LEAPSANDBOUNDS: A Method for Approximately Optimal Algorithm Configuration

Gellért Weisz [1] **András György** [1 2] **Csaba Szepesvári** [1 3]

Abstract

We consider the problem of configuring general-purpose solvers to run efficiently on problem instances drawn from an unknown distribution. The goal of the configurator is to find a configuration that runs fast on average on most instances, and do so with the least amount of total work. It can run a chosen solver on a random instance until the solver finishes or a timeout is reached. We propose LEAPSANDBOUNDS, an algorithm that tests configurations on randomly selected problem instances for longer and longer time. We prove that the capped expected runtime of the configuration returned by LEAPSANDBOUNDS is close to the optimal expected runtime, while our algorithm's running time is near-optimal. Our results show that LEAPSANDBOUNDS is more efficient than the recent algorithm of Kleinberg et al. (2017), which, to our knowledge, is the only other algorithm configuration method with non-trivial theoretical guarantees. Experimental results on configuring a public SAT solver on a new benchmark dataset also stand witness to the superiority of our method.

1. Introduction

For computational problems of major practical interest (satisfiability, planning, etc.) the computing science community has developed a large number of highly configurable "solvers." The reason is that while the hardest problem instances take a long time to solve by any of the solvers, the instances that one encounters in practical applications may exhibit specific properties so that the appropriate solver with an appropriate configuration may finish much faster. The plethora of solvers and their configurations, which for simplicity of presentation we will just treat as configurations from this point on, is explained by the diversity of applications. Which configuration to use in a specific application can then be treated as a learning problem, where an application is identified with an unknown distribution over problem instances that one can sample from, the learning algorithm can run any configuration on any sampled instance until a timeout of its choice, and the goal is to find a configuration with nearly optimal expected runtime while using the least amount of time during the search.[1] There has been much practical success on designing such black-box configuration search methods, especially in the context of satisfiability problems. Examples of successful methods include ParamILS (Hutter, 2007; Hutter et al., 2009), SMAC (Hutter et al., 2011; 2013), irace (Birattari et al., 2002; López-Ibánez et al., 2011), and GGA (Ansótegui et al., 2009; 2015). These methods themselves rely on many heuristics and as such lack theoretical guarantees.

Recently, Kleinberg et al. (2017) explored this problem, presenting a general-purpose configuration optimizer called *Structured Procrastination*, with guarantees on both *(i)* how close to the optimal configuration the algorithm's result is, and *(ii)* how long it takes to find such a configuration. For *(ii)*, Kleinberg et al. (2017) prove that the expected runtime of their algorithm is within a logarithmic factor of the optimal runtime in a worst-case sense. Furthermore, they show that the gap between worst-case runtimes of existing algorithms (SMAC, ROAR, ParamILS, GGA, irace) and their solution can be arbitrarily large. Structured Procrastination attempts to refine the runtime guarantee for the empirically fastest solver and solves tasks in increasing order of difficulty, postponing difficult tasks until all simpler tasks have been solved. The main novelty of their work is that it comes with theoretical guarantees (lower and upper bounds on the runtime), but no empirical illustration is provided.

This paper builds on the results of Kleinberg et al. (2017), and our problem statement closely follows theirs. Our

[1]DeepMind, London, UK. [2]On leave from Imperial College London, London, UK. [3]On leave from University of Alberta, Edmonton, AB, Canada. Correspondence to: Gellért Weisz <gellert@google.com>, András György <agyorgy@google.com>, Csaba Szepesvári <szepi@google.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

[1]Related, but different problems are considered, e.g., by Luby et al. (1993); Adam (2001); Mnih et al. (2008); Audibert & Bubeck (2010); György & Kocsis (2011); Li et al. (2016).

main technical contributions are as follows: We present an (arguably simpler) algorithm (LEAPSANDBOUNDS) that finds an approximately optimal configuration with a worst-case runtime bound that improves upon that of Kleinberg et al. (2017), while we consider a broader class of problems (we don not need their global runtime cap). We also present instance-dependent runtime bounds that show that LEAPSANDBOUNDS finishes faster if the runtime of the configurations over different problem instances has small variance. Experiments were carried out to assess practical performance of both Structured Procrastination and LEAPSANDBOUNDS on configuring the open-source `minisat` solver. LEAPSANDBOUNDS runs every configuration for less time than Structured Procrastination, and returns significantly faster. Finally, to facilitate further research and enable direct comparison to our results, our large-scale measurements on running times of the `minisat` solver are published together with the paper.[2]

The rest of the paper is organized as follows: The problem is introduced formally in Section 2. For clarity, the most basic version of our algorithm is presented first in Section 3, and its performance is analyzed in Section 4. Improvements to our method, together with their analyses, are presented in Section 5. Experimental results are presented in Section 6, followed by some notes on parallel implementation in Section 7. Finally, conclusions are drawn in Section 8.

2. Problem Statement

Following Kleinberg et al. (2017), the algorithm configuration problem is defined by a tuple $(\mathcal{N}, \Gamma, R, \kappa_0)$ as follows:[3] Here, $\mathcal{N}$ is a family of configurations and Γ is a distribution over input instances.[4] For now, we consider the case when $\mathcal{N}$ is a finite set. If we have a benchmark set of instances, we let Γ be the uniform distribution over these benchmark instances. For configuration $i \in \mathcal{N}$ and instance j, $R(i,j) \in [0,\infty]$ is the execution time of configuration i on instance j. Finally, $\kappa_0 > 0$ is the minimum runtime: For all i, j pairs, $R(i,j) \geq \kappa_0$.

We let $R(i) = \mathbb{E}_{J\sim\Gamma}[R(i,J)]$ denote the average runtime of configuration i on instances distributed according to Γ, and define $\mathrm{OPT} = \min_i R(i)$ as the mean runtime of an optimal configuration. *Our goal is to find such an optimal, or at least nearly optimal configuration while spending as little time as possible–proportional to the runtime of the optimal configuration–on this task.* For this, a search algorithm can *(i)* sample instances J at random from Γ; *(ii)* enumerate the configurations in $\mathcal{N}$; *(iii)* run a configuration i on an instance j until it finishes, or the execution time exceeds a fixed timeout $\tau \geq 0$, chosen by the search algorithm. Practically, this means observing $R(i,j,\tau) \doteq \min(R(i,j),\tau)$ after time $R(i,j,\tau)$, and also whether the calculation has finished with a solution or it timed out.

