AutoML: Beyond AutoML

Racing for Algorithm Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

State-of-the-art Algorithm Configuration

SMAC: Sequential Model-based Algorithm Configuration [Hutter et al. 2011]

- Bayesian Optimization +
- aggressive racing +
- adaptive capping (for optimizing runtime)

SMAC: Overview [Hutter et al. 2011]

Algorithm 1 SMAC

Input : instance set $\mathcal{I}$, Algorithm $\mathcal{A}$ with configuration space $\boldsymbol{\Lambda}$, Initial configuration $\boldsymbol{\lambda}_0$, performance metric c, Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on $\boldsymbol{\lambda}_0$; // $\mathcal{D}_{\texttt{Hist}} = (\boldsymbol{\lambda}, i, c(i, \boldsymbol{\lambda}))_i$

while b *remains* **do**

SMAC: Overview [Hutter et al. 2011]

Algorithm 2 SMAC

Input : instance set $\mathcal{I}$, Algorithm $\mathcal{A}$ with configuration space $\boldsymbol{\Lambda}$, Initial configuration $\boldsymbol{\lambda}_0$, performance metric c, Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on $\boldsymbol{\lambda}_0$; // $\mathcal{D}_{\texttt{Hist}} = (\boldsymbol{\lambda}, i, c(i, \boldsymbol{\lambda}))_i$

while b *remains* **do**

  $\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;

SMAC: Overview [Hutter et al. 2011]

Algorithm 3 SMAC

Input : instance set $\mathcal{I}$, Algorithm $\mathcal{A}$ with configuration space $\mathbf{\Lambda}$, Initial configuration $\boldsymbol{\lambda}_0$, performance metric c, Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on $\boldsymbol{\lambda}_0$; // $\mathcal{D}_{\texttt{Hist}} = (\boldsymbol{\lambda}, i, c(i, \boldsymbol{\lambda}))_i$

while b *remains* **do**

  $\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;

  $\mathbf{\Lambda}_{challengers} \leftarrow$ select configurations based on $\hat{c}$;

SMAC: Overview [Hutter et al. 2011]

Algorithm 4 SMAC

Input : instance set $\mathcal{I}$, Algorithm $\mathcal{A}$ with configuration space $\mathbf{\Lambda}$, Initial configuration $\boldsymbol{\lambda}_0$, performance metric c, Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on $\boldsymbol{\lambda}_0$; // $\mathcal{D}_{\texttt{Hist}} = (\boldsymbol{\lambda}, i, c(i, \boldsymbol{\lambda}))_i$
while b *remains* **do**
    $\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;
    $\mathbf{\Lambda}_{challengers} \leftarrow$ select configurations based on $\hat{c}$;
    $\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\text{Hist}} \leftarrow$ intensify($\mathbf{\Lambda}_{challengers}$, $\hat{\boldsymbol{\lambda}}$); `// racing and capping`

SMAC: Overview [Hutter et al. 2011]

Algorithm 5 SMAC

Input : instance set $\mathcal{I}$, Algorithm $\mathcal{A}$ with configuration space $\mathbf{\Lambda}$, Initial configuration $\boldsymbol{\lambda}_0$, performance metric c, Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on $\boldsymbol{\lambda}_0$; // $\mathcal{D}_{\texttt{Hist}} = (\boldsymbol{\lambda}, i, c(i, \boldsymbol{\lambda}))_i$

while b *remains* **do**

  $\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;

  $\mathbf{\Lambda}_{challengers} \leftarrow$ select configurations based on $\hat{c}$;

  $\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\text{Hist}} \leftarrow$ intensify($\mathbf{\Lambda}_{challengers}$, $\hat{\boldsymbol{\lambda}}$); // racing and capping

return $\hat{\boldsymbol{\lambda}}$

Comparisons on N instances [Hutter et al. 2009]

- Basic(N) uses a pretty basic comparison: $\mathit{better}_N(\boldsymbol{\lambda}', \boldsymbol{\lambda})$:
  - Compare $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}$ based on N instances

Comparisons on N instances [Hutter et al. 2009]

- Basic(N) uses a pretty basic comparison: $\mathit{better}_N(\boldsymbol{\lambda}', \boldsymbol{\lambda})$:
  - Compare $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}$ based on N instances
  - How does this relate to cross-validation?

