14}$.

LPSOLVE is one of the most prominent open-source MIP solvers. We determined 52 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. 17 parameters concern presolving; 9 concern pivoting; 14 concern the branch & bound strategy; and 12 concern other functions. After disallowing problematic parts of configuration space (see Section 4.2), we considered 47 of these 52 parameters. Taking into account one conditional parameter, these gave rise to $1.22 \cdot 10^{15}$ distinct parameter configurations.

4 Experimental Setup

We now describe our experimental setup: benchmark sets, how we identified problematic parts in the configuration spaces of GUROBI and LPSOLVE, 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 these 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).[2] On average, these instances contain 2 769 variables and 2 255 constraints (with standard deviations 2 133 and 1 592, respectively).

MIK This set comprises 120 mixed-integer knapsack instances encoded as mixed integer linear programming (MILP) problems [4]; we also obtained it from BCOL. On average, these instances contain 384 variables and 151 constraints (with standard deviations 309 and 127, respectively).

CLS This set of 100 MILP-encoded capacitated lot-sizing instances [5] was also obtained from BCOL. Each instance contains 181 variables and 180 constraints.

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

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

REGIONS100 This set comprises 2 000 instances of the combinatorial auction winner determination problem, encoded as MILP instances. We generated them using the `regions` generator from the Combinatorial Auction Test Suite [22], with parameters *goods*=100 and *bids*=500. On average, the resulting MILP instances contain 501 variables and 193 inequalities (with standard deviations 1.7 and 2.5, respectively).

REGIONS200 This set contains 2 000 instances similar to those in REGIONS100 but larger; we created it with the same generator using *goods*=200 and *bids*=1 000. On average, the resulting MILP instances contain 1 002 variables and 385 inequalities (with standard deviations 1.7 and 3.4, respectively).

MASS This set comprises 100 integer programming instances modelling multi-activity shift scheduling [10]. On average, the resulting MILP instances contain 81 994 variables and 24 637 inequalities (with standard deviations 9 725 and 5 391, respectively).

CORLAT This set comprises 2 000 MILP instances based on real data used for the construction of a wildlife corridor for grizzly bears in the Northern Rockies region (the instances were described by Gomes et al. [11] and made available to us by Bistra Dilkina). All instances had 466 variables; on average they had 486 constraints (with standard deviation 25.2).

4.2 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. (CPLEX did not show these problems on any of the instances studied here.) To deal with them, we took the following measures in our experimental protocol. 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.[3] (For some instances, neither of the two solvers could find a solution within this time; for those instances, we skipped the correctness check described in the following.)

In order to identify problematic parts of a given configuration space, we ran 10 PARAMILS runs (with a time limit of 5 hours each) until one of them encountered a target algorithm run that either produced an incorrect result (as compared to our reference solution for the respective MIP instance), or a segmentation fault. We call the parameter configuration $\boldsymbol{\theta}$ of such a run *problematic*. Starting from this problematic configuration $\boldsymbol{\theta}$, we then identified what we call a *minimal problematic configuration* $\boldsymbol{\theta}_{min}$. In particular, we iteratively changed the value of one of $\boldsymbol{\theta}$'s parameters to its respective default value, and repeated the algorithm run with the same instance, captime, and random seed. If the run still had problems with the modified parameter value, we kept the parameter at its default value, and otherwise changed it back to the value it took in $\boldsymbol{\theta}$. Iterating this process converges to a problematic configuration $\boldsymbol{\theta}_{min}$ that is minimal in the following sense: setting any single non-default parameter value of $\boldsymbol{\theta}_{min}$ to its default value resolves the problem in the current target algorithm run.

