Getting the most out of your planner(s): from static to dynamic algorithm configuration

Frank Hutter
University of Freiburg & Bosch Center for AI
fh@cs.uni-freiburg.de

@FrankRHutter
@AutoMLFreiburg

These slides are available at www.automl.org/talks -- all references are hyperlinks
Motivation

- **Algorithm configuration (AC)** finds good settings of your parameters
  - But it is limited: the parameter setting is fixed

- **We propose dynamic algorithm configuration (DAC)**
  - This can change parameters based on the instance at hand, search progress, time, etc.
Outline

• Part 1: an overview of previous meta-algorithmic approaches
  – Algorithm Configuration
  – Algorithm Portfolios

• Part 2: Dynamic Algorithm Configuration
The Algorithm Configuration (AC) Problem

Definition: Algorithm Configuration (AC)

Given:
- a parameterized algorithm $A$ with configuration space $\Theta$
- a distribution $D$ over problem instances with domain $I$
- a cost metric $c : \Theta \times I \rightarrow \mathbb{R}$ assessing the cost of a config. $\theta \in \Theta$ on a instance $i \in I$

Find: $\theta^* \in \arg\min_{\theta \in \Theta} \mathbb{E}_{i \sim D}[c(\theta, i)]$
What Can be Parameters in Planning?

**Examples**

- **Heuristics**
  - Which heuristics to use
  - Subparameters of each heuristic
  - How to combine the heuristics

- **Search strategy**
  - Global / local search
  - Randomization
  - How to combine them

- **Problem encoding**
  - Domain model
  - Problem model

**In general**

- Any design decision for which you have more than 1 alternative

- **Parameter types**
  - Boolean, categorical, integer, continuous
  - Conditional: only active dependent on setting of other parameters

- Often, parameters give rise to a high-dimensional structured space
  - E.g., LPG: 62 parameters, $6.5 \times 10^{17}$ configurations
AC is a Useful Abstraction: Improvements in Many Areas

<table>
<thead>
<tr>
<th>Domain</th>
<th>Algorithm</th>
<th>#params</th>
<th>#configurations</th>
<th>Speedup factor</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>Spear</td>
<td>26</td>
<td>$8.3 \times 10^{17}$</td>
<td>$4.50 \times -500 \times$</td>
<td>[Hutter et al, FMCAD 2007]</td>
</tr>
<tr>
<td>MIP</td>
<td>CPLEX</td>
<td>76</td>
<td>$1.9 \times 10^{17}$</td>
<td>$2.0 \times -52 \times$</td>
<td>[Hutter et al, CPAIOR 2010]</td>
</tr>
<tr>
<td>MPE</td>
<td>GLS+</td>
<td>5</td>
<td>1680</td>
<td>$&gt;360$</td>
<td>[Hutter et al, AAAI 2007]</td>
</tr>
<tr>
<td>Time-tableting</td>
<td>UBC-TT</td>
<td>18</td>
<td>$1.0 \times 10^{13}$</td>
<td>$\geq 28 \times$</td>
<td>[Fawcett et al, TR 2009]</td>
</tr>
<tr>
<td>AI Planning</td>
<td>FastDownward</td>
<td>45</td>
<td>$3.0 \times 10^{13}$</td>
<td>$1.0 \times -23 \times$</td>
<td>[Fawcett et al, ICAPS-PAL 2011]</td>
</tr>
<tr>
<td>AI Planning</td>
<td>LPG</td>
<td>62</td>
<td>$6.5 \times 10^{17}$</td>
<td>$3.0 \times -118 \times$</td>
<td>[Vallati et al, SOCS 2013]</td>
</tr>
<tr>
<td>AI Planning</td>
<td>Domain configuration</td>
<td>109</td>
<td>$\infty$</td>
<td>$1.0 \times -339 \times$</td>
<td>[Vallati et al, IJCAI 2015]</td>
</tr>
<tr>
<td>AI Planning</td>
<td>Problem configuration</td>
<td>26</td>
<td>$\infty$</td>
<td>$1.0 \times -39 \times$</td>
<td>[Vallati &amp; Serina, ICAPS 2018]</td>
</tr>
</tbody>
</table>

