Appendix

A Proofs

A.1 Proof of Theorem 1

Proof 1 *Using the definition of $p(\boldsymbol{\theta}|f, \mathcal{D})$, the probability of improvement is computed as:*

$$p(f \leq f^\gamma \mid \boldsymbol{\theta}, \mathcal{D}) = \int_{-\infty}^{f^\gamma} p(f|\boldsymbol{\theta}, \mathcal{D})df = \int_{-\infty}^{f^\gamma} \frac{p(\boldsymbol{\theta}|f, \mathcal{D})p(f|\mathcal{D})}{p(\boldsymbol{\theta}|\mathcal{D})}df = \frac{l(\boldsymbol{\theta}|\mathcal{D}^{(l)})}{p(\boldsymbol{\theta}|\mathcal{D})} \int_{-\infty}^{f^\gamma} p(f|\mathcal{D})df$$

Since the expected improvement for TPE is computed as:

$$\mathrm{EI}[\boldsymbol{\theta}|\mathcal{D}] = \frac{l(\boldsymbol{\theta}|\mathcal{D}^{(l)})}{p(\boldsymbol{\theta}|\mathcal{D})} \int_{-\infty}^{f^\gamma} (f^\gamma - f)p(f|\mathcal{D})dy$$

and both equations have the common part $l(\boldsymbol{\theta}|\mathcal{D}^{(l)})/p(\boldsymbol{\theta}|\mathcal{D})$, this part cancels out when we take the ratio. For this reason, the ratio of the two equations is computed as:

$$\frac{\int_{-\infty}^{f^\gamma} (f^\gamma - f)p(f|\mathcal{D})df}{\int_{-\infty}^{f^\gamma} p(f|\mathcal{D})df} = \text{const w.r.t. } \boldsymbol{\theta}, \tag{8}$$

where, since we assume that the support of $p(f \leq f^\gamma|\boldsymbol{\theta}, \mathcal{D})$ covers the whole domain $\boldsymbol{\Theta}$, i.e. $\forall \boldsymbol{\theta} \in \boldsymbol{\Theta}, p(f \leq f^\gamma|\boldsymbol{\theta}, \mathcal{D}) \neq 0$ and f is Lebesgue integrable, i.e. the expectation of f exists and $\int |f|d\mu < \infty$, both numerator and denominator always take a positive finite value and thus the LHS of Eq. (8) takes a finite positive constant value.

A.2 Proof of Theorem 2

To prove Theorem 2, we first show two lemmas.

Lemma 1 *Given a Γ-feasible domain with constraint thresholds of $c_i^\star$ for all $i \in [1, C]$, each constraint satisfies*

$$\forall i \in [1, C], \gamma_{c_i^\star} \geq \Gamma.$$

Proof 2 *Let the feasible domain for the i-th constraint be $\boldsymbol{\Theta}'_{c_i^\star} = \{\boldsymbol{\theta} \in \boldsymbol{\Theta}|c_i \leq c_i^\star\}$. Then the feasible domain is $\boldsymbol{\Theta}' = \bigcap_{i=1}^C \boldsymbol{\Theta}'_{c_i}$. Since $\boldsymbol{\Theta}'_{c_i^\star}$ is a measurable set by definition and $\boldsymbol{\Theta}' \subseteq \boldsymbol{\Theta}'_{c_i^\star}$ holds, $\Gamma/\gamma_{c_i^\star} = \mu(\boldsymbol{\Theta}')/\mu(\boldsymbol{\Theta}'_{c_i}) \leq 1$ holds. Γ is a positive number, so $\gamma_{c_i^\star} \geq \Gamma$.*

Lemma 2 *The domain is $(\Gamma = 1)$-feasible domain iff:*

$$\forall i \in [1, C], \gamma_{c_i^\star} = 1.$$

Proof 3 *Suppose $\gamma_{c_i^\star} < 1$ for some $i \in [1, C]$, we immediately obtain $\Gamma \leq \gamma_{c_i^\star} < 1$ from Lemma 1 and thus the assumption does not hold. For this reason, $\gamma_{c_i^\star} \geq 1$ for all $i \in [1, C]$ and since $\gamma_{c_i^\star} \leq 1$ by definition, $\gamma_{c_i^\star} = 1$ for all $i \in [1, C]$.*

Using Lemma 2, we prove Theorem 2.

Proof 4 *From the assumption, $\Gamma = 1$ holds. Then $\gamma_{c_i^\star} = 1$ holds for all $i \in [1, C]$ from Lemma 2. By replacing $\gamma_{c_i^\star}$ with 1 in Eq. (6), we obtain the following:*

$$\mathrm{ECI}_{f^\gamma}[\boldsymbol{\theta}|\boldsymbol{c}^\star, \mathcal{D}] \propto \left(\gamma + (1-\gamma)\frac{g(\boldsymbol{\theta}|\mathcal{D}^{(g)})}{l(\boldsymbol{\theta}|\mathcal{D}^{(l)})}\right)^{-1} \prod_{i=1}^{C} 1 = \left(\gamma + (1-\gamma)\frac{g(\boldsymbol{\theta}|\mathcal{D}^{(g)})}{l(\boldsymbol{\theta}|\mathcal{D}^{(l)})}\right)^{-1} \propto \mathrm{EI}_{f^\gamma}[\boldsymbol{\theta}|\mathcal{D}] \tag{9}$$

Since $\mathrm{ECI}_{f^\gamma}[\boldsymbol{\theta}|\boldsymbol{c}^\star, \mathcal{D}], EI_{f^\gamma}[\boldsymbol{\theta}|\mathcal{D}] \in \mathbb{R}_{\geq 0}$ holds by definition, Eq. (9) indicates the statement of the theorem.

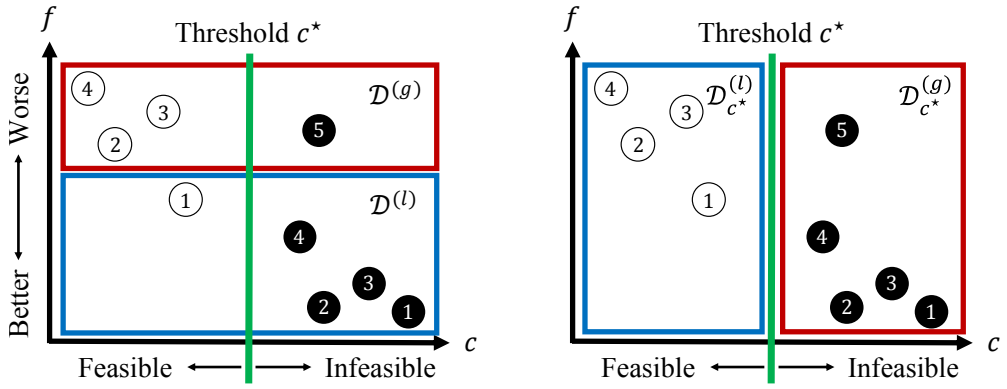

Figure 3: The conceptual visualizations of the split algorithm for the objective (left) and for each constraint (right). The black circles in the figures represent infeasible solutions and the white circles represent feasible solutions. The numberings for white and black objects stand for the ranking of the objective value in feasible and infeasible domains, respectively. The configurations enclosed by the red rectangle belong to the bad group and those enclosed by the blue rectangle belong to the good group.

A.3 Proof of Theorem 3

Proof 5 *Let the priority factor of the i-th constraint over the j-th constraint be $P \in \mathcal{P}_{i,j}(\boldsymbol{\theta})$ such that the following holds:*

$$\mathcal{P}_{i,j}(\boldsymbol{\theta}) = \left\{ P \middle| \left(\gamma_{c_i^\star} + (1-\gamma_{c_i^\star})(r_i(\boldsymbol{\theta}))^{-1}\right)^{-1} - \left(\gamma_{c_j^\star} + (1-\gamma_{c_j^\star})(r_i(\boldsymbol{\theta})P)^{-1}\right)^{-1} \geq 0, P \in \mathbb{R}_+ \cup \{\infty\} \right\}. \tag{10}$$

Then Theorem 3 will be rephrased into the following:

Proposition 1 *Under the given conditions in Theorem 3, the supremum of the priority factor is:*

$$P_{i,j}(\boldsymbol{\theta}) = \sup_{P \in \mathcal{P}_{i,j}(\boldsymbol{\theta})} P = \lim_{\epsilon \to +0} \frac{1-\gamma_{c_j^\star}}{\max\{\epsilon, 1-\gamma_{c_i^\star} - r_i(\boldsymbol{\theta})\Delta\}}.$$

The supremum is achieved when the equality holds in the condition of Eq. (10). By transforming the condition, we obtain the following:

$$\begin{aligned}
\gamma_{c_i^\star} + (1-\gamma_{c_i^\star})(r_i(\boldsymbol{\theta}))^{-1} &\leq \gamma_{c_j^\star} + (1-\gamma_{c_j^\star})(r_i(\boldsymbol{\theta})P)^{-1} \\
\gamma_{c_i^\star} r_i(\boldsymbol{\theta})P + (1-\gamma_{c_i^\star})P &\leq \gamma_{c_j^\star} r_i(\boldsymbol{\theta})P + 1 - \gamma_{c_j^\star} \\
P(1-\gamma_{c_i^\star} - \Delta r_i(\boldsymbol{\theta})) &\leq 1 - \gamma_{c_j^\star}.
\end{aligned}$$

When $1-\gamma_{c_i^\star} - r_i(\boldsymbol{\theta})\Delta \leq 0$, since the LHS will be negative and the RHS will be positive, P can be infinitely large. Now, let's assume $1-\gamma_{c_i^\star} - r_i(\boldsymbol{\theta})\Delta > 0$. Then we obtain the following:

$$P \leq \frac{1-\gamma_{c_j^\star}}{1-\gamma_{c_i^\star} - r_i(\boldsymbol{\theta})\Delta} = P_{i,j}(\boldsymbol{\theta}).$$

When we assume the equality, Theorem 3 holds.

