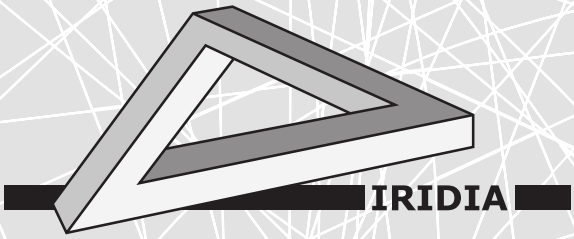

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Automatic Configuration of MIP Solver: Case Study in Vertical Flight Planning

Z. Yuan

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2016-001

February 2016
Last revision: May 2016

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2016-001

Revision history:

TR/IRIDIA/2016-001.001 February 2016
TR/IRIDIA/2016-001.002 May 2016

Automatic Configuration of MIP Solver: Case Study in Vertical Flight Planning

Zhi Yuan

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium

Abstract. Mixed Integer Programming (MIP) solvers are highly parameterized and randomized matheuristic algorithms. How to properly design experiments for algorithm comparison using MIP solvers is an important topic. In this work, we analyze the performance variability of MIP solvers due to random seed settings and parameter configurations. The application domain is the vertical flight planning problem (VFP), which concerns assigning optimal cruise altitude and speed to each trajectory-composing segment, such that the fuel consumption is minimized, and the arrival time constraints are satisfied. Three classic piecewise linear techniques underlying VFP are compared, and their effectiveness highly depends on the solver parameter setting.

Keywords: Automatic configuration; mixed integer programming; performance variability; flight planning; piecewise linear interpolation

1 Introduction

Mixed Integer Programming (MIP) is a general way for modelling many real-world optimization problems. A MIP model can be solved by a general-purpose matheuristic solver such as Cplex, Gurobi, or SCIP, among others. Such MIP solvers usually contain a large number of parameters. For example, IBM ILOG Cplex 12.6 has a total of 159 parameters [14]. Most of these parameters are related to hardware or tolerance, e.g., available working memory, number of threads, random seed, time limit, optimum tolerance, and constraint violation tolerance, etc. These should be fixed according to the hardware in use and the practicality of the problem. Besides, there are algorithmic parameters that can potentially be automatically configured. These parameters feature different types, e.g., categorical parameters such as options of branching strategies, LP method, whether to use certain heuristic; numerical parameters, such as how often a certain heuristic or perturbation is applied, cut limits, and so on; and conditional parameters, such as perturbation constant is only used when perturbation is switched on, or limit of strong candidate list or strong candidate iteration is only used when strong branching is selected. These parameters could potentially be configured for each particular class of problems. There exists work on automatic configuration of the MIP solvers [12] using a general-purpose configuration software ParamILS [13]. Other general-purpose configurators for such task also include iterated racing [3], GGA [1], SMAC [10], etc.

In this work, we empirically compare the three classic piecewise linear formulations as it arises from the vertical flight planning problem [27, 25] using MIP solver. We investigate the performance variability of the MIP solver due to both random seed settings and parameter configurations, and analyze the influence of parameter settings to the algorithmic comparison.

2 Vertical Flight Planning

The vertical flight planning (VFP) problem concerns assigning optimal cruise altitude and speed to each trajectory-composing segment, such that the fuel consumption is minimized, and the arrival time constraints are satisfied. The original MIP models for VFP [27, 26] that assign continuous speed consist of second-order cone constraints, which lead to prohibitively long computation time. The speed discretization scheme proposed in [25] transforms the nonlinear model into linear, and significantly reduces the computation time to within seconds to minutes.

2.1 Mixed Integer Linear Programming model

The MIP model of VFP using discrete speed [25] is as follows.

$$\begin{aligned}
\min \quad & w_0 - w_n && (1)\\
\text{s.t.} \quad & t_0 = 0, \quad \underline{T} \le t_n \le \overline{T} && (2)\\
\forall i \in S: \quad & \Delta t_i = t_i - t_{i-1} && (3)\\
\forall i \in S: \quad & \sum_{v \in V} \mu_{i,v} = 1 && (4)\\
\forall i \in S: \quad & \Delta t = \sum_{v \in V} \mu_{i,v} \cdot \Delta T_{i,v} && (5)\\
& w_n = W^{dry} && (6)\\
\forall i \in S: \quad & w_{i-1} = w_i + f_i && (7)\\
\forall i \in S: \quad & f_i = \sum_{v \in V} \mu_{i,v} \cdot \widehat{F}_{i,v}(w_i). && (8)
\end{aligned}$$