Comparisons on N instances [Hutter et al. 2009]

- Basic(N) uses a pretty basic comparison: *better*$_N(\boldsymbol{\lambda}', \boldsymbol{\lambda})$:
  - Compare $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}$ based on N instances
  - How does this relate to cross-validation?
- Problem: How to set N? Problems of large N? Small N?

Comparisons on N instances [Hutter et al. 2009]

- Basic(N) uses a pretty basic comparison: *better*$_N(\boldsymbol{\lambda}', \boldsymbol{\lambda})$:
  - Compare $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}$ based on N instances
  - How does this relate to cross-validation?

- Problem: How to set N? Problems of large N? Small N?
  - Problem of large N: evaluations are slow
  - Problem of small N: overfitting to a small set of instances

  $\rightsquigarrow$ Tradeoff: Choose N of moderate size

Comparisons on N instances [Hutter et al. 2009]

Question: Which N instances should we use?

1. N different instances for each configuration
2. The same set of N instances for the entire run

Comparisons on N instances [Hutter et al. 2009]

Question: Which N instances should we use?

1. N different instances for each configuration
2. The same set of N instances for the entire run

Answer: the same N instances, so that we compare apples with apples

(but: using the same instances can also yield overtuning)

Comparisons on N instances [Hutter et al. 2009]

Question: Which N instances should we use?

1. N different instances for each configuration
2. The same set of N instances for the entire run

Answer: the same N instances, so that we compare apples with apples

(but: using the same instances can also yield overtuning)

If we sampled different instances for each configuration:

- Some configurations would randomly get easier instances
- Those configurations would look better than they really are

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

1. Sample a new seed for each algorithm run
2. Fix the seeds together with the instances

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

1. Sample a new seed for each algorithm run
2. Fix the seeds together with the instances

Answer: just like for instances, fix them to compare apples to apples

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

1. Sample a new seed for each algorithm run
2. Fix the seeds together with the instances

Answer: just like for instances, fix them to compare apples to apples

In summary, for each run of Basic(N):
pick N (instance, seed) pairs and use them for evaluating each $\boldsymbol{\lambda}$.

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

1. Sample a new seed for each algorithm run
2. Fix the seeds together with the instances

Answer: just like for instances, fix them to compare apples to apples

In summary, for each run of Basic(N):
pick N (instance, seed) pairs and use them for evaluating each $\boldsymbol{\lambda}$.
(Different Basic(N) runs can use different instances and seeds.)

The concept of overtuning

Very related to overfitting in machine learning

- Performance improves on the training set
- Performance does not improve on the test set, and may even degrade

The concept of overtuning

Very related to overfitting in machine learning

- Performance improves on the training set
- Performance does not improve on the test set, and may even degrade

More pronounced for heterogeneous benchmark sets

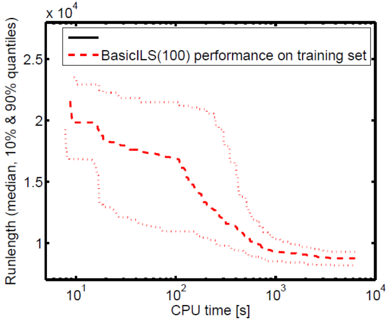
- But it even happens for very homogeneous sets
- Indeed, one can even overfit on a single instance, to the seeds used for training

Overtuning Visualized

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=100:
  average runlengths across 100 runs with different seeds
- Test cost of $\hat{\boldsymbol{\lambda}}$ here based on 1000 new seeds

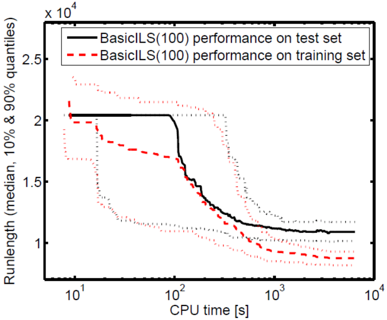
Overtuning Visualized

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=100:
  average runlengths across 100 runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds



Overtuning Visualized

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=100:
  average runlengths across 100 runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds

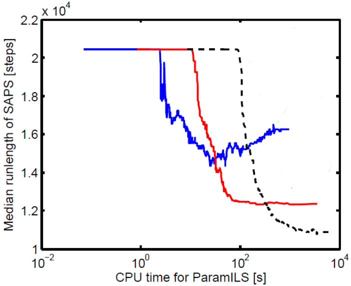


Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=?:
  average runlengths across N runs with different seeds
- Test cost of $\hat{\boldsymbol{\lambda}}$ here based on 1000 new seeds

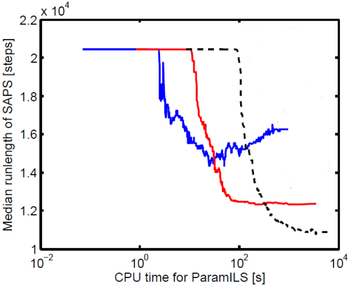
Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=?:
  average runlengths across N runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds



Basic(N) Test Results with Various N

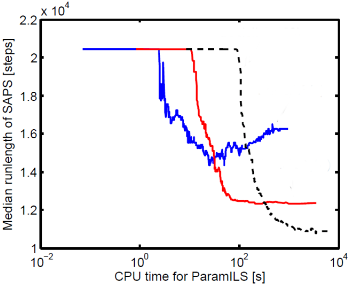
- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=?:
  average runlengths across N runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds



Which of these results corresponds to $N = 1$, $N = 10$, and $N = 100$?

Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=?:
  average runlengths across N runs with different seeds
- Test cost of $\hat{\boldsymbol{\lambda}}$ here based on 1000 new seeds

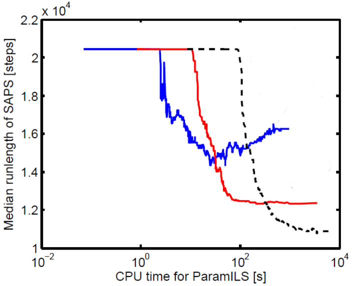


Which of these results corresponds to $N = 1$, $N = 10$, and $N = 100$?

1. N=1: blue, N=10: red,
   N=100 dashed black
2. N=1: dashed black,
   N=10: red, N=100 blue

Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with N=?:
  average runlengths across N runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds



Which of these results corresponds to $N = 1$, $N = 10$, and $N = 100$?

1. N=1: blue, N=10: red, N=100 dashed black
2. N=1: dashed black, N=10: red, N=100 blue

Correct Answer: 1

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Intuition: get the best of both worlds

- Perform more runs for good configurations
  - to avoid overtuning
- Quickly reject poor configurations
  - to make progress more quickly

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Intuition: get the best of both worlds

- Perform more runs for good configurations
  - to avoid overtuning
- Quickly reject poor configurations
  - to make progress more quickly

Definition: $N(\boldsymbol{\lambda})$ and $c_N(\boldsymbol{\lambda})$

$N(\boldsymbol{\lambda})$ denotes the number of runs executed for $\boldsymbol{\lambda}$ so far.
$\hat{c}_N(\boldsymbol{\lambda})$ denotes the cost estimate of $\boldsymbol{\lambda}$ based on N runs.

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Intuition: get the best of both worlds

- Perform more runs for good configurations
  - to avoid overtuning
- Quickly reject poor configurations
  - to make progress more quickly

Definition: $N(\boldsymbol{\lambda})$ and $c_N(\boldsymbol{\lambda})$

$N(\boldsymbol{\lambda})$ denotes the number of runs executed for $\boldsymbol{\lambda}$ so far.
$\hat{c}_N(\boldsymbol{\lambda})$ denotes the cost estimate of $\boldsymbol{\lambda}$ based on N runs.

In the beginning: $N(\boldsymbol{\lambda}) = 0$ for every configuration $\boldsymbol{\lambda}$

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\boldsymbol{\lambda}^{(1)}$ dominates $\boldsymbol{\lambda}^{(2)}$ if

- $N(\boldsymbol{\lambda}^{(1)}) \geq N(\boldsymbol{\lambda}^{(2)})$ and
- $\hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(1)}) \leq \hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(2)})$.

I.e.: we have at least as many runs for $\boldsymbol{\lambda}^{(1)}$ and its cost is at least as low.