Note that the denominator of the RHS is $1-\gamma_{c_i^\star} - r_i(\boldsymbol{\theta})\Delta \leq 1-\gamma_{c_j^\star}$ because $\Delta = \gamma_{c_j^\star} - \gamma_{c_i^\star}$. For this reason, the supremum is always larger than or equal to 1.

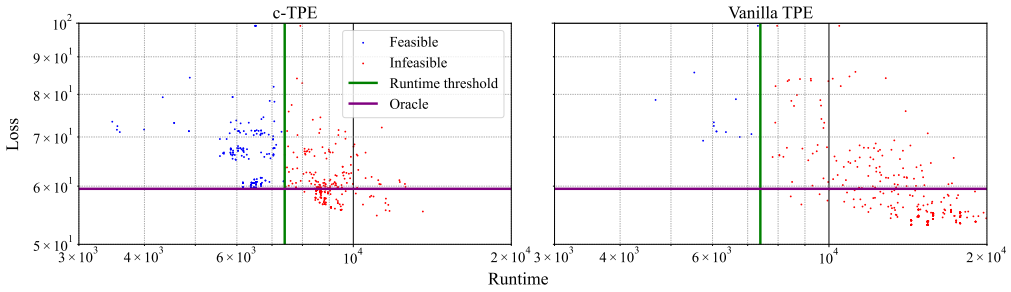

Figure 4: The visualization of the observations obtained by c-TPE (left) and the vanilla TPE (right) in the optimization of NAS-Bench-201 on ImageNet16-120 with $\gamma_{c_i^\star}^{\text{true}} = 0.1$. We include the observations in the latter half of each optimization to see the pure learning effect of each method and each optimization was run five times. Runtime threshold is chosen so that $\gamma_{c_i^\star}^{\text{true}} = 0.1$ will hold and oracle is the best loss value that can be achieved given a constraint value. The red dots are the observations that belong to the infeasible domain and the blue dots are the observations that belong to the feasible domain.

B Further details of the split algorithm

In this section, we describe the intuition and more details on how the split algorithm works.

B.1 Split criterion of objective

Figure 3 presents how to split observations into good and bad groups. The left figure shows the split for the objective. In this example there are $N = 9$ observations and thus we will include $\lceil \sqrt{N}/4 \rceil = \lceil \sqrt{9}/4 \rceil = 1$ feasible solution in $\mathcal{D}^{(l)}$. For this reason, we first need to find the feasible observation with the best objective value and the white-circled observation 1 in the figure is the corresponding observation in this example. Then we split observations at the white observation 1 along the horizontal axis and $\mathcal{D}^{(l)}$ and $\mathcal{D}^{(g)}$ are obtained. The reasons behind this modification are from the fact that observations with the best objective values are often, especially for tighter constraints, far from the feasible domain (e.g. the black-circled observations 1 and 3 in Figure 3) and it guarantees at least one feasible observation to be in the good group unless there is no feasible observation. For example, Figure 4 visualizes the observations by c-TPE and the vanilla TPE on ImageNet16-120 of NAS-Bench-201 with $\gamma_{c_i^\star}^{\text{true}} = 0.1$. As seen in the figure, there are many observations with better performance than the oracle that are far from the feasible domain in the result of the vanilla TPE. When c-TPE prioritizes only such observations, c-TPE ends up searching the infeasible domain. For this reason, we consider the splits by the number of feasible observations rather than greedy selections from all the observations. This selection guarantees the overlap of the domain of the better group and the feasible domain as long as we already have feasible observations. This property prevents c-TPE from spending many evaluations that are far from feasible domains, but with good objective values. Notice that when the whole domain is feasible, all the observations will be feasible and thus this selection naturally converges to the same behavior as the original TPE.

B.2 Split criterion of each constraint

The right figure of Figure 3 shows the split of each constraint. Note that for simplicity, we show the 1D example and abbreviate $c_i, c_i^\star, \mathcal{D}_{c_i^\star}^{(l)}, \mathcal{D}_{c_i^\star}^{(g)}$ as $c, c^\star, \mathcal{D}_{c^\star}^{(l)}, \mathcal{D}_{c^\star}^{(g)}$, respectively. As illustrated in the figure, we take the observations with constraint values less than $c^\star$ into $\mathcal{D}_{c^\star}^{(l)}$ and vice versa. When the observations in the feasible domain do not exist, we only take the observation with the best constraint value among all the observations into $\mathcal{D}_{c^\star}^{(l)}$ and the rest into $\mathcal{D}_{c^\star}^{(g)}$. This selection increases the priority of this constraint as mentioned in Theorem 3 and thus raises the probability of yielding feasible solutions quickly.

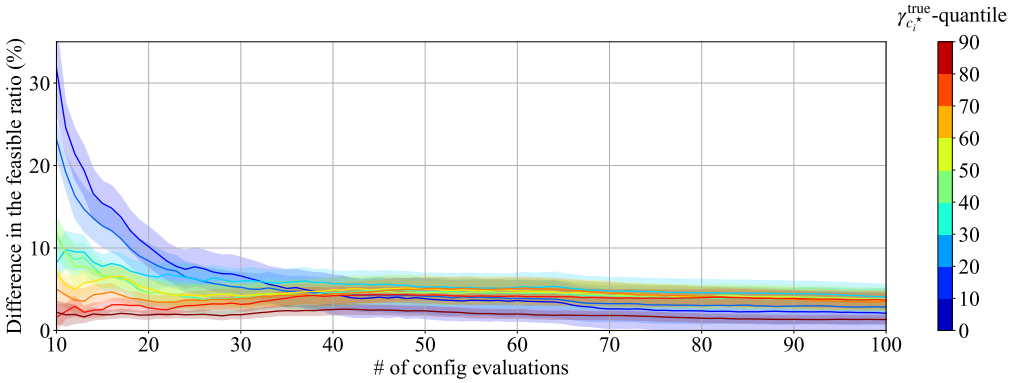

Figure 5: The effect of KA in the optimizations with a constraint for network size. The horizontal axis shows the number of evaluated configurations in optimizations and the vertical axis shows the difference in the cumulated ratio of feasible solutions between c-TPE and c-TPE with KA using 200 randomly sampled configurations. The weak-color bands show the standard error of mean values of 50 runs for 10 benchmarks.

C Integration of partial observations

In this section, we discuss the integration of partial observations for BO and we name the integration "Knowledge augmentation".

C.1 Knowledge augmentation

When some constraints can be precisely evaluated with a negligible amount of time compared to others, practitioners typically would like to use Knowledge augmentation (KA). For example, the network size of deep learning models is trivially computed in seconds while the final validation performance requires several hours to days. In this case, we can obtain many observations only for network size and augment the knowledge of network size prior to the optimization so that the constraint violations will be reduced in the early stage of optimizations.

To validate the effect of KA, all of the additional results in the appendix include the results obtained using c-TPE with KA. In the experiments, we augmented the knowledge only for network size and we did not include runtime as a target of KA because although runtime can be roughly estimated from a 1-epoch training, such estimations are not precise. However, practitioners can include such rough estimations into partial observations as long as they can accept errors caused by them.

C.2 Algorithm of knowledge augmentation

Algorithm 2 is the pseudocode of c-TPE with KA. We first need to specify a set of indices for cheap constraints $I = \{i_j\}_{j=1}^{C_p}$ where $C_p (< C)$ is the number of cheap constraints and I must be an element of the power set of $\{j\}_{j=1}^{C}$. In Lines 4–6, we first collect partial observations $\mathcal{D}_p$. Then we augment observations in Lines 16–19 if partial observations are available for the corresponding constraint. We denote the augmented set of observations $\mathcal{D}_{\text{aug}}$. When the acquisition function follows Eq. (5), the predictive models for each constraint are independently trained due to conditional independence. It enables us to introduce different amounts of observations for each constraint. Since c-TPE follows Eq. (5), we can employ KA. As discussed in Section 5, it is hard to apply KA to evolutionary algorithms due to their algorithm nature and KA causes a non-negligible bottleneck for GP-based BO as the number of observations grows.

C.3 Empirical results of knowledge augmentation

In this experiment, we optimized each benchmark with a constraint for network size, and constraints for runtime and network size. To see the effect, we measured how much KA increases the chance of drawing feasible configurations and tested the performance difference by the Wilcoxon signed-rank