Using PARAMILS's mechanism of forbidden partial parameter instantiations, we then forbade any parameter configurations that included the partial configuration defined

[3] These reference solutions were established before we had access to CPLEX 12.1.

by $\boldsymbol{\theta}_{min}$'s non-default parameter values. (When all non-default values for a parameter became problematic, we did not consider that parameter for configuration, clamping it to its default value.) We repeated this process until no problematic configuration was found in the PARAMILS runs: 4 times for GUROBI and 14 times for LPSOLVE. Thereby, for GUROBI we removed one problematic parameter and disallowed two further partial configurations, reducing the size of the configuration space from $1.32 \cdot 10^{15}$ to $3.84 \cdot 10^{14}$. For LPSOLVE, we removed 5 problematic binary flags and disallowed 8 further partial configurations, reducing the size of the configuration space from $8.83 \cdot 10^{16}$ to $1.22 \cdot 10^{15}$. Details on forbidden parameters and partial configurations, as well as supporting material, can be found at `http://www.cs.ubc.ca/labs/beta/Projects/MIP-Config/`.

While that first stage resulted in concise bug reports we sent to GUROBI and LPSOLVE, it is not essential to algorithm configuration. Even after that stage, in the experiments reported here, target algorithm runs occasionally disagreed with the reference solution or produced segmentation faults. We considered the empirical cost of those runs to be ∞, thereby driving the local search process underlying PARAMILS away from problematic parameter configurations. This allowed PARAMILS to gracefully handle target algorithm failures that we had not observed in our preliminary experiments. We could have used the same approach without explicitly identifying and forbidding problematic configurations.

4.3 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. 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.

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, 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 quadratically constrained MJA instances could only be solved with CPLEX).

For each configuration scenario, we allowed a total configuration time budget of 2 CPU days for each of our 10 PARAMILS runs, with a captime of $\kappa_{max} = 300$ seconds for each MIP solver run. In order to penalize timeouts, during configuration we used the penalized average runtime criterion (dubbed "PAR-10" in our previous work [19]), counting each timeout as $10 \cdot \kappa_{max}$. 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):

> "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, our approach improved CPLEX's performance. Specifically, we achieved speedups ranging from 2-

Algorithm	Scenario	% test instances unsolved in 24h		mean runtime for solved [CPU s]		Speedup
		default	PARAMILS	default	PARAMILS	factor
CPLEX	MJA	0%	0%	3.40	**1.72**	**1.98**×
	MIK	0%	0%	4.87	**1.61**	**3.03**×
	REGIONS100	0%	0%	0.74	**0.35**	**2.13**×
	REGIONS200	0%	0%	59.8	**11.6**	**5.16**×
	CLS	0%	0%	47.7	**12.1**	**3.94**×
	MASS	0%	0%	524.9	**213.7**	**2.46**×
	CORLAT	0%	0%	850.9	**16.3**	**52.3**×
GUROBI	MIK	0%	0%	2.70	**2.26**	**1.20**×
	REGIONS100	0%	0%	2.17	**1.27**	**1.71**×
	REGIONS200	0%	0%	56.6	**40.2**	**1.41**×
	CLS	0%	0%	58.9	**47.2**	**1.25**×
	MASS	0%	0%	493	**281**	**1.75**×
	CORLAT	0.3%	**0.2**%	103.7	**44.5**	**2.33**×
LPSOLVE	MIK	63%	63%	21 670	21 670	1×
	REGIONS100	0%	0%	9.52	**1.71**	**5.56**×
	REGIONS200	12%	**0**%	19 000	**124**	**153**×
	CLS	86%	**42**%	39 300	**1 440**	**27.4**×
	MASS	83%	83%	8 661	8 661	1×
	CORLAT	50%	**8**%	7 916	**229**	**34.6**×

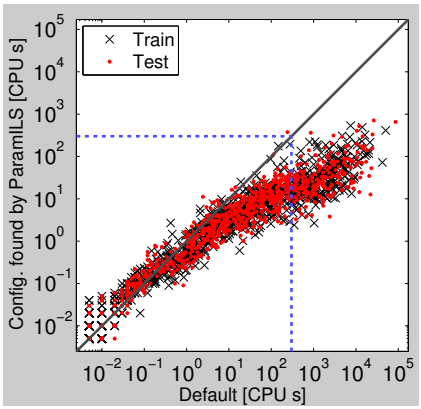
Table 2. Results for minimizing the runtime required to find an optimal solution and prove its optimality. All results are for test sets disjoint from the training sets used for the automated configuration. We report the percentage of timeouts after 24 CPU hours as well as the mean runtime for those instances that were solved by both approaches. Bold-faced entries indicate better performance of the configurations found by PARAMILS than for the default configuration. (To reduce the computational burden, results for LPSOLVE on REGIONS200 and CORLAT are only based on 100 test instances sampled uniformly at random from the 1000 available ones.)