The main difficulty in organizing the search is that some configurations may take a long, or even infinite time to execute on some instances. Since an algorithm that claims to *find* a near-optimal configuration must verify that no other configuration can finish significantly faster than the chosen configuration, the total runtime is at least proportional to $n \times \mathrm{OPT}$, where $n = |N|$ is the number of configurations to be tested. Since knowing the mean runtime up to a multiplicative accuracy of $(1+\varepsilon)$ requires $\Omega(1/\varepsilon^2)$ samples even when the runtime distributions are light-tailed, relaxing the requirement to find a configuration i with runtime $R(i) \leq (1+\varepsilon)\mathrm{OPT}$, we get that the total runtime is at least $\Omega(n \times \mathrm{OPT}/\varepsilon^2)$. The situation worsens for heavy-tailed runtime distributions: If the runtime of an algorithm is $b > 1$ with probability $1/b$ and 0 otherwise, with b unknown, all sampling methods need to see at least one positive runtime to estimate the expected runtime up to any fixed accuracy. Thus, any sampling method needs to use at least $\Omega(b)$ time, despite that the expected runtime is constant 1 (independently of b). This implies that in the face of heavy-tailed runtime distributions, the runtime of any sound configuration search algorithm would be unbounded in the worst-case, regardless the value of OPT, n and ε. Since heavy tailed runtime distributions are quite common in practice, rather than constraining the problem by ruling these out, following Kleinberg et al. (2017), we relax the search criterion to that of finding an (ε,δ)-optimal configuration. Introducing the τ-capped version of $R(i)$ as $R_\tau(i) = \mathbb{E}_{J\sim\Gamma}[R(i,J,\tau)]$, we have the following definition for (ε,δ)-optimality:

Definition 1 ((ε,δ)-optimality)**.** *A configuration i^* is (ε,δ)-optimal if there exists some threshold τ such that $R_\tau(i^*) \leq (1+\varepsilon)\mathrm{OPT}$, and $\Pr_{J\sim\Gamma}(R(i^*,J) > \tau) \leq \delta$. Otherwise, we say i^* is (ε,δ)-suboptimal.*

In words, given (ε,δ), a sound configuration search algorithm must find a configuration i whose τ-capped mean runtime is at most $(1+\varepsilon)\mathrm{OPT}$, with a τ larger than the δ-quantile of the runtime distribution of configuration i. This is a reasonable criterion when OPT is reasonably small.[5]

In this paper we introduce the algorithm LEAPSANDBOUNDS that identifies an (ε,δ)-optimal configuration with

[2] https://github.com/deepmind/leaps-and-bounds

[3] Compared to their problem statement, we removed the global runtime cap from the definition as it is not required for our results.

[4] For randomized solvers, input instances can mean (input instance, random seed) pairs.

[5] If some problem instances are hopelessly hard, the expected runtime of even an optimal configuration can be infinite, in which case any configuration becomes (ε,δ)-optimal. To alleviate this problem, it would be more meaningful to define (ε,δ)-optimality with respect to the optimal capped runtime; this is left for future work (see Section 8 for more details).

probability $1-\zeta$ for a failure parameter ζ and has an expected runtime of $\mathcal{O}\left(\mathrm{OPT}\frac{n}{\varepsilon^2\delta}\log\left(\frac{n\log\mathrm{OPT}}{\zeta}\right)\right)$. The method of Kleinberg et al. (2017) has an additional assumption that all runtimes of any configuration on any instance sampled from Γ are below a maximum $\bar{\kappa}$ that must also be known by the algorithm. While this renders the runtime distributions light-tailed, a nice feature of their method is that its runtime $\mathcal{O}\left(\mathrm{OPT}\frac{n}{\varepsilon^2\delta}\log\left(\frac{n\log\bar{\kappa}}{\zeta\delta\varepsilon^2}\right)\right)$ has only a mild dependence on $\bar{\kappa}$. LEAPSANDBOUNDS does not require a runtime cap and we shave off a few terms from their bound: We replace the doubly logarithmic dependence on $\bar{\kappa}$ with an identical dependence on the practically much smaller OPT, and remove logarithmic terms that depend on δ^{-1} and ε^{-2}. Kleinberg et al. (2017) also prove that the minimum worst-case runtime for any algorithm is $\Omega\left(\mathrm{OPT}\frac{n}{\varepsilon^2\delta}\right)$, so both methods are within a logarithmic factor of the optimum.

The above results make sense when $n = |N|$ is small enough to allow running each algorithm configuration. Similarly to Kleinberg et al. (2017), LEAPSANDBOUNDS can be extended to the case of an arbitrarily large number of configurations: by sampling n configurations randomly from the set of all configurations, the probability that none of the fastest γ fraction of configurations have been sampled is at most $Ce^{-n\gamma}$ for a universal constant $C > 0$. Thus, by letting $n = \left\lceil \frac{1}{\gamma}\log(C/\zeta)\right\rceil$, with probability $1-2\zeta$, LEAPSANDBOUNDS returns an (ε,δ)-optimal configuration with respect to a configuration from the fastest γ fraction of configurations from the entire space.

3. Algorithm

The main problem in finding a near-optimal solver configuration is that solving some instances may take arbitrarily long. To alleviate the problem, (ε,δ)-optimality only considers the mean of runtimes capped at a timeout, ensuring that at most a δ fraction of the worst instances run longer than this timeout. This makes estimating the average runtime of a configuration (over random instances) possible through sampling. The main issue with sampling is that computing the average runtime over the samples can be slowed down arbitrarily if we accidentally select a problem instance with a very large running time. This could be avoided if an oracle told us the runtime threshold τ in the definition of (ε,δ)-optimality, but this is not available of course. To solve the problem, we present a configuration optimization algorithm called LEAPSANDBOUNDS (Algorithm 1).

LEAPSANDBOUNDS attempts to guess a rough value of OPT, starting from a low value. Calling its guess θ, the algorithm then tries to find a configuration with a mean runtime less than θ. If this succeeds, it returns the configuration with the smallest mean found. Otherwise, θ is doubled and a new phase is started. The simplest way of measuring