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\boldsymbol{\lambda}^{(1)}$ dominates $\boldsymbol{\lambda}^{(2)}$ if

- $N(\boldsymbol{\lambda}^{(1)}) \geq N(\boldsymbol{\lambda}^{(2)})$ and
- $\hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(1)}) \leq \hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(2)})$.

I.e.: we have at least as many runs for $\boldsymbol{\lambda}^{(1)}$ and its cost is at least as low.

better$(\boldsymbol{\lambda}', \hat{\boldsymbol{\lambda}})$ in a nutshell

- $\hat{\boldsymbol{\lambda}}$ is the current configuration to beat (incumbent)

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\boldsymbol{\lambda}^{(1)}$ dominates $\boldsymbol{\lambda}^{(2)}$ if

- $N(\boldsymbol{\lambda}^{(1)}) \geq N(\boldsymbol{\lambda}^{(2)})$ and
- $\hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(1)}) \leq \hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(2)})$.

I.e.: we have at least as many runs for $\boldsymbol{\lambda}^{(1)}$ and its cost is at least as low.

better$(\boldsymbol{\lambda}', \hat{\boldsymbol{\lambda}})$ in a nutshell

- $\hat{\boldsymbol{\lambda}}$ is the current configuration to beat (incumbent)
- Perform runs of $\boldsymbol{\lambda}'$ until either
  - $\hat{\boldsymbol{\lambda}}$ dominates $\boldsymbol{\lambda}' \rightsquigarrow$ reject $\boldsymbol{\lambda}'$, or
  - $\boldsymbol{\lambda}'$ dominates $\hat{\boldsymbol{\lambda}} \rightsquigarrow$ change current incumbent $(\hat{\boldsymbol{\lambda}} \leftarrow \boldsymbol{\lambda}')$

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\boldsymbol{\lambda}^{(1)}$ dominates $\boldsymbol{\lambda}^{(2)}$ if

- $N(\boldsymbol{\lambda}^{(1)}) \geq N(\boldsymbol{\lambda}^{(2)})$ and
- $\hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(1)}) \leq \hat{c}_{N(\boldsymbol{\lambda}^{(2)})}(\boldsymbol{\lambda}^{(2)})$.

I.e.: we have at least as many runs for $\boldsymbol{\lambda}^{(1)}$ and its cost is at least as low.

better$(\boldsymbol{\lambda}', \hat{\boldsymbol{\lambda}})$ in a nutshell

- $\hat{\boldsymbol{\lambda}}$ is the current configuration to beat (incumbent)
- Perform runs of $\boldsymbol{\lambda}'$ until either
  - $\hat{\boldsymbol{\lambda}}$ dominates $\boldsymbol{\lambda}'$ $\rightsquigarrow$ reject $\boldsymbol{\lambda}'$, or
  - $\boldsymbol{\lambda}'$ dominates $\hat{\boldsymbol{\lambda}}$ $\rightsquigarrow$ change current incumbent ($\hat{\boldsymbol{\lambda}} \leftarrow \boldsymbol{\lambda}'$)
- Over time: perform extra runs of $\hat{\boldsymbol{\lambda}}$ to gain more confidence in it

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$			

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$	2		

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$	2	10	

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$	2	10	
	$\rightarrow$ reject, since $\hat{c}_2(\boldsymbol{\lambda}') = 6 > \hat{c}_2(\hat{\boldsymbol{\lambda}}) = 2.5$		

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$	2	10	
	$\rightarrow$ reject, since $\hat{c}_2(\boldsymbol{\lambda}') = 6 > \hat{c}_2(\hat{\boldsymbol{\lambda}}) = 2.5$		
$\boldsymbol{\lambda}''$	3		

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$	2	10	
	$\rightarrow$ reject, since $\hat{c}_2(\boldsymbol{\lambda}') = 6 > \hat{c}_2(\hat{\boldsymbol{\lambda}}) = 2.5$		
$\boldsymbol{\lambda}''$	3	1	

Toy Example

- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$	2	10	
	$\rightarrow$ reject, since $\hat{c}_2(\boldsymbol{\lambda}') = 6 > \hat{c}_2(\hat{\boldsymbol{\lambda}}) = 2.5$		
$\boldsymbol{\lambda}''$	3	1	5