fold to 52-fold. For GUROBI, the speedups were also consistent, but less pronounced (1.2-fold to 2.3-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 maximum captime of $\kappa_{max} = 300s$ 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. For the other benchmarks, speedups were very substantial, reaching up to a factor of 153 (on REGIONS200).

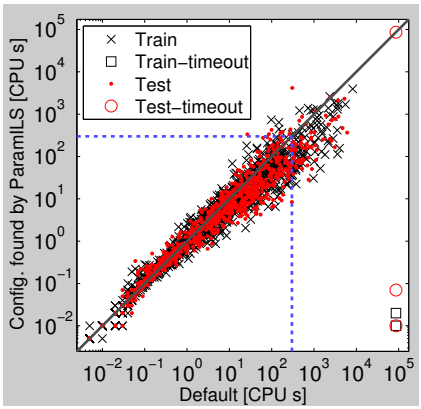
Figure 2 shows the speedups for 4 configuration scenarios. Figures 2(a) to (c) show the scenario with the largest speedup for each of the solvers. In all cases, PARAMILS's configurations scaled better to hard instances than the algorithm defaults, which in some cases timed out on the hardest instances.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

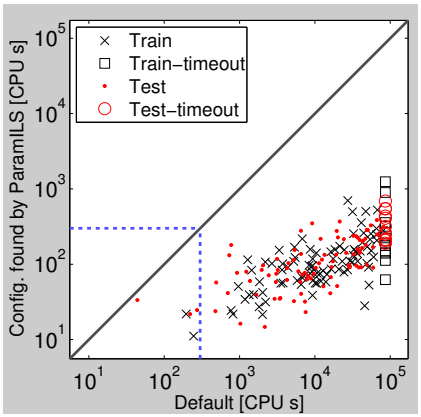
Sometimes, we are interested in minimizing a criterion other than mean runtime. Algorithm configuration procedures such as PARAMILS can in principle deal with various optimization objectives; in our own previous work, for example, we have optimized median runlength, average speedup over an existing algorithm, and average solution quality [20, 15]. In the MIP domain, constraints on the time available for solving a given



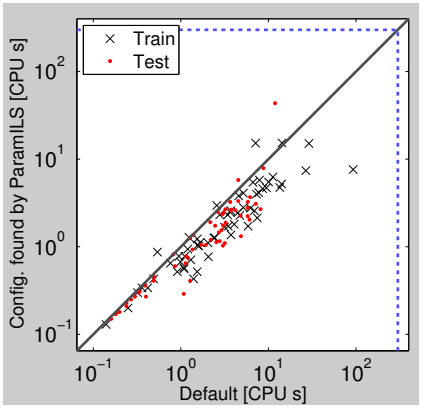
(a) CPLEX, CORLAT. Speedup factors: train 48.4×, test 52.3×.



(b) GUROBI, CORLAT. Speedup factors: train 2.24×, test 2.33×



(c) LPSOLVE, REGIONS200. Speedup factors: train 162×, test 153×.



(d) GUROBI, MIK. Speedup factors: train 2.17×, test 1.20×.

Fig. 2. Results for configuration of MIP solvers to reduce the time for finding an optimal solution and proving its optimality. The dashed blue line indicates the captime ($\kappa_{max} = 300s$) used during configuration.

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, GUROBI and LPSOLVE on the 5 benchmark distributions with the longest average runtimes, with the objective of minimizing the average relative optimality gap achieved within $T = 10$ CPU seconds. To deal with runs that did not find feasible solutions, we used a lexicographic objective function that 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.

