Configuração de Parâmetros em Algoritmos de Otimização

Configuração de Parâmetros em Algoritmos de Otimização

Conteúdo

1. Introdução a Configuração de Parâmetros

2. Algoritmos de Configuração Automática de Parâmetros

Introdução a Configuração de Parâmetros

Algoritmos Exatos ou Heurísticos de Otimização

- Técnicas de branch-and-cut e geração de colunas em softwares de programação inteira mista, bem como meta-heurísticas para problemas de otimização combinatória e contínua possuem mecanismos heurísticos e estratégicos.
- A ativação, interação e comportamento desses mecanismos são controlados por parâmetros cuja configuração tem um impacto substancial na eficácia dos algoritmos de otimização

Classes de Parâmetros

- ***Numéricos*** : valores reais, por exemplo, valor do coeficiente no resfriamento geométrico de *simulated annealing*, tamanho da lista tabu, probabilidade de recombinação em algoritmos genéticos.

- ***Categóricos*** : número discreto de valores não ordenados, que selecionam uma opção dentre componentes ou mecanismos alternativos, por exemplo, vizinhanças distintas em busca local, seleção de nós e variáveis em *branch-and-cut*.

- ***Ordinais*** : número discreto de valores ordenados, por exemplo, baixo, médio, alto.

- ***Booleanos*** : dois valores discretos, por exemplo, liga ou desliga um componente do algoritmo.

Exemplos de Parâmetros em Otimização Combinatória

- Parâmetros do método *branch-and-cut* em *softwares* incluem:
  - Seleção de nós e variáveis
  - Seleção de técnicas de pré-processamento
  - Seleção de cortes
  - Balanceamento entre ramificação e cortes
  - Escolha de ênfase em factibilidade ou otimalidade
- CPLEX 12.1 : 135 parâmetros e gera uma configuração automática de parâmetros.
  - Em um experimento (Hutter et al., 2010) 76 parâmetros foram selecionados que produzem $1,9 \cdot 10^{47}$ configurações.
- Metaheurísticas
  - Simulated annealing : escolha de vizinhança, definição dos componentes e parâmetros do programa de resfriamento.
  - Busca tabu de curto prazo : escolha da vizinhança, definição de atributo e regra de proibição, e tamanho da lista tabu.
  - Algoritmo genético: definição de estratégias de seleção e substituição, e de operadores de recombinação e mutação.

Problema de Configuração Automática de Parâmetros

- Interesse por este problema iniciado, provavelmente, pelo software CALIBRA (Adenso-Días e Laguna, 2006) com limite de otimizar no máximo 5 parâmetros numéricos.

- Interesse cresceu bastante e hoje existem diversos softwares disponíveis livremente.

- Vamos apresentar dois métodos recentes de configuração *offline* de parâmetros que usam duas fases.

- Na fase de treinamento, estes métodos determinam a melhor configuração de parâmetros para um conjunto representativo de instâncias.

- Na fase de teste, esta configuração é aplicada a instâncias distintas (generalização).

Youssef Hamadi • Eric Monfroy • Frédéric Saubion
Editors

Autonomous Search

Springer

Chapter 3
Automated Algorithm Configuration and Parameter Tuning

Holger H. Hoos

3.1 Introduction

Computationally challenging problems arise in the context of many applications, and the ability to solve these as efficiently as possible is of great practical, and often also economic, importance. Examples of such problems include scheduling, time-tabling, resource allocation, production planning and optimisation, computer-aided design and software verification. Many of these problems are $\mathscr{NP}$-hard and considered computationally intractable, because there is no polynomial-time algorithm that can find solutions in the worst case (unless $\mathscr{NP}=\mathscr{P}$). However, by using carefully crafted heuristic techniques, it is often possible to solve practically relevant instances of these 'intractable' problems surprisingly effectively (see, e.g., 55, 3, 54) [1].