The total fuel consumption (1) measured by the difference of aircraft weight before and after the flight is minimized; (2) ensures the flight duration within a given time window; (3) preserves the time consistency; Only one speed is assigned to each segment by (4), and the travel time on each segment depends on the speed assignment by (5). (6) initializes the weight vector by assuming all trip fuel is burnt during the flight; weight consistency is ensured in (7) and the fuel consumption of each segment in (8) is calculated based on the speed selection μ and a piecewise linear function $\widehat{F} : [w_0, w_{max}] \to \mathbb{R}$ interpolating the fuel consumption F based on weight. The piecewise linear function can be modelled by three different formulations.

The Convex Combination (Lambda) Method. A variant of the *convex combination* (*Lambda method*) [5] can be formulated as follows. To interpolate F

we introduce binary decision variables $\tau_k \in \{0,1\}$ for each $k \in K$, and continuous decision variables $\lambda_k^l, \lambda_k^r \in [0,1]$ for each $k \in K$.

$$\sum_{k \in K} \tau_k = 1 \tag{9a}$$

$$\forall\, k \in K: \quad \lambda_k^l + \lambda_k^r = \tau_k \tag{9b}$$

$$w = \sum_{k \in K} (w_{k-1} \cdot \lambda_k^l + w_k \cdot \lambda_k^r) \tag{9c}$$

$$\widehat{F}(w) = \sum_{k \in K} (F(w_{k-1}) \cdot \lambda_k^l + F(w_k) \cdot \lambda_k^r) \tag{9d}$$

The Incremental (Delta) Method. The *incremental (Delta) method* was introduced by Markowitz and Manne [18]. It uses binary decision variable $\tau_k \in \{0,1\}$ for $k \in K$ and continuous decision variables $\delta_k \in [0,1]$ for $k \in K$, and

$$\forall\, k \in K: \quad \tau_k \geq \delta_k \tag{10a}$$

$$\forall\, k \in K \setminus \{n\}: \quad \delta_k \geq \tau_{k+1} \tag{10b}$$

$$w = w_0 + \sum_{k \in K} (w_k - w_{k-1}) \cdot \delta_k \tag{10c}$$

$$\widehat{F}(w) = F(w_0) + \sum_{k \in K} (F(w_k) - F(w_{k-1})) \cdot \delta_k \tag{10d}$$

The Special Ordered Set of Type 2 (SOS) Method. Instead of introducing binary variables for the selection of a particular interval, we mark the lambda variables as belonging to a special ordered set of type 2 (SOS). That is, at most two *adjacent* variables from an ordered set $(\lambda_0, \lambda_1, \ldots, \lambda_m)$ are positive. Such special ordered sets are treated by the solver with a special SOS branching [2]. We introduce continuous decision variables $0 \leq \lambda_k \leq 1$ for each $k \in K_0$, and:

$$\mathrm{SOS}(\lambda_0, \lambda_1, \ldots, \lambda_m) \tag{11a}$$

$$w = \sum_{k \in K} (w_{k-1} \cdot \lambda_{k-1} + w_k \cdot \lambda_k) \tag{11b}$$

$$\widehat{F}(w) = \sum_{k \in K} (F(w_{k-1}) \cdot \lambda_{k-1} + F(w_k) \cdot \lambda_k) \tag{11c}$$

The comparison of these classic piecewise linear formulations has been the topic in many scientific publications on various optimization problems, including gas network design [19], water network design [9], transportation [23], process engineering [8], flight planning [26], etc. SOS method was found the best in [19], while Delta method was best in [9], and mixed results among the three methods were presented in [23, 8, 26], despite the superior theoretical property of the Delta method over the Lambda method [22].

