ImpatientCapsAndRuns: Approximately Optimal Algorithm Configuration from an Infinite Pool

Anonymous Author(s)
Affiliation
Address
email

Abstract

Algorithm configuration procedures optimize parameters of a given algorithm to perform well over a distribution of inputs. Recent theoretical work focused on the case of selecting between a small number of alternatives. In practice, parameter spaces are often very large or infinite, and so successful heuristic procedures discard parameters "impatiently", based on very few observations. Inspired by this idea, we introduce IMPATIENTCAPSANDRUNS, which quickly discards less promising configurations, significantly speeding up the search procedure compared to previous algorithms with theoretical guarantees, while still achieving optimal runtime up to logarithmic factors under mild assumptions. Experimental results demonstrate a practical improvement.

1 Introduction

Solvers for computationally hard problems (e.g., SAT, MIP) often expose many parameters that only affect runtime rather than solution quality. Choosing values for these parameters is seldom easy or intuitive, and different settings can lead to drastically different runtimes—days versus seconds—for a given input instance. Such parameters are exposed in the first place because they do not have known, globally optimal settings, instead typically expressing tradeoffs between different heuristic mechanisms or implicit assumptions about problem structure. In practice, solver end-users typically need to repeatedly solve similar problems: e.g., integer programs modeling airline crew scheduling problems; or SAT formulae used to formally verify a sequence of related hardware or software designs. This gives rise to the problem of *algorithm configuration*: finding a joint setting of parameters for a given algorithm so that it performs well on input instances drawn from a given distribution. We make no restrictions on the space of possible parameters or its structure: they may be continuous, categorical, subject to arbitrary constraints, and may contain jump discontinuities. We refer to a joint setting of all the algorithm's parameters as a *configuration* to stress this generality. A common metric of performance for a configuration, and the one we consider in this work, is mean runtime: we prefer configurations that are faster, on average, on the problems we care about solving. An algorithm configuration method can sample instances from the distribution underlying an application and can run any configuration (possibly also sampled from the set possible configurations) on any sampled instance until a timeout of its choice, and the goal is to find a configuration with nearly optimal mean runtime while using the least amount of time during the search.[1]

Heuristic methods for algorithm configuration such as ParamILS [17, 18], GGA [2, 3], irace [11, 28] and SMAC [20, 21] have been used with great success for more than a decade, but they do not come with any rigorous performance guarantees. More recently, algorithm configuration has also

[1]As usual, we treat the cumulative runtime of all the configurations tried as the total search time. One could also consider including the overhead imposed by the configuration algorithm itself. However, beyond being difficult to model, this cost is typically negligible compared to the runtime of the configurations.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

been considered from a theoretical perspective. Kleinberg et al. [23] introduced a framework to analyze algorithm configuration methods theoretically, and presented the first configuration procedure, STRUCTURED PROCRASTINATION (SP), which is guaranteed to find an approximately optimal solution with a non-trivial worst-case runtime bound. Since then algorithms with better theoretical guarantees have been developed [34, 35, 24]. Overall, these theoretically-motivated configuration procedures have nice properties, such as achieving near-optimal asymptotic worst-case running times. However, none of them yet achieves competitive performance on practical problem benchmarks, for two key reasons: (i) heuristic methods usually iteratively select candidate configurations that appear likely to perform well given previous samples from the configuration space (e.g., leveraging structure in the parameter space, such as smoothness or low pseudodimension [19, 29]), whereas the theoretical algorithms select configurations randomly; and (ii) heuristic methods often impatiently discard less promising configurations based on just a few runtime observations, while the theoretical algorithms are more conservative and continue evaluating them until they demonstrate, with high probability, that another configuration is better. Such early discard strategies are particularly effective when the configuration space contains one or a few configurations that drastically outperform all others. This "needle-in-a-haystack" scenario is common in practice, perhaps in part explaining the success of these heuristic methods.

In this paper we take a significant step towards theoretically grounded *and* practical algorithm configuration by addressing the second problem. We build on CAPSANDRUNS (CAR) [35], a simple and intuitive algorithm that continuously discards configurations that perform poorly relative to a global upper bound on the best achievable mean runtime. Here we introduce IMPATIENTCAPSANDRUNS (ICAR), which equips CAR with the ability to quickly discard less-promising configurations by applying an initial "precheck" mechanism that allows poorly performing configurations to be discarded quickly. Additionally, via a more careful analysis we are able speed up a key subroutine from CAR. While ICAR retains the favorable optimality and runtime guarantees of CAR under mild assumptions, it is also provably faster in needle-in-a-haystack scenarios where most configurations are considerably weaker than the best ones (these are the cases where good algorithm configuration procedures are the most useful, because identifying a good configuration is the most consequential.) Because of its precheck procedure, ICAR is able to examine more configurations than CAR, and hence finds configurations with better mean runtime. Furthermore, not wasting time on examining bad configurations, the total runtime of ICAR is significantly smaller than that of CAR and any other existing procedure with theoretical guarantees, making a step towards closing the performance gap relative to heuristic procedures.

Finally, we briefly survey some less closely related work. Gupta & Roughgarden [13] initiated the study of algorithm configuration from a learning-theoretic perspective. Rather than seek general purpose configuration procedures, as we do in this work, this and subsequent approaches seek to bound the number of training samples required to guarantee good generalization for specific classes of problems. Examples include combinatorial partitioning problems such as max-cut and clustering [6], branching strategies in tree search algorithms [7], and general algorithm configuration when the runtime is piecewise-constant over its parameter space [8]. Hyperparameter-search methods based on multi-armed bandit algorithms are also related. The main difference is that this literature focuses on settings where every configuration run costs the same amount or where there is a tradeoff between how long each configuration is run and the accuracy with which its performance is estimated [5, 27]; thus, these methods do not face questions like how many instances to consider and how to cap runs.

The rest of the paper is organized as follows. The formal model of algorithm configuration is given in Section 2. The ICAR algorithm is presented and analyzed in Section 3. Experiments on some algorithm configuration benchmarks are given in Section 4. Proofs and additional experimental results are deferred to the appendix.

2 The Model

Following Kleinberg et al. [23], the algorithm configuration problem is defined by a triplet (Π, Γ, R), where Π is a distribution over possible configurations, Γ is a distribution over input instances, and $R(i, j)$ is the runtime of a configuration i on a problem instance j. For example Π and Γ may simply be uniform distributions, respectively over the space of hyperparameters and the set of past problem instances seen. The mean runtime of a configuration i is defined as $R(i) = \mathbb{E}_{j \sim \Gamma}[R(i, j)]$, and the ultimate goal of an algorithm configuration method is to find a configuration i minimizing $R(i)$.

During this search the configuration method needs to explore new configurations, which can be sampled from Π.[2] The configuration method can also sample problem instances from Γ and run a configuration i on an instance j until it finishes, or the execution time exceeds a specified timeout $\tau \geq 0$. The use of such a timeout allows for a tradeoff between learning more about the runtime of a single configuration–instance pair and considering a larger number of such pairs.

To this end, for any configuration i we consider the τ-capped expected runtime $R_\tau(i) = \mathbb{E}_{j\sim\Gamma}[\min\{R(i,j),\tau\}]$. Furthermore, for any $\delta \in (0,1)$, let $t_\delta(i) = \inf_t\{t : \Pr_{j\sim\Gamma}(R(i,j) > t) \leq \delta\}$ denote the δ-quantile of i's runtime, and define $R^\delta(i) = R_{t_\delta(i)}(i)$ the δ-capped expected runtime of i.[3] That is, $R^\delta(i)$ is the mean runtime of i if we cap the slowest δ-fraction of its runtimes.

Since a globally optimal configuration may be arbitrarily hard to find, we instead seek a solution that is competitive with the performance of the top γ-fraction of the configurations for a $\gamma \in (0,1)$. That is, instead of finding a configuration close to $\mathrm{OPT} = \min_i\{R(i)\}$, we search for one close to $\mathrm{OPT}^\gamma = \inf_{x\in\mathbb{R}^+}\{x : \Pr_{i\sim\Pi}(R(i) \leq x) \geq \gamma\}$. Additionally, since the average runtime of any configuration, including the optimal one, could be totally dominated by a few incredibly unlikely but arbitrarily large runtime values, we seek solutions whose expected δ-capped runtime is close to the δ-capped optimum. However, it turns out that this relaxed property is still impossible to verify [34]. Following Weisz et al. [34], we address this by adding a small amount of slack to the benchmark, comparing to the $(\delta/2)$-capped optimum rather than the δ-capped optimum. Putting this together, we seek solutions whose expected δ-capped runtime is close to the $(\delta/2)$-capped optimum, after excluding the best γ-fraction of configurations: $\mathrm{OPT}^\gamma_{\delta/2} = \inf_{x\in\mathbb{R}^+}\left\{x : \Pr_{i\sim\Pi}[R^{\frac{\delta}{2}}(i) \leq x] \geq \gamma\right\}$.

Definition 1 ($(\varepsilon,\delta,\gamma)$-optimality)**.** *A configuration i is $(\varepsilon,\delta,\gamma)$-optimal if* $R^\delta(i) \leq (1+\varepsilon)\mathrm{OPT}^\gamma_{\delta/2}$.

This definition generalizes the notion of (ε,δ)-optimality of Weisz et al. [35] for a finite set of configurations, where instead of the top-γ portion, we aim to achieve the performance of the best configuration (up to ε): for a finite set of N configurations, configuration i is (ε,δ)-optimal if it is $(\varepsilon,\delta,1/N)$-optimal when Π is the uniform distribution over the N configurations.

3 The Algorithm

Recent theoretically-sound algorithm configuration procedures make several runtime measurements for every configuration in a finite pool $\mathcal{N}$, and stop when they can confirm, with high probability, that one configuration is close enough to the best one. The main challenge is to avoid wasting time on (a) hard input instances with large runtimes; and (b) bad configurations that will be eliminated later. To this end, STRUCTURED PROCRASTINATION (SP) [23] and its improved version STRUCTURED PROCRASTINATION WITH CONFIDENCE (SPC) [24] gradually increase the runtime cap for every configuration-instance pair, while carefully determining an order to evaluate these pairs, depending on the configurations' empirical average runtime (SP) or empirical confidence bounds on the mean runtimes (SPC). LEAPSANDBOUNDS (LAB) [34], which introduced empirical confidence bounds to the algorithm configuration problem, works with a much simpler schedule, and tests all configurations for a given time budget, which is increased gradually.

On the other hand, CAPSANDRUNS (CAR) [35] first measures the runtime cap for each configuration guaranteeing that at least a $(1-\delta)$-portion of the instances can be solved within that cap, then runs a racing algorithm (based on continuously recomputing confidence bounds on the mean runtimes) to select which capped configuration is the best. During the race, all configurations are run in parallel on more and more problem instances, and their mean runtime is continuously estimated. This makes it possible to maintain a high-probability upper bound T on the optimal capped runtime, and any configuration with a runtime lower bound above T can be eliminated. The algorithm stops when it can prove that a configuration is (ε,δ)-optimal.

To apply any of the above methods to an infinite pool of configurations, one can simply select a pool of $\left\lceil \frac{\log(\zeta)}{\log(1-\gamma)} \right\rceil$ configurations randomly from Π to ensure that with probability at least $1-\zeta$ it contains a configuration that belongs to the top γ-fraction of all the configurations. Thus the above

[2]We can see Π as reflecting beliefs about the distribution of good configurations in the parameter space. This implicitly neglects any search procedure that leverages structural assumptions about the parameter space.