Toy Example

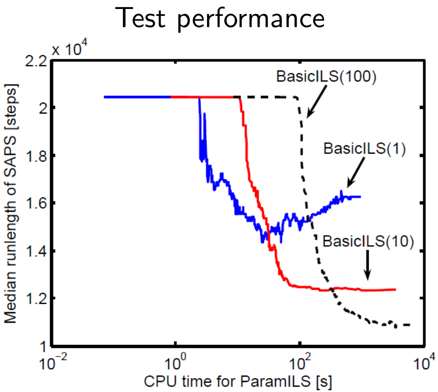
- Let $\hat{\boldsymbol{\lambda}}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers $\boldsymbol{\lambda}'$ and $\boldsymbol{\lambda}''$

	$i^{(1)}$	$i^{(2)}$	$i^{(3)}$
$\hat{\boldsymbol{\lambda}}$	3	2	10
$\boldsymbol{\lambda}'$	2	10	
		$\rightarrow$ reject, since $\hat{c}_2(\boldsymbol{\lambda}') = 6 > \hat{c}_2(\hat{\boldsymbol{\lambda}}) = 2.5$	
$\boldsymbol{\lambda}''$	3	1	5

- new incumbent: $\hat{\boldsymbol{\lambda}} \leftarrow \boldsymbol{\lambda}''$
- Perform an additional run for new $\hat{\boldsymbol{\lambda}}$ to increase confidence over time

Racing achieves the best of both worlds

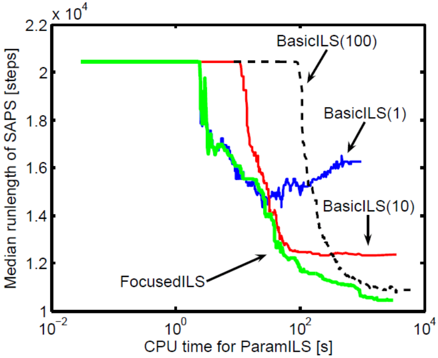
Aggressive racing (aka FocusedILS): Fast progress and no overtuning



Racing achieves the best of both worlds

Aggressive racing (aka FocusedILS): Fast progress and no overtuning

Test performance



Overview of Racing

Input : candidate configurations $\boldsymbol{\Lambda}_{new}$, cutoff κ_{max}, previously evaluated runs $\mathcal{D}_{\text{Hist}}$, budget T, incumbent $\hat{\boldsymbol{\lambda}}$

while $\boldsymbol{\Lambda}_{new}$ *not empty* **do**

$\boldsymbol{\lambda}^{(t)} \leftarrow \text{getNext}(\boldsymbol{\Lambda}_{new})$;

Overview of Racing

Input : candidate configurations $\mathbf{\Lambda}_{new}$, cutoff κ_{max}, previously evaluated runs $\mathcal{D}_{\text{Hist}}$, budget T, incumbent $\hat{\boldsymbol{\lambda}}$

while $\mathbf{\Lambda}_{new}$ *not empty* **do**

$\boldsymbol{\lambda}^{(t)} \leftarrow$ getNext($\mathbf{\Lambda}_{new}$);

$i, s \leftarrow$ instance and seed drawn uniformly at random;

$c \leftarrow$ EvaluateRun($\hat{\boldsymbol{\lambda}}, i, s, \kappa_{max}$);

$\mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\hat{\boldsymbol{\lambda}}, i, s, c)$;

Overview of Racing

Input : candidate configurations $\boldsymbol{\Lambda}_{new}$, cutoff κ_{max}, previously evaluated runs $\mathcal{D}_{\text{Hist}}$, budget T, incumbent $\hat{\boldsymbol{\lambda}}$

while $\boldsymbol{\Lambda}_{new}$ *not empty* **do**

$\quad \boldsymbol{\lambda}^{(t)} \leftarrow \text{getNext}(\boldsymbol{\Lambda}_{new})$;

$\quad i, s \leftarrow$ instance and seed drawn uniformly at random;

$\quad c \leftarrow \text{EvaluateRun}(\hat{\boldsymbol{\lambda}}, i, s, \kappa_{max})$;

$\quad \mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\hat{\boldsymbol{\lambda}}, i, s, c)$;