Algorithm 2 c-TPE with knowledge augmentation

$N_{\text{init}}, N_s, N_p$ ▷ Control parameters
$I = \{i_j\}_{j=1}^{C_p}$ ▷ Indices of cheap constraints
$\mathcal{D}_p \leftarrow \emptyset, \mathcal{D} \leftarrow \emptyset$
for $n = 1, \ldots, N_p$ **do**
  Randomly pick $\boldsymbol{\theta}$
  $\mathcal{D}_p \leftarrow \mathcal{D}_p \cup \{(\boldsymbol{\theta}, c_{i_1}(\boldsymbol{\theta}), \ldots, c_{i_{C_p}}(\boldsymbol{\theta}))\}$
for $n = 1, \ldots, N_{\text{init}}$ **do**
  Randomly pick $\boldsymbol{\theta}$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{\theta}, c_1(\boldsymbol{\theta}), \ldots, c_C(\boldsymbol{\theta}))\}$
while Budget is left **do**
  $\mathcal{S} = \emptyset$
  Split $\mathcal{D}$ into $\mathcal{D}^{(l)}$ and $\mathcal{D}^{(g)}$, $\gamma \leftarrow |\mathcal{D}^{(l)}|/|D|$
  Construct $l(\boldsymbol{\theta}|\mathcal{D}^{(l)}), g(\boldsymbol{\theta}|\mathcal{D}^{(g)})$
  $\{\boldsymbol{\theta}_j\}_{j=1}^{N_s} \sim l(\boldsymbol{\theta}|\mathcal{D}^{(l)}), \mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{\theta}_j\}_{j=1}^{N_s}$
  for $i = 1, \ldots, C$ **do**
    if $i \in I$ **then**
      $\mathcal{D}_{\text{aug}} = \mathcal{D} \cup \mathcal{D}_p$
    else
      $\mathcal{D}_{\text{aug}} = \mathcal{D}$
    Split $\mathcal{D}_{\text{aug}}$ into $\mathcal{D}^{(l)}_{c_i^\star}$ and $\mathcal{D}^{(g)}_{c_i^\star}$, $\gamma_{c_i^\star} \leftarrow |\mathcal{D}^{(l)}_{c_i^\star}|/|D_{\text{aug}}|$
    Construct $l_{c_i^\star}(\boldsymbol{\theta}|\mathcal{D}^{(l)}_{c_i^\star}), g_{c_i^\star}(\boldsymbol{\theta}|\mathcal{D}^{(g)}_{c_i^\star})$
    $\{\boldsymbol{\theta}_j\}_{j=1}^{N_s} \sim l_{c_i^\star}(\boldsymbol{\theta}|\mathcal{D}^{(l)}_{c_i^\star}), \mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{\theta}_j\}_{j=1}^{N_s}$
  Pick $\boldsymbol{\theta}_{\text{opt}} \in \text{argmax}_{\boldsymbol{\theta} \in \mathcal{S}} \text{ECI}'_{f^\gamma}[\boldsymbol{\theta}|\boldsymbol{c}^\star, \mathcal{D}]$ ▷ Eq. (6)
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{\theta}_{\text{opt}}, f(\boldsymbol{\theta}_{\text{opt}}), c_1(\boldsymbol{\theta}_{\text{opt}}), \ldots, c_C(\boldsymbol{\theta}_{\text{opt}}))\}$

Table 3: The table shows (Wins/Loses/Ties) of c-TPE with KA against c-TPE for optimizations with different constraint levels. Bold numbers indicate $p < 0.05$ of the hypothesis "c-TPE is better than c-TPE with KA" by the Wilcoxon signed-rank test.

Quantiles	$\gamma^{\text{true}}_{c_i^\star} = 0.1$				$\gamma^{\text{true}}_{c_i^\star} = 0.5$				$\gamma^{\text{true}}_{c_i^\star} = 0.9$			
# of configs	50	100	150	200	50	100	150	200	50	100	150	200
Wins/Loses/Ties	**12/5/1**	**11/5/2**	7/5/6	6/6/6	6/12/0	5/11/2	7/6/5	5/5/8	9/9/0	10/6/2	8/5/5	6/9/3

test on 18 settings (9 benchmarks $\times$ 2 constraint choices). According to Figure 5, the tighter the constraint becomes, the more KA helps to obtain feasible solutions, especially in the early stage of the optimizations. Additionally, Table 3 shows the statistically significant speedup effects of KA in $\gamma^{\text{true}}_{c_i^\star} = 0.1$. Although KA did not exhibit the significant speedup in loose constraint levels, it did not deteriorate the optimization quality significantly. At the later stage of the optimizations, the effect gradually decays as c-TPE becomes competent enough to detect violations. In summary, KA significantly accelerates optimizations with tight constraints and it does not deteriorate the optimization quality in general, so it is practically recommended to use KA as much as possible.

D Additional results for Section 4.1

In this section, we show the additional results for Section 4.1. The main goal of those results is to show how robust c-TPE is over various levels of constraints. Note that we picked only network size as a cheap constraint and did not pick runtime as discussed in Appendix C and we used $N_p = 200$ throughout all the experiments.

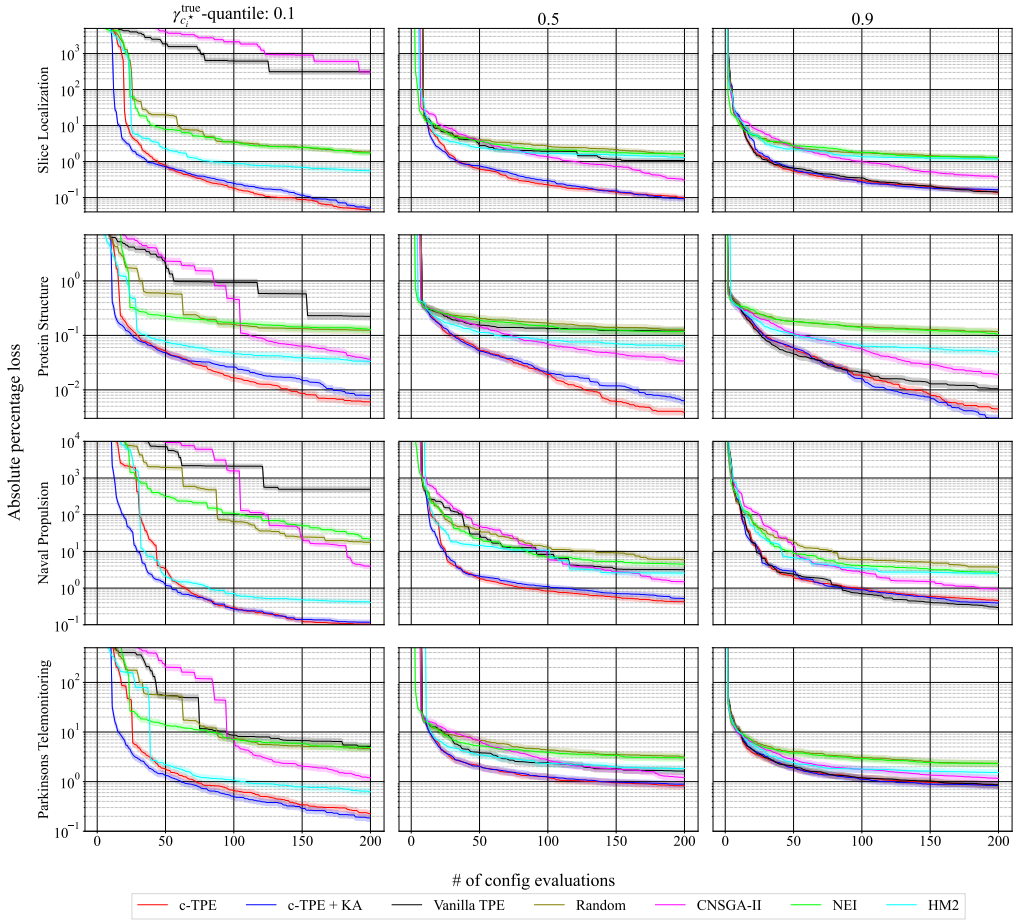

Figure 6: Figures show the performance curves on four benchmarks in HPOlib with a constraint of network size. We picked $\gamma_{c_i^\star}^{\text{true}} = 0.1$ (left), 0.5 (center), 0.9 (right). The horizontal axis shows the number of evaluated configurations in optimizations and the vertical axis shows the absolute percentage error in each experiment.

D.1 Performance over number of evaluations for all the datasets

The performance in each figure was measured by absolute percentage loss that is computed by:

$$\frac{f_{\text{observed}} - f_{\text{oracle}}}{f_{\text{oracle}}}$$

Note that f_{oracle} is collected by checking all the available validation metric values, which satisfy given constraints, in each benchmark.

D.1.1 Results on HPOlib

Figures 6, 7, and 8 show the time evolution of absolute percentage loss of each optimization method on HPOlib, NAS-Bench-101, and NAS-Bench-201 with the $\gamma_{c_i^\star}^{\text{true}}$-quantile of 0.1, 0.5, and 0.9 for the network size constraint.

For tighter constraint settings, c-TPE outperformed other methods and KA accelerated c-TPE in the early stage. For looser constraint settings, CNSGA-II improves its performance in the early stage of optimizations although c-TPE still exhibits quicker convergence. On the other hand, the performance of NEI and HM2 was degraded. As mentioned in Theorem 2, c-TPE approaches the performance of the vanilla TPE in the settings of $\gamma_{c_i^\star}^{\text{true}} = 0.9$ and thus such degradation does not happen to c-TPE. Furthermore, the contributions to the acquisition function from looser constraints decay and KA does not disrupt the performance of c-TPE thanks to this property.