[3]With a slight abuse of terminology, throughout we use the same expression for capping with timeouts (τ) and quantiles (δ), when the interpretation is clear from the context; we specify the type of capping otherwise.

methods can select $(\varepsilon, \delta, \gamma)$-optimal configurations from an infinite pool, with attractive theoretical guarantees. Our focus in this paper is on extending CAR, due to its conceptual simplicity and good practical performance. However, in contrast to LAB and SPC, which try to assign little runtime to bad configurations from the very beginning, at the start CAR spends the same amount of time testing all configurations. This is because the estimation of the runtime caps is done in parallel, so every configuration is run for an equally long time until the first cap is found for any configuration (only after this can the algorithm start eliminating configurations with large mean runtimes). As a result, CAR spends more time testing the worst configurations than LAB or SPC.

IMPATIENTCAPSANDRUNS (ICAR) addresses this problem, introducing a "precheck" mechanism to ensure that bad configurations are eliminated early. The PRECHECK function estimates the mean capped runtime (up to a constant multiplicative factor) needed by a configuration to solve at least a constant fraction of the problem instances (less than $1 - \delta/2$). If this capped runtime is large compared to the upper bound T on the $(\varepsilon, \delta, \gamma)$-optimal runtime (maintained similarly as in CAR), the configuration is rejected and eliminated from further analysis. This procedure is very similar to the CAR algorithm (with some fixed, constant ε and δ); only the specific rejection conditions differ mildly. Note that the runtime estimated by PRECHECK is a lower bound to the $\delta/2$-capped runtime, ensuring that good configurations are unlikely to be rejected. The efficiency of PRECHECK crucially depends on the quality of the bound T on the optimal runtime. Therefore, similarly to SPC, ICAR gradually introduces more and more configurations in batches $\mathcal{N}_k, k = K-1, \ldots, 0$: if a configuration passes PRECHECK, a (rough) estimate of its capped runtime is calculated (up to a multiplicative constant, for a cap slightly larger than the δ quantile), again by first measuring the runtime cap, then estimating the mean runtime using the measured cap. This runtime estimate is then used to reduce the bound T, which improves the performance of PRECHECK for the next batch of configurations, $\mathcal{N}_{k-1}$. The size of batch $\mathcal{N}_k$ is of order $1/\gamma_k$ with $\gamma_k = 2^k\gamma$, ensuring that with high probability it contains an $(\varepsilon, \delta, \gamma_k)$-optimal configuration (whose mean runtime is then bounded by $\mathrm{OPT}^{\gamma_k}_{\delta/2}$). As a consequence, after batch $\mathcal{N}_k$, T is at most $2\mathrm{OPT}^{\gamma_k}_{\delta/2}$, gradually reducing towards $2\mathrm{OPT}^{\gamma}_{\delta/2}$. Finally, the racing part of CAR is run over all surviving configurations, further reducing T towards $\mathrm{OPT}^{\gamma}_{\delta/2}$, and stopping when an $(\varepsilon, \delta, \gamma)$-optimal configuration is found.

Now we are ready to present the main theoretical result of the paper, a performance guarantee for ICAR. The components of the algorithm are presented in Algorithms 1–5. We then discuss each and present a proof sketch for the theorem (the detailed proof is given in Appendix A).

Theorem 1. *For input parameters $\varepsilon \in (0, 1/3), \delta \in (0, 0.2), \gamma \in (0, 1)$, integer $K \geq 1$, and failure parameter $\zeta \in (0, 1/12)$, with probability at least $1 - 12\zeta$,* IMPATIENTCAPSANDRUNS *finds an $(\varepsilon, \delta, \gamma)$-optimal configuration with total work[4] bounded by[5]*

$$\tilde{\mathcal{O}}\left(\frac{\mathrm{OPT}^{\gamma}_{\delta/2}}{\varepsilon^2\delta\gamma} \cdot F(38\mathrm{OPT}^{\gamma}_{\delta/2}) + \sum_{k=0}^{K-2} \frac{\mathrm{OPT}^{\gamma_k}_{\delta/2}}{\gamma_k}\left(1 + \frac{F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})}{\delta}\right) + \frac{\mathrm{OPT}^{\gamma_{K-1}}_{\delta/2}}{\delta\gamma_{K-1}}\right), \quad (1)$$

where $\gamma_k = 2^k\gamma$, and $F(x) = \Pr_{i\sim\Pi}(R^{0.35}(i) \leq x) + 4\zeta/K$.

Discussion. *(i)* To illustrate the advantages captured by the theorem, consider a situation where configuration runtimes are distributed exponentially, with their mean distributed uniformly over an interval $[A, A+B]$. When the number of near-optimal configurations is small (i.e., B/A is large enough), the bound on the fraction of configurations surviving PRECHECK, $F(38\mathrm{OPT}^{\gamma}_{\delta/2})$, roughly scales with γ, resulting in a runtime $\mathrm{OPT}^{\gamma}_{\delta/2}/(\varepsilon^2\delta)$, providing a γ-factor speedup over typical bounds in other work (which scale with $\mathrm{OPT}^{\gamma}_{\delta/2}/(\varepsilon^2\delta\gamma)$). (Details are given in Appendix B.)

(ii) The first term in the bound corresponds to the work done in the final racing part of ICAR. The other terms correspond to the work done for each batch $\mathcal{N}_k$ (except that the cost of the last precheck is included in the $k = 0$ term).

(iii) Kleinberg et al. [23] showed that to find an (ε, δ)-optimal configuration out of a pool of size n, the worst-case minimum total runtime is $\tilde{\Omega}(\frac{n\mathrm{OPT}}{\varepsilon^2\delta})$.[6] Since we need to test $\Omega(1/\gamma)$ configurations, in

[4]We use "total work" and "total runtime" interchangeably; both sum over all parallel threads.

[5]We use the standard $\mathcal{O}$ and $\tilde{\mathcal{O}}$ notation, where the latter hides poly-logarithmic factors.

[6]Essentially this holds since we need $\tilde{\Omega}(\frac{1}{\varepsilon^2\delta})$ sample runs to estimate the δ-capped runtime of a configuration with accuracy ε, as the maximum runtime for configuration i on some instance can be as large as $R_\delta(i)/\delta$.

Global variables

Instance distribution Γ
Phase I measurements count b
$T \leftarrow \infty$ ▷ Upper bound on $\mathrm{OPT}^{\gamma}_{\delta/2}$, updated continuously by all parallel processes
Set $\overline{\mathcal{N}}$ of algorithm configurations

Algorithm 1 IMPATIENTCAPSANDRUNS

Inputs: Precision parameter $\varepsilon \in (0, \frac{1}{3})$, Quantile parameter $\delta \in (0, \frac{1}{7})$, Optimality quantile target parameter γ, Failure probability parameter $\zeta \in (0, \frac{1}{12})$, Number of iterations K, Instance distribution Γ, Configuration distribution Π
$\mathcal{N}_k \leftarrow$ Sample $\left\lceil \frac{\log(\zeta/K)}{\log(1-\gamma_k)} \right\rceil - \left\lceil \frac{\log(\zeta/K)}{\log(1-\gamma_{k+1})} \right\rceil$ many configurations from Π for $k \in [0, K-1]$
$b \leftarrow \left\lceil \frac{26}{\delta} \log\left(\frac{2n}{\zeta}\right) \right\rceil$
Reset $T \leftarrow \infty$
$\mathcal{N} \leftarrow \bigcup_{k=0}^{K-1} \mathcal{N}_k$
for $k = K-1$ downto 0 **do**
$\overline{\mathcal{N}}_k \leftarrow$ PRECHECK $(\mathcal{N}_k, \zeta/K)$
for configurations $i \in \overline{\mathcal{N}}_k$ in parallel[a] **do**
$P_i \leftarrow$ CAPSANDRUNS $(i, \varepsilon, \delta, \zeta)$ thread
Start running P_i
Pause P_i when b runs of RUNTIMEEST finished
end for
end for
$\overline{\mathcal{N}} \leftarrow$ PRECHECK $(\mathcal{N}, \zeta/K)$
Continue runing P_i for $i \in \overline{\mathcal{N}}$
// CAPSANDRUNS eliminates the threads
Wait until all threads finish, abort if $|\overline{\mathcal{N}}| = 1$
return $i^* = \mathrm{argmin}_{i \in \overline{\mathcal{N}}} \bar{Y}(i)$ and τ_{i^*}

Algorithm 2 CAPSANDRUNS thread

Inputs: Configuration i, precision ε, quantile parameter δ, failure probability parameter ζ
// Phase I:
Run $\tau_i \leftarrow$QUANTILEEST (i, δ)
// Phase II:
if QUANTILEEST (i, δ) aborted **then**
Remove i from $\overline{\mathcal{N}}$
else
$\bar{Y}(i) \leftarrow$ RUNTIMEEST$(i, \tau_i, \varepsilon, \delta, \zeta)$
if RUNTIMEEST rejected i **then**
Remove i from $\overline{\mathcal{N}}$
end if
end if

Algorithm 3 QUANTILEEST

Inputs: i, δ
Initialize: $m \leftarrow \lceil (1 - \frac{3}{4}\delta) b \rceil$
Run configuration i on b instances, in parallel, until m of these complete. Abort and return *abort* if total work $\geq 1.5Tb$.
$\tau \leftarrow$ runtime of m^{th} completed instance
return τ

[a] When running CAPSANDRUNS threads in parallel, we allocate the same amount of time for every running thread, regardless of the number of parallel tasks they themselves may be performing.

Algorithm 4 PRECHECK

Inputs: Configurations $\mathcal{M}$, error parameter ζ/K
$\mathcal{M}' \leftarrow \{\}$ ▷ empty set
$b' \leftarrow \left\lceil 32.1 \log\left(\frac{2K}{\zeta}\right) \right\rceil$
if $T = \infty$ **then**
return $\mathcal{M}$
end if
for $i \in \mathcal{M}$ **do**
if T last set when evaluating i **then**
append i to $\mathcal{M}'$ ▷ Add automatically
Continue
end if
// Phase I:
Run i on b' instances in parallel until $\lceil 0.8b' \rceil$ complete. Abort if total work $\geq 1.9Tb'$.
if not aborted **then**
$\tau' \leftarrow$ runtime of $\lceil 0.8b' \rceil^{th}$ completed instance
// Phase II:
for $l = 1, l \leq b'$ **do**
$Y_l \leftarrow$ runtime of configuration i on instance $j \sim \Gamma$, with timeout τ'
if $\sum_{m=1}^{l} Y_m > 2.99Tb'$ **then**
// Stop measuring if total work too large
Break
end if
end for
Sample mean $\bar{Y} \leftarrow \frac{1}{|Y|} \sum_{y \in Y} y$
Sample variance $\bar{\sigma}^2 \leftarrow \frac{1}{|Y|} \sum_{y \in Y} (y - \bar{Y})^2$
Confidence $C \leftarrow \bar{\sigma} \sqrt{\frac{2 \log(\frac{3K}{\zeta})}{l}} + \frac{3\tau' \log(\frac{3K}{\zeta})}{l}$
if $\bar{Y} - C \leq T$ **then**
append i to $\mathcal{M}'$
end if
end if
end for
return $\mathcal{M}'$

Algorithm 5 RUNTIMEEST

Inputs: $i, \tau_i, \varepsilon, \delta, \zeta$
Initialize: $j \leftarrow 0$
while True **do**
Sample j^{th} instance J from Γ
$Y_{i,j} \leftarrow$ runtime of configuration i on instance J, with timeout τ_i
Sample mean $\bar{Y}(i) \leftarrow \frac{1}{j} \sum_{j'=1}^{j} Y_{i,j'}$
Sample variance $\bar{\sigma}_i^2 \leftarrow \frac{1}{j} \sum_{j'=1}^{j} (Y_{i,j'} - \bar{Y}(i))^2$
// Calculate confidence:
$C_i \leftarrow \bar{\sigma}_i \sqrt{\frac{2 \log(\frac{3nj(j+1)}{\zeta})}{j}} + \frac{3\tau_i \log(\frac{3nj(j+1)}{\zeta})}{j}$
if $\bar{Y}(i) - C_i > T$ **then**
return reject i
end if
if j=b **then**
$T \leftarrow \min\{T, 2\bar{Y}(i)\}$.
end if
$T \leftarrow \min\{T, \bar{Y}(i) + C_i\}$ ▷ upper confidence
if $C_i \leq \frac{\varepsilon}{3}(2\bar{Y}(i) - C_i)$ **then**
return accept i with runtime estimate $\bar{Y}(i)$.
end if
$j \leftarrow j + 1$
end while

the worst case the total runtime needed to find an $(\varepsilon, \delta, \gamma)$-optimal configuration is about $\frac{\mathrm{OPT}^{\gamma}_{\delta/2}}{\varepsilon^2 \delta \gamma}$. The first term in our bound matches this, except that it is multiplied by (an upper bound on) the fraction of configurations surviving PRECHECK, $F(38\mathrm{OPT}^{\gamma}_{\delta/2})$. Under typical parameter settings, this is the main term of the bound—the only one scaling with $1/(\varepsilon^2\delta\gamma)$—and the performance improvement of ICAR over CAR comes from this additional factor of $F(38\mathrm{OPT}^{\gamma}_{\delta/2})$. Note that this term, and all the others, scale with a bound on the *optimal* runtime for the set of configurations they correspond to (e.g., for batch $\mathcal{N}_k$ they scale with $\mathrm{OPT}^{\gamma_k}_{\delta/2}$).