AC is also a key enabling technology in automated machine learning (AutoML), e.g.:

- Auto-WEKA [Thornton et al, KDD 2013]
- Auto-sklearn [Feurer et al, NeurIPS 2015]
- Auto-PyTorch [Zimmer et al, arXiv 2020]
AC is a Useful Abstraction: Increasingly Popular

AC is increasingly popular (citation numbers from Google scholar)

- **Iterated F-Race**
  - Sampling based
- **GGA/GGA++**
  - Genetic algorithm
- **ParamILS**
  - Local search
- **SMAC**
  - Bayesian optimization
- All these algorithms are available through a unified interface in **AClib**
## Empirical Evaluation of AC Methods in AClib

[Hutter et al, 2020]

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Default</th>
<th>SMAC</th>
<th>ParamILS</th>
<th>GGA++</th>
<th>GGA</th>
<th>IRACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPLEX on Regions200</td>
<td>10.98</td>
<td>3.45</td>
<td>3.66</td>
<td>10.98</td>
<td>10.98</td>
<td>7.33</td>
</tr>
<tr>
<td>CPLEX on COR-LAT</td>
<td>22.71</td>
<td>7.44</td>
<td>23.89</td>
<td>22.71</td>
<td>22.71</td>
<td>22.27</td>
</tr>
<tr>
<td>CPLEX on RCW2</td>
<td>72.66</td>
<td>64.29</td>
<td>71.38</td>
<td>72.66</td>
<td>72.66</td>
<td>72.66</td>
</tr>
<tr>
<td>LPG on Depots</td>
<td>35.01</td>
<td>0.82</td>
<td>4.52</td>
<td>--</td>
<td>--</td>
<td>35.01</td>
</tr>
<tr>
<td>LPG on Satelite</td>
<td>18.68</td>
<td>6.30</td>
<td>6.54</td>
<td>--</td>
<td>--</td>
<td>18.68</td>
</tr>
<tr>
<td>LPG on Zenotravel</td>
<td>26.7</td>
<td>1.75</td>
<td>3.23</td>
<td>--</td>
<td>--</td>
<td>26.70</td>
</tr>
<tr>
<td>Cadical on Circuit Fuzz</td>
<td>397.11</td>
<td>303.12</td>
<td>302.24</td>
<td>537.54</td>
<td>408.73</td>
<td>445.95</td>
</tr>
<tr>
<td>Lingeling on Circuit Fuzz</td>
<td>319.73</td>
<td>258.33</td>
<td>281.37</td>
<td>574.83</td>
<td>430.55</td>
<td>--</td>
</tr>
<tr>
<td>Clasp on Queens</td>
<td>713.5</td>
<td>6.33</td>
<td>28.41</td>
<td>--</td>
<td>--</td>
<td>58.80</td>
</tr>
<tr>
<td>Clasp on 3CNF-v350</td>
<td>332.33</td>
<td>43.68</td>
<td>50.88</td>
<td>--</td>
<td>--</td>
<td>47.81</td>
</tr>
<tr>
<td>ProbSAT on 5SAT500</td>
<td>3000</td>
<td>1.94</td>
<td>1.96</td>
<td>5.32</td>
<td>4.15</td>
<td>2.06</td>
</tr>
<tr>
<td>LKH on TSP-Rue-1000-3000</td>
<td>340.31</td>
<td>253.20</td>
<td>325.02</td>
<td>647.80</td>
<td>340.31</td>
<td>335.03</td>
</tr>
<tr>
<td>Clasp on Ricochet</td>
<td>83.78</td>
<td>56.93</td>
<td>84.38</td>
<td>--</td>
<td>--</td>
<td>93.64</td>
</tr>
<tr>
<td>Clasp on Riposte</td>
<td>5.65</td>
<td>0.81</td>
<td>3.03</td>
<td>--</td>
<td>--</td>
<td>5.65</td>
</tr>
<tr>
<td>Clasp on Weighted Sequence</td>
<td>979.13</td>
<td>96.63</td>
<td>575.53</td>
<td>--</td>
<td>--</td>
<td>857.08</td>
</tr>
</tbody>
</table>
Algorithm 1: SMAC (high-level overview)