Algorithm 1 LEAPSANDBOUNDS

Inputs:
  Set $\mathcal{N}$ of n algorithm configurations
  Precision parameter $\varepsilon \in (0, \frac{1}{3})$
  Quantile parameter $\delta \in (0,1)$
  Failure probability parameter $\zeta \in (0,1)$
  Lower runtime bound $\kappa_0 > 0$
  Instance distribution Γ
Initialize:
  $\theta \leftarrow \frac{16}{7}\kappa_0$, $k \leftarrow 0$, $\mathcal{J} \leftarrow$ empty list
while True **do**
  $k \leftarrow k+1$ ▷ phase count
  $b \leftarrow \left\lceil 44\log\left(\frac{6nk(k+1)}{\zeta}\right)\frac{1}{\delta\varepsilon^2}\right\rceil$ ▷ instance bound
  Add $b - |\mathcal{J}|$ new instances sampled from Γ to $\mathcal{J}$
  for $i \in \mathcal{N}$ **do**
    $\bar{Q}_i \leftarrow$RUNTIMEEST $(i, \mathcal{J}, \delta, \theta)$
  end for
  if $\min_i \bar{Q}_i < \theta$ **then**
    return $\operatorname{argmin}_i \bar{Q}_i$
  end if
  $\theta \leftarrow 2\theta$
end while

the mean runtime while guaranteeing (ε,δ)-optimality is to take runtime samples with timeout $\frac{\theta}{\delta}$ and reject any algorithm that times out for any instance. Then, a concentration bound on the measurements could be used to ensure that the mean is close to the empirical mean. If the mean is less than θ, Markov's inequality can be used to bound the tail probability for (ε,δ)-optimality. However, by rejecting any configuration that ever times out, we fail to measure the capped mean–which could be significantly lower–, and thus the algorithm may not stop at the right time. To fix this, we would ideally allow a δ fraction of runs to time out, but we use $\frac{3}{4}\delta$ instead, to achieve a high-confidence tail bound with a Chernoff bound (replacing Markov's inequality). Still, the measurements could take a long time: if we perform b measurements for a reliable mean estimation with timeout τ, then we spend up to $b\tau$ time. A key observation is that if we spend more than $b\theta$ time on measurements, the average would have to be above θ, and we would reject the configuration. Thus, we can specify an overall time budget of $T = b\theta$, and reject any configuration early if they run over this limit. These ideas are embodied in our algorithm (RUNTIMEEST).

4. Theoretical Analysis

In this section we explore the theoretical properties of LEAPSANDBOUNDS. We show that the estimates computed by the algorithm are reliable with high probability. Then we prove that if the estimates are reliable, the running time cannot be too large and the algorithm returns an (ε,δ)-optimal

Algorithm 2 The RUNTIMEEST subroutine

Inputs:
  Configuration i
  Instance list $\mathcal{J} = (J_1, \ldots, J_b)$ of length b
  Quantile parameter $\delta \in (0, 1)$
  Average runtime bound θ
Initialize:
  $T \leftarrow b\theta$ ▷ overall runtime budget
  $\tau \leftarrow \frac{4\theta}{3\delta}$ ▷ individual runtime budget
  $j \leftarrow 1$ ▷ instance index
while True **do**
  Run configuration i on J_j with timeout $\min\{T, \tau\}$
  $Q_j \leftarrow R(i, J_j, \min\{T, \tau\})$
  $T \leftarrow T - Q_j$
  // Stopping rules:
    if $T = 0$ **then** ▷ Stop if overall budget zero
      return θ
    else if $j = b$ **then** ▷ Stop after $b = |\mathcal{J}|$ samples
      return $\bar{Q} = 1/b \sum_{m=1}^{b} Q_m$ ▷ Return mean
    end if
  $j \leftarrow j + 1$
end while

solution. We start with a few important observations about RUNTIMEEST.

4.1. Guarantees for algorithm RUNTIMEEST

Consider the execution of RUNTIMEEST with the inputs $(i, \mathcal{J}, \theta, b)$. Noting that the loop is stopped if the budget $T_0 = b\theta$ gets exhausted, it follows that the total runtime of the (optimized) algorithm is bounded by $b\theta$:

Lemma 2. *The runtime of one call to* RUNTIMEEST *is* $\mathcal{O}(b\theta)$.

With $T_0 = b\theta$, for $j \geq 1$, define $T_j = T_{j-1} - Q_j = b\theta - (Q_1 + \cdots + Q_j)$. If the budget $b\theta$ is not exhausted (i.e., $T_b = b\theta - (Q_1 + \cdots + Q_b) > 0$), each instance J_j runs within its $\min\{T_{j-1}, \tau\} \leq \tau$ individual budget, and so $Q_j = R(i, J_j) = R(i, J_j, \tau) =: R_j$. Clearly, $T_b > 0$ is equivalent to $\bar{Q} < \theta$. Furthermore, in any case, $Q_j = R(i, J_j, \min(T_{j-1}, \tau)) \leq R_j$. Defining $\bar{R} = (R_1 + \cdots + R_b)/b$, we can summarize these findings as follows:

Lemma 3. *If* RUNTIMEEST *returns with* $\bar{Q}{<}\theta$*, then* $\forall j, Q_j{=}R_j$ *and* $\bar{Q}{=}\bar{R}$*. Otherwise,* $\forall j, Q_j{\leq}R_j$ *and* $\bar{Q}{\leq}\bar{R}$*.*

Let us now turn to analyzing Algorithm 1. For this, we need some extra notation.

4.2. Notation

Let θ_k, τ_k and b_k denote the respective values of θ, τ and b in phase k (Line 6 of Algorithm 1), noting that $\tau_k = \frac{4\theta_k}{3\delta}$. Let J_j denote the jth instance (ever) sampled in Line 6 of Algorithm 1. Note that for k large enough so that $b_k \geq j$, J_j is the jth instance that is passed on to RUNTIMEEST by Algorithm 1 in phase k (for any configuration i). Let $R_{i,j,k} = R(i, J_j, \tau_k)$ be the τ_k-capped runtime of configuration i on instance J_j and let $\bar{R}_{i,k}$ be the average of these values: $\bar{R}_{i,k} = \frac{1}{b_k} \sum_{j=1}^{b_k} R_{i,j,k}$. Similarly, let $\bar{Q}_{i,k}$ be the return value of algorithm RUNTIMEEST in phase k for configuration i, which is also the mean of $(Q_{i,j,k})_j$, the runtimes observed at Line 5 of RUNTIMEEST. Let $\hat{\sigma}^2_{i,k}$ be the empirical variance of $(R_{i,j,k})_j$: $\hat{\sigma}^2_{i,k} = \frac{1}{b_k} \sum_{j=1}^{b_k} (R_{i,j,k} - \bar{R}_{i,k})^2$.

4.3. Good events

Let $p_{i,k} = \Pr_{J \sim \Gamma}(R(i, J) > \tau_k)$ denote the probability that configuration i does not finish on instance j in time τ_k. Next we define two events that ensure that the algorithm works well. First, let

$$E_{1,i,k} = \{\bar{Q}_{i,k} = \theta_k\} \cup \{p_{i,k} \leq \delta\};$$

if $E_{1,i,k}$ holds then if Algorithm 1 returns, the probability that the corresponding configuration fails to solve a random task within τ_k time is small (note that $\bar{Q}_{i,k} \leq \theta_k$).

