

AutoML: Beyond AutoML

Structured Procastination

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Structured Procrastination [Kleinberg et al. 2017]

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime

Structured Procrastination [Kleinberg et al. 2017]

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime
- task: for a fix set of configuration, identify the one with the best average runtime
- instead of top-down capping, use bottom up capping

Structured Procrastination [Kleinberg et al. 2017]

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime
- task: for a fix set of configuration, identify the one with the best average runtime
- instead of top-down capping, use bottom up capping
- start with a minimal cap-time and increase it step by step

Structured Procrastination [Kleinberg et al. 2017]

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime
 - task: for a fix set of configuration, identify the one with the best average runtime
 - instead of top-down capping, use bottom up capping
 - start with a minimal cap-time and increase it step by step
 - unsuccessful runs (with too small cap-time) are procrastinated to later
- ~> worst-case runtime guarantees

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 1 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 2 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

|

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 3 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while *b remains* **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 4 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while *b remains* **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 5 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while *b remains* **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if *terminates* **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 6 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while *b remains* **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if *terminates* **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

else

$R(\hat{\lambda}, i^{(k)}) := \kappa$;

 Insert $(i^{(k)}, 2 \cdot \kappa)$ at the end of $Q_{\hat{\lambda}}$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 7 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if *terminates* **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

else

$R(\hat{\lambda}, i^{(k)}) := \kappa$;

 Insert $(i^{(k)}, 2 \cdot \kappa)$ at the end of $Q_{\hat{\lambda}}$;

 Replenish queue $Q_{\hat{\lambda}}$ if too small;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 8 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if *terminates* **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

else

$R(\hat{\lambda}, i^{(k)}) := \kappa$;

 Insert $(i^{(k)}, 2 \cdot \kappa)$ at the end of $Q_{\hat{\lambda}}$;

 Replenish queue $Q_{\hat{\lambda}}$ if too small;

return $\hat{\lambda} := \arg \min_{\lambda \in \Lambda} \sum_{k=1}^N R(\lambda, i^{(k)})$

- We can derive theoretical optimality guarantees with structured procrastination (SP)

- We can derive theoretical optimality guarantees with structured procrastination (SP)
- In practice, plain SP is rather slow and requires the setting of some hyperparameters

- We can derive theoretical optimality guarantees with structured procrastination (SP)
- In practice, plain SP is rather slow and requires the setting of some hyperparameters
- Several extensions and similar ideas:
 - ▶ [Procrastinating with Confidence: Near-Optimal, Anytime, Adaptive Algorithm Configuration]
 - ▶ [LeapsAndBounds: A Method for Approximately Optimal Algorithm Configuration]
 - ▶ [CapsAndRuns: An Improved Method for Approximately Optimal Algorithm Configuration]