Algorithm	Scenario	% test instances for which no feas. sol. was found		mean gap when feasible		Gap reduction
		default	PARAMILS	default	PARAMILS	factor
CPLEX	MIK	0%	0%	0.15%	**0.02%**	**8.65×**
	CLS	0%	0%	0.27%	**0.15%**	**1.77×**
	REGIONS200	0%	0%	1.90%	**1.10%**	**1.73×**
	CORLAT	28%	**1%**	4.43%	**1.22%**	**2.81×**
	MASS	88%	**86%**	1.91%	**1.52%**	**1.26×**
GUROBI	MIK	0%	0%	0.02%	**0.01%**	**2.16×**
	CLS	0%	0%	0.53%	**0.44%**	**1.20×**
	REGIONS200	0%	0%	3.17%	**2.52%**	**1.26×**
	CORLAT	14%	**5%**	3.22%	**2.87%**	**1.12×**
	MASS	68%	68%	76.4%	**52.2%**	**1.46×**
LPSOLVE	MIK	0%	0%	652%	**14.3%**	**45.7×**
	CLS	0%	0%	29.6%	**7.39%**	**4.01×**
	REGIONS200	0%	0%	10.8%	**6.60%**	**1.64×**
	CORLAT	68%	**13%**	4.19%	**3.42%**	**1.20×**
	MASS	100%	100%	-	-	-

Table 3. Results for configuration of MIP solvers to reduce the relative optimality gap reached within 10 CPU seconds. We report the percentage of test instances for which no feasible solution was found within 10 seconds and the mean relative gap for the remaining test instances. 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).

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 14 scenarios the relative gaps were smaller (sometimes substantially so; consider, *e.g.*, the 45-fold reduction for LPSOLVE, and note that the gap is not bounded by 100%). 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 in the available 10 seconds, and the same turned out to be the case for the thousands of configurations considered by PARAMILS.

7 Comparison to CPLEX Tuning Tool

The CPLEX tuning tool is a built-in CPLEX function available in versions 11 and above.[4] It allows the user to minimize CPLEX's runtime on a given set of instances. As in our approach, the user specifies a per-run captime, the default for which is $\kappa_{max} = 10\,000$ 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 the mean is usually dominated by the runtimes of the hardest instances.) By default, the objective for tuning is to minimize 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'. Which configurations it tested depended on the benchmark set.

[4] 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 [14]. At that time, no other mechanism for automatically configuring CPLEX was available; CPLEX 11 was released Nov. 2007.

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 by PARAMILS. For this comparison, we used the tuning tool's default settings to optimize mean runtime on the same training sets used for PARAMILS, and tested performance on the same test sets (disjoint from the training sets). We ran both configuration approaches with a time limit of 2 CPU days. In most cases, the CPLEX tuning tool finished before that time limit was reached and—in contrast to PARAMILS—could not use the remaining time in order to further improve performance. As before, we used 10 independent parallel runs of PARAMILS, at each time step reporting the performance of the one with best training performance.

First, we discuss the performance of the CPLEX tuning tool, summarized in Table 4. We note that in two cases (REGIONS200 and CORLAT), it reached the time limit of 2 CPU days and returned the algorithm defaults in both cases. Out of the remaining 5 cases, it returned the default configuration in 1 (CLS), yielded a configuration with slightly *worse* performance than the default in 1 (REGIONS100), and moderately improved performance in the remaining 3 (up to a factor of 1.37). The 3 non-default configurations it returned only differed in the following few 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); 'long_test1' (use aggressive probing and aggressive settings for 8 types of cuts); and 'branch_dir' (at each node, select the up branch first).

PARAMILS outperformed the tuning tool for 6 of the 7 configuration scenarios, sometimes substantially so. Specifically, PARAMILS found configurations with up to 5.2 times lower mean runtime when its total time budget was set to exactly the amount of time t the CPLEX tuning tool ran before terminating (*i.e.*, $t/10$ for each of the 10 PARAMILS runs; t varied widely across the scenarios, see Table 4). For the one remaining scenario, MASS, the configuration it found was slower by a factor of $1/0.68 = 1.47$ (which we attribute to an over-tuning effect to be discussed shortly). With a fixed time budget of two days for each PARAMILS run, PARAMILS's performance improved for all seven domains, reaching a speedup factor of up to 46.