The next event guarantees that the average capped running time is close to its expectation: let

$$E_{2,i,k} = \{|\bar{R}_{i,k} - R_{\tau_k}(i)| \leq C_{i,k}\}$$

with

$$C_{i,k} = \hat{\sigma}_{i,k} \sqrt{\frac{2 \log(\frac{6nk(k+1)}{\zeta})}{b_k}} + \frac{3\tau_k \log(\frac{6nk(k+1)}{\zeta})}{b_k} .$$

The main result of this section is to show that $E_{1,i,k}$ and $E_{2,i,k}$ hold with high probability for all i and k simultaneously:

Lemma 4. *Let* $E = \bigcap_{i \in \{1,\ldots,n\}, k \in \mathbb{Z}_+} (E_{1,i,k} \cap E_{2,i,k})$*. Then,* $\Pr(E) \geq 1 - \zeta$*.*

To prove the lemma, we individually bound the probabilities that the events do not hold:

Lemma 5. $\Pr(E^c_{1,i,k}) \leq \frac{\zeta}{2nk(k+1)}$.

Proof. If $p_{i,k} \leq \delta$, then $\Pr(E^c_{1,i,k}) = 0$ and the statement holds trivially. For the rest of this proof, we assume that $p_{i,k} > \delta$. From the algorithm, we have that $b_k \geq \frac{32}{\delta} \log(\frac{2nk(k+1)}{\zeta})$. Define $B_{i,j,k}$ as the Bernoulli random variable indicating whether configuration i on input J_j takes more time than τ_k to finish (value 1), or not (value 0). For $\hat{\delta}_{i,k} = \frac{1}{b_k} \sum_{j=1}^{b_k} B_{i,j,k}$, observe that $\mathbb{E}(\hat{\delta}_{i,k}) = p_{i,k}$. If the algorithm returns with $\bar{Q}_{i,k} < \theta$, as necessary for event $E^c_{1,i,k}$, then $\bar{R}_{i,k} = \bar{Q}_{i,k}$ according to Lemma 3. Noting that $B_{i,j,k} = \mathbb{I}[R_{i,j} \geq \tau_k]$, we have $\frac{4\theta}{3\delta} \sum_j B_{i,j,k} \leq \sum_j R_{i,j}$ (since $\tau_k = \frac{4\theta}{3\delta}$). Therefore, $\frac{4\theta}{3\delta} \hat{\delta}_{i,k} \leq \bar{R}_{i,k} = \bar{Q}_{i,k} < \theta$, so $\hat{\delta}_{i,k} \leq \frac{3}{4}\delta$.

Applying a Chernoff bound on the b_k independent Bernoulli random variables $B_{i,j,k}$, the probability of the latter event can be bounded, giving

$$\begin{aligned}\Pr(E_{1,i,k}^c) = \Pr(\bar{Q}_{i,k} < \theta) &\le \Pr\left(\hat{\delta}_{i,k} \le \frac{3}{4}\delta\right) \\ &\le \Pr\left(\hat{\delta}_{i,k} \le \frac{3}{4}\mathbb{E}(\hat{\delta}_{i,k})\right) \le \exp\left(-\frac{\mathbb{E}(\hat{\delta}_{i,k})b_k}{32}\right) \\ &< \exp\left(-\frac{1}{32}\delta b_k\right) \le \frac{\zeta}{2nk(k+1)},\end{aligned}$$

where the second and second to last inequalities follow from $\mathbb{E}(\hat{\delta}_{i,k}) > \delta$. □

Lemma 6. $\Pr(E_{2,i,k}^c) \le \frac{\zeta}{2nk(k+1)}$.

Proof. The samples $(R_{i,j,k})_j$ are independent and identically distributed with mean $\bar{R}_{i,k}$ and expectation $R_{\tau_k}(i)$. Thus, the lemma holds by the empirical Bernstein bound (cf. Audibert et al., 2009, Theorem 1 and Appendix A). □

Now Lemma 4 follows from Lemmas 5 and 6 and the union bound (details are given in Appendix B).

4.4. Bounding the average runtime

Note that when the algorithm finishes, $\bar{Q}_{i,k} = \bar{R}_{i,k}$. Hence, in this section we focus on $\bar{R}_{i,k}$ and its deviation from its mean. In particular, we show that $|R_{\tau_k}(i) - \bar{R}_{i,k}| \le \frac{3}{7}\varepsilon R_{\tau_k}(i)$ holds on event E when phase k is *preterm*. Here, a phase k is called preterm if $\min_i R_{\tau_k}(i) \ge \frac{7}{16}\theta_k$. The idea is that if a phase is preterm then the best capped expected runtime is large compared to the guess on the optimal runtime. We then show that on E, any phase executed by the algorithm is preterm.

Since on E, $|R_{\tau_k}(i) - \bar{R}_{i,k}| \le C_{i,k}$ by the definition of E, we need to bound $C_{i,k}$. We start with a bound on the empirical variance $\hat{\sigma}_{i,k}^2$.

Lemma 7. *For any preterm phase k, $\hat{\sigma}_{i,k}^2 \le \frac{32}{21\delta}(\bar{R}_{i,k} + R_{\tau_k}(i))^2$.*

Proof. First we show that for any $c > 0$,

$$\hat{\sigma}_{i,k}^2 \le c\left(\bar{R}_{i,k} + \frac{\tau_k}{2c}\right)^2 . \tag{1}$$

Notice that $\hat{\sigma}_{i,k}^2 = \frac{1}{b_k}\sum_{j=1}^{b_k}(R_{i,j,k} - \bar{R}_{i,k})^2 \le \frac{1}{b_k}\sum_{j=1}^{b_k} R_{i,j,k}^2 \le \frac{1}{b_k}\sum_{j=1}^{b_k}\tau_k R_{i,j,k} = \tau_k \bar{R}_{i,k}$ because $\bar{R}_{i,k} \le \tau_k$ by definition. Now (1) follows from the obvious $\bar{R}_{i,k}\tau_k \le c\left(\bar{R}_{i,k} + \frac{\tau_k}{2c}\right)^2$.