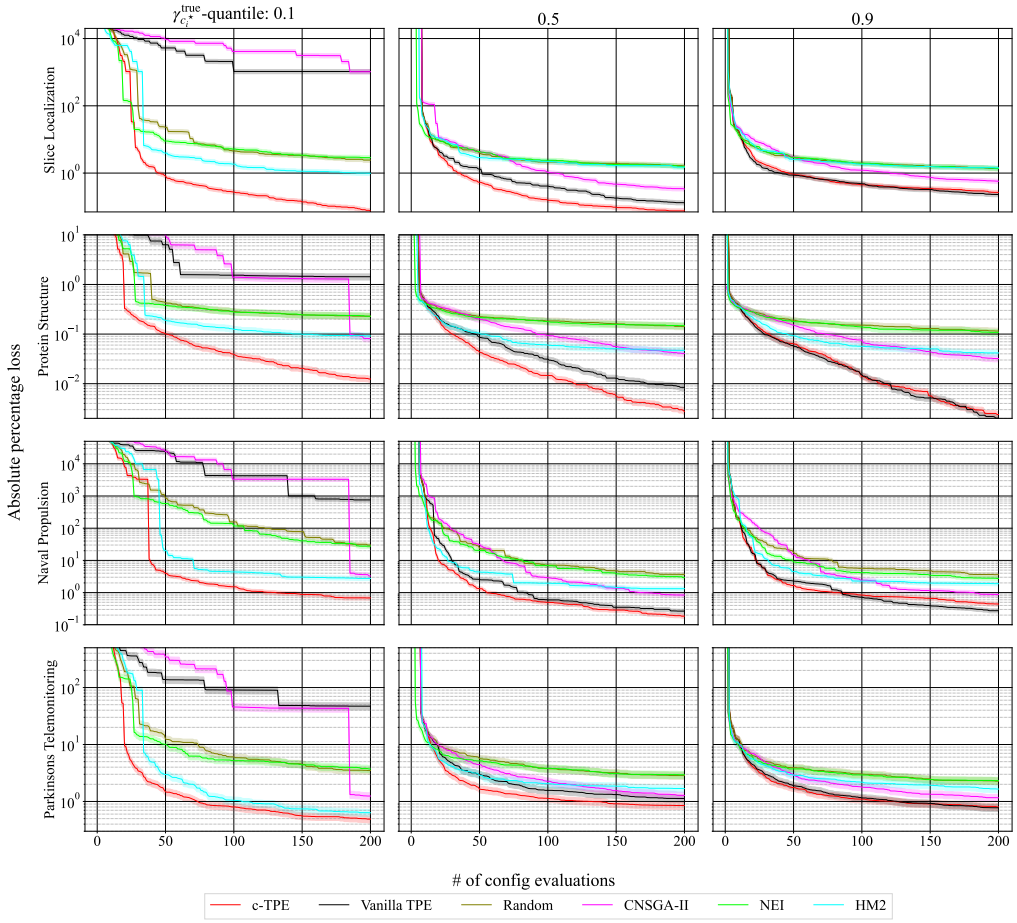

Figure 7: Figures show the performance curves on four benchmarks in HPOlib with a constraint of runtime.

For multiple constraints settings shown in Figure 8, both CNSGA-II and HM2 show slower convergence compared to single constraint settings. On the other hand, c-TPE shows quicker convergence in the settings as well.

D.1.2 Results on NAS-Bench-101

Figures 9, 10, and 11 show the time evolution of absolute percentage loss of each optimization method on HPOlib, NAS-Bench-101, and NAS-Bench-201 with the $\gamma_{c_i^\star}^{\text{true}}$-quantile of 0.1, 0.5, and 0.9 for the runtime constraint. Note that since we could not run NEI and HM2 on CIFAR10C in our environment, the results for CIFAR10C do not have the performance curves of NEI and HM2.

The results on NAS-Bench-101 look different from those on HPOlib and NAS-Bench201. For example, random search outperforms other methods on the tighter constraint settings of CIFAR10C. This is because high-dimensional search space and tight constraints made the information collection harder and thus each method could not guide itself although c-TPE still outperformed other methods on average. Additionally, KA still helps to yield better configurations quickly except CIFAR10C with runtime and network size constraints. As seen in the figures, the vanilla TPE exhibited better performance on loose constraint settings and thanks to the c-TPE's property discussed in Theorem 2, c-TPE improves its performance in loose constraint levels.

D.1.3 Results on NAS-Bench-201

Figures 12, 13, and 14 show the time evolution of absolute percentage loss of each optimization method on HPOlib, NAS-Bench-101, and NAS-Bench-201 with the $\gamma_{c_i^\star}^{\text{true}}$-quantile of 0.1, 0.5, and

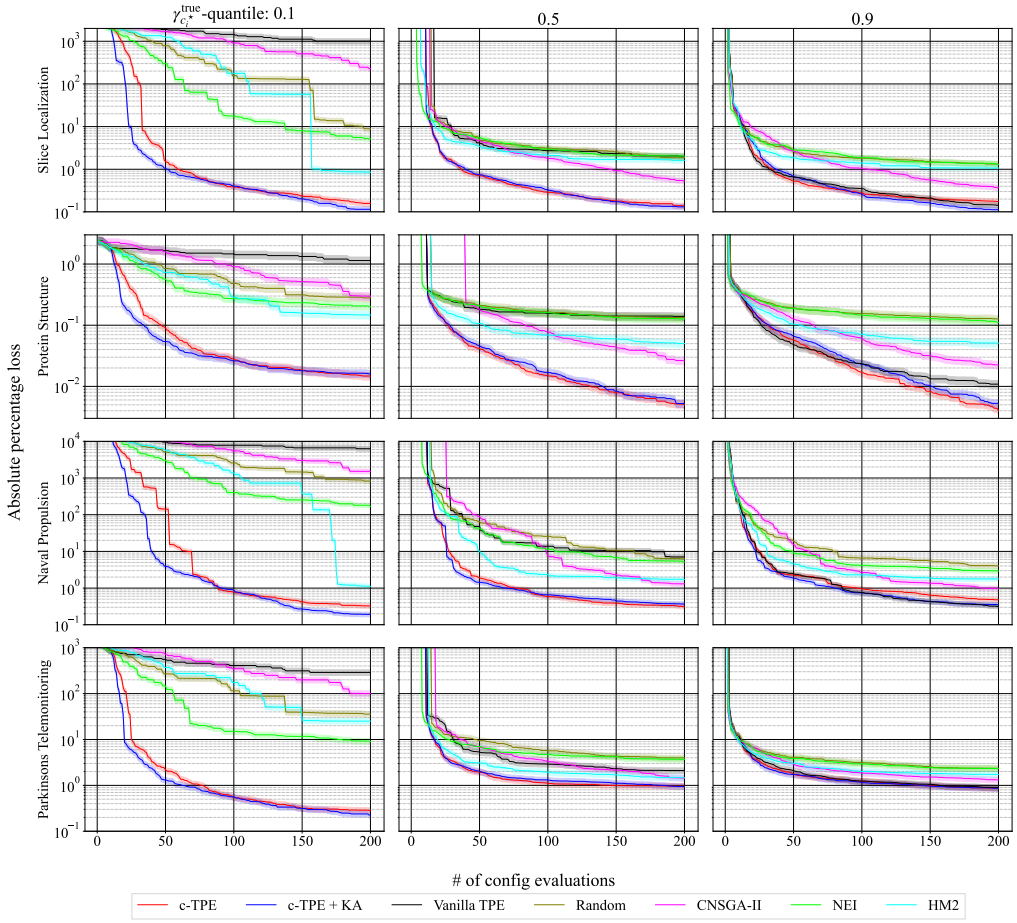

Figure 8: Figures show the performance curves on four benchmarks in HPOlib with constraints of runtime and network size.

0.9 for the runtime and network size constraints. Note that the search space of NAS-Bench-201 is composed of six categorical parameters.

According to the figures, the discrepancy between c-TPE and the vanilla TPE is larger than HPOlib and NAS-Bench-101 settings. This means that there are many violated configurations that exhibit good performance. For this reason, the tighter constraint settings on NAS-Bench-201 are harder than the other benchmarks. However, c-TPE and HM2 showed better performance on tighter constraint settings. Additionally, c-TPE maintained the performance even over looser constraint settings while CNSGA-II and HM2 did not. This robustness is also from the property mentioned in Theorem 2.

D.2 Performance over $\gamma^{\text{true}}_{c^\star_i}$-quantile for all the datasets

Figures 15, 16, and 17 show the mean and standard error of the best performance in each optimization with various $\gamma^{\text{true}}_{c^\star_i}$. The results are visualized for each benchmark problem unlike other visualization so that we can see the best optimization methods for each problem and how close the final performance of each method is. In other words, a method is better when it is always close to the best performance rather than exhibiting only the best performance for some settings and the worst settings for others. Note that since we plot the final performance, KA, which accelerates the optimizations in the early stage, does not make big difference.

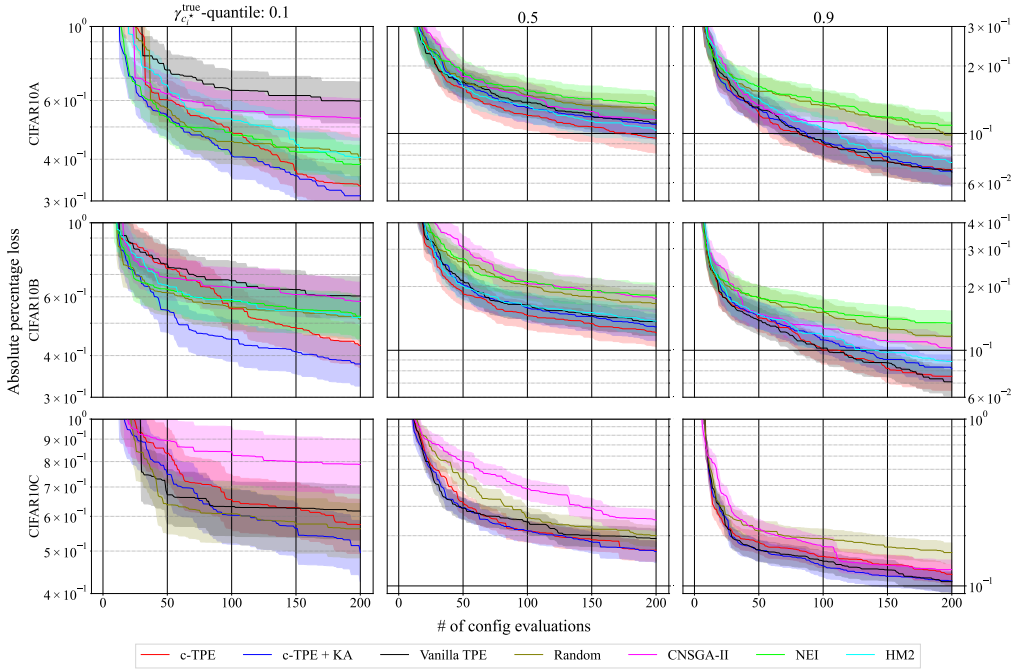

Figure 9: Figures show the performance curves on three benchmarks in NAS-Bench-101 with a constraint of network size. Note that since the scale of the results in $\gamma_{c_i^\star}^{\text{true}}$-quantile of 0.1 is different from others, we separately scaled for the readability.