Initialize by executing some runs and collecting their performance data

repeat
    Learn a model $\hat{m}$ from performance data so far: $\hat{m} : \Theta \times \mathcal{I} \rightarrow \mathbb{R}$
    Use model $\hat{m}$ to select promising configurations $\Theta_{new} \sim$ Bayesian optimization
    Compare $\Theta_{new}$ against best configuration so far by executing new algorithm runs
until time budget exhausted
Bayesian Optimization Visualized

\[ f(\theta) \]

\[ \theta \]
Algorithm 1: SMAC (high-level overview)

Initialize by executing some runs and collecting their performance data

repeat

Learn a model $\hat{m}$ from performance data so far: $\hat{m} : \Theta \times \mathcal{I} \to \mathbb{R}$

Use model $\hat{m}$ to select promising configurations $\Theta_{new}$

$\leadsto$ **Bayesian optimization with random forests**

Compare $\Theta_{new}$ against best configuration so far by executing new algorithm runs

$\leadsto$ **How many instances to evaluate for $\theta \in \Theta_{new}$?**

until *time budget exhausted*
Using a fixed number of $N$ instances is suboptimal
- Large $N$: too slow
- Small $N$: too noisy, overfitting

Adaptive choice of $N$ (in FocusedILS & SMAC)
- Start with $N=1$, reject aggressively
- Increase only for good configurations

**Theorem**

Let $\Theta$ be finite. Then, when using aggressive racing, the probability that ParamILS and SMAC find the true optimal parameter configuration approaches 1.
- Poor configurations often take a very long time (e.g., 1h vs. 1s)
- We can cap their evaluation when we know them to be worse than the incumbent

\[ c(\theta_{inc}, \pi) = c_{inc} \quad c(\theta', \pi) > c_{inc} \]

**Theorem**

Let \( \Theta \) be finite. Then, when using aggressive racing and adaptive capping, the probability that ParamILS and SMAC find the true optimal parameter configuration approaches 1.
Algorithm 1: SMAC

Initialize by executing some runs and collecting their performance data

repeat

  Learn a model $\hat{m}$ from performance data so far: $\hat{m} : \Theta \times \mathcal{I} \rightarrow \mathbb{R}$

  Use model $\hat{m}$ to select promising configurations $\Theta_{new}$

  $\leadsto$ **Bayesian optimization with random forests**

  Compare $\Theta_{new}$ against best configuration so far by executing new algorithm runs

  $\leadsto$ **Aggressive racing and adaptive capping**

until *time budget exhausted*
All of these components matter for performance

Example: optimizing CPLEX on combinatorial auctions (Regions-100)

Hutter et al, 2020
Parameter space for Fast Downward

- Choice of heuristics & subparameters
  - $h_{lm} (\times 12)$
  - $h_{lmcut} (\times 2)$
  - $h_{add} (\times 3)$
  - $h_{cg} (\times 3)$
  - $h_{cea} (\times 3)$
  - $h_{ff} (\times 3)$
  - $h_{goal\_count} (\times 3)$
  - $h_{mas} (\times 4)$
  - $h_{hm} (\times 2)$
  - $h_{blind}$
  - $h_{max}$

- Search
  - 8 additional parameters

- In total: 45 params, $2.99 \times 10^{13}$ configs

Domain-wise configuration with FocusedILS

Result: over 10x speedup on average
Per domain: 1x – 23x speedup
AC Application #2: Configuration of LPG

[Vallati et al, SOCS 2013]

- Parameter space for LPG (local search on linear action graph)
  - Preprocessing ($\times 6$)
  - Search strategy ($\times 15$)
  - Flaw selection strategy ($\times 8$)
  - Search neighbourhood ($\times 6$)
  - Heuristic function ($\times 17$)
  - Reachability information ($\times 7$)
  - Search randomization ($\times 3$)