(iv) $F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})$ is an upper bound on the number of configurations surviving PRECHECK from $\mathcal{N}_k$. Due to the a worst-case nature of our analysis, the bound is conservative, and in practice the number of surviving configurations is much smaller. In essence, this term measures how many configurations are competitive with a very good ($\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}$-optimal) configuration. In other words, it measures the "needle-in-a-haystack" property of the configuration task.

(v) The first term can be replaced with the problem-dependent bound of Weisz et al. [35, Equation 1] for $n = F(38\mathrm{OPT}^{\gamma}_{\delta/2})\frac{1}{\gamma}$ configurations. This bound depends on the characteristics of the runtime distributions of the configurations, and show that the algorithm can run much faster if the problem is easy, e.g., adapting to the relative variance of the runtime distributions. However, for simplicity, we only present the worst-case form here.

(vi) The rest of the terms represent the cost of iteratively selecting only the best configurations to evaluate. None of these terms depends on $1/\varepsilon^2$. Note $1/\gamma_k$ is roughly the number of configurations in batch $\mathcal{N}_k$, and each configuration is run essentially as long as the best configuration in that batch ($\mathrm{OPT}^{\gamma_k}_{\delta/2}$). Each of these configurations is run on constantly many instances in PRECHECK, and the surviving fraction of $F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})$ configurations is also run on $1/\delta$ instances to measure an accurate cap and set the bound T. These terms scale with $\mathrm{OPT}^{\gamma_k}_{\delta/2}/\gamma_k = 2^{-k}\mathrm{OPT}^{\gamma_k}_{\delta/2}/\gamma$. Thus, the bound is only meaningful when $2^{-k}\mathrm{OPT}^{\gamma_k}_{\delta/2}$ is not too large. While in principle they can be infinite, in realistic scenarios this is not the case. Nevertheless, this requires the practitioner to choose γ_{K-1} such that it guarantees a small-enough optimal runtime $\mathrm{OPT}^{\gamma_{K-1}}_{\delta/2}$, which is essentially the same task as choosing a proper γ. The terms also scale with $1/\delta$, but the effect of this is mitigated by the success of PRECHECK: for $k \neq K-1$, each term is multiplied by the upper bound $F(38\mathrm{OPT}^{\gamma_k}_{\delta/2})$ on the fraction of configurations surviving PRECHECK.

(vii) Our analysis shows that CAR can be sped up significantly without sacrificing any of its guarantees from Weisz et al. [35], by measuring the runtime caps on fewer samples (i.e., replacing the original value of b from Weisz et al. [35] with the one in Line 3 of Algorithm 1). We call this improved algorithm CAR ++. This effect is also partly responsible for the improved performance of ICAR.

Insights into the algorithm and proof sketch We start with a brief description of the CAR algorithm, which runs parallel threads of Algorithm 2 for all configurations it considers. As described before, one thread, working on configuration i, has two phases: In the first phase, implemented in QUANTILEEST (Algorithm 3), a runtime cap τ_i is determined such that i is guaranteed, with high probability, to solve a random instance with probability between $1-\delta$ and $1-\delta/2$ (i.e. $t_\delta(i) \le \tau_i < t_{\delta/2}(i)$).[7] This is achieved by solving sufficiently many instances in parallel, and τ_i is selected to be the time when a $(1-3\delta/4)$-fraction of the instances are solved. If measuring this cap takes too long, then QUANTILEEST stops measuring and eliminates configuration i. Unless this happens, in the second phase, the method RUNTIMEEST (Algorithm 3) is used to estimate the mean τ_i-capped runtime $R_{\tau_i}(i)$ of i, by solving successively selected random instances and computing the average runtime $\bar{Y}(i)$. Then the empirical Bernstein inequality [4] is used to guarantee that $R_{\tau_i}(i) \in [\bar{Y}(i) - C_i, \bar{Y}(i) + C_i]$ for C_i calculated in Line 9 of Algorithm 5. This confidence interval is used continuously in multiple ways: (i) to reduce a global upper bound T on the best possible runtime of all the configurations (Line 16); (ii) to eliminate a configuration if it shows that $R_{\tau_i}(i) > T$ (Line 10); and (iii) to check if $R_{\tau_i}(i)$ is estimated accurately enough (Line 17). The procedure (which is an instance of a so-called Bernstein race [30]) continues until each configuration is either measured accurately or eliminated. The continuous elimination (also in QUANTILEEST) and parallel execution

[7] Almost all guarantees provided in this paper are based on random sampling and hence hold with high probability. For brevity, when it is clear from the context, we often omit the 'high-probability' qualifier.

guarantees that when the procedure stops, every configuration is run for at most $\tilde{\mathcal{O}}(\mathrm{OPT}/(\varepsilon^2\delta))$ time, and eventually an (ε, δ)-optimal configuration is found, where OPT is the minimum mean $\delta/2$-quantile capped runtime of the configurations.

As explained before, ICAR (Algorithm 1) starts to examine new configurations in batches. For any batch $\mathcal{N}_k$, first each configuration is quickly tested to see if it can be excluded from the set of potentially optimal configurations. This is done by the PRECHECK function, given in Algorithm 4. PRECHECK is very similar to CAR, but works with constant accuracy and quantile parameters instead of ε and δ, ensuring that it runs quickly, in time independent of these parameters. Also, the conditions to reject configurations are slightly different. For a configuration i, PRECHECK first estimates a cap τ' that guarantees solving random instances with constant probability $p_i \in [0.1, 0.35]$; then the mean τ'-capped runtime is estimated roughly up to a constant multiplicative error. Since $\delta/2 \leq 0.1$ (the lower bound on p_i), PRECHECK can compute multiplicative lower bounds on the runtime $R_{\delta/2}(i)$. These are then used to set the rejection conditions such that at least one of the best configurations from this batch i with $R_{\delta/2}(i) \leq T$ is not rejected. Combining with the fact that $\bigcup_{j=k}^{K-1} \mathcal{N}_j$ contains a top-γ_k configuration, such a configuration survives PRECHECK and the corresponding CAPSANDRUNS-thread in ICAR (Algorithm 1) ensures that T is set to at most $2\mathrm{OPT}^{\gamma_k}_{\delta/2}$ in Line 11 of Algorithm 1, that is, T is continuously refined as new batches are evaluated. The number of configurations surviving PRECHECK can be bounded by looking at mean runtimes capped at the 0.35-quantile (upper bound on p_i). Together with the setting of T, this implies that at most a $\tilde{\mathcal{O}}(F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})$ fraction of the $|\mathcal{N}_k| = \tilde{\mathcal{O}}(1/\gamma_k)$ configurations survive PRECHECK. Considering that the number of runs carried out for each configuration is constant in PRECHECK, $\tilde{\mathcal{O}}(1/\delta)$ in the loop of Algorithm 1, and $\tilde{\mathcal{O}}(1/(\varepsilon^2\delta))$ in the last full CAR procedure, since the average runtime per configuration for $\mathcal{N}_k$ is $\mathrm{OPT}^{\gamma_k}_{\delta/2}$ (by the analysis of CAR), the runtime bound of the theorem follows. Correctness (i.e., the fact that the procedure finds an $(\varepsilon, \delta, \gamma)$-optimal configuration) follows from that of CAR and because PRECHECK retains good configurations, as just shown.

4 Experiments

The basic setup and main results of our experimental analysis of ICAR are given below, while details are presented in Appendix C, along with a synthetic experiment examining ICAR's speedup as good configurations become increasingly rare. We compared against the best available configurators that come with theoretical guarantees. We used the improved version of CAR (CAR++), derived in this paper, which uses a smaller b-value than the original version, thanks to our improved analysis (see Section 3 and Appendix A for details). Including CAR++ in the experiments allowed us to separately examine the effects of two improvements we introduced: (i) the smaller number of samples b needed in CAR, and (ii) the main conceptual innovation of this paper, the impatient discarding of configurations using PRECHECK. We attempted to compare against SPC [24] as well. However, in the experiments presented in Table 1, although SPC identified good configurations, it usually was not able to provide the required guarantees on ε and δ even after running for twice as long as the slowest alternative considered (CAR): SPC did not provide guarantees for 7 out of the 9 scenarios while also being the slowest in the other two cases (1.56 and 1.91 times slower than CAR). Therefore, we decided not to include SPC in our further comparisons.

Datasets. We looked at two datasets from MIP and one from SAT. We considered true runtime data from the `minisat` SAT solver on instances generated by CNFuzzDD (`http://fmv.jku.at/cnfuzzdd`), which was examined in past work [34, 35, 24]. For the MIP scenarios, we looked at the CPLEX integer program solver on combinatorial auction instances (Regions200 [26]) and problems from wildlife conservation (RCW [1]). To generate sufficient MIP runtime data, following Hutter et al. [22], we used an *Empirical Performance Model (EPM)*—a random forest model trained on existing runtime data—to predict the runtime of new configurations on new instances. EPMs can do surprisingly well at predicting individual runtimes, particularly on the MIP datasets we consider. More importantly for our purposes, Eggensperger et al. [12] showed that such EPMs are effective surrogates for algorithm configuration, capturing key properties of runtime distributions such as the relative quality of configurations. We note that similar surrogates have also been used to guide search procedures [19, 9, 33, 37], to build algorithm portfolios [31, 36], to impute missing data [10], and to optimize hyperparameters from limited observations [32].

