Markus Wagner

Tuning your algorithms by brains and CPU
Markus Wagner

Tuning your algorithms/implementations by brains and CPU
Real-World Optimisation, a reminder

Premier’s Research and Industry Fund: $14.6m Research Consortium “Unlocking Complex Resources through Lean Processing” (BHP, OZ Minerals, …)

…and and ARC ITTC (starting now)

Everybody is after decision support!
(to be faster, decrease wear, deliver on time, …)

Problem-specific (or exact mathematical) algorithms not always available
... problem is not entirely understood
... objective function is based on a simulation
... not enough resources
Heuristic Optimisation – my field

Heuristic Approaches
... do not rely on gradient information.
... are less likely to get stuck due to inherent parallelism.

General Template

Examples: Local Search, Simulated Annealing, Evolutionary Algorithms, Ant-Colony Optimisation, ...

Successful applications in: automobile industry, clustering, computer vision, cognitive systems, data mining, energy production, hardware design, logistics, network design and security, predictive modeling, robotics, scheduling, software engineering (energy consumption), ...
0.1 Data-Driven Search-Based Software Engineering
0.2 "DUO": Data Mining Algorithms Using/Used-by Optimizers
Why did these people meet in Japan in Dec’17?

DSE = Data-Driven Search-based SE
DSE = Data-Driven Search-based SE

- Conceptually, common higher-level goal
  - supporting and giving insights to software engineers
## Some technical differences

<table>
<thead>
<tr>
<th></th>
<th>MSR (or: data mining, machine learning, …)</th>
<th>SBSE (or: “optimisation”)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inference</strong></td>
<td>induction, visualize</td>
<td>optimisation</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td>Faster, often more scalable</td>
<td>Becoming faster</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Collected before inference</td>
<td>Sampling controlled by inference</td>
</tr>
<tr>
<td><strong>Tools</strong></td>
<td>R, SciKitLearn, WEKA</td>
<td>jMetal, AutoWeka, AutoSklearn, Opt4j, DEAP</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>● e.g. defect prediction; ● StackOverflow mining</td>
<td>● minimize a test suite ● configure software</td>
</tr>
<tr>
<td><strong>problems</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Goals</strong></td>
<td>e.g. just a few: recall, precision, MRE</td>
<td>● domain-specific goals. ● meta-criteria (hypervolume, spread, IGD)</td>
</tr>
</tbody>
</table>
Optimization = surfing the landscape

*murmuration* of starlings
(learn safe “shapes” to avoid predators)

Particle Swarm Optimization:
new = old + $\phi_1 \times \text{rand}(\text{ourBest} - \text{now})$ ;; social cognition
+ $\phi_2 \times \text{rand}(\text{myBest} - \text{now})$ ;; private cognition

use data miners to learn the landscape, guide our optimizers?
Data-Driven Search-based SE (DSE)

- To solve an SE problem:
  - Insert a data miner into an optimizer;
  - Or use an optimizer to improve a data miner.

- A new era for MSR (better MSR)
- A new era for SBSE (better SBSE)

Note: in 2018, we called this “DSE” (following Japan ‘17)
In 2019, we decided to broaden it to “DUO: Data Mining Algorithms Using/Used-by Optimizers”.
A new era for SBSE: Supercharging MSR

- Black art: hyperparameter optimization
- E.g. learning how many trees in a random forest
- E.g. learning how many “k” in kth-nearest neighbours
- Thanks to SBSE: massive improvements in, say, defect prediction
  - e.g. Agrawal & Menzies, ICSE 2018
  - performance details (after - before) tuning
A new era for SBSE: Let MSR help you run faster

- Landscape analysis
  - Find the lay of the land (shape of data)
  - Jump faster to better conclusions
  - e.g. GALE, TSE 2015
- Note that this “optimizer” is really a “data miner”
  - clustering, PCA
Q: Why explore MSR+SBSE?
A: So many application areas