The practically observed efficacy of these heuristic mechanisms remains typically inaccessible to the analytical techniques used for proving theoretical complexity results, and therefore needs to be established empirically, on the basis of carefully designed computational experiments. In many cases, state-of-the-art performance is achieved using several heuristic mechanisms that interact in complex, non-intuitive ways. For example, a DPLL-style complete solver for SAT (a prototypical $\mathscr{NP}$-complete problem with important applications in the design of reliable soft- and hardware) may use different heuristics for selecting variables to be instantiated and the values first explored for these variables, as well as heuristic mechanisms for managing and using logical constraints derived from failed solution attempts. The activation, interaction and precise behaviour of those mechanisms is often controlled by parameters, and the settings of such parameters have a substantial impact on the

Holger H. Hoos
Department of Computer Science, University of British Columbia, 2366 Main Mall, Vancouver, BC, V6T 1Z4, Canada, e-mail: hoos@cs.ubc.ca

[1] We note that the use of heuristic techniques does not imply that the resulting algorithms are necessarily incomplete or do not have provable performance guarantees, but often results in empirical performance far better than the bounds guaranteed by rigorous theoretical analysis.

Y. Hamadi et al. (eds.), *Autonomous Search*, 37
DOI 10.1007/978-3-642-21434-9_3,

Enunciado do Problema

Definição Uma instância do algoritmo de configuração de parâmetros é uma 6-tupla $\langle A, \Theta, D, k_{\text{max}}, o, m \rangle$, em que:

- A é um algoritmo parametrizado;
- Θ é o espaço de configuração de parâmetros de A;
- D é a distribuição sobre as instâncias do problema com domínio Π;
- k_{max} é o tempo de corte de cada rodada de A;
- o é uma função que mede o custo observado de rodar $A(\theta)$ em uma instância $\pi \in \Pi$ com tempo de corte k (por exemplo o custo da solução encontrada);

Enunciado do Problema

- m é um parâmetro estatístico populacional (média, mediana, variância);

- O_θ : distribuição de custos induzidos pela função o, aplicada a instâncias π retiradas de distribuição D e múltiplas rodadas para algoritmos aleatorizados, em que $k = k_{max}$;

- O custo de uma solução candidata θ é definida por

$$c(\theta) = m(O_\theta)$$

F-Race

- Método inspirado em algoritmos de competição (*racing*) em aprendizado de máquina (*machine learning*).

- Idéia essencial de métodos de *racing* é avaliar um conjunto de configurações candidatas em um conjunto de instâncias. O conjunto de instâncias é gerado a partir de distribuição uniforme.

- Quando há evidência estatística suficiente contra configurações candidatas, estas são eliminadas e a competição com as sobreviventes continua.

- Teste de Friedman é usado para avaliar configurações.

- Se a hipótese nula de diferenças é rejeitada, então testes a posteriori de Friedman são aplicados para eliminar configurações que são muito piores que a melhor.

F-Race

- ni_{min} : número mínimo de instâncias


```
procedure F-Race
    input target algorithm A, set of configurations C, set of problem instances I,
          performance metric m;
    parameters integer ni_min;
    output set of configurations C*;
    C* := C; ni := 0;
    repeat
        randomly choose instance i from set I;
        run all configurations of A in C* on i;
        ni := ni + 1;
        if ni ≥ ni_min then
            perform rank-based Friedman test on results for configurations in C* on all instances
                in I evaluated so far;
            if test indicates significant performance differences then
                c* := best configuration in C* (according to m over instances evaluated so far);
                for all c ∈ C* \ {c*} do
                    perform pairwise Friedman post hoc test on c and c*;
                    if test indicates significant performance differences then
                        eliminate c from C*;
                    end if;
                end for;
            end if;
        end if;
    until termination condition met;
    return C*;
end F-Race
```

Fig. 3.1: Outline of F-Race for algorithm configuration (original version, according to 11). In typical applications, ni_{min} is set to values between 2 and 5; further details are explained in the text. When used on its own, the procedure would typically be modified to return $c^* \in C^*$ with the best performance (according to m) over all instances evaluated within the race

post hoc tests between the incumbent and all other configurations is performed. All configurations found to have performed significantly worse than the incumbent are eliminated from the race. An outline of the F-Race procedure for algorithm configuration, as introduced by [11], is shown in Figure 3.1; as mentioned by [5], runs on a fixed number of instances are performed before the Friedman test is first applied. The procedure is typically terminated either when only one configuration remains, or when a user-defined time budget has been exhausted.

The Friedman test involves ranking the performance results of each configuration on a given problem instance; in the case of ties, the average of the ranks that would have been assigned without ties is assigned to each tied value. The test then determines whether some configurations tend to be ranked better than others when considering the rankings for all instances considered in the race up to the given iteration. Following Birattari et al. [11], we note that performing the ranking separately for each problem instance amounts to a blocking strategy on instances. The use of

F-Race *Iterado*

- Em cada iteração, as configurações sobreviventes são usadas para enviesar (*bias*) a distribuição de probabilidade de geração de novas instância.