In total: 62 params, $6.5 \times 10^{17}$ configs

- Domain-wise configuration with FocusedILS

Configuration can also improve quality

Result: over 10x speedup on average
Per domain: $3x – 118x$ speedup
AC Application #3: Domain Model Configuration

[Vallati et al, IJCAI 2015]

- Parameter space for any planner, for how to rewrite the PDDL file
  - Order of domain predicates
  - Order of operators
  - Within each operator:
    - Order of preconditions
    - Order of postconditions
  - Up to **109 continuous parameters** configured with SMAC

- Analysis can provide useful information to effectively engineer domain models
  - fANOVA parameter importance suggests:
    - First list operators that are used most/early
    - First list preconditions unlikely to be satisfied

Yahsp on Depots
Per domain: **1x – 339x speedup**
• For any planner, how to rewrite the problem model file

Original: (on-table A), (on-table B), (on C A), (clear C), (clear B), (handempty)

Configured: (on C A), (on-table B), (on-table A), (clear B), (clear C), (handempty)

• Need a domain-specific heuristic that applies for all problems in the domain
  – Construct a parameterized heuristic using features of facts in the planning encoding graph (PEG)
  – Configure the heuristic’s 26 parameters by SMAC

• Per domain: 1x – 39x speedup

• Analysis can provide useful information to effectively engineer problem models
  – fANOVA parameter importance suggests:
    • Initial and goal states’ ordering should be aligned
    • First list propositional facts that often occur in preconditions & often occur positively
    • First list propositional facts that are most connected in the PEG
• Part 1: an overview of previous meta-algorithmic approaches
  – Algorithm Configuration
  – Algorithm Portfolios

• Part 2: Dynamic Algorithm Configuration
No single algorithm or parameter setting works best everywhere

→ Exploit the complementary strengths of different planners

**Algorithm schedules**
- Very popular in planning
- First work on schedules already goes back two decades! [Howe et al, ECP 1999]
- Fast Downward Stonesoup [Helmert et al, ICAPS-WS 2011] has been very successful in the IPC

**Algorithm selection**
- Has been less popular in planning
- **IBaCoP** [Cenamor et al, IPC 2012, ICAPS-PAL 2013 & JAIR 2016]
  - Per-instance selection of top algorithms (to be combined in a schedule)
• How can we characterize the fingerprint of a planning instance?

• **311 features** from several categories
  – PDDL features by Roberts et al [ICAPS 2008]
  – FDR features (from translation to finite domain representation)
  – Causal and domain transition graph features by Cenamor et al [ICAPS-PAL 2013]
  – LPG preprocessing
  – Torchlight search sampling
  – FD probing from running FastDownward for 1s
  – SAT representation
  – Success & timing

• Better results with more features (based on random forests [Hutter et al, AIJ 2014])
• **Delfi** [Katz et al, IPC 2018; Sievers et al, AAAI 2020]
  – Algorithm selection with CNNs based on an image encoding of abstract structure graph

• **Simple graph features** [Ferber, ICAPS WS 2020; Ferber & Seipp, ICAPS WS 2020]
  – Simple ML techniques perform similarly with simple statistics of the graph

• **Graph convolutional neural networks (GCNs)** [Ma et al, AAAI 2020]
  – Perform better than CNNs on graph encoding
Outside of planning, many more algorithm selection methods exist.

We spanned a design space over them: 54 parameters.

Used SMAC to find instantiation with best cross-validation performance.

Won ICON challenge on algorithm selection, categories #solved & PAR10.
Combining AC + Portfolios

• AC and portfolios have opposite strengths
  – AC finds great configurations for homogeneous instance distributions
  – Portfolios take these as inputs to address heterogeneous distributions

• Combining AC & algorithm selection (per-instance AC)
  – ISAC [Kadioglu et al, ECAI 2010]
    • Cluster instances, use AC for each cluster
  – Hydra [Xu et al, AAAI 2010; IJCAI-RCRA 2011]
    • Use AC to search for the configuration maximally improving an algorithm selector

• Combining AC & algorithm schedules
  – Seipp et al [ICAPS 2012]
    • Use AC for several planning domains; combine the result into a schedule with uniform time shares
  – Cedalion [Seipp et al, AAAI 2015]
    • Similar to Hydra, but for schedules: search for configuration + time slot to add
• Part 1: an overview of previous meta-algorithmic approaches
  – Algorithm Configuration
  – Algorithm Portfolios

Part 2: Dynamic Algorithm Configuration
Which Planning Parameters Could be Adapted?