Main Results. Table 1 shows the total CPU time needed to find a $(0.05, 0.1, \gamma)$-optimal configuration on each dataset with the same total failure probability (0.05) and with different values of γ. The

		Total CPU Time (days)			Number of Conf. Before/After PRECHECK			R^δ of returned conf. (secs)		
		$\gamma = 0.05$	$\gamma = 0.02$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.02$	$\gamma = 0.01$	$\gamma = 0.05$	$\gamma = 0.02$	$\gamma = 0.01$
Minisat CNFuzzDD	ICAR	101 (13)	243 (15)	467 (25)	134 / 74	351 / 197	724 / 395	5.0 (0.1)	4.9 (0.1)	4.9 (0.1)
	CAR++	**92 (5)**	**224 (16)**	**452 (18)**	97	245	492	5.2 (0.1)	4.9 (0.1)	4.9 (0.1)
	CAR	158 (18)	368 (7)	771 (22)	97	245	492	5.2 (0.1)	4.9 (0.1)	4.9 (0.1)
CPLEX Regions200	ICAR	**164 (91)**	**275 (101)**	**420 (103)**	134 / 10	351 / 15	724 / 26	34.8 (4.3)	29.8 (2.2)	28.5 (1.8)
	CAR++	229 (20)	567 (28)	1098 (88)	97	245	492	35.3 (4.3)	32.0 (2.2)	29.8 (1.8)
	CAR	524 (53)	1295 (64)	2549 (199)	97	245	492	35.3 (4.5)	31.9 (1.6)	29.8 (2.2)
CPLEX RCW	ICAR	**1284 (391)**	**2030 (302)**	**4072 (239)**	134 / 18	351 / 44	724 / 97	156.1 (11.9)	146.5 (4.1)	143.3 (4.9)
	CAR++	1728 (375)	3644 (185)	7526 (131)	97	245	492	162.1 (11.9)	149.1 (4.1)	143.3 (4.9)
	CAR	3306 (502)	7591 (192)	15658 (258)	97	245	492	160.1 (13.3)	149.1 (4.7)	143.3 (4.9)

Table 1: Total CPU time in days to find a $(0.05, 0.1, \gamma)$-optimal configuration, the number of configurations before and after PRECHECK, and the quality of the returned configurations, as measured by δ-capped mean runtime with $\delta = 0.1$. For CAR and CAR++, the number of configurations sampled is reported. Error terms (in parentheses) are standard deviations over five runs.

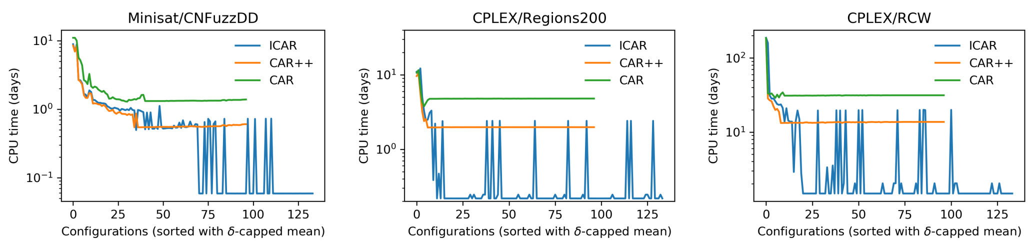

Figure 1: CPU time spent on each configuration while searching for a $(0.05, 0.1, 0.05)$-optimal one (note the log scale on the y-axis). CAR and CAR++ allocated a significant amount of time to evaluating bad configurations, while ICAR discarded many of these with near minimal work via its PRECHECK routine. The large spikes in the ICAR curve are those configurations that fail to be rejected by the first call to PRECHECK. Smaller spikes are configurations that were also rejected by PRECHECK, but the decision took more time (e.g., T was larger in PRECHECK or the configuration was rejected in the second phase of PRECHECK).

parameters were not specifically chosen; results for varying ε and δ are reported in Appendix C. ICAR consistently outperformed CAR in all cases; ICAR outperformed CAR++ on the MIP datasets and was competitive on the SAT one. The performance improvement was largest when the PRECHECK mechanism managed to discard the most configurations; the MIP datasets have relatively more weak configurations, enabling PRECHECK to filter out more configurations quickly (see Fig. 2 in the Appendix for the distribution of configuration means). When γ is relatively small, ICAR was more likely to sample a really good configuration, making it easier to discard weak ones. In this case its runtime was as little as half that of CAR++, a significant improvement. Despite taking less total CPU time, ICAR actually sampled *more* configurations than CAR did. To understand this phenomenon better, Fig. 1 shows the time spent running each configuration. For all datasets the plots nearly overlap for the very best few configurations, indicating that ICAR treated these good configurations in much the same way as CAR or CAR++. However, the effect of the PRECHECK mechanism is clear, as ICAR ran many bad configurations for near-zero time, discarding them quickly. In cases where a bad configuration made it past PRECHECK (largest spikes in the blue curve), ICAR ran it for an amount of time similar to CAR++. Finally, the empirical mean δ-capped runtime (R^δ) of the returned configuration is reported in Table 1. All configurators returned solutions with similar quality, but thanks to its ability to examine more configurations, ICAR often did slightly better.

5 Conclusions

This paper presented a novel algorithm configuration method, ICAR, that selects configurations from an infinite pool with optimal theoretical guarantees up to logarithmic factors under mild conditions. While earlier theoretically grounded methods thoroughly test all configurations, ICAR—like successful heuristic approaches—quickly discards less promising ones. As a result, ICAR achieves significant speedups, particularly in needle-in-a-haystack scenarios. It thus constitutes an important step towards closing the gap between theoretical and heuristic procedures.

A key limitation is that our work focuses simply on evaluating randomly sampled configurations. We do note that state-of-the-art heuristic methods also evaluate many random configurations to avoid getting stuck in local optima, so analyzing such procedures is of obvious practical importance. Furthermore, ICAR can be understood as a way of weighing different candidate configurations against each other, which could be proposed by model- or gradient-based methods as well as by random sampling (see, e.g., an argument to this effect in [23, Theorem 7.1]).

Broader Impact

We expect that our theorems will guide the design of future algorithm configuration procedures. We note that speeding up computationally expensive algorithms saves time, money, and electricity, arguably reducing carbon emissions and yielding social benefit. The algorithms we study can be be applied to a limitless range of problems and so could yield both positive and negative impacts; however, we do not foresee our work particularly amplifying such impacts beyond the computational speedups already discussed.

References

[1] Ahmadizadeh, K., Dilkina, B., Gomes, C. P., and Sabharwal, A. An empirical study of optimization for maximizing diffusion in networks. In *International Conference on Principles and Practice of Constraint Programming*, pp. 514–521. Springer, 2010. `http://www.cs.cornell.edu/~kiyan/rcw/generator.htm`.

[2] Ansótegui, C., Sellmann, M., and Tierney, K. A gender-based genetic algorithm for automatic configuration of algorithms. In *Principles and Practice of Constraint Programming (CP)*, pp. 142–157, 2009.

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

[4] Audibert, J.-Y., Munos, R., and Szepesvári, C. Tuning bandit algorithms in stochastic environments. In *ALT*, volume 4754, pp. 150–165. Springer, 2007.

[5] 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.

[6] Balcan, M.-F., Nagarajan, V., Vitercik, E., and White, C. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In *Conference on Learning Theory*, pp. 213–274, 2017.

[7] Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. Learning to branch. *International Conference on Machine Learning*, 2018.

[8] Balcan, M.-F., DeBlasio, D., Dick, T., Kingsford, C., Sandholm, T., and Vitercik, E. How much data is sufficient to learn high-performing algorithms? *arXiv preprint arXiv:1908.02894*, 2019.

[9] Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. Collaborative hyperparameter tuning. In *International conference on machine learning*, pp. 199–207, 2013.

[10] Biedenkapp, A., Marben, J., Lindauer, M., and Hutter, F. Cave: Configuration assessment, visualization and evaluation. In *International Conference on Learning and Intelligent Optimization*, pp. 115–130. Springer, 2018.

[11] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. A racing algorithm for configuring metaheuristics. In *Genetic and Evolutionary Computation Conference (GECCO)*, pp. 11–18, 2002.

[12] Eggensperger, K., Lindauer, M., Hoos, H. H., Hutter, F., and Leyton-Brown, K. Efficient benchmarking of algorithm configurators via model-based surrogates. *Machine Learning*, 107 (1):15–41, 2018.

[13] Gupta, R. and Roughgarden, T. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

[14] Hoos, H. H. *Stochastic local search-methods, models, applications*. IOS Press, 1998.

[15] Hoos, H. H. A mixture-model for the behaviour of SLS algorithms for SAT. In *AAAI/IAAI*, pp. 661–667, 2002.

[16] Hoos, H. H. and Stützle, T. Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112(1-2):213–232, 1999.

[17] Hutter, F., Hoos, H., and Stützle, T. Automatic algorithm configuration based on local search. In *AAAI Conference on Artificial Intelligence*, pp. 1152–1157, 2007.

[18] Hutter, F., Hoos, H., Leyton-Brown, K., and Stützle, T. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[19] 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.

[20] Hutter, F., Hoos, H., and Leyton-Brown, K. Bayesian optimization with censored response data. In *NIPS workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits (BayesOpt'11)*, 2011.

[21] Hutter, F., Hoos, H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Conference on Learning and Intelligent Optimization (LION)*, pp. 507–523, 2011.

[22] Hutter, F., Xu, L., Hoos, H., and Leyton-Brown, K. Algorithm runtime prediction: Methods and evaluation. *AIJ*, 206:79–111, 2014.

[23] Kleinberg, R., Leyton-Brown, K., and Lucier, B. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[24] Kleinberg, R., Leyton-Brown, K., Lucier, B., and Graham, D. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[25] Kroc, L., Sabharwal, A., and Selman, B. An empirical study of optimal noise and runtime distributions in local search. In *International Conference on Theory and Applications of Satisfiability Testing*, pp. 346–351. Springer, 2010.

[26] Leyton-Brown, K., Pearson, M., and Shoham, Y. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 66–76, 2000. `https://www.cs.ubc.ca/~kevinlb/CATS`.

[27] Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.

[28] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. The irace package, iterated race for automatic algorithm configuration. Technical report, IRIDIA, Université Libre de Bruxelles, 2011. `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`.

[29] Maclaurin, D., Duvenaud, D., and Adams, R. P. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pp. 2113–2122, 2015.

[30] Mnih, V., Szepesvári, C., and Audibert, J.-Y. Empirical Bernstein stopping. In *Proceedings of the 25th international conference on Machine learning*, pp. 672–679. ACM, 2008.

[31] Nudelman, E., Leyton-Brown, K., Andrew, G., Gomes, C., McFadden, J., Selman, B., and Shoham, Y. Satzilla 0.9. Solver description, International SAT Competition, 2003.

[32] Probst, P., Bischl, B., and Boulesteix, A.-L. Tunability: Importance of hyperparameters of machine learning algorithms. *arXiv preprint arXiv:1802.09596*, 2018.

[33] Swersky, K., Snoek, J., and Adams, R. P. Multi-task Bayesian optimization. In *Advances in neural information processing systems*, pp. 2004–2012, 2013.

[34] Weisz, G., György, A., and Szepesvári, C. LeapsAndBounds: A method for approximately optimal algorithm configuration. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[35] Weisz, G., György, A., and Szepesvári, C. CapsAndRuns: An improved method for approximately optimal algorithm configuration. In *International Conference on Machine Learning*, pp. 6707–6715, 2019.

[36] Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research (JAIR)*, 32:565–606, 2008.

[37] Yogatama, D. and Mann, G. Efficient transfer learning method for automatic hyperparameter tuning. In *Artificial intelligence and statistics*, pp. 1077–1085, 2014.

A Proof of Theorem 1

The first step of the proof improves the analysis of CAPSANDRUNS given in [35]. In [35], the value of b was $\left\lceil \frac{48}{\delta} \log\left(\frac{3n}{\zeta}\right)\right\rceil$, which we replace here with $\left\lceil \frac{26}{\delta} \log\left(\frac{2n}{\zeta}\right)\right\rceil$. This value is used in the original analysis of CAPSANDRUNS twice, in Lemma 2 and Lemma 3 of [35]. The analysis of Lemma 2 still holds with the new value without any change, while we give a new proof for Lemma 3 of [35]: the difference is that in the new proof we use the Bernstein inequality rather than its empirical version. . The new version of the lemma, Lemma 2, is slightly stronger, which means we can replace $2Tb$ with $1.5Tb$ in Line 3 of the sub-routine QUANTILEEST. Note that this change of the value of b itself improves the runtime of CAR, and we call the resulting algorithm CAR++, which will also be examined in the experiment section.