Figure 3 visualizes the anytime test performance of PARAMILS compared to the default and the configuration found by the CPLEX tuning tool. Typically, PARAMILS found good configurations quickly and improved further when given more time. The main exception was configuration scenario MASS (see Figure 3(e)), the one scenario

Scenario	CPLEX tuning tool stats		CPLEX mean runtime [CPU s] on test set, with respective configuration			
	Tuning time t	Name of result	Default	CPLEX tuning tool	10× PARAMILS(t/10)	10× PARAMILS(2 days)
CLS	104 673	'defaults'	48.4	48.4	**15.1(3.21×)**	**10.1(4.79×)**
REGIONS100	3 117	'easy'	0.74	0.86	**0.48(1.79×)**	**0.34(2.53×)**
REGIONS200	172 800*	'defaults'	59.8	59.8*	**14.2(4.21×)**	**11.9(5.03×)**
MIK	36 307	'long_test1'	4.87	3.56	**1.46(2.44×)**	**0.98(3.63×)**
MJA	2 266	'easy'	3.40	3.18	**2.71(1.17×)**	**1.64(1.94×)**
MASS	28 844	'branch_dir'	524.9	**425.8**	627.4(0.68×)	478.9(0.89×)
CORLAT	172 800*	'defaults'	850.9	850.9*	**161.1(5.28×)**	**18.2(46.8×)**

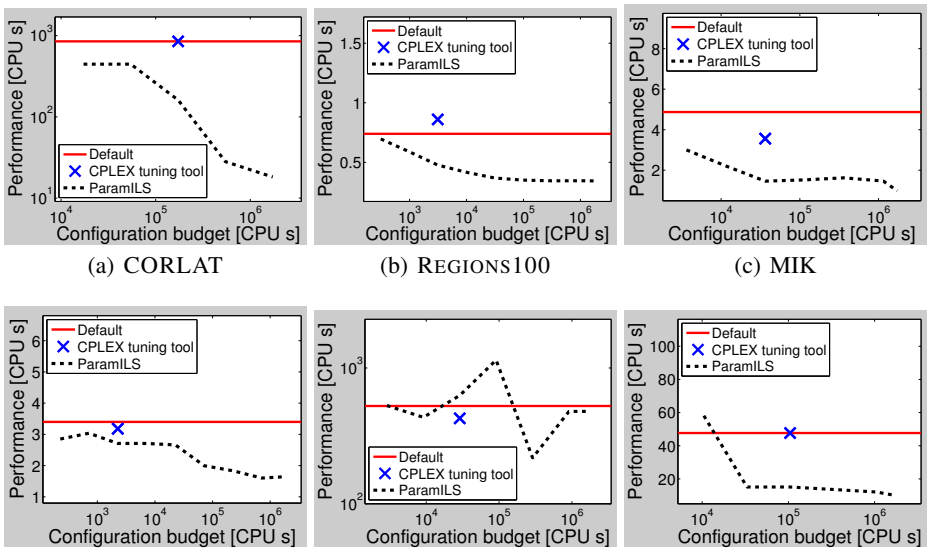
Table 4. Comparison of our approach against the CPLEX tuning tool. For each benchmark set, we report the time t required by the CPLEX tuning tool (it ran out of time after 2 CPU days for REGIONS200 and CORLAT, marked by '*') and the CPLEX name of the configuration it judged best. We report the mean runtime of the default configuration; the configuration the tuning tool selected; and the configurations selected using 2 sets of 10 PARAMILS runs, each allowed time $t/10$ and 2 days, respectively. For the PARAMILS runs, in parentheses we report the speedup over the CPLEX tuning tool. Boldface indicates improved performance.

where PARAMILS performed worse than the CPLEX tuning tool in Table 4. Here, test performance did not improve monotonically: given more time, PARAMILS found configurations with better training performance but worse test performance. This example of the over-tuning phenomenon mentioned in Section 2.3 clearly illustrates the problems arising from benchmark sets that are too small (and too heterogeneous): good results for 50 (rather variable) training instances are simply not enough to confidently draw conclusions about the performance on additional unseen test instances. On all other 6 configuration scenarios, training and test sets were similar enough to yield near-monotonic improvements over time, and large speedups over the CPLEX tuning tool.