2.2 Problem Instance

Two of the most common aircraft types, Airbus 320 (A320) and Boeing 737 (B737), are used for our empirical study. The aircraft performance data are pro-

vided by Lufthansa Systems AG. We generated random instances including three flight ranges for A320: 1000, 2000, and 3000 nautical miles (NM), and one flight range of 1500 NM for B737. Two types of segment lengths are generated, the homogeneous instances include segment lengths uniformly randomly generated from 40 to 60 NM, while the heterogeneous instances include segment lengths generated from a uniform distribution from 10 to 90 NM. The expected numbers of segments are 20, 30, 40, and 60 for flight ranges of 1000, 1500, 2000, and 3000 NM, respectively. Three time constraints are considered which require the aircraft to accelerate over its unconstrained optimal speed by factors of 2%, 4%, and 6%. For each of the four aircraft ranges with two homogeneities and three acceleration factors, 10 random instances were generated, totalling 240 instances for testing purpose. Besides, another 240 instances are generated in the same way with different random seeds for training purpose. It is worth noting that these instances represent a broad range of the vertical flight planning problem.

3 Performance Variability of MIP solver

The MIP solvers were believed by many researchers and practitioners for a long time to be deterministic and perfect for benchmarking. The issue of *performance variability* in MIP solvers was first introduced to the MIP community by [4], where she reported a seemly neutral change in the computing environment can result in a drastic drastic change in solver performance. An experiment in [6] also reported a drastic performance variability of up to 25% by adding redundant constraints. A performance variability of up to a factor of 915 for the MIPLIB instances is also observed in [16] by randomly permuting rows or columns of a MIP model. The roots of such performance variability were first explained in [16] to be *imperfect tie-breaking*. There are many decisions to make in a branch-and-cut process, e.g., the cut separation, cut filtering, and the order of the variables to branch on, etc. Such decisions are made based on ordering the candidates by a score. However, it is impossible to have a score that uniquely distinguishes all candidates at each step, thus a deterministic choice was always made when a tie in the score occurs, e.g., by taking always the first candidate. Therefore, changing the order of the variables or constraints will lead to a change of path in the tree search, thus very different behavior and performance in MIP solver. It was further argued in [17] that even if a perfect score for tie-breaking exists, it may be too expensive to compute. Therefore, randomness is intrinsic in MIP solvers, and one can exploit the randomness to improve performance as in [7], where a *bet-and-go* approach is proposed that tries out a number of different neutrally perturbed settings for a short runtime, and then pick the best setting and continue for a long run. However, performance variability must be taken into account in benchmarking, scalability study, or comparing different formulations or new algorithmic ideas to avoid misinterpretation. A more detailed discussion on performance variability can be found in [17].

In response to the issue of performance variability, MIP solvers start to break ties randomly, and allow users to specify a random seed, e.g., Cplex since 12.5.

Table 1. The performance variability of Cplex with default setting under two random seeds for each testing instance: a fixed default seed and a randomly generated seed.

Method	1 thread					12 threads				
	#solved	var.coef.		$\frac{max}{min}$		#solved	var.coef.		$\frac{max}{min}$	
	def./ran.seed	avg.	max.	avg.	max.	def/ran.seed	avg.	max.	avg.	max.
Lambda	233/236	0.28	1.84	1.91	23.79	240/240	0.15	0.82	1.18	2.39
SOS	48/49	0.33	1.06	1.50	3.23	64/64	0.37	1.72	1.82	13.49
Delta	227/230	0.34	1.48	1.59	6.70	220/220	0.22	1.37	1.31	5.31

Ready or not, it is time for us to accept the fact that MIP solvers are randomized algorithms. An experimental study of the MIP solvers should follow a proper experimental setup for randomized algorithms, cf. [21, 15]. E.g., an empirical study of a (randomized) MIP solver based on a fixed random seed (as done in e.g. [19, 23, 9, 8, 26]) will bias the performance evaluation by a fixed sequence of random numbers obtained by the random number generator, which will limit the empirical study to a specific implementation of the MIP solver with a peculiar sequence of random numbers for making random choices. A proper setup for empirical study of a specific problem instance should be performed by running MIP solver with multiple random seeds and collecting proper statistics; empirical study of a problem class should include a wide range of instances from the problem class, and assign to each instance a different random seed. Each instance can be paired with a unique random seed, such that algorithmic ideas to be compared by MIP solver can be evaluated on the same instance with the same seed. The use of common random seed is a variance reduction technique [20].