- **Heuristics**
  - Which heuristics to use
  - Subparameters of each heuristic
  - How to combine the heuristics

- **Search strategy**
  - Global / Local search
  - Randomization
  - How to combine them

- **Problem encoding**
  - Domain model
  - Problem model

- **LPG’s local search parameters**

- **Further promising parameters**
  - Merge strategies for merge & shrink
  - In general, when to do X
    - E.g., when to derive a new heuristic, when to discard an old one
• Early pioneering work by Lagoudakis & Littmann
  – Lagoudakis, Littmann & Parr [2001]: State-specific selection of sorting algorithm
  – Lagoudakis & Littmann [2004a]: State-specific selection of branching rules in DPLL for SAT
  – Lagoudakis & Littmann [2004b]: Reinforcement Learning for Algorithm Selection

• Very recent related work in AI planning by Gomoluch et al
  – Policy gradient for learning to switch between search methods [Gomoluch et al, ICAPS 2019]
    • Tabular state space (4 states) and action space (5 search methods)
  – Blackbox optimization of neural search policy [Gomoluch et al, ICAPS 2020]
    • Adaptive parameterization of mix between global & local best first search and random moves
Dynamic Algorithm Configuration (DAC)

Definition: Algorithm Configuration (AC)
Given:
- a parameterized algorithm $A$ with configuration space $\Theta$
- a distribution $\mathcal{D}$ over problem instances with domain $\mathcal{I}$
- a cost metric $c : \Theta \times \mathcal{I} \rightarrow \mathbb{R}$ assessing the cost of a config. $\theta \in \Theta$ on a instance $i \in \mathcal{I}$

Find: $\theta^* \in \arg \min_{\theta \in \Theta} \mathbb{E}_{i \sim \mathcal{D}} [c(\theta, i)]$

Definition: Dynamic AC (DAC)
Given:
- a parameterized algorithm $A$ with configuration space $\Theta$
- a distribution $\mathcal{D}$ over problem instances with domain $\mathcal{I}$
- a space of dynamic configuration policies $\pi \in \Pi$ with $\pi : S \times \mathcal{I} \rightarrow \Theta$ that adaptively choose a configuration $\theta \in \Theta$ for each instance $i \in \mathcal{I}$ and state $s \in S$ of $A$
- a cost metric $c : \Pi \times \mathcal{I} \rightarrow \mathbb{R}$ assessing the cost of a policy $\pi \in \Pi$ on a instance $i \in \mathcal{I}$

Find: $\pi^* \in \arg \min_{\pi \in \Pi} \mathbb{E}_{i \sim \mathcal{D}} [c(\pi, i)]$
DAC as a Contextual Markov Decision Process

Definition: Dynamic AC (DAC)

Given:
- a parameterized algorithm $A$ with configuration space $\Theta$
- a distribution $D$ over problem instances with domain $I$
- A space of dynamic configuration policies $\pi \in \Pi$ with $\pi : S \times I \rightarrow \Theta$ that adaptively choose a configuration $\theta \in \Theta$ for each instance $i \in I$ and state $s \in S$ of $A$

A cost metric $c : \Pi \times I \rightarrow \mathbb{R}$ assessing the cost of a policy $\pi \in \Pi$ on a instance $i \in I$

Find: $\pi^* \in \arg \min_{\pi \in \Pi} \mathbb{E}_{i \sim D}[c(\pi, i)]$

DAC as a contextual MDP

DAC can be formalized as a contextual MDP $\mathcal{M}_I = \{\mathcal{M}_i\}_{i \sim I}$, where each $\mathcal{M}_i$ is an MDP:
- State Space $S$
- Action Space $\Theta$
- Transition Function $T_i$
- Reward Function $R_i$
DAC Strictly Generalizes All Formulations We’ve Seen