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 a broad range of benchmark sets, in terms of minimizing runtime for proving optimality (with speedup factors of up to 52), and of minimizing the optimality gap given a fixed runtime (with gap reduction factors of up to 45). This is particularly noteworthy considering the effort that has gone into optimizing the default configurations for commercial MIP solvers, such as CPLEX and GUROBI. Our approach also clearly outperformed the CPLEX tuning tool. The success of our fully automated approach depends on the availability of training benchmark sets that are large enough to allow generalization to unseen test instances. Not surprisingly, when using relatively small benchmark sets, performance improvements on training instances sometimes do not fully translate to test instances; we note that this effect can be avoided by using bigger benchmark sets (in our experience, about 1000 instances are typically sufficient).

In future work, we plan to develop more robust and more efficient configuration procedures. In particular, here (and in past work) we ran our configurator PARAMILS 10 times per configuration scenario and selected the configuration with best performance on the training set in order to handle poorly-performing runs. We hope to develop more robust approaches that do not suffer from large performance differences between independent runs. Another issue is the choice of captimes. Here, we chose rather large captimes during training to avoid the risk of poor scaling to harder instances; the downside is a potential increase in the time budget required for finding good configurations.



(a) CORLAT (b) REGIONS100 (c) MIK

(d) MJA (e) MASS (f) CLS

Fig. 3. Comparison of the default configuration and the configurations returned by the CPLEX tuning tool and by our approach. The x-axis gives the total time budget used for configuration and the y-axis the performance (CPLEX mean CPU time on the test set) achieved within that budget. For PARAMILS, we perform 10 runs in parallel and count the total time budget as the sum of their individual time requirements. The plot for REGIONS200 is qualitatively similar to the one for REGIONS100, except that the gains of PARAMILS are larger.

We therefore plan to investigate strategies for automating the choice of captimes during configuration. 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 [17, 18, 15].

Acknowledgements

We thank the authors of the MIP benchmark instances we used for making them available, in particular Louis-Martin Rousseau and Bistra Dilkina, who provided the previously unpublished instance sets MASS and CORLAT, respectively. We also thank IBM and Gurobi Optimization for making a full version of their MIP solvers freely available for academic purposes; and Westgrid for support in using their compute cluster. FH gratefully acknowledges support from a postdoctoral research fellowship by the Canadian Bureau for International Education. HH and KLB gratefully acknowledge support from NSERC through 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] Bartz-Beielstein, T. (2006). *Experimental Research in Evolutionary Computation: The New Experimentalism*. Natural Computing Series. Springer Verlag, Berlin.

[8] Birattari, M. (2004). *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium.

[9] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proc. of GECCO-02*, pages 11–18.

[10] 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.

[11] Gomes, C. P., van Hoeve, W.-J., and Sabharwal, A. (2008). Connections in networks: A hybrid approach. In *Proc. of CPAIOR-08*, pages 303–307.

[12] Gratch, J. and Chien, S. A. (1996). Adaptive problem-solving for large-scale scheduling problems: A case study. *JAIR*, 4:365–396.

[13] 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.

[14] 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.

[15] Hutter, F. (2009). *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University Of British Columbia, Department of Computer Science, Vancouver, Canada.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] Hutter, F., Hoos, H. H., and Stützle, T. (2007b). Automatic algorithm configuration based on local search. In *Proc. of AAAI-07*, pages 1152–1157.

[21] 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.

[22] 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.

[23] Mittelmann, H. (2010). Mixed integer linear programming benchmark (serial codes). `http://plato.asu.edu/ftp/milpf.html`. Version last visited on January 26, 2010.