All experiments ran on a computing node with a 12-core Intel Xeon X5675 CPU at 3.07 GHz and 48 GB RAM. We use Cplex 12.6 with both single thread and 12 threads. The default MIP optimality tolerance 0.01% corresponds to a maximum fuel error of 1 kg for B737, and 2 kg for A320. We fixed the memory parameters such as working memory (`WorkMem`) to 40 GB, and the node storage file switch (`NodeFileInd`) to 3. We first run Cplex with the default setting on the 240 testing instances. Two random seed settings are run, one with Cplex default fixed seed, and the other assigns a different random number to each instance as seed to initialize the random number generator. The goal is to evaluate the performance variability of Cplex due to different random seed settings. We followed the variability measure used in Two types of variability measure are used in this work: (i) *variation coefficient* (termed variability score in [16]), which is a relative variability measure defined as the standard deviation divided by the mean, i.e., $\frac{2\cdot|t_1-t_2|}{t_1+t_2}$ where t_1 and t_2 are the solver computation time with and without specifying a random seed, respectively; (ii) the *ratio* of the maximum and minimum computation time between the two random seeds for each instance, which is also mentioned in [16]. Both the average and the maximum of both variability measures across all instances are presented in Table 1. The variability measure is only calculated with the instances that are solved within 300 seconds. The variability of Cplex due to random seed on vertical flight planning problem is

certainly not negligible. The average variation coefficient is between 0.15 to 0.37, while average ratio is between 1.18 to 1.91. In general, the variability is higher in single thread than parallel computation, especially in Lambda method: both its average measures are 60% to 85% higher in single thread, and it has the highest max-min-ratio of 23.79, which is on an instance that is solved by a randomly generated seed in 12.6 seconds, but takes the Cplex with default seed 300 seconds and still leaves a 0.02% gap. The variability appears to increase for models that are harder to solve, i.e., the variability of SOS is higher than Delta and then higher than Lambda method. Note that the variability measure for especially the SOS method is probably an underestimate, since it is only calculated on a small set of solved instances. Hence although the parallel SOS seems more variable than sequential, the high variability score are mainly contributed by the 15 additional instances that are solved in parallel SOS but not in sequential SOS. Although using the randomly generated seed solves a few more instances to optimality than using the default seed in 1-thread case, no statistical significant difference is found by Wilcoxon's signed rank test or binomial test.

4 Automatic configuration of MIP solver

4.1 Automatic solver configuration

The MIP solvers are highly parameterized matheuristic algorithms. Cplex 12.6 has a total of 159 parameters, where around 70 to 80 parameters can influence algorithmic behaviors. Although Cplex claimed that "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." [14, p. 222], this configuration needs not be a good choice for a specific problem class of interest. Experimental study using MIP solver with only the default configuration will limit the study to only a specific implementation of the MIP solver with a peculiar parameter setting rather than a general algorithm. Especially when empirically comparing algorithmic ideas, there will be high risk of misinterpretation due to the interaction of the algorithmic idea with certain algorithmic parameters of Cplex. Each algorithmic idea to be compared should be given a fair amount of configuration effort by the same procedure.

The automatic configuration experiments are performed on the 240 training instances. 11 copies of the training set of the instances are generated, each using different random instances order and different random seed for each instance. 10 copies are used for 10 independent *training* trials, respectively, and one copy is used for *validation* where the best configurations found in the 10 training trials are compared. The best configuration identified by the validation phase is applied to the *testing* set of instances to assess the quality of the automatically trained configuration. The automatic configuration of Cplex on MIPLIB instances done by Hutter et al. [12] has sped up over the use of default setting by a factor of 1.3 to 52. We followed the algorithmic parameter file for solving MILP by Cplex listed at [11], which has a total of 74 parameters. Two parameters are removed from the list: `lpmethod` since no pure LP exists in our problem (LP relaxation in MIP

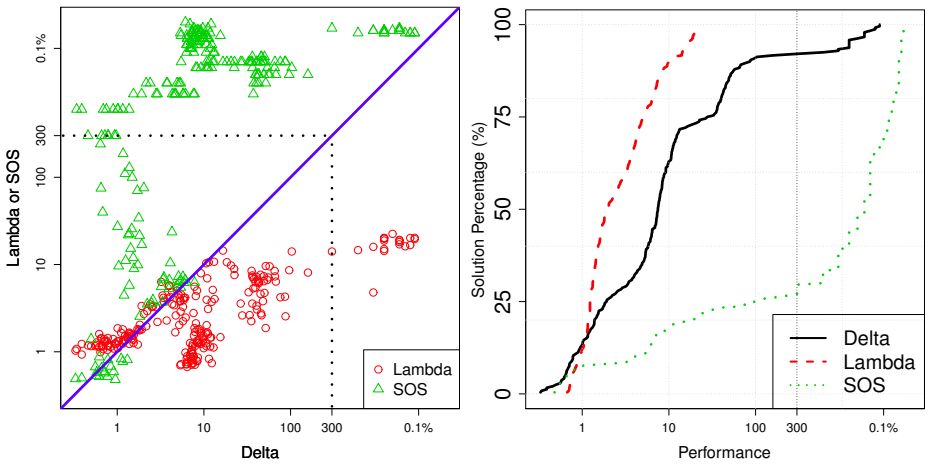

Fig. 1. The comparison of three piecewise linear formulations: Delta, Lambda, and SOS methods, solved by Cplex with default setting.