To prove Theorem 1, we need to (i) prove the correctness of IMPATIENTCAPSANDRUNS, that is, the $(\varepsilon, \delta, \gamma)$-optimality of the configuration returned by the algorithm; and (ii) give a bound on the total runtime. Starting with the correctness, we note that the algorithm proceeds in iterations from $K-1$ to 0 in decreasing order, sampling bigger and bigger sets of configurations $\mathcal{N}_k$. Each new set $\mathcal{N}_k$, together with those configurations sampled before for $k' > k$, contains an $\mathrm{OPT}^{\gamma_k}_{\delta/2}$-optimal configuration with high probability (Lemma 4), in other words, a configuration from an exponentially decreasingly small quantile of the best configurations. The size of $\mathcal{N}_k$, for all $k \in [0, K-1]$ is roughly $\log(K/\zeta)/\gamma_k$ (Lemma 6). Next, we prove in Lemma 8 that PRECHECK does not reject a good configuration, and does reject a truly bad configuration. Unlike other parts of the proof, we do not guarantee this to hold with high probability for all configurations, instead we guarantee it to hold with high probability for any one configuration per each iteration k; this will be chosen later to be one of the $\mathrm{OPT}^{\gamma_k}_{\delta/2}$-optimal configurations. Then, Lemma 10 shows that there remains an $\mathrm{OPT}^{\gamma_k}_{\delta/2}$-optimal configuration after each iteration k (Line 11 of Algorithm 1) that is not rejected by QUANTILEEST or RUNTIMEEST. This is because even if our designated configuration was rejected by PRECHECK, that means that there was an even better configuration, which from the proof of CAPSANDRUNS, by Lemma 9, will not be rejected by QUANTILEEST or RUNTIMEEST. Several corollaries follow from this. Corollary 11 shows that with high probability, the configuration IMPATIENTCAPSANDRUNS returns in the end is $(\varepsilon, \delta, \gamma)$-optimal, showing the correctness of the algorithm To prove the runtime bound, we start by showing that in every iteration k, T is set to at most $2\mathrm{OPT}^{\gamma_k}_{\delta/2}$, after evaluating a configuration for no more than $4b\mathrm{OPT}^{\gamma_k}_{\delta/2}$ time (Corollary 12). From this, Corollary 13 deduces a runtime bound for CAR in each iteration, which depends on the number of configurations surviving PRECHECK. Using the correctness analysis of PRECHECK (Lemma 8), Lemma 14 gives an upper bound on this number, essentially saying that roughly only $F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})$ fraction of the $\mathcal{N}_k$ configurations survive PRECHECK in round k, where $F(x)$ is roughly the probability of a random configuration sampled from Π having a larger 0.35th quantile-capped[8] runtime than x. That is, essentially only those configurations survive which can solve at least 65% of the problem instances reasonably fast.

This is complemented by Lemma 15, which gives a runtime bound for PRECHECK, relying on Lines 13 and 21 of PRECHECK (Algorithm 4) stopping lengthy evaluations. To finish the proof, we combine the runtime bounds for all the components of ICAR discussed above. The lemmas above introduce various high-probability events under which their statements hold (by guaranteeing mostly that our bounds on the runtime caps and on the average runtimes hold), and a union bound over them proves that all those events hold simultaneously with probability at least $1 - 12\zeta$, proving Theorem 1.

Lemma 2 (Improved version of Lemma 3 of [35])**.** *Let τ be a constant satisfying $0 \le \tau \le t_{\delta/2}(i)$, and let $Z_\tau(i,j)$, $j \in [1, b]$, be b runtime measurements of configuration i with timeout τ. Let $\bar{Z}_\tau(i)$ be their average and $R_\tau(i)$ their expectation. Then for $c > 0$, $\Pr\left(|\bar{Z}_\tau(i) - R_\tau(i)| \ge cR_\tau(i)\right) \le 2\exp\left(\frac{b\delta c^2}{4(1+c/3)}\right)$. In particular, for $S_i = \{\frac{1}{2}R_\tau(i) \le \bar{Z}_\tau(i) \le 1.5R_\tau(i)\}$ and $b = \left\lceil \frac{26}{\delta}\log\left(\frac{2n}{\zeta}\right)\right\rceil$, we have $\Pr(S_i^c) \le \frac{\zeta}{n}$ (by substituting $c = \frac{1}{2}$).*

[8]This constant 0.35 can be set arbitrarily, and it only affects other constants in the algorithm. It was set to 0.35 so that these constant do not increase beyond how large they have to be to guarantee other statements with high probability.

Proof. Since $Z_\tau(i,j) \le \tau$, $\mathrm{Var}(Z_\tau(i)) = \mathrm{Var}_{j\sim\Gamma}[Z_\tau(i,j)] \le \mathbb{E}_{j\sim\Gamma} Z_\tau^2(i,j) \le \tau R_\tau(i)$. As at least $\delta/2$ fraction of instances run longer than τ, we have that $R_\tau(i) \ge \frac{\delta}{2}\tau$, so $\mathrm{Var}(Z_\tau(i)) \le \frac{2}{\delta} R_\tau^2(i)$. Using the Bernstein inequality,

$$\begin{aligned}\Pr\left(|\bar{Z}_\tau(i) - R_\tau(i)| \ge cR_\tau(i)\right) &\le 2\exp\left(-\frac{bc^2 R_\tau^2(i)/2}{\frac{1}{3}\tau c R_\tau(i) + \mathrm{Var}(Z_\tau(i))}\right)\\ &\le 2\exp\left(-\frac{bc^2 R_\tau^2(i)/2}{\frac{2}{3\delta} c R_\tau^2(i) + \frac{2}{\delta} R_\tau^2(i)}\right)\\ &= 2\exp\left(\frac{b\delta c^2}{4(1+c/3)}\right).\end{aligned}$$

□

Remark 3. *From Lemma 6 of [35], there is an event E_1 (with the notation of [35],this event is $E_1 \cap E_2 \cap E_3$) with $\Pr(E_1) \ge 1 - 6\zeta$, under which all the high-probability statements in the analysis of* CAPSANDRUNS *hold for the algorithm with the constants improved as above. In particular, E_1 guarantees that the average runtime estimates of CAR are close to their expectations, and that* QUANTILEEST *measures an accurate cap for each configuration such that $t_\delta(i) \le \tau_i \le t_{\delta/2}(i)$.*

Lemma 4. *There is an event E_2 with $\Pr(E_2) \ge 1 - \zeta$ such that under E_2, for all integers $k \in [0, K-1]$, there is a configuration $i \in \bigcup_{j=k}^{K-1} \mathcal{N}_j$ with $R^{\frac{\delta}{2}}(i) \le \mathrm{OPT}_{\delta/2}^{\gamma_k}$ after Line 2 of* IMPATIENTCAPSANDRUNS *(Algorithm 1).*

Proof. For any i chosen randomly from the distribution Π, $R^{\frac{\delta}{2}}(i) \le \mathrm{OPT}_{\delta/2}^{\gamma_k}$ with probability at least γ_k. As the configurations are sampled independently, the probability that none of the sampled configurations are optimal for γ_k is at most $(1-\gamma_k)^{|\bigcup_{j=k}^{K-1}\mathcal{N}_j|} = (1-\gamma_k)^{\left\lceil\frac{\log(\zeta/K)}{\log(1-\gamma_k)}\right\rceil} \le \zeta/K$. Applying the union bound over $k \in [0, K-1]$, with probability at least $1-\zeta$, for all $k \in [0, K-1]$, there is a configuration i with $R^{\frac{\delta}{2}}(i) \le \mathrm{OPT}_{\delta/2}^{\gamma_k}$ sampled into $\bigcup_{j=k}^{K-1}\mathcal{N}_j$ at Line 2 of IMPATIENTCAPSANDRUNS. □

Remark 5. *Noting that $\gamma_0 = \gamma$, and $\bigcup_{k=0}^{K-1}\mathcal{N}_k = \mathcal{N}$, the previous lemma with $k = 0$ states that under E_2, a configuration i with $R^{\frac{\delta}{2}}(i) \le \mathrm{OPT}_{\delta/2}^{\gamma}$ is sampled into $\mathcal{N}$.*

In the following we refer to the last part of Algorithm 1 (Lines 14 to 18) iteration -1, denote it with $k = -1$, and accordingly define $\overline{\mathcal{N}}_{-1} = \overline{\mathcal{N}}$ and $\mathcal{N}_{-1} = \mathcal{N}$.

Lemma 6. *After Line 2 of Algorithm 1, for all $k \in [-1, K-1]$, $|\mathcal{N}_k| \le \log(K/\zeta)/\gamma_k + 1$.*

Proof. Using that for any $x \in (0,1)$, $x \le -\log(1-x)$, we have for any $k \in [-1, K-1]$ that

$$|\mathcal{N}_k| \le \left\lceil \frac{\log(\zeta/K)}{\log(1-\gamma_k)} \right\rceil \le \frac{\log(K/\zeta)}{-\log(1-\gamma_k)} + 1 \le \log(K/\zeta)/\gamma_k + 1.$$

□

By [35, Lemma 2], under E_1, and noting that T can only be set by RUNTIMEEST evaluating a configuration after the cap τ for that configuration has already been measured by QUANTILEEST:

Lemma 7. *If a configuration i sets T, then $t_\delta(i) \le \tau_i \le t_{\delta/2}(i)$.*

Lemma 8. *Suppose $\delta \le 0.2$ and assume that we are in the* PRECHECK *call in iteration $-1 \le k < K-1$ of Algorithm 1 (recall that iteration -1 refers to the last part of the algorithm after the iteration loop is finished). Let $\Pr_k$ denote the conditional probability conditioned on all the random events before the call to* PRECHECK*. Let i' be the configuration that was last evaluated to set T by* RUNTIMEEST *(in an iteration $k' > k$). For any $i \in \mathcal{M}$, there is an event $E_{3,k,i}$ with $\Pr_k(E_{3,k,i}) \ge 1 - 4\zeta/K$ such that under E_1 and $E_{3,i}$, (1) if $R^{\frac{\delta}{2}}(i) \le R_{\tau_{i'}}(i')$, i won't be rejected by* PRECHECK*, and (2) if $R^{0.35}(i) \ge 19T$, then i will be rejected by* PRECHECK*.*

Proof. Consider the evaluation of configuration i in PRECHECK. Let I_l be the indicator that the l^{th} instance in Phase I of PRECHECK takes at least $t_{1/10}(i)$ time to complete (without capping). The I_l are independent and identically distributed Bernoulli random variables with $\Pr_k(I_l = 1) = 1/10$. We use the Chernoff bound to get that $\Pr_k\left(\sum_{l=0}^{b'} I_l > 0.2b'\right) = \Pr_k\left(\sum_{l=0}^{b'} I_l > \frac{1}{10}b'(1+1)\right) \leq \exp\left(-\frac{1}{30}b'\right) \leq \zeta/(2K)$. Let $E_{3,k,i,1}$ be the event that $\sum_{l=0}^{b'} I_l \leq 0.2b'$. Similarly, defining J_l to be the indicator that the l^{th} instance in Phase I of PRECHECK takes at least $t_{0.35}$ time to complete (without capping), noting that $\Pr_k(J_l = 1) = 0.35$, the Chernoff bound implies that $\Pr_k\left(\sum_{l=0}^{b'} J_l < 0.2b'\right) = \Pr_k\left(\sum_{l=0}^{b'} J_l > 0.35b'(1 - \frac{0.35-0.2}{0.35})\right) \leq \zeta/(2K)$. Let $E_{3,k,i,2}$ be the event that $\sum_{l=0}^{b'} J_l \geq 0.2b'$.