1. Requirements Menzies, Feather, Bagnall, Mansouri, Zhang
2. Transformation Cooper, Ryan, Schielke, Subramanian, Fatiregun, Williams
3. Effort prediction Aguilar-Ruiz, Burgess, Dolado, Lefley, Shepperd
4. Management Alba, Antoniol, Chicano, Di Pentam Greer, Ruhe
5. Heap allocation Cohen, Kooi, Srisa-an
6. Regression test Li, Yoo, Elbaum, Rothermel, Walcott, Soffa, Kampfhamer
7. SOA Canfora, Di Penta, Esposito, Villani
8. Refactoring Antoniol, Briand, Cinneide, O’Keeffe, Merlo, Seng, Tratt
9. Test Generation Alba, Binkley, Bottaci, Briand, Chicano, Clark, Holcombe, Harrold, Jones, Korel, Pargass, Xanthakis, Sthamer, Tracy, Tonella, Xanthakis, Xiao, Wegener, Wilkins
10. Maintenance Antoniol, Lutz, Di Penta, Madhavi
11. Model checking Alba, Chicano, Godefroid
12. Probing Cohen, Elbaum
13. Comprehension Gold, Li, Mahdavi
14. Protocols Alba, Clark, Jacob, Troya
15. Component sel Baker, Skaliotis, Steinhofel, Yoo
16. Agent Oriented Haas, Peysakhov, Sinclair, Shami, Mancoridis
Q: Why explore MSR+SBSE?
A2: cause you got to

- How to get a paper rejected (in 2020+):
  - Publish data mining results without hyper-parameter optimization

- Coming to the end of “merely mining”
  - See debates on “unsupervised learning”
    - Too easy to just chase precision, recall etc.

- Complex problems need complex inference
  - e.g. minimizing #false alarms before first defect [Huang et al. ICSME’17]
http://tiny.cc/data-SE: A new resource for MSR researchers
89 DSE artifacts, in 13 groups
(e.g. RE, software product lines, software processes)

<table>
<thead>
<tr>
<th>Domain</th>
<th>Problem</th>
<th>Decision Space</th>
<th>C/D</th>
<th>Projects</th>
<th>Description</th>
<th>Links</th>
<th>Related Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSR</td>
<td>Defect Prediction</td>
<td>Numeric</td>
<td>D</td>
<td>10</td>
<td>CK Metric</td>
<td>raise_data_defect</td>
<td>[29]</td>
</tr>
<tr>
<td></td>
<td>Text Classification</td>
<td>Text</td>
<td>-</td>
<td>6</td>
<td>Citemap</td>
<td>raise_data_pits</td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td>Performance Optimization</td>
<td>Mixed</td>
<td>D</td>
<td>22</td>
<td>Performance Configuration optimization</td>
<td>raise_data_perf</td>
<td>[66, 67, 69]</td>
</tr>
<tr>
<td></td>
<td>Software Product Lines</td>
<td>Boolean</td>
<td>D</td>
<td>5</td>
<td>Product Lines</td>
<td>raise_data_SPL</td>
<td>[15]</td>
</tr>
<tr>
<td></td>
<td>General Optimization</td>
<td>Numeric</td>
<td>C</td>
<td>7</td>
<td>DTLZ</td>
<td>raise_dtlz_zdt</td>
<td>[65]</td>
</tr>
<tr>
<td></td>
<td>Workflow</td>
<td>Numeric</td>
<td>D</td>
<td>20</td>
<td>Workflow</td>
<td>raise_gen_workflow</td>
<td>[14]</td>
</tr>
<tr>
<td></td>
<td>Text Discovery</td>
<td>Text</td>
<td>-</td>
<td>4</td>
<td>Reading Faster</td>
<td>raise_data_fastread</td>
<td>[103]</td>
</tr>
<tr>
<td></td>
<td>Software Processes</td>
<td>Numeric</td>
<td>C</td>
<td>5</td>
<td>Xomo</td>
<td>raise_pom_xomo</td>
<td>[16, 65]</td>
</tr>
<tr>
<td></td>
<td>Requirement Engineering</td>
<td>Numeric</td>
<td>D</td>
<td>8</td>
<td>Requirement Engineering</td>
<td>raise_short</td>
<td>[57]</td>
</tr>
</tbody>
</table>

existing results; useful for testing new methods
DUO: Data Mining Algorithms Using/Used-by Optimizers

Follow-up paper to DSE, [https://arxiv.org/abs/1812.01550](https://arxiv.org/abs/1812.01550)

- Systematic literature review
- Comes with tutorial, and “A Dozen Tips for Using Optimisers” & “A Dozen Tips for Using Data Mining”
- Broadening of claims... most importantly: “Claim 4 Data Mining without optimisation is deprecated not recommended”
  - “Algorithm X is better than Y” invalidated when tuning
  - “feature X is relevant” invalidated when tuning
  - “Problem insights” invalidated when algorithms were tuned
  - And: tuning can result in better results faster
Something is changing. Things are .... Different ....