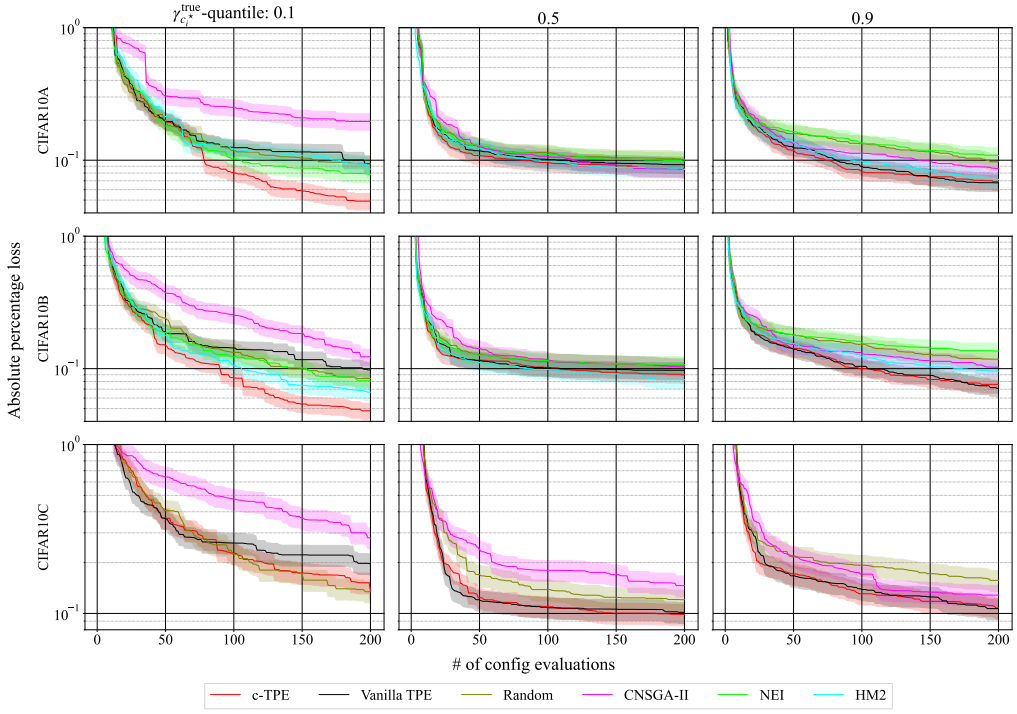

Figure 10: Figures show the performance curves on three benchmarks in NAS-Bench-101 with a constraint of runtime.

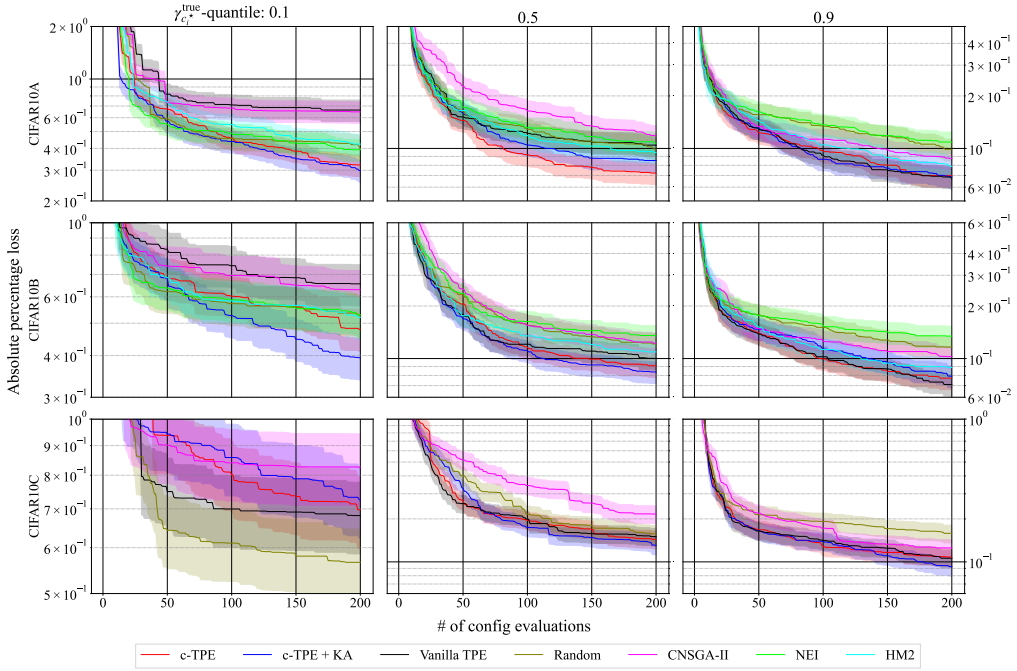

Figure 11: Figures show the performance curves on three benchmarks in NAS-Bench-101 with constraints of runtime and network size. Note that since the scale of the results in $\gamma^{\text{true}}_{c^\star_i}$-quantile of 0.1 is different from others, we separately scaled for the readability.

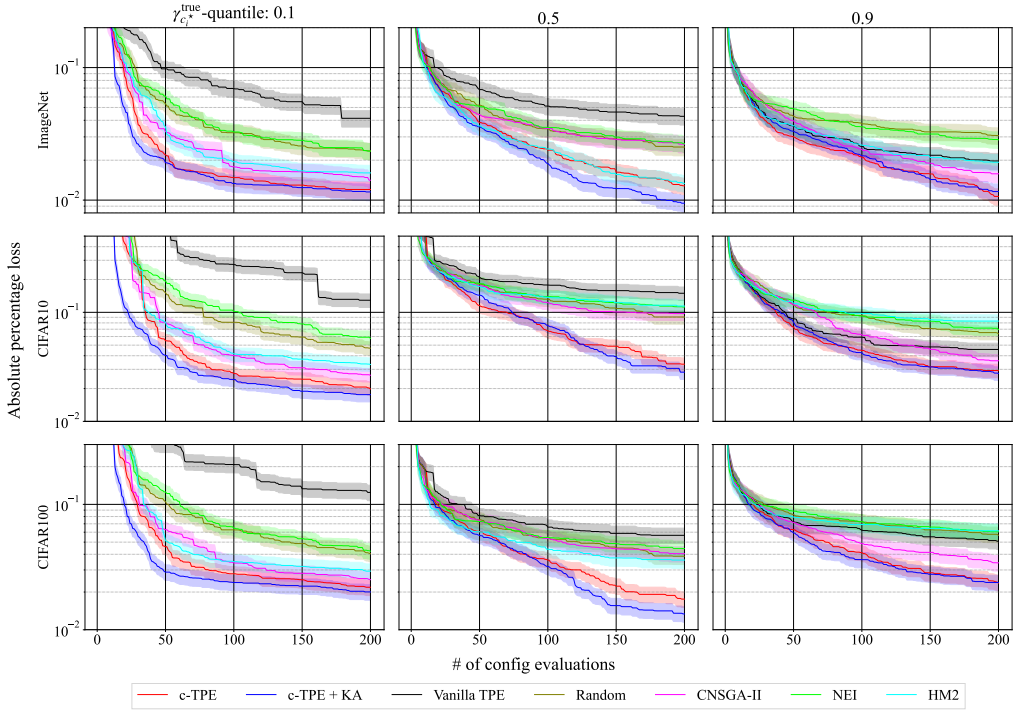

Figure 12: Figures show the performance curves on three benchmarks in NAS-Bench-201 with a constraint of network size.

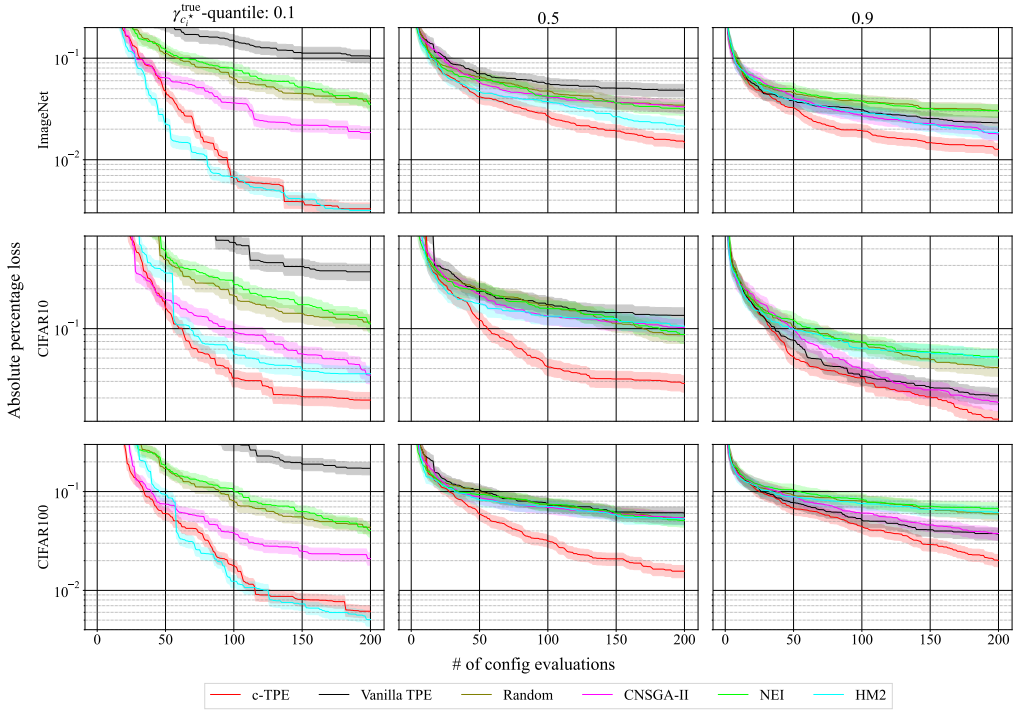

Figure 13: Figures show the performance curves on three benchmarks in NAS-Bench-201 with a constraint of runtime.