By the assumption on k, $\theta_k \le \frac{16}{7}R_{\tau_k}(i)$. Since $\tau_k = \frac{4\theta_k}{3\delta}$, this means that $\tau_k \le \frac{64}{21\delta}R_{\tau_k}(i)$. Thus, applying (1) with $c = \frac{32}{21\delta}$ completes the proof as $\hat{\sigma}_{i,k}^2 \le \frac{32}{21\delta}\left(\bar{R}_{i,k} + \frac{21\delta}{64}\tau_k\right)^2 \le \frac{32}{21\delta}\left(\bar{R}_{i,k} + R_{\tau_k}(i)\right)^2$. □

Combining the above result with the upper bound $\tau_k = \frac{4\theta_k}{3\delta} \le \frac{64}{21\delta}R_{\tau_k}(i)$, which holds for any preterm phase k, simple algebra yields the following bound on $C_{i,k}$ (the full proof is given in Appendix C):[6]

Lemma 8. *For any preterm phase k, it holds that $C_{i,k} \le \frac{\varepsilon}{3}(\bar{R}_{i,k} + R_{\tau_k}(i))$.*

Now we give the promised bound on $|R_{\tau_k}(i) - \bar{R}_{i,k}|$.

Lemma 9. *Assume E holds and $C_{i,k} \le \frac{\varepsilon}{3}(\bar{R}_{i,k} + R_{\tau_k}(i))$. Then, $|R_{\tau_k}(i) - \bar{R}_{i,k}| \le \frac{3}{7}\varepsilon R_{\tau_k}(i)$ for all configurations i.*

Proof. Let us define x such that $\bar{R}_{i,k} = (1+x)R_{\tau_k}(i)$. Because $E_{2,i,k}$ holds, $|x|R_{\tau_k}(i) = |R_{\tau_k}(i) - \bar{R}_{i,k}| \le C_{i,k} \le \frac{\varepsilon}{3}(\bar{R}_{i,k} + R_{\tau_k}(i)) = \frac{\varepsilon}{3}(1+2x)R_{\tau_k}(i)$. So $|x| \le \frac{\varepsilon}{3}(1+2x)$. If $x < 0$, then $x \ge -\frac{\varepsilon/3}{1+2\varepsilon/3} > -\frac{3}{7}\varepsilon$. If $x \ge 0$, then $x \le \frac{\varepsilon/3}{1-2\varepsilon/3} \le \frac{3}{7}\varepsilon$ because $\varepsilon \le \frac{1}{3}$. □

In the analysis of the correctness and the running time of the algorithm, we only need the slightly weaker corollary of Lemma 8 and Lemma 9 (which also holds for another variant of our algorithm, as opposed to Lemma 8):

Corollary 10. *Assume E holds and phase k is preterm. Then, for each i, if $\bar{R}_{i,k} < \theta_k$, then $|R_{\tau_k}(i) - \bar{R}_{i,k}| \le \frac{3}{7}\varepsilon R_{\tau_k}(i)$; otherwise, if $\bar{R}_{i,k} \ge \theta_k$, then $\theta_k < (1+\frac{3}{7}\varepsilon)R_{\tau_k}(i)$.*

4.5. Correctness and runtime

In this section we show that Algortihm 1 returns an (ε, δ)-optimal configuration, and give an upper bound on its running time. First we show the following result promised earlier:

Lemma 11. *If E holds then every phase k executed is preterm.*

Proof. The first phase is preterm as $\frac{7}{16}\theta_1 = \kappa_0 \le R_{\tau_1}(i)$. For a phase $k \ge 2$ that is executed, since the algorithm did not return in phase $k-1$, by Lemma 3, $\bar{R}_{i,k-1} \ge \bar{Q}_{i,k-1} = \theta_{k-1}$. If E holds and phase $k-1$ was preterm, by Corollary 10, $\theta_{k-1} < (1+\frac{3}{7}\varepsilon)R_{\tau_{k-1}}(i)$. Moreover,

$$\frac{7}{16}\theta_k = \frac{7}{8}\theta_{k-1} \le \frac{7}{8}(1+\tfrac{3}{7}\varepsilon)R_{\tau_{k-1}}(i) \le R_{\tau_{k-1}}(i) \le R_{\tau_k}(i),$$

since $\varepsilon \le \frac{1}{3}$. By induction, any phase executed is preterm. □

Lemma 12. *If E holds and Algorithm 1 returns with a configuration I in phase K, then I is (ε, δ)-optimal.*

Proof. We prove the statement by contradiction. Thus, assume I is (ε, δ)-suboptimal. At stopping, $\bar{Q}_{I,K} < \theta_K$, hence on E, $p_{I,K} = \Pr_{J\sim\Gamma}(R(I,J) > \tau_K) \le \delta$ must hold.

[6] The multiplicative constant in the proof is not optimized carefully to promote simplicity. Nevertheless, in our experiments the empirical effect of this constant is negligible.

Since I is (ε, δ)-suboptimal, it follows that there exists an instance j such that $R_{\tau_K}(I) > (1+\varepsilon)R(j) > (1+\varepsilon)R_{\tau_K}(j)$. Take such an index j. Since Algorithm 1 returned I instead of j, $\bar{Q}_{I,K} \leq \bar{Q}_{j,K}$. By Lemma 3, $\theta_k > \bar{Q}_{I,K} = \bar{R}_{I,K}$ and $\bar{R}_{j,K} \geq \bar{Q}_{j,K}$. Applying Corollary 10 and Lemma 11, if $\bar{R}_{j,K} < \theta_k$, then $(1+\frac{3}{7}\varepsilon)R_{\tau_K}(j) \geq \bar{R}_{j,K} \geq \bar{Q}_{j,K} \geq \bar{Q}_{I,K}$. Otherwise, $(1+\frac{3}{7}\varepsilon)R_{\tau_k}(i) \geq \theta_k \geq \bar{Q}_{I,K}$. Using this,

$$R_{\tau_K}(j)(1+\tfrac{3}{7}\varepsilon) \geq \bar{Q}_{I,K} = \bar{R}_{I,K} > R_{\tau_K}(I)(1-\tfrac{3}{7}\varepsilon) > R_{\tau_K}(j)(1+\varepsilon)(1-\tfrac{3}{7}\varepsilon).$$

Therefore, $1+\frac{3}{7}\varepsilon > (1+\varepsilon)(1-\frac{3}{7}\varepsilon)$ which leads to a contradiction since $\varepsilon \leq \frac{1}{3}$. □

Theorem 13. *Algorithm 1 identifies an (ε, δ)-optimal solution in time $\mathcal{O}\left(\mathrm{OPT}\frac{n}{\varepsilon^2\delta}\log(\frac{n\log \mathrm{OPT}}{\zeta})\right)$ with probability at least $1-\zeta$, where $\mathrm{OPT} = \min_i R(i)$.*

Proof. By Lemma 4, E holds with probability at least $1-\zeta$. The rest of the proof assumes that E holds.