- Cada iteração tem três passos:
  - Amostre uma configuração inicial Θ_0^l baseado na probabilidade p_X.
  - Avalie o conjunto Θ_0^l pelo uso de F-Race.
  - Selecione configurações elite de F-Race e atualize p_X.

Aplicações `F-Race` *Iterado*

- 338 citações no Google Scholar do primeiro artigo ”A Racing Algorithm for Configuring Metaheuristics”, publicado em 2002.

A Racing Algorithm for Configuring Metaheuristics

Mauro Birattari[†]
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium

Thomas Stützle, Luis Paquete, and Klaus Varrentrapp
Intellektik/Informatik
Technische Universität Darmstadt
Darmstadt, Germany

Abstract

This paper describes a racing procedure for finding, in a limited amount of time, a configuration of a metaheuristic that performs as good as possible on a given instance class of a combinatorial optimization problem. Taking inspiration from methods proposed in the machine learning literature for model selection through cross-validation, we propose a procedure that empirically evaluates a set of candidate configurations by discarding bad ones as soon as statistically sufficient evidence is gathered against them. We empirically evaluate our procedure using as an example the configuration of an ant colony optimization algorithm applied to the traveling salesman problem. The experimental results show that our procedure is able to quickly reduce the number of candidates, and allows to focus on the most promising ones.

1 INTRODUCTION

A metaheuristic is a general algorithmic template whose components need to be instantiated and properly tuned in order to yield a fully functioning algorithm. The instantiation of such an algorithmic template requires to choose among a set of different possible components and to assign specific values to all free parameters. We will refer to such an instantiation as a *configuration*. Accordingly, we call *configuration problem* the problem of selecting the optimal configuration.

Practitioners typically configure their metaheuristics in an iterative process on the basis of some runs of different configurations that are felt as promising. Usually, such a pro-

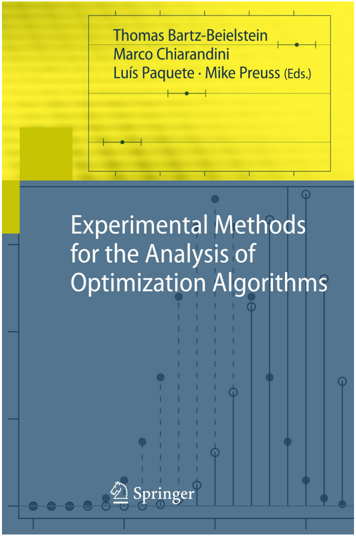
by a mixture of rules of thumb. Most often this leads to tedious and time consuming experiments. In addition, it is very rare that a configuration is selected on the basis of some well defined statistical procedure.

The aim of this work is to define an automatic hands-off procedure for finding a good configuration through statistically guided experimental evaluations, while minimizing the number of experiments. The solution we propose is inspired by a class of methods proposed for solving the model selection problem in memory-based supervised learning (Maron and Moore, 1994; Moore and Lee, 1994). Following the terminology introduced by Maron and Moore (1994), we call *racing method for selection* a method that finds a good configuration (model) from a given finite pool of alternatives through a sequence of steps.[1] As the computation proceeds, if sufficient evidence is gathered that some candidate is inferior to at least another one, such a candidate is dropped from the pool and the procedure is iterated over the remaining ones. The elimination of inferior candidates, speeds up the procedure and allows a more reliable evaluation of the promising ones.

Two are the main contributions of this paper. First, we give a formal definition of the metaheuristic configuration problem. Second, we show that a metaheuristic can be tuned efficiently and effectively by a racing procedure. Our results confirm the general validity of the racing algorithms and extend their area of applicability. On a more technical level, left aside the specific application to metaheuristics, we give some contribution to the general class of racing algorithms. In particular, our method adopts blocking design (Dean and Voss, 1999) in a nonparametric setting. In some sense, therefore, the method fills the gap between Hoeffding race (Maron and Moore, 1994) and BRACE (Moore and Lee, 1994): similarly to Hoeffding race it features a nonparametric test, and similarly to BRACE it considers a

Aplicações `F-Race` *Iterado*

- 338 citações no Google Scholar do primeiro artigo ”A Racing Algorithm for Configuring Metaheuristics”, publicado em 2002.