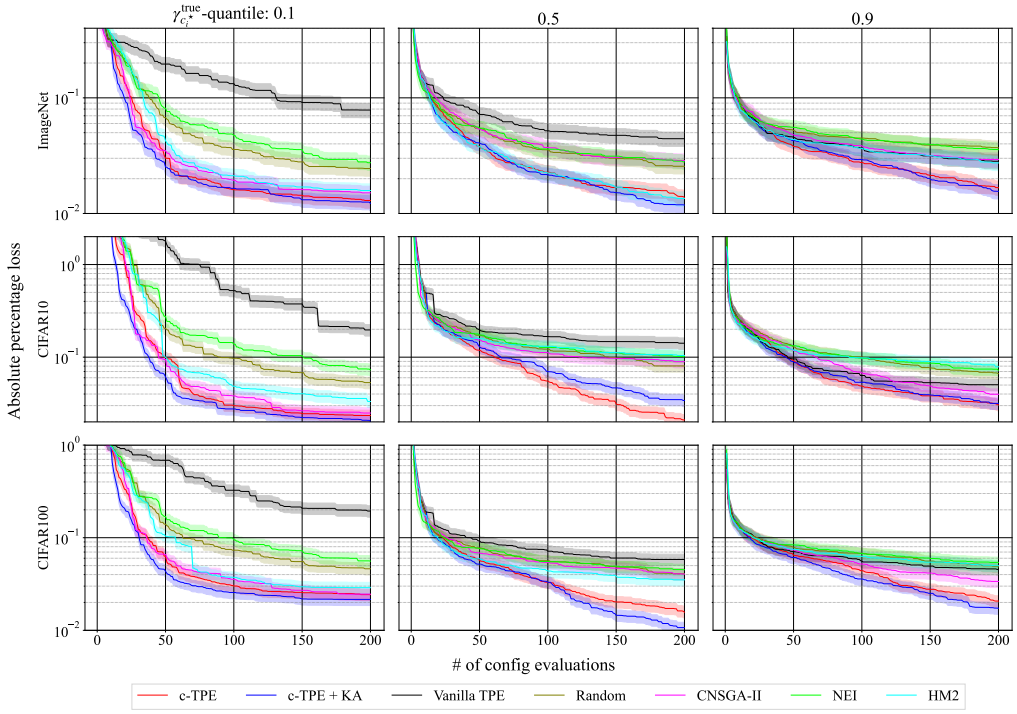

Figure 14: Figures show the performance curves on three benchmarks in NAS-Bench-201 with constraints of runtime and network size.

D.2.1 Results on HPOlib

According to Figure 15, while c-TPE was outperformed for some settings, the differences from the best performance are not large compared to those for CNSGA-II and HM2. CNSGA-II typically has larger differences in tighter constraints and HM2 has them in looser settings. Another point is that the performance of NEI and HM2 was not superior to the vanilla TPE when constraint quantiles are higher. This is due to the fact that NEI and HM2 do not converge to a single objective optimization unless the approximated probability improvements of each constraint return 1 over the whole domain. On the other hand, such performance degradation does not happen to c-TPE due to Theorem 2. However, KA for c-TPE does not help, except in tighter constraint settings although it does not disrupt the optimizations. This is because the results show the final performance of 200 evaluations and thus when c-TPE reaches 200 evaluations, it is competent enough to be able to find good solutions.

D.2.2 Results on NAS-Bench-101

Figure 16 shows the performance on NAS-Bench-101. As seen in the figures, c-TPE or c-TPE with KA exhibited the best performance except on CIFAR10C with runtime and network size constraints. This is because NAS-Bench-101 has high-dimensional search spaces and tighter constraints enforce each method to have only a fraction of observations that are useful to guide the optimization. The experiments on NAS-Bench-101 conclude that although the combination of tight constraints and higher dimensions might block the quick convergence of c-TPE, it is better to use c-TPE rather than other methods because the performance of c-TPE is stable over all the constraint levels.

D.2.3 Results on NAS-Bench-201

Figure 17 presents the results on NAS-Bench-201. As seen in the figure, while c-TPE and c-TPE with KA yielded the best performance for most settings or competitive performance if they are not the best, CNSGA-II and HM2 exhibited larger differences in some settings. This result implies that c-TPE is more robust compared to other methods although others also yield good performances in some settings.

E Additional results for Section 4.2

Figures 18, 19, and 20 show the average rank of each method over the number of evaluations. Each figure shows the performance of different constraint settings with 0.1 to 0.9 of $\gamma_{c_i^\star}^{\mathrm{true}}$.

As the constraint becomes tighter, c-TPE converges quicker in the early stage of the optimizations in all the settings due to KA. On the other hand, KA does not accelerate the optimizations as constraints become looser. This is because it is easy to obtain information about feasible domains even by random samplings. However, KA does not degrade the performance of c-TPE and thus it is recommended to add KA as much as possible.

Furthermore, it is worth noting that although the performance of HM2 and NEI outperformed the vanilla TPE in the tighter constraint settings, their performance is degraded as constraints become looser and they did not exhibit better performance than the vanilla TPE with $\gamma_{c_i^\star}^{\mathrm{true}} = 0.9$. On the other hand, c-TPE adapts the optimization based on the estimated $\gamma_{c^\star}$-quantile and thus it exhibited better performance than the vanilla TPE even in the settings of $\gamma_{c_i^\star}^{\mathrm{true}} = 0.9$.

F The performance of the vanilla TPE

As described in Section 3.3, since our TPE implementation uses multivariate kernel density estimation, it is different from the Hyperopt implementation that is used in most prior works. For this reason, we demonstrate that our TPE implementation performs significantly better than Hyperopt and how our TPE compares with other BO methods in our experiment settings. In this experiment, we compare our TPE with Hyperopt and the following methods:

1. **TuRBO** (Eriksson *et al.* (2019)) [1], and

[1] Implementation: `https://github.com/uber-research/TuRBO`

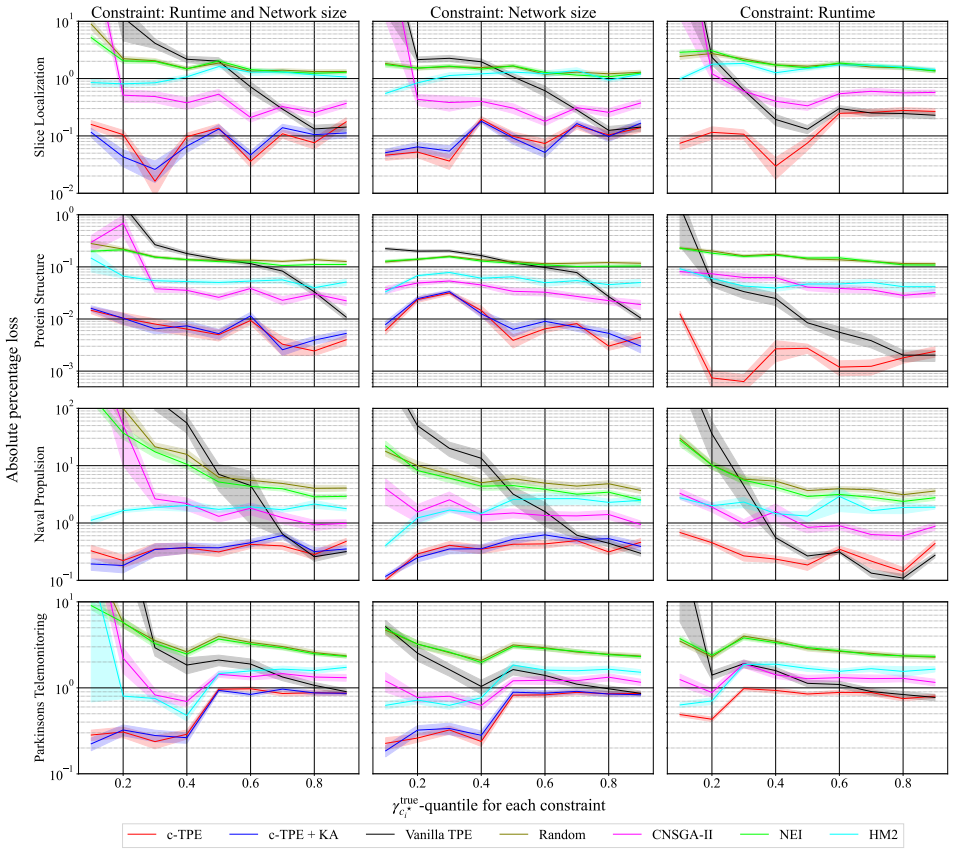

Figure 15: The best performance on HPOlib over various constraint settings and various constraint levels. Each optimization evaluated 200 configurations. Weak-color bands represent the standard error of the best performance over 50 random seeds. The horizontal axis represents $\gamma_{c_i^\star}^{\text{true}}$-quantile and the vertical axis represents the absolute percentage loss.

2. **CoCaBO** (Ru *et al.* (2020)) [2].