<table>
<thead>
<tr>
<th>Meta-algorithmic problem</th>
<th>Formally</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>$\pi : \emptyset \rightarrow \Theta$</td>
</tr>
<tr>
<td>Selection</td>
<td>$\pi : \mathcal{I} \rightarrow \mathcal{A}$</td>
</tr>
<tr>
<td>Schedules</td>
<td>$\pi : \mathbb{R}^+ \rightarrow \mathcal{A}$</td>
</tr>
<tr>
<td>Selection + schedules</td>
<td>$\pi : \mathcal{I} \times \mathbb{R}^+ \rightarrow \mathcal{A}$</td>
</tr>
<tr>
<td>AC + selection</td>
<td>$\pi : \mathcal{I} \rightarrow \Theta$</td>
</tr>
<tr>
<td>AC + schedules</td>
<td>$\pi : \mathbb{R}^+ \rightarrow \Theta$</td>
</tr>
<tr>
<td>DAC</td>
<td>$\pi : \mathcal{I} \times \mathbb{R}^+ \times \mathcal{S} \rightarrow \Theta$</td>
</tr>
</tbody>
</table>

Notation reminder:
- $\mathcal{I}$: Instances
- $\Theta$: Configuration space
- $\mathcal{A}$: Set of algorithms
- $\mathbb{R}^+$: Positive real numbers (time steps)
- $\mathcal{S}$: State space
- $\pi$: Policy

**Proposition**
The optimal DAC policy is **at least as good** as the optimal solution of any of the above.

**Theorem**
The optimal DAC policy can be **exponentially better** than the optimal selector or schedule.
Evidence that DAC Might Be a Useful Abstraction

- **Experiments on whitebox/toy benchmarks** [Biedenkapp et al, ECAI 2020]
  - Strong generalization across instances
  - Moderate scaling with number of parameters
  - Found optimal solution in a task that required using both instance & state features

- **DAC for controlling the step size in CMA-ES** [Shala et al, PPSN 2020]
  - Guided policy search [Levine & Kuhn, 2013], learning from an existing heuristic

- **DAC for selecting heuristics in AI planning** [Speck et al, ICAPS-PRL 2020]
DAC for Selecting Heuristics in AI planning

[Speck et al, ICAPS-PRL 2020]

What state should I expand next?

I will tell you from my experience!

State $s$!

State $t$!

State $u$!

State $v$!

$h_1$

$h_2$

$h_3$

$h_4$

Who is correct?

Frank Hutter: Getting the most out of your planner(s)

Slides available at http://www.automl.org/talks
• Satisficing planning
  – Search for a good plan
  – Inadmissible heuristics are difficult to combine

• Greedy search with multiple heuristics [Helmert, JAIR 2006]
  – One separate open list for each heuristic
  – Each heuristic is evaluated at each step
  – Alternation strategy can be better than any single heuristic [Röger & Helmert, ICAPS 2010]
  – Can we do better than alternation?

Theorem
For each algorithm schedule $\pi_{sched}$ and each algorithm selector $\pi_{sel}$, there exists a family of planning instances $i_n$, a collection of heuristics $H$ and a dynamic control policy $\pi_{dac}$, so that greedy best-first search with $H$ and $\pi_{dac}$ expands exponentially less states in $i_n$ than greedy best-first search with $H$ and $\pi_{sched}$ or $\pi_{sel}$ until a plan is found.
• **Action Space**
  – 4 different heuristic functions: $h_{ff}$, $h_{cg}$, $h_{cea}$, $h_{add}$

• **State space**
  – Time step $t$
  – Simple features over the states in the open list of each considered heuristic:
    • $\max_h$, $\min_h$, $\mu_h$, $\sigma^2_h$, $\#h$
    • Actually taking the difference of each feature between $t-1$ and $t$