is controlled by `MIP startalgorithm` or `MIP subalgorithm`); `NodeFileInd` is fixed to 3, such that it allows Cplex to write the node files to hard disk rather than using a swap when the working memory (`WorkMem`) is exceeded.

There exists software for such automatic algorithm configuration tasks, e.g. ParamILS [13] is used in [12]. We have used JRace, which is a Java software for racing based configurators such as iterated racing [3] and post-selection [24, 28]. Each of the three piecewise linear function formulations, Lambda method, Delta method, and the SOS method are trained separately. The cutoff time for the validation and testing phase is set to 300 seconds, and the cutoff time for training phase is set to a much smaller value of 10 seconds. For the instances that are unsolved with a gap of $\delta\%$ after the cutoff time $\kappa = 10$ seconds, we modified the penalized average runtime (`PAR-10`) of $10 \cdot \kappa$ in [12] to `mPAR-10`: $10 \cdot \kappa \cdot (1 + (\delta - 0.01)/10)$, such that the configurations that are unsolved during training can still compare with each other with the optimality gap. A configuration budget of 3 000 evaluations is allowed for each training trial, which amounts to maximum around 8 hours per trial.

4.2 Formulation Comparison with Default Setting

Figure 1 compares the three piecewise linear formulations, Delta, Lambda, and SOS methods, using the Cplex default setting on the testing set with randomly generated seed. The performance distribution plot on the left shows that the Delta method is clearly better performing than SOS, while it is also clearly outperformed by the Lambda method except a few smallest instances. The runtime development plot agrees with the clear trend that Lambda method is the overall best performing formulation of the three. In fact, the Lambda method is the only one that solves all the instances within the cutoff time of 300 seconds. Delta

Table 2. The comparison of the default configuration and the automatically tuned configuration of Cplex and the performance variability induced by using the two configurations. Number of solved instances, average runtime for solved instances, and the number of wins (out of 240) over the other configuration are shown in the comparison of the two configurations. The average speedup factor is also presented.

Method	default			tuned			avg.	var.coef.		$\frac{max}{min}$	
	#solved	avg.time	#win	#solved	avg.time	#win	spdup.	avg.	max.	avg.	max.
Lambda	240	4.24	84	240	3.49	151	1.21	0.20	0.83	1.25	2.42
SOS	64	38.72	39	67	18.87	170	2.05	0.61	1.81	2.83	20.62
Delta	220	38.43	14	240	2.49	226	15.37	1.19	1.92	9.24	51.99

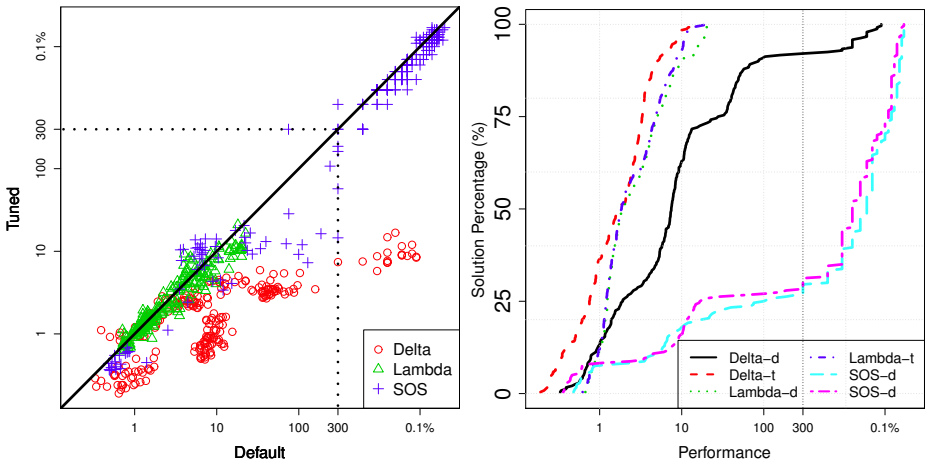

Fig. 2. The comparison of default and tuned parameter setting of Cplex on the three piecewise linear formulations: Delta, Lambda, and SOS methods.

method solves 220 out of 240, and SOS method solves only 64. The difference between the three is also statistically significant by the Wilcoxon's signed rank test or binomial test with $\alpha = 0.01$.