$\quad$ **while** *true* **do**

$\quad\quad \mathcal{I}^{+}, \mathbf{s}^{+} \leftarrow \text{getAlreadyEvaluatedOn}(\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\text{Hist}})$;

$\quad\quad \mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow \text{getAlreadyEvaluatedOn}(\boldsymbol{\lambda}^{(t)}, \mathcal{D}_{\text{Hist}})$;

$\quad\quad i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^{+} \setminus \mathcal{I}^{(t)}$ and $\mathbf{s}^{+} \setminus \mathbf{s}^{(\mathbf{t})}$;

Overview of Racing

Input : candidate configurations $\boldsymbol{\Lambda}_{new}$, cutoff κ_{max}, previously evaluated runs $\mathcal{D}_{\mathsf{Hist}}$, budget T, incumbent $\hat{\boldsymbol{\lambda}}$

while $\boldsymbol{\Lambda}_{new}$ *not empty* **do**

$\boldsymbol{\lambda}^{(t)} \leftarrow$ getNext($\boldsymbol{\Lambda}_{new}$);

$i, s \leftarrow$ instance and seed drawn uniformly at random;

$c \leftarrow$ EvaluateRun($\hat{\boldsymbol{\lambda}}, i, s, \kappa_{max}$);

$\mathcal{D}_{\mathsf{Hist}} \leftarrow \mathcal{D}_{\mathsf{Hist}} \cup (\hat{\boldsymbol{\lambda}}, i, s, c)$;

while *true* **do**

$\mathcal{I}^+, \mathbf{s}^+ \leftarrow$getAlreadyEvaluatedOn($\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\mathsf{Hist}}$);

$\mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow$getAlreadyEvaluatedOn($\boldsymbol{\lambda}^{(t)}, \mathcal{D}_{\mathsf{Hist}}$);

$i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^+ \setminus \mathcal{I}^{(t)}$ and $\mathbf{s}^+ \setminus \mathbf{s}^{(\mathbf{t})}$;

$c_i \leftarrow$ EvaluateRun($\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max}$);

$\mathcal{D}_{\mathsf{Hist}} \leftarrow \mathcal{D}_{\mathsf{Hist}} \cup (\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, c^{(t)})$;

Overview of Racing

Input : candidate configurations $\mathbf{\Lambda}_{new}$, cutoff κ_{max}, previously evaluated runs $\mathcal{D}_{\text{Hist}}$, budget T, incumbent $\hat{\boldsymbol{\lambda}}$

while $\mathbf{\Lambda}_{new}$ *not empty* **do**

  $\boldsymbol{\lambda}^{(t)} \leftarrow$ getNext($\mathbf{\Lambda}_{new}$);

  $i, s \leftarrow$ instance and seed drawn uniformly at random;

  $c \leftarrow$ EvaluateRun($\hat{\boldsymbol{\lambda}}, i, s, \kappa_{max}$);

  $\mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\hat{\boldsymbol{\lambda}}, i, s, c)$;

  while *true* **do**

    $\mathcal{I}^+, \mathbf{s}^+ \leftarrow$getAlreadyEvaluatedOn($\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\text{Hist}}$);

    $\mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow$getAlreadyEvaluatedOn($\boldsymbol{\lambda}^{(t)}, \mathcal{D}_{\text{Hist}}$);

    $i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^+ \setminus \mathcal{I}^{(t)}$ and $\mathbf{s}^+ \setminus \mathbf{s}^{(\mathbf{t})}$;

    $c_i \leftarrow$ EvaluateRun($\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max}$);

    $\mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, c^{(t)})$;

    if *average cost of* $\boldsymbol{\lambda}^{(t)} >$ *average cost of* $\hat{\boldsymbol{\lambda}}$ *across* $\mathcal{I}^{(t)}$ *and* $\mathbf{s}^{(t)}$ **then**

      break;