Strange new words:
- “hyper-parameter optimization”
- “evolutionary algorithms”
- “differential evolution”
- “model-based reasoning”

What is going on?

Other tools: irace, smac, gga+, ... auto-sklearn (= sklearn + smac), ...
...Intermezzo...

Cool technology:
- **Algorithm configuration (AC):** irace, smac, ...
- **Algorithm selection (AS):** given a portfolio of algorithms (or configs), which algorithm should you run when given a new instance? *(very disruptive at SAT competitions)*
- **Per-instance-algorithm-configuration (PIAC):** briefly look at an instance, and prescribe a good configuration of your highly configurable solver
- **SBSE at the source-code level:** optimise code! (aka “genetic improvement of software”, GI, [https://tinyurl.com/gitutorial](https://tinyurl.com/gitutorial))

Note: why not tune your single solver several times, for classes of instances? --
--> create your own, custom portfolio of overfitted customised configurations
A Generic Bet-and-run Strategy for Speeding Up Stochastic Local Search

HELLO IT
HAVE YOU TRIED TURNING IT OFF AND ON AGAIN?
**Restarts**

- A desktop PC does not work properly → we restart it.

- Performance of stochastic algorithm and randomized search heuristics unsatisfactory → we restart it again and again.

- While this approach is well-known, few algorithms directly incorporate such restart strategies.

- Potential reason: added complexity of designing an appropriate restart strategy that is advantageous for the considered algorithm.

- We are looking for: a generic framework for restart strategies that is not overly dependent on the algorithm used and the problem considered.
**Related work**

**Bet-and-Run by Fischetti and Monaci (2014)**

- **Phase 1**: of length $k \cdot t_1$
- **Phase 2**: of length $t_2 = t - k \cdot t_1$

---

**Notes**

- **Single-run**: $k=1$
- **Multi-run with restarts from scratch**: $t_1 = t/k$ and $t_2 = 0$

**Fischetti and Monaci (2014)**

“Exploiting erraticism in search”

- $k=5$, CPLEX, diversity, MIPlib 2010

**de Perthuis de Laillevault, Doerr, and Doerr (2015)**

- 1+1-EA on OneMax
- Possible additive runtime gain of order $\sqrt{n \log n}$
Related work
Bet-and-Run by Fischetti and Monaci (2014)

Implementation Detail:
The initial runs can be run sequentially – they don’t have to be in parallel. Keep in mind: our goal is to make best use of some total computation budget \( t \), not of some wallclock time.

Single-run:
\[
k = 1
\]

Multi-run with restarts from scratch:
\[
t_1 = t/k \quad \text{and} \quad t_2 = 0
\]

Fischetti and Monaci (2014)
“Exploiting erraticism in search”
k=5, CPLEX, diversity, MIPlib 2010

de Perthuis de Laillevault, Doerr, and Doerr (2015)
1+1-EA on OneMax
possible additive runtime gain of order \( \sqrt{n \log n} \)
A Generic Bet-and-Run Strategy (AAAI 2017)

Example: FastVC on MVC instance shipsec1.mtx
Total budget $t=240s$
Shown in colour is absolute distance to best-found (117,366).

- This column: 1 restart (single naïve run)
- This area: bet-and-run advantage
- This diagonal: $k$ runs of $t/k$ length (naïve restarts)
Cross Domain Study Summary (~200 instances, 1 Bet-and-Run strategy vs 1 single run, AAAI 2017)

Universally good (given our experiments): Restarts\(^{40\%}\)
- Phase 1: 40 runs, each with a time budget of 1% of the total time budget
- Phase 2: use the remaining 60% to continue the best run of Phase 1

Comparison of our “universal” Restarts\(^{40\%}\) with a single run:
Wilcoxon-rank-sum test (p=0.05): green shows where Restarts\(^{40\%}\), is significantly better, grey (identical or insignificant), red (single run is better)

Exploitable erraticism using restarts:

Follow-up work: GECCO 2017 (theory), LION 2017 (reactive restarts), AAAI 2019 (more general...)