Aplicações `F-Race` *Iterado*

- Diversas aplicações citadas no artigo de 2010 abaixo.

Chapter 13

`F-Race` and Iterated `F-Race`: An Overview

Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle

Abstract Algorithms for solving hard optimization problems typically have several parameters that need to be set appropriately such that some aspect of performance is optimized. In this chapter, we review `F-Race`, a racing algorithm for the task of automatic algorithm configuration. `F-Race` is based on a statistical approach for selecting the best configuration out of a set of candidate configurations under stochastic evaluations. We review the ideas underlying this technique and discuss an extension of the initial `F-Race` algorithm, which leads to a family of algorithms that we call iterated `F-Race`. Experimental results comparing one specific implementation of iterated `F-Race` to the original `F-Race` algorithm confirm the potential of this family of algorithms.

13.1 Introduction

Many state-of-the-art algorithms for tackling computationally hard problems have a number of parameters that influence their search behavior. Such algorithms include exact algorithms such as branch-and-bound algorithms, algorithm packages for integer programming, and approximate algorithms such as stochastic local search (SLS) algorithms. The parameters can roughly be classified into numerical and categorical parameters. Examples of numerical parameters are the tabu tenure in tabu search algorithms or the pheromone evaporation rate in ant colony optmization (ACO) algorithms. Additionally, many algorithms can be seen as being composed of a set of specific components that are often interchangeable. Examples are different branching strategies in branch-and-bound algorithms, different types of crossover operators in evolutionary algorithms, and different types of local search algorithms in

Mauro Birattari · Zhi Yuan · Prasanna Balaprakash · Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: {mbiro,zyuan,pbalapra,stuetzle}@ulb.ac.be

T. Bartz-Beielstein et al. (eds.), *Experimental Methods for the Analysis of Optimization Algorithms*, DOI 10.1007/978-3-642-02538-9_13, © Springer-Verlag Berlin Heidelberg 2010 311

Aplicações F−Race *Iterado*

330 M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle

Fine-tuning algorithms The by far most common use of F−Race is to use it as a method to fine-tune an existing or a recently developed algorithm. Often, tuning through F−Race is also done before comparing the performance of various algorithms. In fact, this latter usage is important to make reasonably sure that performance differences between algorithms are not simply due to uneven tuning.

A significant fraction of the usages of F−Race is due to researchers either involved in the development of the F−Race method or by their collaborators. In fact, F−Race has been developed in the research for the Metaheuristics Network, an EU-funded research and training network on the study of metaheuristics. Various applications there have been for configuring different metaheuristics for the university-course timetabling problem (Chiarandini and Stützle 2002, Manfrin 2003, Rossi-Doria et al. 2003) and also for various other problems (Chiarandini 2005, Chiarandini and Stützle 2007, den Besten 2004, Risler et al. 2004, Schiavinotto and Stützle 2004).

Soon after these initial applications, F−Race was also adopted by a number of other researchers. Most applications focus on configuring SLS methods for combinatorial optimization problems (Bin Hussin et al. 2007, Balaprakash et al. 2009a, Di Gaspero and Roli 2008, Di Gaspero et al. 2007, Lenne et al. 2007, Pellegrini 2005, Philemotte and Bersini 2008). However, also other applications have been considered, including the tuning of algorithms for training neural networks (Blum and Socha 2005, Socha and Blum 2007) or the tuning of parameters of a control system for simple robots (Nouyan 2008, Nouyan et al. 2008).

Industrial applications Few researches have evaluated F−Race in pilot studies for industrial applications. The first has been a feasibility study, where F−Race was used to configure a commercial solver for vehicle routing and scheduling problems, which has been developed by the software company SAP. In this research, six configuration tasks have been considered that ranged from the study of specific parameters, which determined the frequency of the application of some important operators of the program, to the configuration of the SLS method that was used in the software package. F−Race was compared with a strategy that after each fixed number of instances discarded a fixed percentage of the worst candidate configurations, showing, as expected, advantages for F−Race when the performance differences between configurations were stronger. Some results of this study have been published by Becker et al. (2005); more details are available in a master thesis (Becker 2004).

Yuan et al. (2008) have adopted F−Race to configure several algorithms for a highly constrained train scheduling problem arising at Deutsche Bahn AG. A comparison of various tuned algorithms identified an iterated greedy algorithm as the most promising one.