4.3 The Automatically Tuned Setting versus the Default Setting

The comparison of the default configuration and the automatically tuned configuration is listed in Table 2. the tuned configuration significantly outperforms the Cplex default setting for each of the three formulations. The tuned setting of the Lambda method speeds up in average 21% over the default setting, and performs better in 151 instances (out of 240) while worse in 84. The tuned setting of SOS method solves 3 more instances to optimality, and is more than twice as fast as the default setting in the solved instances. It also wins in 170 instances and loses in 39. The tuned setting of Delta method solves all the 20 remaining unsolved problems, improves the default Cplex setting by an average speedup factor of over 15, and performs better than the default setting in 226

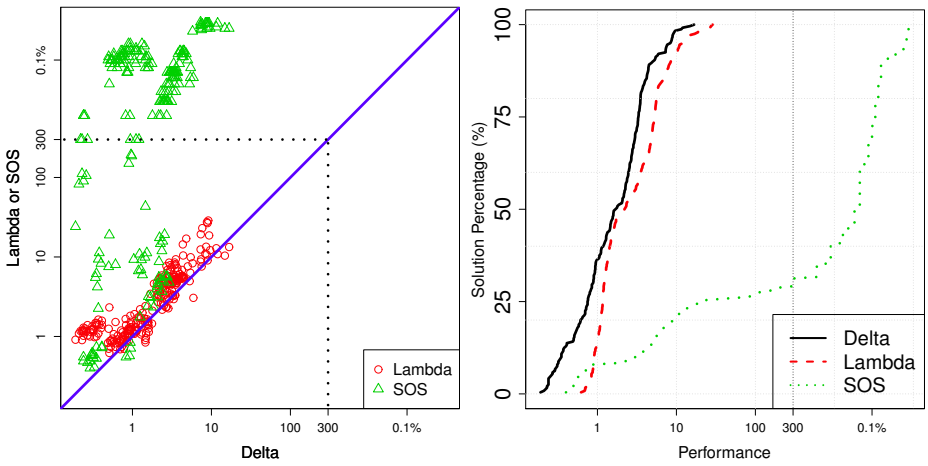

Fig. 3. The comparison of three piecewise linear formulations: Delta, Lambda, and SOS methods, solved by Cplex with tuned configuration.

out of 240 instances. The performance difference between the default and tuned configuration in each of the three formulations is statistically significant. It is worth noting that different formulations benefit from automatic configuration differently. This is best illustrated in Figure 2, the tuned configuration improves the Delta method much more than SOS method which benefits more than the Lambda method. The performance variability measures due to the two different Cplex configurations shown in the last four columns of Table 2 is also drastically higher than the ones shown in Table 1. The highest ratio is ca. 52 times, with an instance solved in 5.8 seconds by the tuned configuration while taking Cplex with default setting 300 seconds with a gap of 0.03%.

4.4 Formulation Comparison with Tuned Setting

With the automatically tuned configuration, the ranking of the three piecewise linear formulations shown in Figure 3 looks quite different from Figure 1 with default setting. The Delta method clearly and statistically significantly outperforms Lambda method, and the average speedup is 40%. This shows that the comparison of different MIP formulations heavily depends on the setting of the MIP solver. Comparing them using only the default setting may limit the conclusion to only a particular implementation of the MIP solver. Generalizing such conclusion may lead to misinterpretation. A more reliable empirical formulation comparison should use a proper experimental setup by applying automatic configuration with equal tuning effort.