• **Reward**
  – Simply -1 for each expansion step until solution is found

• **RL strategy**
  – $\epsilon$-greedy deep Q-learning with a double DQN [van Hasselt et al, 2015]
  – Simple feed-forward network with 2 hidden layers of 75 units each
Experiments: Coverage on Unseen Test Instances

[Speck et al, ICAPS-PRL 2020]

- Experimental setup: domain-wise training on 6 domains
  - 100 train & 100 test instances each

- Baselines
  - All single heuristics & oracle per-instance selector
  - Random and alternating heuristic

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>CONTROL POLICY</th>
<th>SINGLE HEURISTIC</th>
<th>BEST AS (ORACLE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain (# Inst.)</td>
<td>RL</td>
<td>RND</td>
<td>ALT</td>
</tr>
<tr>
<td>BARMAN (100)</td>
<td>84.4</td>
<td>83.8</td>
<td>83.3</td>
</tr>
<tr>
<td>BLOCKSWORLD (100)</td>
<td>92.9</td>
<td>83.6</td>
<td>83.7</td>
</tr>
<tr>
<td>CHILDSNACK (100)</td>
<td>88.0</td>
<td>86.2</td>
<td>86.7</td>
</tr>
<tr>
<td>ROVERS (100)</td>
<td>95.2</td>
<td>96.0</td>
<td>96.0</td>
</tr>
<tr>
<td>SOKOBAN (100)</td>
<td>87.7</td>
<td>87.1</td>
<td>87.0</td>
</tr>
<tr>
<td>VISITALL (100)</td>
<td>56.9</td>
<td>51.0</td>
<td>51.5</td>
</tr>
<tr>
<td><strong>SUM (600)</strong></td>
<td><strong>505.1</strong></td>
<td><strong>487.7</strong></td>
<td><strong>488.2</strong></td>
</tr>
</tbody>
</table>
Further Work Under Way on DAC

Exploiting that actions often need to be repeated many times
- Learn when to act
- TempoRL [Biedenkapp et al, 2020]

Active selection of instances that are helpful in learning
- Self-paced reinforcement learning
- Making use of changes in the value function [Eimer et al, 2020]

Creating a library of DAC benchmarks
- OpenAI gym format
- We would love to include your DAC problems
Choosing the right problem

– Where is DAC likely to help most?
– Which parameters are crucial to adapt?

Constrain DAC to simple strategies

– To aid interpretability

Combinations of AC & DAC

– Configuring many static parameters and some dynamic ones
Opportunities for Planning Experts #2: Feature Space

Instance features
- We have good instance features, but we don’t actually use them yet
- These might directly allow for domain-independent planning by DAC

State features
- We only had a very first shot at these
- Better state features will improve domain-specific & domain-independent planning

Let’s parse Fast Downward‘s log file?
Opportunities for Planning Experts #3: Insights

Use these data-driven tools to gain scientific understanding

1. Use AC/DAC to improve planner’s performance

2. Use meta-algorithmic tools to understand why performance improved
   - For AC, we have automated parameter importance analysis methods
     - Forward selection [Hutter et al, LION 2013]
     - Ablation analysis [Fawcett & Hoos, 2016; Biedenkapp et al, AAAI 2017]
     - Functional ANOVA [Hutter et al, ICML 2014] → [Vallati et al, IJCAI 2015] and [Vallati & Serina, ICAPS 2018]
     - CAVE framework to automatically generate reports [Biedenkapp et al, LION 2018]
   - For DAC, we still need to come up with such methods
     - E.g., can strong yet complex policies be approximated with a simpler one? (→ Ferber & Seipp, ICAPS WS 2020)

3. Use the gained insights to develop new & better algorithms
• Algorithm configuration (AC) is a reliable workhorse
  – Often leads to speedups of orders of magnitudes

• Dynamic algorithm configuration (DAC) is the new kid on the block
  – Strict generalization of AC, selection & schedules
  – Also much harder (RL setting)
  – First success stories, but still at an early state

• Please join us in making DAC a great thing for the community
  – Try DAC, break DAC, improve DAC 😊
  – We’re building a team of postdocs on DAC in Freiburg