Algorithm development F−Race has occasionally also been used to explicitly support the algorithm development process. A first example is described by Chiarandini et al. (2006) who used F−Race to design a hybrid metaheuristic for the university-course timetabling problem. In their work they have adopted F−Race in a semi-automatic way. They observed the algorithm candidates that were maintained in a

Aplicações F−Race *Iterado*

race and based on this information they generated new algorithm candidates that were then manually added to the ongoing race. In fact, one of these newly injected candidate algorithms was finally the best performing algorithm in an international timetabling competition (see also http://www.idsia.ch/Files/ttcomp2002).

The PhD work of den Besten (2004) provides an empirical investigation into the application of ILS to solve a range of deterministic scheduling problems with tardiness penalties. Racing in general, and F−Race in particular, is a very important ingredient throughout the algorithm development and calibration. The ILS algorithms are built in a modular way and F−Race is applied to assess each combination of modular components of the algorithm.

Comparison of F−Race with other methods There have been some comparisons of F−Race with other racing algorithms. Some preliminary results comparing F−Race and *t*-test-based racing techniques are presented by Birattari (2004b, 2009), showing that F−Race typically performs best.

Yuan and Gallagher (2004) discuss the use of F−Race for the empirical evaluation of evolutionary algorithms. They also use an algorithm called *A-Race*, where the family-wise test is based on the *analysis of variance* (ANOVA) method. From the experiments they conduct, they conclude that their version of F−Race obtains better results than *A-Race*.

In their work Caelen and Bontempi (2005) compare five techniques from various communities on a model selection task. The techniques compared are (i) a two-stage selection technique proposed in the stochastic simulation community, (ii) a stochastic dynamic programming approach conceived to address the multi-armed bandit problem, (iii) a racing method, (iv) a greedy approach, and (v) a round-search technique. F−Race is mentioned and applied for comparison purposes. The comparison results shows that the bandit strategy yields the most promising performance when the sample size is small, but F−Race outperforms other techniques when the sample size is sufficiently large.

Extensions and hybrids of F−Race The F−Race algorithm has been adopted as a module integrated into an ACO algorithm framework for tackling combinatorial optimization problems under uncertainty (Birattari et al. 2007). The resulting algorithm is called ACO/F−Race and it uses F−Race to determine the best of a set of candidate solutions generated by the ACO algorithm. In later work by Balaprakash et al. (2009b) on the application of estimation-based ACO algorithms to the probabilistic traveling salesman problem the Friedman test is replaced by an ANOVA.

Yuan and Gallagher (2005, 2007) propose an approach to tune evolutionary algorithms by hybridizing Meta-EA and F−Race. Meta-EA is an approach that uses various genetic operators to tune the parameters of EAs. It is reported that one major difficulty in Meta-EA is that it cannot effectively handle categorical parameters. These categorical parameters are usually handled in Meta-EA by pure random search. The proposed hybridization uses Meta-EA to evolve part of the numerical parameters and leave the categorical parameters for F−Race. Experiment show that

ParamILS

- Outro método de configuração automática de parâmetros baseado na meta-heurística busca local iterada (*iterated local search*-ILS).
- ILS é simples, derivada da heurística de Lin-Kernighan (1973) para TSP simétrico, e tem sido aplicada com sucesso.
- ILS parte de uma solução factível e segue uma trajetória determinada por uma vizinhança até chegar a um ótimo local x.
- A solução x é *perturbada aleatóriamente* para escapar do ótimo local, e a busca continua até o próximo ótimo local x', que pode se aceito ou rejeitado.
- Aceitar x' somente se for melhor que x corresponde a uma intensificação da busca, enquanto aceitar sempre x' corresponde a uma diversificação da busca.
- Um critério intermediário é aceitar x' com uma probabilidade similar àquela usada em *simulated annealing*.

ParamILS Básico

- Inicialização: uma dada configuração de partida θ_0, r configurações $\theta_i, i = 1, \ldots, r$ obtidas por distribuição uniforme, e s movimento aleatórios para perturbação, N instâncias.

- Compara θ_0 com θ_i em N instâncias e escolhe a de melhor estimativa $\hat{c}_N(\theta)$ do custo $c(\theta)$.

- Busca local: aceita o primeiro movimento de melhoria do custo e usa s movimentos aleatórios de perturbação.

- Sempre aceita configurações melhores ou de igual qualidade, mas pode reinicializar a busca de forma aleatória com probabilidade $p_{restart}$ (uma ”diversificação”).