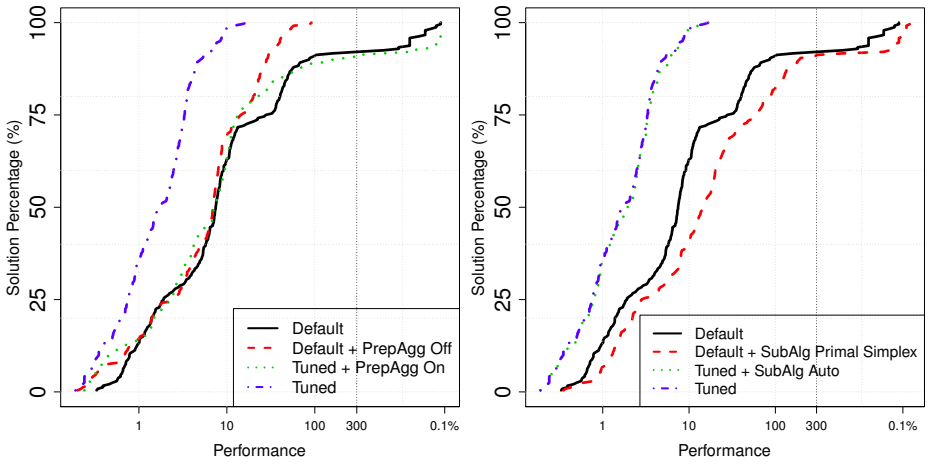

Fig. 4. Solver performance when changing one parameter `preprocessing aggregator` (left) or `MIP subalgorithm` (right) while other parameters are set to either default or tuned value.

4.5 Further Analysis on the Tuned Configuration

Since the tuned configuration drastically improves Delta method over the default one for Cplex, a further analysis is conducted. 38 out of the 72 parameters have been varied by the automatic configuration (the tuned setting of Lambda and SOS differs more to the tuned setting of Delta than default). We tried to vary from the default (tuned) configuration one of these 38 parameters to its tuned (default) value, respectively, evaluated them on the testing instances to assess their influence. The list of changed parameters and their default and tuned value can be found in `http://www2.hsu-hh.de/am/yuanz/downloads/deltaParam.csv`. There are 15 parameter changes from the default setting that can lead to a significant performance improvement (by Wilcoxon's signed rank test with $\alpha = 0.01$). The most influential one is to turn off the preprocessing aggregator (`on` by default). Varying only this parameter already solves the 20 by default unsolved instances, and has an average runtime of 11.4 seconds, which speeds up the default setting by a factor of 3.4. However, 3 single parameter variations from the default setting leads to significant performance deterioration. The most worsening one is changing the `MIP subalgorithm` from default value of `auto` (i.e., always select `dual simplex`) to `primal simplex`, which consumes 2.2 times more runtime than the default setting. On the other hand, there are 7 parameters, changing which from the tuned configuration towards default leads to statistically significantly worse performance. Again, the most influential is to turn the `preprocessing aggregator on` from the tuned configuration, which performs even worse than the default configuration for the hardest instances, as shown in the left of Figure 4. An interesting observation is that the parameter `MIP subalgorithm` also belongs to one of the 7 most influential parameters,

varying which from `primal simplex` to default `auto` statistically significantly worsens the performance (p-value 10^{-8}), and takes in average 5% more runtime, as shown in the right of Figure 4. This shows that it can be misleading to analyze the influence of a single parameter to a given problem, or to set parameters in a one-factor-at-a-time fashion, since it also depends on other parameters. Such parameter correlation should be taken into account, and a sophisticated automatic configuration tool should be applied.

5 Conclusions

MIP solvers are highly parameterized and randomized matheuristic algorithms. In this article, we analyze the performance variability of a MIP solver Cplex, and apply an automatic configuration to Cplex for comparing three classic piecewise linear formulations in the vertical flight planning problem. The performance variability in Cplex due to random seed setting is certainly not negligible, the average variation coefficient ranges from 0.15 to 0.37. The performance variability due to different Cplex parameter settings is even higher, and the formulation comparison depends heavily on the Cplex setting. The experiments are conducted using different random seed and automatic configuration tool JRace. The automatically tuned configuration significantly improves the Cplex default setting in all the three formulations by a factor of up to 15, and makes an inferior formulation by default setting stand out as the best performing formulation.