So for any configuration $i \in \mathcal{M}$, under $E_{3,k,i,1}$, the number of samples from the $1/10$-tail will be at most $\lfloor 0.2b' \rfloor$, and under $E_{3,k,i,2}$, the number of samples from the 0.35-tail will be at least $\lceil 0.2b' \rceil$, so picking the $\lceil 0.8b' \rceil^{th}$ finished run and denoting it by τ' (in Line 15 of Algorithm 4) ensures under and $E_{3,k,i,1}$ and $E_{3,k,i,2}$ that $t_{0.35}(i) \leq \tau' \leq t_{1/10}(i)$. For $\delta \leq 0.2$ (which we have by assumption), this implies that $\tau' \leq t_{\delta/2}(i)$.

By [35, Lemma 7], under E_1, $R_{\tau_{i'}}(i') \leq T$. Assuming that $R^{\frac{\delta}{2}}(i) \leq R_{\tau_{i'}}(i')$ for (1), we have $R^{\frac{\delta}{2}}(i) \leq T$. Then under $E_{3,k,i,1}$, $R_{\tau'}(i) \leq R^{1/10}(i) \leq T$ (as by assumption $\delta \leq 0.2$ and $\tau' \leq t_{1/10}(i)$, so $R_{\tau'}(i) \leq R^{1/10}(i) \leq R^{\frac{\delta}{2}}(i) \leq R_{\tau_{i'}}(i') \leq T$). There are two cases in which we reject configuration i. First, if $\operatorname{avg}(Y) - C \geq T$. By the empirical Bernstein bound [4], there is an event $E_{3,k,i,3}$ such that $\Pr_k(E_{3,k,i,3}) \geq 1 - \zeta/K$, and under $E_{3,k,i,3}$, $|\operatorname{avg}(Y) - \mathbb{E}_k[\operatorname{avg}(Y)|\tau']| \leq C$. As $\mathbb{E}_k[\operatorname{avg}(Y)|\tau'] = R_{\tau'}(i) \leq T$, we have that under E_1, $E_{3,k,i,1}$ and $E_{3,k,i,3}$, configuration i in iteration k will not be rejected in Line 27 of PRECHECK if (1) holds.

The second type of rejection happens in Line 13 of PRECHECK when Phase I of PRECHECK runs for at least $1.9Tb'$ time. For each run l that is performed in Phase I, denote by X_l the hypothetical runtime of that instance if the cap were $t_{1/10}(i)$, and by Y_l the run if the runtime cap were $t_{0.35}(i)$. From the above, under $E_{3,k,i,1}$ $\tau' \leq t_{1/10}(i)$, and if we had to abort then that means we haven't run any instance for τ' time yet, so by denoting measurements performed so far by Phase I of PRECHECK by $\bar{X}_l$, we have $\bar{X}_l \leq X_l$, so when we abort we have that $\operatorname{avg}(X) \geq 1.9T$.

Applying Lemma 2 with $c = 0.9$, $b = b' = \left\lceil 32.1 \log\left(\frac{2K}{\zeta}\right)\right\rceil$, $\delta = 0.2$, and $\tau = t_{1/10}(i)$, we get that $\Pr_k\left(|\operatorname{avg}(X) - R^{1/10}(i)| \geq 0.9R^{1/10}(i)\right) \leq 2\exp\left(b'\frac{81}{5\cdot 520}\right) \leq \frac{\zeta}{K}$. Denote by $E_{3,k,i,4}$ the event that $\operatorname{avg}(X) \leq 1.9T$. Then, for (1), under E_1 and $E_{3,k,i,1}$, by the above $R^{1/10}(i) \leq T$, we have $\Pr_k(\operatorname{avg}(X) \geq 1.9T) \leq \Pr_k(\operatorname{avg}(X) - R^{1/10}(i) > 0.9R^{1/10}(i)) \leq \frac{\zeta}{K}$, so $\Pr_k(E_{3,k,i,4}) \geq 1 - \frac{\zeta}{K}$, and under E_1 and $E_{3,k,i,4}$, this configuration won't be rejected in Line 13 of PRECHECK if it satisfies (1).

For (2), let $E_{3,k,i,5}$ the event that $\operatorname{avg}(Y) \geq 0.1R^{0.35}(i))$. Apply Lemma 2 with the same parameters except $\tau = t_{0.35}(i)$, to get that $\Pr_k(\operatorname{avg}(Y) \leq 0.1R^{0.35}(i)) = \Pr_k(R^{0.35}(i) - \operatorname{avg}(Y) \geq 0.9R^{0.35}(i)) \leq \Pr_k(|\operatorname{avg}(Y) - R^{0.35}(i)| \geq 0.9R^{0.35}(i)) \leq 2\exp\left(b'\frac{81}{5\cdot 520}\right) \leq \frac{\zeta}{K}$. For PRECHECK to not reject a configuration i, it measures a cap $\tau' \geq t_{0.35}(i)$ (under $E_{3,k,i,2}$), and so the measurements $\bar{Y}_l$ satisfy $\bar{Y}_l \geq Y_l$, so we spend $b' \operatorname{avg}(\bar{Y}_l) \geq b' \operatorname{avg}(Y_l) \geq 0.1R^{0.35}(i)$ time for configuration i under $E_{3,k,i,5}$. Thus, with probability at least $\Pr_k(E_{3,k,i,4}) \geq 1 - \zeta/K$, a configuration where $R^{0.35}(i) \geq 19T$ is rejected in Line 13. Taking a union bound and letting $E_{3,k,i} = E_{3,k,i,1} \cap E_{3,k,2} \cap E_{3,k,i,3} \cap E_{3,k,i,4} \cap E_{3,k,i,5}$ (the event that all the high-probability statements above hold for configuration i and iteration k), we have that $\Pr_k(E_{3,k,i}) \geq 1 - 4\zeta/K$. $\square$

From the proof of [35, Theorem 1] we can extract the following result:

Lemma 9. *Let $\mathcal{N}$ be the set of configurations* CAPSANDRUNS *is called with, and $\mathcal{N}'$ the ones among these that are not rejected in* QUANTILEEST. *Let $i_* = \min_{i \in \mathcal{N}'} R_{\tau_i}(i)$. Under E_1, i_* is not rejected in* RUNTIMEEST *and* CAPSANDRUNS *returns a configuration I for which $R_{\tau_I}(I) \leq (1+\varepsilon)R_{\tau_{i_*}}(i_*)$.*

To proceed, we instantiate the events $E_{3,k,i}$ of Lemma 8 for one of the best configurations i in $\mathcal{N}_k$. By Lemma 4 and Remark 5, under E_2, for every iteration k of IMPATIENTCAPSANDRUNS, there is a configuration $\hat{i}_k^* \in \bigcup_{j=k}^{K-1} \mathcal{N}_j$ such that $R^{\frac{\delta}{2}}(\hat{i}_k^*) \leq \mathrm{OPT}_{\delta/2}^{\gamma_k}$. Furthermore, this guarantees that

$\hat{\imath}_0^* \in \mathcal{N}$ satisfies $R^{\frac{\delta}{2}}(\hat{\imath}_0^*) \le \mathrm{OPT}^{\gamma}_{\delta/2}$, which also implies, through the first part of Lemma 9, that under E_1, there is a configuration $\hat{\imath}_{-1}^*$ satisfying $R^{\frac{\delta}{2}}(\hat{\imath}_{-1}^*) \le \mathrm{OPT}^{\gamma}_{\delta/2}$. Now we define the following event $E_4 \subset E_1 \cap E_2$ as $E_4 = \cap_{k=-1}^{K-2} E_{3,k,\hat{\imath}_k^*} \cap E_1 \cap E_2$.

Lemma 10. *Under E_4, for all integer $0 \le k \le K-1$, there is a configuration i_k^* remaining in $\bigcup_{j=k}^{K-1} \overline{\mathcal{N}}_j$ at the end of the k^{th} iteration (after Line 11 in Algorithm 1), that is not rejected by* QUANTILEEST *or* RUNTIMEEST, *for which $R_{\tau_{i_k^*}}(i_k^*) \le \mathrm{OPT}^{\gamma_k}_{\delta/2}$. Similarly, there is a configuration i_* remaining in $\overline{\mathcal{N}}$ at the end of the final* CAPSANDRUNS *call (after Line 17 in Algorithm 1), for which $R_{\tau_{i_*}}(i_*) \le \mathrm{OPT}^{\gamma}_{\delta/2}$.*

Proof. Suppose E_4 holds (this also means that E_1 and E_2 hold). Let i' denote the configuration that last set T.

For $k = K-1$ there is no PRECHECK as $T = \infty$, in other words nothing is rejected by PRECHECK. For iterations $0 \le k \le K-2$, and for the final CAPSANDRUNS call, either $R^{\frac{\delta}{2}}(\hat{\imath}_k^*) \le R_{\tau_{i'}}(i')$, in which case by Lemma 8, under E_1 and $E_{3,k,\hat{\imath}_k^*}$, $\hat{\imath}_k^*$ is not rejected, or $R^{\frac{\delta}{2}}(\hat{\imath}_k^*) > R_{\tau_{i'}}(i')$. Thus under E_4, $R^{\frac{\delta}{2}}(\hat{\imath}_k^*) > R_{\tau_{i'}}(i')$ holds whenever $\hat{\imath}_k^*$ is rejected. We assume this for the rest of the proof.

The remainder of this proof handles iterations $0 \le k \le K-1$, but the arguments transfer for the final CAPSANDRUNS call case by writing γ and $\hat{\imath}_{-1}^*$ instead of γ_k and $\hat{\imath}_k^*$. We investigate the two possible cases:

- If $\hat{\imath}_k^*$ is not rejected by PRECHECK, then under E_1 by Lemma 8 in [35], there is an i in the set of configurations CAPSANDRUNS is called with, that will not be rejected by QUANTILEEST, and for which $R_{\tau_i}(i) \le R^{\frac{\delta}{2}}(\hat{\imath}_k^*) \le \mathrm{OPT}^{\gamma_k}_{\delta/2}$.
- If $\hat{\imath}_k^*$ is rejected by PRECHECK, i' is a configuration not rejected by QUANTILEEST (as it set T), for which $R_{\tau_{i'}}(i') \le R^{\frac{\delta}{2}}(\hat{\imath}_k^*) \le \mathrm{OPT}^{\gamma_k}_{\delta/2}$.