Overview of Racing

Input : candidate configurations $\mathbf{\Lambda}_{new}$, cutoff κ_{max}, previously evaluated runs $\mathcal{D}_{\text{Hist}}$, budget T, incumbent $\hat{\boldsymbol{\lambda}}$

while $\mathbf{\Lambda}_{new}$ *not empty* **do**

- $\boldsymbol{\lambda}^{(t)} \leftarrow$ getNext($\mathbf{\Lambda}_{new}$);
- $i, s \leftarrow$ instance and seed drawn uniformly at random;
- $c \leftarrow$ EvaluateRun($\hat{\boldsymbol{\lambda}}, i, s, \kappa_{max}$);
- $\mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\hat{\boldsymbol{\lambda}}, i, s, c)$;
- **while** *true* **do**
  - $\mathcal{I}^{+}, \mathbf{s}^{+} \leftarrow$getAlreadyEvaluatedOn($\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\text{Hist}}$);
  - $\mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow$getAlreadyEvaluatedOn($\boldsymbol{\lambda}^{(t)}, \mathcal{D}_{\text{Hist}}$);
  - $i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^{+} \setminus \mathcal{I}^{(t)}$ and $\mathbf{s}^{+} \setminus \mathbf{s}^{(\mathbf{t})}$;
  - $c_i \leftarrow$ EvaluateRun($\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max}$);
  - $\mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, c^{(t)})$;
  - **if** *average cost of* $\boldsymbol{\lambda}^{(t)} >$ *average cost of* $\hat{\boldsymbol{\lambda}}$ *across* $\mathcal{I}^{(t)}$ *and* $\mathbf{s}^{(t)}$ **then**
    - break;
  - **else if** *average cost of* $\boldsymbol{\lambda}^{(t)} <$ *average cost of* $\hat{\boldsymbol{\lambda}}$ *and* $\mathcal{I}^{+} = \mathcal{I}^{(t)}$ *and* $\mathbf{s}^{(t)} = \mathbf{s}^{+}$ **then**
    - $\hat{\boldsymbol{\lambda}} \leftarrow \boldsymbol{\lambda}^{(t)}$;

Overview of Racing

Input : candidate configurations $\mathbf{\Lambda}_{new}$, cutoff κ_{max}, previously evaluated runs $\mathcal{D}_{\text{Hist}}$, budget T, incumbent $\hat{\boldsymbol{\lambda}}$

while $\mathbf{\Lambda}_{new}$ *not empty* **do**

$\boldsymbol{\lambda}^{(t)} \leftarrow$ getNext($\mathbf{\Lambda}_{new}$);

$i, s \leftarrow$ instance and seed drawn uniformly at random;

$c \leftarrow$ EvaluateRun($\hat{\boldsymbol{\lambda}}, i, s, \kappa_{max}$);

$\mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\hat{\boldsymbol{\lambda}}, i, s, c)$;

while *true* **do**

$\mathcal{I}^+, \mathbf{s}^+ \leftarrow$getAlreadyEvaluatedOn($\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\text{Hist}}$);

$\mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow$getAlreadyEvaluatedOn($\boldsymbol{\lambda}^{(t)}, \mathcal{D}_{\text{Hist}}$);

$i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^+ \setminus \mathcal{I}^{(t)}$ and $\mathbf{s}^+ \setminus \mathbf{s}^{(\mathbf{t})}$;

$c_i \leftarrow$ EvaluateRun($\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max}$);

$\mathcal{D}_{\text{Hist}} \leftarrow \mathcal{D}_{\text{Hist}} \cup (\boldsymbol{\lambda}^{(t)}, i^{(t)}, s^{(t)}, c^{(t)})$;

if *average cost of* $\boldsymbol{\lambda}^{(t)} >$ *average cost of* $\hat{\boldsymbol{\lambda}}$ *across* $\mathcal{I}^{(t)}$ *and* $\mathbf{s}^{(t)}$ **then**

break;

else if *average cost of* $\boldsymbol{\lambda}^{(t)} <$ *average cost of* $\hat{\boldsymbol{\lambda}}$ *and* $\mathcal{I}^+ = \mathcal{I}^{(t)}$ *and* $\mathbf{s}^{(t)} = \mathbf{s}^+$ **then**

$\hat{\boldsymbol{\lambda}} \leftarrow \boldsymbol{\lambda}^{(t)}$;

if *time spent exceeds* T *or* $\mathbf{\Lambda}_{new}$ *is empty* **then**

return $\hat{\boldsymbol{\lambda}}, \mathcal{D}_{\textit{Hist}}$