Let $i^* = \operatorname{argmin}_i R(i)$. If $\theta_k \geq (1+\frac{3}{7}\varepsilon)\mathrm{OPT} \geq (1+\frac{3}{7}\varepsilon)R_{\tau_k}(i^*)$, then only the first case of Corollary 10 can hold. Together with Lemma 11, we have that $\bar{Q}_{i^*,k} \leq \bar{R}_{i^*,k} \leq (1+\frac{3}{7}\varepsilon)R_{\tau_k}(i^*) \leq \theta_k$, so Algorithm 1 terminates for $\theta_k \geq (1+\frac{3}{7}\varepsilon)\mathrm{OPT}$. Let the total number of phases of the outer loop of Algorithm 1 be L. Then $L = \mathcal{O}(\log \mathrm{OPT})$.

The for loop on Line 7 of Algorithm 1 adds a factor of n to the runtime. By Lemma 2, calling algorithm RUNTIMEEST on Line 8 adds a factor of $b_k\theta_k$ to the runtime. Now $b_k \leq \left\lceil\left(44\log(\frac{6nL(L+1)}{\zeta})\frac{1}{\delta\varepsilon^2}\right)\right\rceil = \mathcal{O}\left(\frac{1}{\varepsilon^2\delta}\log(\frac{6n\log^2 \mathrm{OPT}}{\zeta})\right) = \mathcal{O}\left(\frac{1}{\varepsilon^2\delta}\log(\frac{6n\log \mathrm{OPT}}{\zeta})\right)$, so substituting $\theta_k = \frac{16}{7}\kappa_0 2^k$, the total runtime becomes

$$\mathcal{O}\left(\sum_{k=1}^{\left\lceil\log_2\left((1+\frac{3}{7}\varepsilon)\frac{\mathrm{OPT}}{\kappa_0}\right)\right\rceil} \kappa_0 2^k \cdot \frac{n}{\varepsilon^2\delta}\log\left(\frac{6n\log \mathrm{OPT}}{\zeta}\right)\right) = \mathcal{O}\left(\mathrm{OPT}\frac{n}{\varepsilon^2\delta}\log\left(\frac{n\log \mathrm{OPT}}{\zeta}\right)\right).$$

By Lemma 12, when the algorithm returns, it returns with an (ε, δ)-optimal configuration. □

5. Optimizing RUNTIMEEST

Our runtime analysis presented in the previous section used a worst-case upper bound for $\hat{\sigma}_{i,k}$. Some instances may allow faster runtimes if we modify RUNTIMEEST to stop earlier in scenarios where the empirical variance is lower than this worst case bound. To do this, building on the approach of Mnih et al. (2008), we change the stopping rules of algorithm RUNTIMEEST and add two more rules as given in Algorithm 3. The code shown here should replace Lines 8–12 of RUNTIMEEST.

Algorithm 3 Stopping rules

$\bar{Q} \leftarrow \frac{1}{j}\sum_{m=1}^{j} Q_m$
$\hat{\sigma}^2 \leftarrow \frac{1}{j}\sum_{m=1}^{j}\left(Q_m - \bar{Q}\right)^2$
$d_{j,k} \leftarrow 4nk(k+1)j(j+1)/\zeta$
$c \leftarrow \sqrt{\hat{\sigma}^2\frac{2\log(3d_{j,k})}{j}} + \frac{3\tau\log(3d_{j,k})}{j}$
LB $\leftarrow \bar{Q} - c$
if $T = 0$ **then** ▷ Stop if overall budget zero
  return θ
end if
if $j = b$ **then** ▷ Stop after $b = |J|$ samples
  return $\bar{Q}$ ▷ Return mean of Q
end if
if $(1+\frac{3}{7}\varepsilon)\mathrm{LB} \geq \theta$ and $\bar{Q} > \theta$ **then** ▷ LB too large
  return θ
end if
if $j \geq \left\lceil\frac{32}{\delta}\log d_{j,k}\right\rceil$ and $c \leq \frac{\varepsilon}{3}\left(\bar{Q} + \mathrm{LB}\right)$ **then**
  return $\bar{Q}$ ▷ Return mean of Q
end if

We outline a proof sketch that this algorithm is still correct and has the same runtime bound. We define the running averages in iteration j of RUNTIMEEST as $\bar{Q}_{i,j,k}$ and $\bar{R}_{i,j,k}$. As before, we define an event, as a union of other events, that guarantees that the empirical estimates behave well. We keep the previously defined events $E_{1,i,k}$ and $E_{2,i,k}$; note that $E_{1,i,k}$ corresponds to the estimate $\bar{Q}_{i,b,k}$. However, we need a similar event to $E_{1,i,k}$ for all iterations j: $E'_{i,j,k} = \{\bar{Q}_{i,k} \geq \theta_k\} \cup \{p_{i,k} \leq \delta\} \cup \{j < \left\lceil\frac{32}{\delta}\log d_{j,k}\right\rceil\}$.

Let $E = \bigcap_{i\in\{1,\ldots,n\},j,k\in\mathbb{Z}_+}\left(E_{1,i,k} \cap E_{2,i,k} \cap E'_{i,j,k}\right)$. Similarly to the previous section, it is easy to show that $\Pr(E) \geq 1 - 3\zeta/2$. If E holds and the algorithm returns with an average runtime less than θ_k, then $E_{1,i,k}$ and $E'_{i,j,k}$ guarantee that $\Pr(R(i,j) > \tau_k) \leq \delta$ (independently of which stopping condition was activated). Since the original stopping rule is still in place, the runtime of algorithm RUNTIMEEST with the additional stopping rules is still $\mathcal{O}(b_k\theta_k)$. Similarly, it is easy to verify that Lemma 3 still holds.

Furthermore, by a slight modification of Theorem 2 of Mnih (2008), one can show that with probability at least $1-\zeta/2$, $|R_{\tau_k}(i) - \bar{R}_{i,j,k}| \leq c_{i,j,k}$ holds for all i, j, k, and $c_{i,j,k} \leq \frac{\varepsilon}{3}\left(\bar{Q}_{i,j,k} + \mathrm{LB}_{i,j,k}\right) \leq \frac{\varepsilon}{3}\left(\bar{R}_{i,j,k} + R_{\tau_k}(i)\right)$ holds[7] for all

$$j \geq C\cdot\max\left(\frac{\sigma^2_{i,k}}{\varepsilon^2 R^2_{\tau_k}(i)}, \frac{\tau_k}{\varepsilon R_{\tau_k}(i)}\right)\left(\log\frac{1}{\zeta'} + \log\frac{1}{\varepsilon R_{\tau_k}(i)}\right),$$

where C is a universal constant, and $\bar{Q}_{i,j,k} \leq \bar{R}_{i,j,k}$. Denote

[7] Here, Lemma 3 was used additionally in the last inequality.

this event by E'; then $\Pr(E') \geq 1 - \zeta/2$.[8]

Applying Lemma 9, E' also implies that

$$|R_{\tau_k}(i) - \bar{R}_{i,j,k}| \leq \tfrac{3}{7}\varepsilon R_{\tau_k}(i).$$

Thus, if $E \cup E'$ holds, then Corollary 10 holds: if Algorithm 3 returns either because it went through all the b_k samples or because of Line 12, then $|R_{\tau_k}(i) - \bar{R}_{i,k}| \leq \frac{3}{7}\varepsilon R_{\tau_k}(i)$, which implies the first part of the corollary; otherwise Algorithm 3 returns in line 10, implying that the algorithm returns with θ_k and $(1 + \frac{3}{7}\varepsilon)R_{\tau_k} > \theta_k$. Then the runtime bound and the correctness guarantee of Theorem 13 follows as before.