In either case, there is a configuration i not rejected by QUANTILEEST, for which $R_{\tau_i}(i) \le \mathrm{OPT}^{\gamma_k}_{\delta/2}$. Thus by Lemma 9, under E_1, there is a configuration i_k^* not rejected by QUANTILEEST or RUNTIMEEST for which $R_{\tau_{i_k^*}}(i_k^*) \le R_{\tau_i}(i) \le \mathrm{OPT}^{\gamma_k}_{\delta/2}$. ☐

Corollary 11. *Under E_4, the configuration returned by* IMPATIENTCAPSANDRUNS *is $(\varepsilon, \delta, \gamma)$-optimal.*

Proof. By Lemma 9 and Lemma 10, under E_4, the final CAPSANDRUNS call returns with a configuration I for which $R_{\tau_I}(I) \le (1+\varepsilon) R_{\tau_{i_*}}(i_*) \le (1+\varepsilon)\mathrm{OPT}^{\gamma}_{\delta/2}$. Under E_1, $R^{\delta}(I) \le R_{\tau_I}(I)$, so I is $(\varepsilon, \delta, \gamma)$-optimal. ☐

Corollary 12. *Under E_4, for all iterations $0 \le k \le K-1$, T is set by* QUANTILEEST *to at most $2\mathrm{OPT}^{\gamma_k}_{\delta/2}$, and the combined time spent by* QUANTILEEST *and* RUNTIMEEST *evaluating the configuration that has set T is bounded by $4b\mathrm{OPT}^{\gamma_k}_{\delta/2}$ when it sets T.*

Proof. Take i_k^* as in Lemma 10. Since i_k^* is not rejected in either QUANTILEEST or RUNTIMEEST, its b measurements in RUNTIMEEST will complete, and by [35, Lemma 4], under E_1, this measurement will be at most $2R_{\tau_{i_k^*}}(i_k^*) \le 2\mathrm{OPT}^{\gamma_k}_{\delta/2}$. T is thus set to at most this value. From the proof of [35, Lemma 5], the work spent by QUANTILEEST and RUNTIMEEST evaluating i_k^* is bounded by $4b\mathrm{OPT}^{\gamma_k}_{\delta/2}$ time. ☐

Corollary 13. *Suppose E_4 holds. Then for all iterations $0 \le k \le K-1$,* CAPSANDRUNS *performs at most $\tilde{\mathcal{O}}\left(b|\overline{\mathcal{N}}_k|\mathrm{OPT}^{\gamma_k}_{\delta/2}\right)$ work.*

Proof. By Corollary 12, under E_4, in iteration k, T is set to at most $2\mathrm{OPT}^{\gamma_k}_{\delta/2}$, after which by the proof of [35, Lemma 5], each configuration performs at most $\tilde{\mathcal{O}}\left(b\mathrm{OPT}^{\gamma_k}_{\delta/2}\right)$ work. Also by

Corollary 12, the work performed by the configuration that set T to this value in iteration k is upper bounded by $4b\mathrm{OPT}^{\gamma_k}_{\delta/2}$. Since configurations are run in parallel, all the other configurations performed at most this amount of work in the meantime. Thus in total CAPSANDRUNS performs at most $\tilde{\mathcal{O}}\left(b|\overline{\mathcal{N}}_k|\mathrm{OPT}^{\gamma_k}_{\delta/2}\right)$ work in iteration k. $\square$

Lemma 14. *There is an event E_5 such that $\Pr(E_5) \geq 1-\zeta$, and under E_5, E_1, and E_4, for all iterations $k \in [-1, K-2]$, the number of configurations not rejected by* PRECHECK *can be bounded as*

$$|\overline{\mathcal{N}}_k| \leq (\log(K/\zeta)+1)\left[F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})\frac{1}{\gamma_k} + \sqrt{2F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})\frac{1}{\gamma_k}} + \frac{2}{3}\right],$$

where $F(x) = \Pr_{i\sim\Pi}(R^{0.35}(i) \leq x) + 4\zeta/K$.

Proof. Note that for the first call of PRECHECK, with $k = K-1$, PRECHECK returns its input without any modification, so $|\mathcal{M}'| = |\mathcal{M}|$. For the rest of the calls, $-1 \leq k < K-1$.

Denoting by B_i the indicator whether configuration $i \in \mathcal{N}_k$ is accepted by PRECHECK. Since elements of $\mathcal{N}_k$ are independent and identically distributed random variables, and there are no interactions between configurations being evaluated by PRECHECK, the outcomes B_i of PRECHECK are also independent and identically distributed. By Lemma 8, under E_1, PRECHECK rejects a configuration i if $R^{0.35}(i) \geq 19T$ with probability at least $1-4\zeta/K$, so the probability of reject is at least $\Pr_{i\sim\Pi}(R^{0.35}(i) \geq 19T \,|\, T)(1-4\zeta/K) \geq \Pr_{i\sim\Pi}(R^{0.35}(i) \geq 19T \,|\, T) - 4\zeta/K = 1-F(19T)$, so the conditional probability of accept is at least $F(19T)$. The number of configurations not accepted is $\sum_{i\in\mathcal{N}_k} B_i$. By the Bernstein inequality,

$$\Pr\left[\sum_{i\in\mathcal{N}_k} B_i \geq F(19T)|\mathcal{N}_k| + \sqrt{2F(19T)|\mathcal{N}_k|\log\frac{K}{\zeta}} + \frac{2}{3}\log\left(\frac{K}{\zeta}\right) \,\Big|\, T\right] \leq \frac{\zeta}{K},$$

so by a union bound over $k \in [-1, K-2]$, this is true for all iterations under an event E_5 with probability at least $1-\zeta$. By Lemma 6, $|\mathcal{N}_k| \leq \log(K/\zeta)/\gamma_k + 1$. By Corollary 12, under E_4, $T \leq 2\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}$ when PRECHECK is run for iteration k. Making these substitutions and reordering the terms gives the result. $\square$

Lemma 15. *For iterations $-1 \leq k < K-1$, under E_4,* PRECHECK *runs for at most* $10\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}\left\lceil 32.1\log\left(\frac{2K}{\zeta}\right)\right\rceil(\log(K/\zeta)/\gamma_k+1)$ *time.*

Proof. By Corollary 12, under E_4, $T \leq 2\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}$ when PRECHECK is run for iteration k. Phase I of PRECHECK is aborted when the total runtime reaches $1.9Tb' \leq 3.8\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}b'$. Phase II of PRECHECK is aborted when the total Phase II runtime exceeds $2.99Tb' \leq 5.98\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}b'$. This abort only happens after the last run, which takes at most τ' time, where τ' is measured in Phase I of PRECHECK. Because of the way τ' is calculated by Phase I, at least $\lfloor 0.2b' \rfloor$ instances were running up until τ' time, which took $\lfloor 0.2b' \rfloor \tau' \leq 1.9Tb'$ time. For any valid setting of ζ, $\lfloor 0.2b' \rfloor \geq 0.19b'$, so $\tau' \leq 10T \leq 20\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2} \leq 0.21\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}b'$, so the work of PRECHECK for each configuration is upper bounded by $(3.8+5.98+0.21)\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}b' < 10\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}b'$. Multiplying this by the number of configurations $|\mathcal{N}_k|$ PRECHECK evaluates, and using Lemma 6, the total work of PRECHECK is bounded by $10\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2}b'(\log(K/\zeta)/\gamma_k+1)$. $\square$

Proof of Theorem 1. Suppose E_4 and E_5 hold. By a union bound, taking also into account the lower bounds on the probabilities of these events and for events E_1, E_2, and $E_{3,k,i^*_k,j}$ (given by Remark 3, Lemma 4, Lemma 8, Lemma 14), we have $\Pr(E_1 \cap E_2 \cap E_4 \cap E_5) \geq 1-12\zeta$. By Corollary 11, under these events, the configuration returned by IMPATIENTCAPSANDRUNS is $(\varepsilon, \delta, \gamma)$-optimal.

Next we consider the runtime of IMPATIENTCAPSANDRUNS. For iteration $k = K-1$, E_1, E_2, and E_4, by Corollary 13 and Lemma 6, the runtime of CAPSANDRUNS is $\tilde{\mathcal{O}}\left(b\mathrm{OPT}^{\gamma_k}_{\delta/2}/\gamma_{K-1}\right)$. For iterations $0 \leq k < K-1$, by Corollary 13, the runtime of CAPSANDRUNS in iteration k is upper bounded as $\tilde{\mathcal{O}}\left(b|\overline{\mathcal{N}}_k|\mathrm{OPT}^{\gamma_{K-1}}_{\delta/2}\right)$. Using the bound $|\overline{\mathcal{N}}_k| = \tilde{\mathcal{O}}\left(F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})/\gamma_k\right)$ given by Lemma 14 the work by CAPSANDRUNS in iteration k is bounded by $\tilde{\mathcal{O}}\left(bF(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})\mathrm{OPT}^{\gamma_k}_{\delta/2}/\gamma_k\right)$.

For the final CAPSANDRUNS call, the total work performed by CAPSANDRUNS would only increase if we didn't do any work on any configurations before, in other words, if we restarted CAPSANDRUNS with the input configurations $\overline{\mathcal{N}}$. By this idea we can upper bound the total work of the final CAPSANDRUNS call using [35, Theorem 1], which states that under E_1, the total work of a restarted CAPSANDRUNS with input configurations $\overline{\mathcal{N}}$ is at most $\tilde{\mathcal{O}}\left(|\overline{\mathcal{N}}|\frac{1}{\varepsilon^2\delta}\min_{i\in\overline{\mathcal{N}}} R^{\frac{\delta}{2}}(i)\right)$, which is a simplified form of the problem-dependent bound (1) in [35]. By Lemma 10, $\min_{i\in\overline{\mathcal{N}}} R^{\frac{\delta}{2}}(i) \leq \mathrm{OPT}^{\gamma}_{\delta/2}$, and by Lemma 14, $|\overline{\mathcal{N}}| = \tilde{\mathcal{O}}\left(F(38\mathrm{OPT}^{\gamma}_{\delta/2})/\gamma\right)$. Plugging these in the bound we get that the final CAPSANDRUNS takes $\tilde{\mathcal{O}}\left(\mathrm{OPT}^{\gamma}_{\delta/2}F(38\mathrm{OPT}^{\gamma}_{\delta/2})\frac{1}{\varepsilon^2\delta\gamma}\right)$ time.

Now we turn our attention to bounding the work done in PRECHECK. By Lemma 15, under E_1, for all the iterations, and including the final PRECHECK call, the total work is $\tilde{\mathcal{O}}\left(\sum_{k=0}^{K-1}\mathrm{OPT}^{\gamma_k}_{\delta/2}/\gamma_k\right)$.

Adding all this work up, noting that $b = \tilde{\mathcal{O}}(1/\delta)$, we get that under E_4 and E_5, the total work performed by IMPATIENTCAPSANDRUNS is

$$\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2\delta\gamma}\mathrm{OPT}^{\gamma}_{\delta/2}F(38\mathrm{OPT}^{\gamma}_{\delta/2}) + \sum_{k=0}^{K-2}\frac{1}{\gamma_k}\mathrm{OPT}^{\gamma_k}_{\delta/2}\left(1+F(38\mathrm{OPT}^{\gamma_{k+1}}_{\delta/2})/\delta\right) + \frac{1}{\delta\gamma_{K-1}}\mathrm{OPT}^{\gamma_{K-1}}_{\delta/2}\right).$$

□

B Runtime bound analysis with exponential distributions

To better understand the runtime bound in Theorem 1, consider a scenario where the runtime of each configuration follows an exponential distribution. Such scenarios are realistic and motivated by practical applications: roughly speaking, many solvers for NP-hard problems (e.g., SAT) proceed by initializing with a random seed and, if they fail, try again with another random seed. The runtimes of such solvers will follow approximately exponential distributions, due to being essentially memoryless. To make the example concrete, suppose that the mean runtime for each configuration is distributed uniformly between A and $A+B$, denoted by $U(A, A+B)$, for some $A, B > 0$. Here A can be thought of as a small runtime associated with the cost of starting the run of any configuration on any problem instance, and B as the maximum "true" mean runtime of the configurations.