CoCaBO is a BO method that focuses on the handling of categorical parameters and TuRBO is one of the strongest BO methods developed recently. Both methods follow the default settings provided in the examples. Note that as both methods are either not extended to constraint optimization or not publicly available, we could not include those methods in Section 4.

Figure 21 shows the average rank over time for each method. As seen in the figure, our TPE is better than Hyperopt. Furthermore, while our TPE is statistically significantly better than other methods in most settings, Hyperopt is better than only CoCaBO. Notice that since most BO papers test performance on toy functions and we use the tabular benchmarks, the discussion here does not generalize and the results only validate why we should use our TPE in our paper.

G Limitations

In this paper, we focus on tabular benchmarks for search spaces with categorical parameters and with one or two constraints. We chose the tabular benchmarks to enable the stability analysis of the performance variations depending on constraint levels. Furthermore, such settings are common in HPO of deep learning. However, practitioners may use c-TPE for other settings, and thus we would like to discuss the following settings which we did not cover in the paper:

1. **Extremely small feasible domain size**

[2] Implementation: `https://github.com/rubinxin/CoCaBO_code`

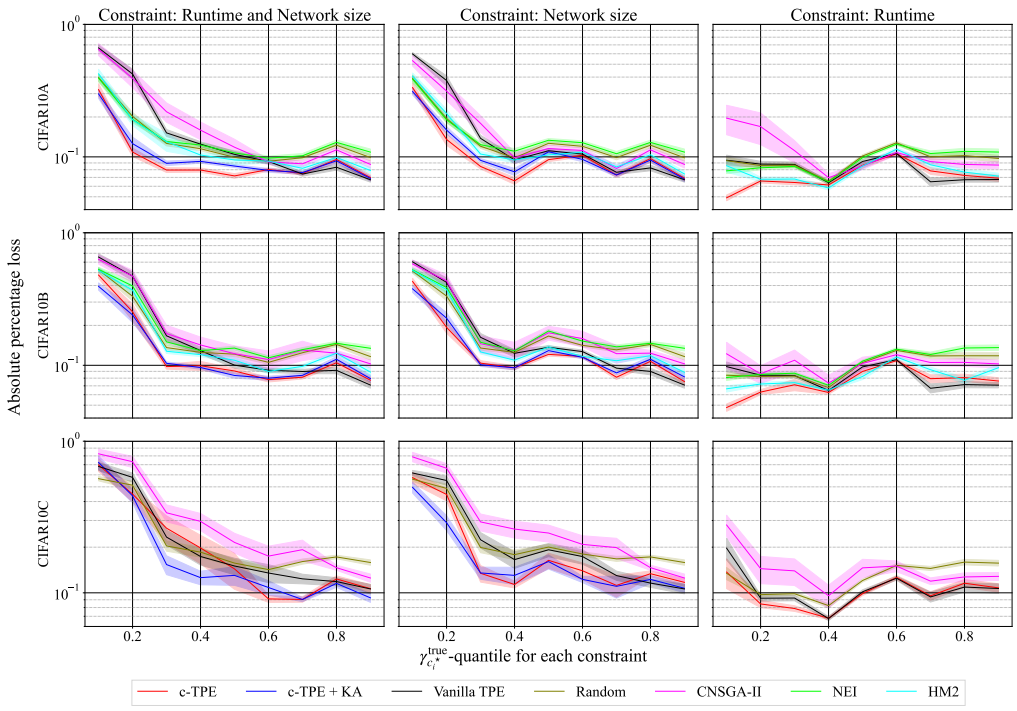

Figure 16: The best performance on NAS-Bench-101 over various constraint settings and various constraint levels. Each optimization evaluated 200 configurations. Weak-color bands represent the standard error of the best performance over 50 random seeds.

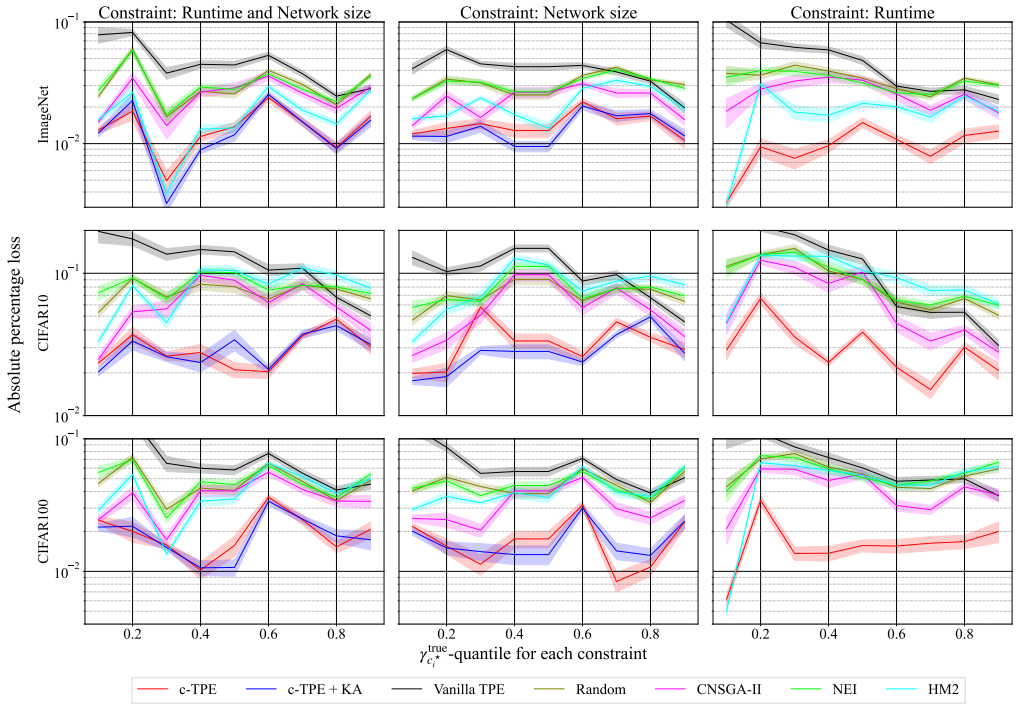

Figure 17: The best performance on NAS-Bench-201 over various constraint settings and various constraint levels. Each optimization evaluated 200 configurations. Weak-color bands represent the standard error of the best performance over 50 random seeds.

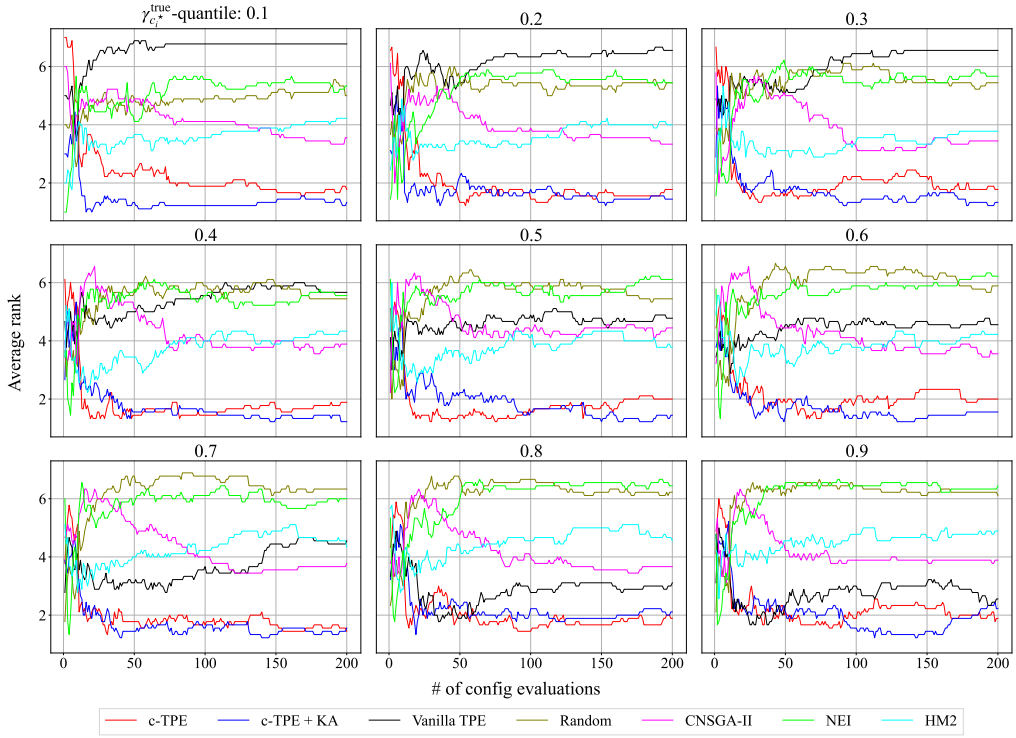

Figure 18: The average rank of each method over the number of evaluations. We evaluated each method on nine benchmarks with the network size constraint and each optimization was repeated over 50 random seeds. Each figure presents the results for $\gamma^{\text{true}}_{c_i^*}$ of 0.1 to 0.9, respectively.