On the other hand, if the variances of the runtimes over instances are low enough, it is possible to prove an improved runtime bound for the whole algorithm. For $\zeta' = \frac{\zeta}{4nk(k+1)}$, there exists a constant C such that if $j \geq C \cdot \frac{1}{\delta}\left(\log\frac{1}{\delta} + \log\frac{1}{\zeta'}\right)$, then $j \geq \left\lceil \frac{32}{\delta}\log d_{j,k} \right\rceil$ holds. Together with the previous lower bound on j and by upper bounding $\tau_k \leq \frac{64}{21\delta}R_{\tau_k}(i)$, by the definition of a preterm phase (see Lemma 8), with probability at least $1 - 2\zeta$, RUNTIMEEST evaluates at most

$$C \cdot \max\left(\frac{\sigma_{i,k}^2}{\varepsilon^2 R_{\tau_k}^2(i)}, \frac{1}{\varepsilon\delta}, \frac{1}{\delta}\log\frac{1}{\delta}\right)\left(\log\frac{1}{\zeta'} + \log\frac{1}{\varepsilon R_{\tau_k}(i)}\right)$$

samples in any phase k for configuration i before the stopping conditions on Line 15 are be satisfied. This bound is usually much lower than the previous $b_k = \left\lceil 44\log\left(\frac{6nk(k+1)}{\zeta}\right)\frac{1}{\delta\varepsilon^2}\right\rceil$: if the variance of runtimes is sufficiently low, this scales as ε^{-1} rather than ε^{-2} (the $\delta^{-1}\log\delta^{-1}$ term is negligible).

Mnih et al. (2008) also describe EBGStop, a slightly improved version of empirical Bernstein stopping, which applies Bernstein inequalities to bound the means of an exponentially increasing number of samples. This allows us to effectively replace $\log\frac{1}{\varepsilon R_{\tau_k}(i)}$ with $\log\log\frac{1}{\varepsilon R_{\tau_k}(i)}$ in the bound presented above. We use this version of the algorithm in our experiments. For completeness, the pseudocode of this version is given in Appendix D.

6. Experiments

To run experiments, we gathered a benchmark set of runtimes of different configurations on generated SAT problems. We used `minisat`[9] (Sorensson & Een, 2005) as the SAT solver. The SAT problems were generated using CNFuzzDD,[10] of which only those 20118 were kept that took at least about a second to solve for `minisat` with the default parameters. This was done so that the data reflects what happens when instances are nontrivial to solve. 972 different configurations were tested for `minisat`, which are described in Appendix E. The solver `minisat` was run with each configuration and instance combination. The unit of computation, κ_0, is one second of CPU time, and the experiments were ran with a timeout of 15 CPU minutes.[11] To get a sense of this data, the capped mean runtimes $R_\tau(i)$ for each configuration are shown in Fig. 1 in a sorted order. Here, the timeout τ was set separately for each configuration so that the tail probability $\Pr_{J\sim\Gamma}(R(i,J) > \tau)$ was approximately δ; the running times are shown for different values of δ ($\delta = 0$ corresponds to the mean runtimes). From this, we can see a large difference between configurations. For a particularly "fast" configuration, Fig. 2 shows the distribution of runtimes on different instances. Note that because of the global time limit for executions, the final bucket includes runs that may take arbitrarily long.

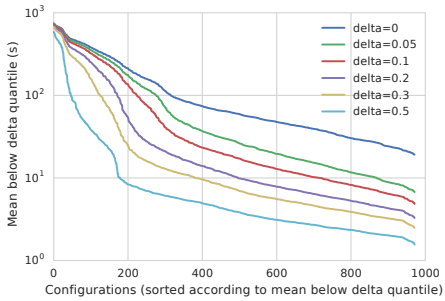


Figure 1. Average of runtimes, capped at the timeout, disregarding the worst δ fraction of samples. Configurations on the x-axis are sorted according to this value. Note the log scale on the y-axis.

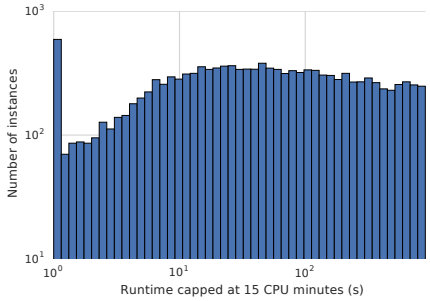


Figure 2. Histogram of runtimes of configuration 898 over the instances in our data. Note the log scale on the x-axis.

The benchmark set of runtimes is used to quickly simulate runs of Structured Procrastination and LEAPSANDBOUNDS, as follows. A simulated environment acts as an oracle,

[8] The original event behind E' guarantees, via Bernstein's inequality, that the estimates for the means and variances are accurate enough.

[9] We used version 2013/09/25. http://minisat.se/

[10] http://fmv.jku.at/cnfuzzdd/

[11] Our measurements have been scaled such that the unit of computation roughly corresponds to a second on commodity hardware as of 2018 rather than our machines. In this unit, about 83 CPU years were spent in total to generate this data.

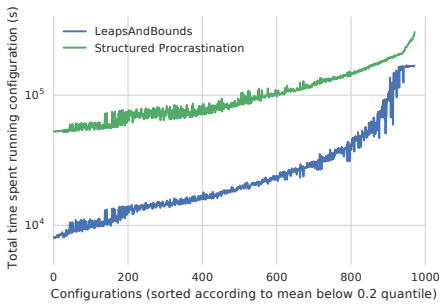


Figure 3. Amount of time LEAPSANDBOUNDS runs each configuration compared to Structural Procrastination on a log scale, in an environment that allows resuming runs.

returning precomputed values of $R(i, j, \tau)$ when queried, accumulating the total time the algorithm under test would have run for.