We can simplify the runtime bound of Theorem 1 with this assumption. The best γ_k fraction of the configurations have mean $A+B\gamma_k$ so $\mathrm{OPT}^{\gamma_k}_{\delta/2} \leq A+B\gamma_k$. Furthermore, for a configuration i with mean $\frac{1}{\lambda}$, $R_\tau(i) = \frac{1}{\lambda}\left(1-e^{-\lambda\tau}\right)$ for any runtime cap τ. Substituting $\tau = t_\delta(i)$, noting that the probability of running over the cap is δ so $e^{-\lambda\tau} = \delta$, we have $R^\delta(i) = \frac{1}{\lambda}(1-\delta)$. Similarly, $R^{0.35}(i) = 0.65\frac{1}{\lambda}$. Then $F(38\mathrm{OPT}^{\gamma_k}_{\delta/2}) - 4\zeta/K = \Pr_{i\sim\Pi}(R^{0.35}(i) \leq 38\mathrm{OPT}^{\gamma_k}_{\delta/2}) \leq \Pr_{\frac{1}{\lambda}\sim U(A,A+B)}(0.65\frac{1}{\lambda} \leq 38(A+B\gamma_k)) \leq \Pr_{\frac{1}{\lambda}\sim U(A,A+B)}(\frac{1}{\lambda} \leq 58.5(A+B\gamma_k)) \leq \frac{58.5(A+B\gamma_k)-A}{B} = \mathcal{O}(\gamma_k + \frac{A}{B})$. This bounds $F(38\mathrm{OPT}^{\gamma_k}_{\delta/2}) - 4\zeta/K$. The extra $4\zeta/K$ is insignificant, as the failure probability ζ can simply be chosen to be $\mathcal{O}(\varepsilon^2\delta)$ with only additional logarithmic factors in the runtime as a result, and then any term multiplied by ζ in the runtime bound disappears in the $\tilde{\mathcal{O}}(\cdot)$ notation. Substituting the bound on F and $\mathrm{OPT}^{\gamma_k}_{\delta/2}$, assuming a choice of ζ as above, the runtime bound (Eq. (1)) becomes

$$\tilde{\mathcal{O}}\left((A+B\gamma)\frac{1}{\varepsilon^2\delta\gamma}\cdot\left(\gamma+\frac{A}{B}\right) + \sum_{k=0}^{K-2}(A+B\gamma_k)\frac{1}{\gamma_k}\left(1+\frac{\left(\gamma_k+\frac{A}{B}\right)}{\delta}\right) + (A+B\gamma_{K-1})\frac{1}{\delta\gamma_{K-1}}\right)$$
$$= \tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2\delta}\left(\frac{A^2}{B\gamma}+A+B\gamma\right)\gamma + \frac{1}{\delta}\left(\frac{A}{\gamma_{K-1}}+B\right)+\frac{A}{\gamma}\right),$$

where the K term disappears in the $\tilde{\mathcal{O}}(\cdot)$ notation as $K \leq \log_2(\frac{1}{\gamma})$. Contrasting this with the typical runtime bound $\tilde{\mathcal{O}}\left(\frac{1}{\varepsilon^2\delta\gamma}\mathrm{OPT}^{\gamma}_{\delta/2}\right) = \tilde{\mathcal{O}}\left(\left(\frac{A}{\gamma}+B\right)\frac{1}{\varepsilon^2\delta}\right)$ of previous works, we see the main term

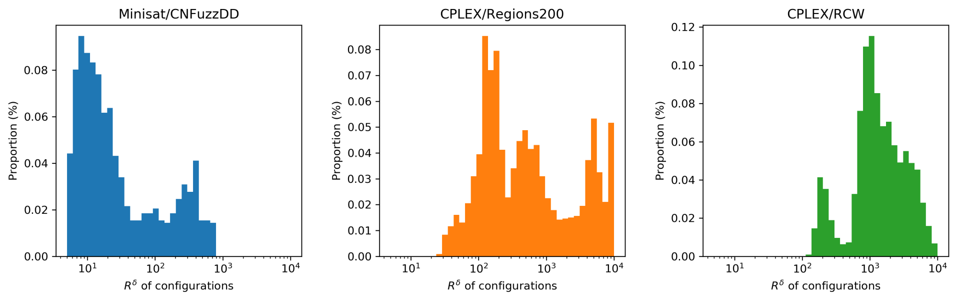

Figure 2: Distribution of δ-capped mean runtime of the sampled configurations, with $\delta = 0.1$. For Minisat/CNFuzzDD, many configurations are close to the optimal one, whereas for CPLEX/Regions200 and CPLEX/RCW, many configurations are significantly worse than the optimal one. Consequently, PRECHECK is able to discard more configurations in the latter two scenarios.

(the one multiplied by $\frac{1}{\varepsilon^2\delta}$) is reduced by a factor of $\max\{\gamma, \frac{A}{B}\}$. The rest of the terms have no dependence on ε and are indeed always much smaller than the typical runtime bound for other works: $\frac{A}{\delta\gamma_{K-1}}$ is smaller than the first term provided that K is chosen to be large enough for γ_{K-1} not to be too small, $\frac{B}{\delta}$ does not depend on the number of configurations evaluated, and $\frac{A}{\gamma}$ is associated with having to evaluate about $\frac{1}{\gamma}$ number of configurations, and this term could not scale better than with the minimum runtime A.

C Details of Experiments

We followed the experimental setup of [35]. Runs were pre-computed and then queried from a simulation environment in which they can be stopped and resumed. In a scenario where this is not possible (e.g., due to memory constraints when performing real runs) the experiments can still be implemented by restarting runs from scratch with doubling cap times, resulting in at most a factor of 2 slowdown.

Parameter values Experiments on all datasets were done with $(\varepsilon, \delta) = (0.05, 0.1)$ and varying $\gamma \in \{0.01, 0.02, 0.05\}$. For each configurator, ζ was set so that the total failure probability is 0.05. The hyperparameter K was set such that $0.25 < \gamma 2^{K-1} \leq 0.5$. This is a somewhat arbitrary choice, but was made so that values of γ_k were neither too big to be trivial, nor too small to be computationally prohibitive.

EPM Setup We used the provided generators for Regions200 and RCW to produce as many new random instances as needed, which were pre-processed using the feature extractors provided with the EPM.[9] Runtime-related features were dropped since they are machine-dependent. We then used the provided configurations and runtime data[10] to train the EPM model, using the parameters suggested in [12]. Finally, the trained EPM was used as a surrogate model to provide runtimes on future scenarios. We can query this model to produce a runtime estimate for any configuration-instance pair. New configurations were sampled by uniformly choosing a value for each parameter of CPLEX. A new instance was then generated and the pair was given to the EPM. Note that ICAR examined more configurations than CAR did. For consistency, sampling was done so that the configurations seen by CAR were a subset of those seen by ICAR.

Datasets Description

- Minisat/CNFuzzDD is a SAT scenario based on the minisat solver, with 6 parameters, applied to the CNFuzzDD[11] instances. The benchmark dataset we use is the same as in [34, 35, 24].

[9] http://www.cs.ubc.ca/labs/beta/Projects/EPMs/
[10] https://www.ml4aad.org/automated-algorithm-design/performance-prediction/epms/
[11] http://fmv.jku.at/cnfuzzdd/

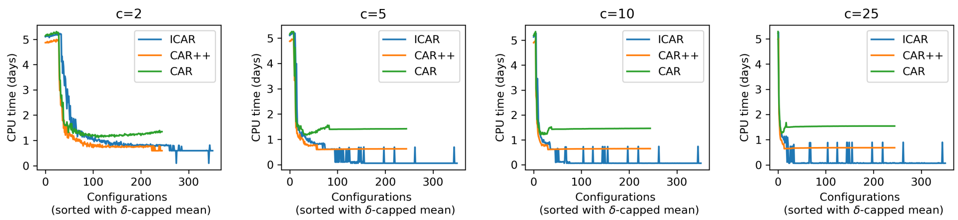

Figure 3: As the proportion of configurations that are far from the optimal gets larger (i.e., as c gets larger), the CPU runtime of CAR was more dominated by the work spent on bad configurations, while ICAR was able to drop more bad configurations with its PRECHECK mechanism.

- CPLEX/Regions200 is a MIP scenario based on the CPLEX, an interger programming solver, applied to the combinatorial auction winner determination problem. There are 74 parameters for CPLEX. The benchmark dataset is generated with the EPM described above.
- CPLEX/RCW is a MIP scenario based on the CPLEX solver, applied to Red-cockaded Woodpecker conservation problem. The configuration space is the same as CPLEX/Regions200. The benchmark dataset is generated with the EPM described above.

D ImpatientCapsAndRuns with varying parameters

We also compared the performance of ICAR and CAR++ with varying values of ε and δ (with fixed $\gamma = 0.02$ and failure probability $\zeta = 0.05$). The speedup (computed as the ratio of the runtimes) achieved by ICAR over CAR++ is reported in Table 2. As we can see, the speedup was fairly stable across a range of ε and δ that a user might be likely to care about.

Table 2: Speedup achieved by ICAR over CAR++ for various values of ε and δ. Values greater than one indicate ICAR is faster.

	Minisat/CNFuzzDD				CPLEX/Regions200				CPLEX/RCW			
δ	0.025	0.05	0.075	0.1	0.025	0.05	0.075	0.1	0.025	0.05	0.075	0.1
$\varepsilon = 0.025$	0.80	0.83	0.71	0.95	2.76	2.44	2.15	2.03	2.27	1.99	1.83	1.63
$\varepsilon = 0.05$	0.83	0.84	0.78	0.92	2.90	2.54	2.24	2.06	2.54	2.21	1.96	1.80
$\varepsilon = 0.075$	0.84	0.86	0.82	0.92	2.94	2.58	2.28	2.09	2.66	2.30	2.03	1.85
$\varepsilon = 0.1$	0.85	0.87	0.85	0.93	2.97	2.60	2.29	2.11	2.73	2.36	2.08	1.88

E Synthetic Experiments

To better understand how well ICAR can exploit a needle-in-a-haystack scenario, we examined its performance on synthetic data. In this way we could choose each configuration's true mean, and thus control their distribution. The runtimes of each configuration were sampled from an exponential distribution, with the means being uniformly chosen from the interval $[\mathrm{OPT},\ c \cdot \mathrm{OPT}]$. We tend to think that real algorithm runtimes do look somewhat exponential, and there is justification for this, at least in certain cases [14, 16, 15, 25].

We ran the simulation for $c \in \{2, 5, 10, 25\}$. The larger the value of c, the more configurations will tend to be far from the best one, creating more and more of a "needle-in-a-haystack" scenario. We used $(\varepsilon, \delta, \gamma) = (0.05, 0.1, 0.02)$ and failure probability of 0.05, as before. Table 3 shows the total CPU time to find a $(0.05, 0.1, 0.02)$-optimal configuration for the range of c values. The degree to which ICAR outperformed CAR increases as c increases, as expected. We see that PRECHECK was able to reject a greater proportion of configurations when many tended to be far from optimal (large c).

Figure 3 shows the CPU time spent on each configuration, sorted by the δ-capped mean runtime. When $c = 2$, ICAR rejected very few configurations in PRECHECK, but as c increases we can see a greater proportion of configurations were being run for minimal time compared to CAR++. Furthermore, the runtime of CAR and CAR++ became dominated by the runs on the bad configurations, as we can see from the area under the curve.

	Total CPU Time (days)				Number of Configurations Before/After Precheck			
	$c = 2$	$c = 5$	$c = 10$	$c = 25$	$c = 2$	$c = 5$	$c = 10$	$c = 25$
ICAR	505 (23)	187 (18)	113 (17)	92 (27)	351 / 349	351 / 114	351/ 54	351/ 27
CAR++	344 (24)	214 (12)	193 (20)	195 (31)	245	245	245	245
CAR	447 (21)	384 (15)	380 (35)	406 (62)	245	245	245	245

Table 3: Total CPU time in days to find a $(0.05, 0.1, 0.02)$-optimal configuration in the synthetic datasets, and the number of configurations before and after precheck. For CAR and CAR++, the number of configurations sampled is reported. CAR++ is the improved version CAR arising from more careful analysis. Error terms are standard deviations over five runs.