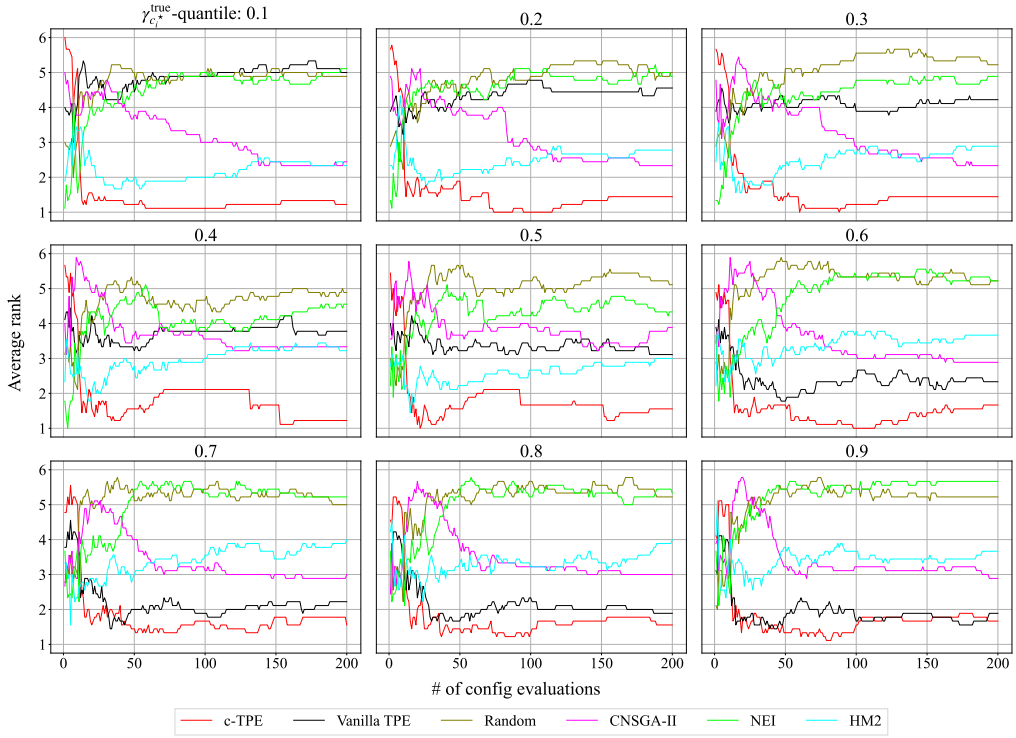

Figure 19: The average rank of each method over the number of evaluations. We evaluated each method on nine benchmarks with the runtime constraint and each optimization was repeated over 50 random seeds.

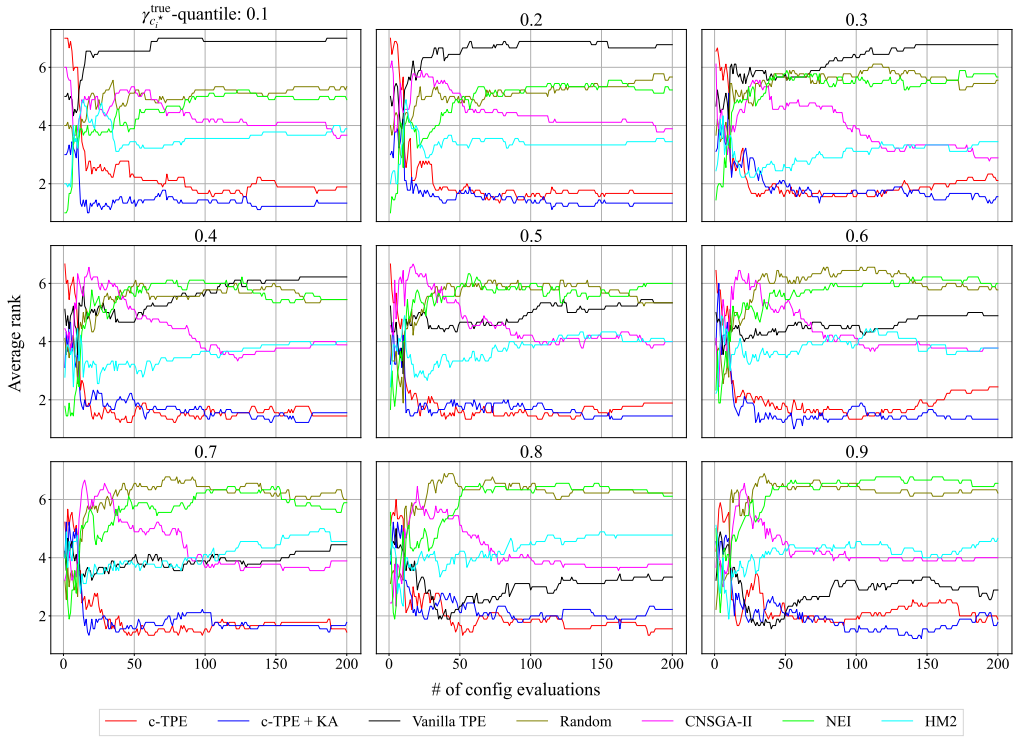

Figure 20: The average rank of each method over the number of evaluations. We evaluated each method on nine benchmarks with the runtime and the network size constraints and each optimization was repeated over 50 random seeds.

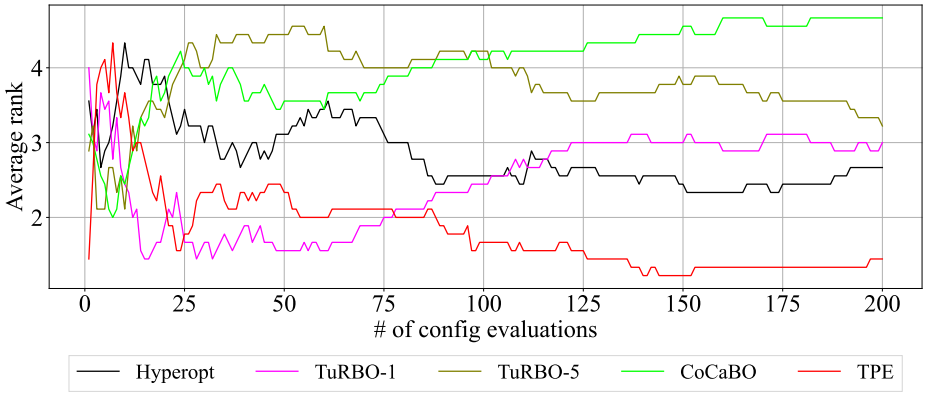

Figure 21: The performance comparison of our TPE implementation against prior works and Hyperopt implementation. The horizontal axis represents the number of configurations and the vertical axis represents the average rank of each method over 9 benchmarks that were used in Section 4.

2. **Many constraints**

3. **Parallel computation**

The first setting is an extremely small feasible domain size. For example, when we have $\Gamma = 10^{-4}$ for 200 evaluations, we get at least one feasible solution with the probability of $1 - (1 - 10^{-4})^{200} = 0.0198\cdots \simeq 2.0\%$ in the case of the uniform sampler. Such settings are generally hard for most optimizers to find even one feasible solution.

Table 4: In the table, we show the test results of The hypothesis "The other method is better than our TPE" for the "v.s. our TPE" column and the hypothesis "The other method is better than Hyperopt" for the "v.s. Hyperopt" column by the Wilcoxon signed-rank test. For example, the "TuRBO-1" row in the "v.s. our TPE" column says "N/N/W/W". It means while we cannot draw any conclusion about the performance difference with $50, 100$ evaluations, TuRBO-1 is significantly worse than our TPE with $150, 200$ evaluations in our settings. Note that we chose $p < 0.01$ as the threshold. Each method was run over 15 random seeds.

Methods # of configs	v.s. our TPE 50/100/150/200	v.s. Hyperopt 50/100/150/200
our TPE	–/–/–/–	N/B/B/B
Hyperopt	N/**W**/**W**/**W**	–/–/–/–
TuRBO-1	N/N/**W**/**W**	B/N/N/N
TuRBO-5	**W**/**W**/**W**/**W**	**W**/N/N/N
CoCaBO	**W**/**W**/**W**/**W**	N/**W**/**W**/**W**

The second setting is tasks with many constraints. In our experiments, we have the constraints of runtime and network size. On the other hand, there might be more constraints in other purposes. Many constraints make the optimization harder because the feasible domain size will be smaller due to the curse of dimensionality. More formally, when we define the feasible domain for the i-th constraint as $\mathbf{\Theta}_i = \{\boldsymbol{\theta} \in \mathbf{\Theta} | c_i(\boldsymbol{\theta}) \leq c_i^\star\}$, the feasible domain size will become exponentially small unless some feasible domains are identical, i.e. $\mathbf{\Theta}_i = \mathbf{\Theta}_j$ for some pairs $(i, j) \in [1, C] \times [1, C]$ such that $i \neq j$. As the objective spaces will be more complicated, optimizers usually require much more observations compared to tasks with one or two constraints.

The third setting is parallel computation. In HPO, since objective functions are usually expensive, it is often preferred to be able to optimize with less regret. For example, evolutionary algorithms evaluate a fixed number G of configurations in one generation and thus they optimize the objective function without any loss compared to the sequential setting up to G parallel processes. Although TPE (and c-TPE) are applicable to asynchronous settings, we cannot conclude c-TPE works nicely in parallel settings from our experiments.

We did not test c-TPE on those three settings and thus practitioners are encouraged to compare c-TPE with other methods if their tasks of interest have the characteristics described above.

H Societal impacts

Since there are not many options for black-box constraint optimization and c-TPE is likely to be integrated into open source software, our method would enable, for example, efficient performance improvements along with a budget constraint and it potentially reduces the energy consumption during both HPO and actual operations. On the other hand, when practitioners run HPO, they are mostly required to design search spaces and to set constraints on their own. As discussed in the paper, the difficulties of constraint optimizations depend on the feasible domain size Γ and users must specify the search spaces and constraint values. It might lead to yielding no feasible solutions in the case of too tight constraint values or no convergence in the case of too high search space dimensions. Such cases waste much energy; therefore, we still need to investigate ways to automate search space design to circumvent such difficulties.

References

D. Eriksson, M. Pearce, J. Gardner, RD. Turner, and M. Poloczek. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*, 2019.

B. Ru, A. Alvi, Vu V. Nguyen, M. Osborne, and S. Roberts. Bayesian optimisation over multiple continuous and categorical inputs. In *International Conference on Machine Learning*, 2020.