Both LEAPSANDBOUNDS and Structured Procrastination often run the same configuration on the same instance with an increased time limit. Thus, both algorithms can benefit largely when the environment allows pausing and resuming of executions. This can be implemented either by saving the state of the execution when the actual runtime limit is reached, or by reloading the state from automatically saved checkpoints. However, resuming execution comes with an additional memory requirement, and may not always be feasible or preferable to restarts. Thus, we report our experiments for both cases.

After each phase, in Line 13 of LEAPSANDBOUNDS, we double θ. In fact, this multiplier is arbitrary, and changing it only affects the worst-case runtime up to a constant factor. In practice, a smaller multiplier, making smaller steps in θ, typically overshoots the best average runtime less, thereby decreasing the total runtime for environments that allow resuming runs. On the other hand, a smaller multiplier leads to more phases, introducing more overheads in resuming jobs and increasing the total runtime if resuming is not allowed by rerunning portions of jobs more frequently. The value of the multiplier can be optimized by taking these effects into account, e.g., by measuring the overheads related to switching and resuming jobs. For simplicity, and since this information is not included in our benchmark dataset, in the experiments below the value of the multiplier was set to 1.25 (see Appendix F for more details).

We simulated LEAPSANDBOUNDS and Structured Procrastination on our benchmark dataset with parameters $\varepsilon = 0.2$, $\delta = 0.2$, and $\zeta = 0.1$. Fig. 3 shows that LEAPSANDBOUNDS runs every configuration for a significantly shorter amount of time than Structured Procrastination. The configurations are sorted in the same order as in Fig. 1, for $\delta = 0.2$. Paradoxically, both algorithms run the faster configurations significantly longer. This is because both algorithms quickly reject slow configurations, whereas they both run fast configurations many more times to ensure (ε, δ)-optimality. In total, LEAPSANDBOUNDS runs for 933.50 CPU days in the environment that does not support resuming execution, and 368.50 days in one that does. The corresponding runtime measurements for Structured Procrastination are 1850.46 and 1169.36, respectively. Both algorithms return with configuration 898, which has the best average runtime below a $\delta = 0.2$ quantile, out of all configurations.

7. Parallelization

One benefit of the simplicity of LEAPSANDBOUNDS is that it is embarrassingly parallel. This is due to the fact that there is little dependency between the runtime measurements that need to be carried out. In phase k, when $\theta_k = \frac{16}{7}\kappa_0 2^k$, runs of the form $R\left(i, j, \frac{\kappa_0 2^{k+6}}{21\delta}\right)$ are carried out. The core of our argument is that this parallelizes over i, j, and k, but there are three further considerations. First, to implement the overall runtime bound of RUNTIMEEST, for any fixed i and k, the runs of $R\left(i, j, \frac{\kappa_0 2^{k+6}}{21\delta}\right)$ should be terminated once the summed running times of these reach the overall budget $b_k\theta_k$. This could be implemented either via interprocess communication or by starting these runs at once on p processors and terminating them after $b_k\theta_k/p$ time. Second, a new phase k of Algorithm 1 should only be started once $\bar{Q}_{i,k}$ is available for all $i \in \mathcal{N}$. Thus, runs should be started in increasing order of k, for each i. Third, the optional empirical Bernstein stopping, as described in Section 5, adds a dependency between runs of different j. This could be resolved either by not parallelizing over j, or by running only a small number of parallel runs over j and checking the stopping conditions after they finish.

8. Conclusions and Future Work

We have introduced an algorithm applying empirical Bernstein stopping with the goal of finding approximately optimal configurations, and provided guarantees for its worst-case runtime as well as correctness. Our runtime guarantee is tighter than that of Structured Procrastination, which, to our knowledge, is the only other method solving this problem. Empirical evaluations suggest that LEAPSANDBOUNDS outperforms Structured Procrastination in realistic, non-adversarial scenarios too, which depends crucially on leveraging the gap between worst-case and realistic scenarios by using empirical Bernstein stopping.

The optimality of the configuration returned by LEAPSANDBOUNDS is, in fact, with respect to configurations *with timeout* τ_K for the final phase K. An important direction of future work is to get guarantees with respect to the best configuration for the fastest $(1 - \delta')$-proportion of instances for any $\delta' < \delta$.

References

Adam, K. Learning while searching for the best alternative. *Journal of Economic Theory*, 101:252–280, 2001.

Ansótegui, C., Sellmann, M., and Tierney, K. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pp. 142–157. Springer, 2009.

Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., and Tierney, K. Model-based genetic algorithms for algorithm configuration. In *International Joint Conference on Artificial Intelligence*, pp. 733–739, 2015.

Audibert, J.-Y. and Bubeck, S. Best arm identification in multi-armed bandits. In *Conference on Learning Theory*, pp. 41–53, 2010.

Audibert, J.-Y., Munos, R., and Szepesvári, C. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19): 1876–1902, 2009.

Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. A racing algorithm for configuring metaheuristics. In *Annual Conference on Genetic and Evolutionary Computation*, pp. 11–18. Morgan Kaufmann Publishers Inc., 2002.

György, A. and Kocsis, L. Efficient multi-start strategies for local search algorithms. *Journal of Artificial Intelligence Research*, 41:407–444, 2011.

Hutter, F. On the potential of automatic algorithm configuration. In *SLS-DS2007: Doctoral Symposium on Engineering Stochastic Local Search Algorithms*, pp. 36–40, 2007.

Hutter, F., H. Hoos, H., Leyton-Brown, K., and Stützle, T. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1): 267–306, 2009.

Hutter, F., H. Hoos, H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pp. 507–523. Springer, 2011.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Bayesian optimization with censored response data. *CoRR*, abs/1310.1947, 2013. URL http://arxiv.org/abs/1310.1947.

Kleinberg, R., Leyton-Brown, K., and Lucier, B. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *International Joint Conference on Artificial Intelligence*, pp. 2023–2031, 2017.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR*, abs/1603.06560, 2016. URL http://arxiv.org/abs/1603.06560.

López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. The irace package, iterated race for automatic algorithm configuration. Technical report, Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

Luby, M., Sinclair, A., and Zuckerman, D. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47:173–180, 1993.

Mnih, V. Efficient stopping rules. Master's thesis, University of Alberta, Edmonton, AB, Canada, 2008.

Mnih, V., Szepesvári, C., and Audibert, J.-Y. Empirical Bernstein stopping. In *International Conference on Machine Learning*, pp. 672–679. ACM, 2008.

Sorensson, N. and Een, N. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005(53):1–2, 2005.