- Vizinho obtido por mudança de um único parâmetro.

ParamILS Focado - FocusedILS

- Seleção adaptativa do número de instâncias de treinamento.
- Número pequeno leva a generalização pobre, número grande faz com que o progresso da busca seja muito lento.
- FocusedILS é uma variante de ParamILS que aborda o problema de variar adaptativamente o número de instâncias de treinamento de uma configuração para outra.
- $N(\theta)$: número de rodadas disponíveis para avaliar a estatística do custo $c(\theta)$ da configuração de parâmetros θ.
- **Definição** (Dominância). θ_1 domina θ_2 se e somente se $N(\theta_1) \geq N(\theta_2)$ e $\hat{c}_{N(\theta_2)} \leq \hat{c}_{N(\theta_1)}$.

ParamILS Focado - FocusedILS

- **Lema** (Número ilimitado de avaliações). Seja $N(J,\theta)$ o número de rodadas que FocusedILS foi executado com configuração de parâmetro θ até a iteração J para estimar $c(\theta)$. Então para qualquer constante K e configuração $\theta \in \Theta(|\Theta|\mathit{finito})$

$$\lim_{J\to\infty} P[N(J,\theta) \geq K] = 1$$

- **Definição** (Estimador consistente). $\hat{c}_N(\theta)$ é um estimador consistente de $c(\theta)$ se e somente se

$$\forall\epsilon > 0 : \lim_{N\to\infty} P|(\hat{c}_N(\theta) - c(\theta)| < \epsilon) = 1$$

- **Lema** (Sem enganos para $N \to \infty$). Sejam θ_1 e θ_2 duas configurações de parâmetros com $c_{\theta_1} < c_{\theta_2}$. Então, para estimadores consistentes $\hat{c}_N$

$$\lim_{N\to\infty} P(\hat{c}_N(\theta_1) \geq \hat{c}_N(\theta_2)) = 0$$

ParamILS - Resultados Computacionais

190 F. Hutter, H.H. Hoos, and K. Leyton-Brown

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

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	

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 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

ParamILS - Resultados Computacionais

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.

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

ParamILS - Resultados Computacionais

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.

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.)

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

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

ParamILS - Resultados Computacionais

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 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.

ParamILS - Resultados Computacionais

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.)

Algorithm	Scenario	% test instances unsolved in 24h		mean runtime for solved [CPU s]		Speedup factor
		default	PARAMILS	default	PARAMILS	
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×**

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."

ParamILS - Resultados Computacionais

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 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

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).

Algorithm	Scenario	% test instances for which no feas. sol. was found		mean gap when feasible		Gap reduction factor
		default	PARAMILS	default	PARAMILS	
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%	-	-	-

ParamILS - Resultados Computacionais

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.

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.

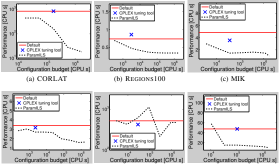
PARAMILS differs from the CPLEX tuning tool in at least three crucial ways. First, it searches in the vast space of all possible configurations, while the CPLEX tuning tool focuses on a small set of handpicked candidates. Second, PARAMILS is a randomized procedure that can be run for any amount of time, 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

[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 - Resultados Computacionais

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.

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×)**



(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.

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.

Referências

Adenzo-Díaz, B. e Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search, Operations Research, 54, 99-114.

Birattari, M., Yuan, Z., Balaprakash, P. e Stützle, T. F-Race and iterated F-Race: an overview, em *Experimental Methods for the Analysis of Optimization Algorithms*, Thomaz Bartz-Beielstein et al. (eds), 311-336, Springer Verlag, 2010.

Hoos, H.H. Automated algorithm configuration and parameter tuning, em *Autonomous Search*, Youssef et al. (eds), 37-72, Springer Verlag, 2011.

Hutter, F., Hoos, H.H., Leyton-Brown, K., e Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework, Journal of Artificial Intelligence Research, 36, 267-306.

Hutter, F., Hoos, H.H. e Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers, LNCS 6140, 183-202.

López-Ibáñez, M. e Stützle, T. (2014). Automatically improving the anytime behaviour of optimisation algorithms, European Journal of Operational Research, 235, 569-582.

Schneider, M. e Hoos H.H. (2012). Quantifying homogeneity of instance sets for algorithm configurations, LNCS 7219, 190-204.