We thank all the reviewers for their helpful feedback. A main point raised was the need for a better comparison between Structured Procrastination (SP) and Structured Procrastination with Confidence (SPC); we will dedicate more space to this. In brief: SPC uses a novel form of lower confidence bound as an indicator of the quality of a particular configuration, while SP simply uses that configuration's sample mean. The consequence is that SPC spends much less time running poorly performing configurations, as other configurations quickly appear better and receive more attention. Since SPC spends less time running bad configurations, we are also able to show an improved runtime bound for it over SP. As Reviewer 3 points out, rather than having an additive term of the form $O(\epsilon^{-2}\delta^{-1})$ for each of n configurations considered, the improved bound has a term of the form $O(\epsilon_i^{-2}\delta_i^{-1})$, for each configuration i that is not (ϵ, δ)-optimal, where $\epsilon_i^{-2}\delta_i^{-1}$ is as small as possible. This can be a significant improvement in runtime in cases where many configurations being considered are far from being (ϵ, δ)-optimal. Using this confidence bound in place of the mean also requires a novel proof technique which leverages the theory of empirical processes.

The following concrete example illustrates the gains SPC can offer over SP. Suppose that there are just two configurations: one that always finishes in 100 milliseconds on every problem instance and another that always takes 1000 milliseconds. Suppose furthermore that κ_0—the minimum time it potentially takes to run a configuration—is equal to one millisecond. SP, configured with approximation parameter $\epsilon = 0.01$ and failure probability $\zeta = 0.1$, will set the initial queue size of each configuration to be at least[1] 7500, because the queue size is initialized with a value that is at least $12\epsilon^{-2}\ln(3\beta n/\zeta)$, where $\beta \geq 10$ is the logarithm of the ratio of maximum to minimum potential running times, and $n = 2$ is the number of configurations. It will run each configuration 7500 times with a timeout of 1ms, then it will run each of them 7500 times with a timeout of 2ms, then 4ms, and so on, progressively doubling the timeout until it reaches 128ms. At that point it exceeds 100ms, so the first configuration will solve all of the instances in its queue. However, for the first $2 \cdot 7500 \cdot (1+2+4+\cdots+64) = 1.9 \times 10^6$ milliseconds of running the algorithm—more than half an hour—essentially nothing happens: SP obtains no evidence of the superiority of the first configuration.

In contrast, SPC maintains more modest queue sizes, and thus runs each configuration on fewer instances before running them with a timeout of 128ms, at which point it can distinguish between the two configurations. In our example, during the first 5000 iterations of SPC, the size of each configuration's instance queue is at most 400. This is because $r_i \leq t$, and $t \leq 5000$, so $q_i \leq 25\log(5000\log(5000)) < 400$. Further, observe that 5000 iterations is sufficient for SPC to attempt to run both configurations on some instance with a cutoff of 128ms, since each configuration will first run at most 400 instances with cutoff 1ms, then at most 400 instances with cutoff 2ms, and so on. Continuing up to 64ms, for both configurations, takes a total of $2 \cdot \log(64) \cdot 400 = 4800 < 5000$ iterations. Thus, it takes at most $2 \cdot 400 \cdot (1+2+4+...+64) = 101,600$ milliseconds (less than two minutes) before SPC runs each configuration on some instance with cutoff time 128ms. We see that SPC requires significantly less time—in this example, almost a factor of 20 less time—to reach the point where it can distinguish between the two configurations.

We agree with Reviewer 3 that it is important for SPC to return the parameters for which its optimality guarantee holds; we will explain how to do this in the paper. In brief: it is not possible to return every (ϵ, δ) pair for which a guarantee is given because there are infinitely many such pairs. The original SP algorithm takes ϵ as a parameter and returns the corresponding δ for which the guarantee holds; we can do the same with SPC. We have also annotated our experimental results with the δ guaranteed as a function of time; see Figure 1.

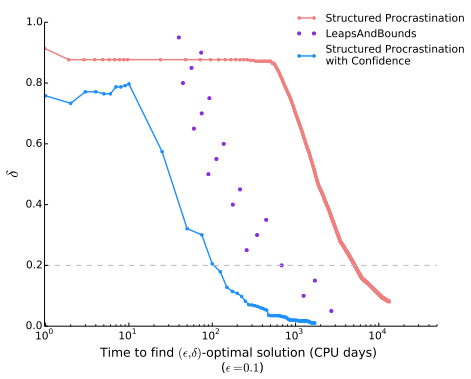

Figure 1: δ for which optimality holds, as a function of runtime.

We experimented only with the MiniSAT data of Weisz et al (2018) since this is the dataset considered by the previous literature on provably near-optimal algorithm configuration. We are, however, also eager to see the results of more comprehensive experiments, including for the case of many configurations (i.e., continuous parameters). This is no small task, requiring significant amounts of coding and compute resources. We thus leave this important step for future work. We will of course make code available to reproduce our experiments as well.

Finally, we will of course fix all typos and other minor issues identified in the reviews.

[1]The exact queue size depends on the number of active instances, but this lower bound suffices for our example.