Acknowledgments This work was partially supported by BMBF Verbundprojekt E-Motion. The author thanks Swen Schlobach and Anton Kaier from Lufthansa Systems for providing aircraft performance data, and Ingmar Vierhaus for literature suggestions. Special thanks to Éric Taillard, Vittorio Maniezzo, and Stefan Voß for the inspiring talks on MIP solver variability in Hamburg, Dec. 2015.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of solvers. In: CP 2009, LNCS, vol. 5732, pp. 142–157. Springer (2009)
2. Beale, E.L.M., Tomlin, J.A.: Global Optimization Using Special Ordered Sets. Mathematical Programming 10, 52–69 (1976)
3. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: An overview. In: Bartz-Beielstein, T., et al. (eds.) Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336. Springer (2010)
4. Danna, E.: Performance variability in mixed integer programming. In: Presentation at Workshop on Mixed Integer Programming (2008)
5. Dantzig, G.B.: On the significance of solving linear programming problems with some integer variables. Econometrica 28(1), 30 – 44 (1960)
6. Fischetti, M., Monaci, M.: On the role of randomness in exact tree search methods. In: Presentation at Matheuristics (2012)

7. Fischetti, M., Monaci, M.: Exploiting erraticism in search. Operations Research 62(1), 114–122 (2014)
8. Fügenschuh, A., Hayn, C., Michaels, D.: Mixed-Integer Linear Methods for Layout-Optimization of Screening Systems in Recovered Paper Production. Optimization and Engineering 15(2), 533–573 (2014)
9. Geißler, B., Martin, A., Morsi, A., Schewe, L.: Using piecewise linear functions for solving MINLPs. In: Mixed Integer Nonlinear Programming, pp. 287–314. Springer (2012)
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of LION-5. pp. 507–523 (2011)
11. Hutter, F.: Automated Configuration of MIP solvers. `http://www.cs.ubc.ca/labs/beta/Projects/MIP-Config`, last accessed: 2016-05-04
12. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Proc. of CPAIOR. LNCS, vol. 9335, pp. 186–202. Springer (2010)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)
14. IBM ILOG CPLEX: 12.6 user's manual. 2014
15. Johnson, D.S.: A theoreticians guide to the experimental analysis of algorithms. Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges pp. 215–250 (2002)
16. Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., et al.: MIPLIB 2010. Mathematical Programming Computation 3(2), 103–163 (2011)
17. Lodi, A., Tramontani, A.: Performance variability in mixed-integer programming. TutORials in Operations Research: Theory Driven by Influential Applications pp. 1–12 (2013)
18. Markowitz, H.M., Manne, A.S.: On the solution of discrete programming problems. Econometrica 25(1), 84 – 110 (1957)
19. Martin, A., Möller, M., Moritz, S.: Mixed integer models for the stationary case of gas network optimization. Mathematical programming 105(2-3), 563–582 (2006)
20. McGeoch, C.: Analyzing algorithms by simulation: variance reduction techniques and simulation speedups. ACM Computing Surveys 24(2), 195–212 (1992)
21. McGeoch, C.C.: Feature article-toward an experimental method for algorithm simulation. INFORMS Journal on Computing 8(1), 1–15 (1996)
22. Padberg, M.: Approximating separable nonlinear functions via mixed zero-one programs. Operations Research Letters 27(1), 1–5 (2000)
23. Vielma, J.P., Ahmed, S., Nemhauser, G.: Mixed-integer models for nonseparable piecewise-linear optimization: unifying framework and extensions. Operations research 58(2), 303–315 (2010)
24. Yuan, Z., Montes de Oca, M., Birattari, M., Stützle, T.: Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. Swarm Intelligence 6(1), 49–75 (2012)
25. Yuan, Z., Amaya Moreno, L., Fügenschuh, A., Kaier, A., Schlobach, S.: Discrete speed in vertical flight planning. In: Corman, F., et al. (eds.) Proc. of ICCL, LNCS, vol. 9335, pp. 734–749. Springer (2015)
26. Yuan, Z., Amaya Moreno, L., Maolaaisha, A., Fügenschuh, A., Kaier, A., Schlobach, S.: Mixed integer second-order cone programming for the horizontal and vertical free-flight planning problem. Tech. rep., AMOS#21, Applied Mathematical Optimization Series, Helmut Schmidt University, Hamburg, Germany (2015)

27. Yuan, Z., Fügenschuh, A., Kaier, A., Schlobach, S.: Variable speed in vertical flight planning. In: Operations Research Proceedings. pp. 635–641. Springer (2014)
28. Yuan, Z., Stützle, T., Montes de Oca, M.A., Lau, H.C., Birattari, M.: An analysis of post-selection in automatic configuration. In: Proceedings of GECCO. pp. 1557–1564. ACM (2013)