Total time limit: \(100 \cdot t_{\text{init}}\), \(400 \cdot t_{\text{init}}\), \(1000 \cdot t_{\text{init}}\)
Simple On-the-Fly Parameter Selection
Evolutionary algorithms and related iterative optimization heuristics are parametrized algorithms

Example: $(\mu + \lambda)$ EAs

Parameters:
- Memory size $\mu$
- Offspring population size $\lambda$
- Crossover rate
- Mutation rate, search radius, etc.
- Selective pressure

How shall I set these parameters to get a well-performing EA?
Parameter Tuning vs. Parameter Control

- **Parameter Tuning:**
  - Initial set of experiments
  - Deduce reasonable parameter settings
  - Does not have to be done manually, but a number of powerful, ready-to-use tools available: irace, SPOT, ParamILS, SMAC, GGA,…

- **Parameter Control:**
  - 2 main differences:
    - Parameters are set *while* optimizing
    - Parameters *change over time*:
      Key motivation: different parameter values can be optimal in different stages of an optimization process
Parameter Tuning vs. Parameter Control

- **Parameter Tuning:**
  - Initial set of experiments
  - Deduce reasonable parameter settings
  - Does not have to be done manually, but a number of powerful, ready-to-use tools available: irace, SPOT, ParamILS, SMAC, GGA,...

- **Parameter Control:**
  - 2 main differences:
    - Parameters are set *while* optimizing
    - Parameters *change over time*:
      Key motivation: different parameter values can be optimal in different stages of an optimization process
Parameter Control

- Example: LeadingOnes: $\text{LO}(110110101010)=2$
- Randomized Local Search: flip $k$ bits, keep the better of parent and offspring

\[ k_{opt}(i) = \left\lfloor \frac{n}{\text{LO}(x)+1} \right\rfloor \]
Parameter Control

- Example: LeadingOnes: LO(110110101010)=2
- Randomized Local Search: flip $k$ bits, keep the better of parent and offspring
- $n=1000$

$$k_{opt}(i) = \left\lfloor \frac{n}{\text{LO}(x)+1} \right\rfloor$$
Parameter Control

- Example: LeadingOnes: \( \text{LO}(110110101010) \cdot 2 \)
- Randomized Local Search: flip offspring
- \( n = 1000 \)

\[
\text{k}_{\text{opt}}(i) = \left\lceil \frac{n}{\text{LO}(x)+1} \right\rceil
\]

\[
E[T(A,i)]
\]

22% smaller optimization time

How can I find/predict such a dependence???
Good News: You Don’t Have to!

- Easy mechanisms which find close-to-optimal parameter values automatically:

![Graph showing optimal and average mutation strength over LO(x) values.](image-url)
Good News: You Don’t Have to!

- With close-to-optimal performance:

![Graph showing mutation strength and hitting time across different mutation strengths and LO(x) values. The graph illustrates an 14% performance gain with optimal mutation strength compared to other strategies.]
Success-Based Multiplicative Update Rule

Algorithm 2: The \( (1 + 1) \) EA\( _\alpha \) with update strengths \( A \) and \( b \) and initial mutation rate \( p_0 \in [1/n^2, 1/2] \) for the maximization of a pseudo-Boolean function \( f : \{0, 1\}^n \rightarrow \mathbb{R} \)

1. **Initialization:** Sample \( x \in \{0, 1\}^n \) uniformly at random and compute \( f(x) \);
2. Set \( p = p_0 \);
3. **Optimization:** for \( t = 1, 2, 3, \ldots \) do
   4. Create offspring \( y \) through standard bit mutation with mutation probability \( p \)
   5. if \( f(y) \geq f(x) \) then
      6. \( x \leftarrow y \) and \( p \leftarrow \min\{A \cdot p, 1/2\} \)
   7. else
      8. \( p \leftarrow \max\{b \cdot p, 1/n^2\} \)

\( A > 1 \)  
\( b < 1 \)
LeadingOnes

- Average optimization time for different combinations of $A$ and $b$ (101 independent runs)

For comparison: RLS needs $n^2/2$ iterations (=0.5 and =3.125 above), (1+1) EA$_{>0}$ needs 0.54 and $3.4 \times 10^4$ iterations, respectively
1/5-th Success Rules

- 1/5-th success rule:
  - originally from continuous optimization [Rechenberg, Devroye, Schumer/Steiglitz]
  - (1+1) ES optimizing sphere \( f(x) = \sum x_i^2 \)
  - When success rate > 1/5: increase search radius
    When success rate < 1/5: decrease search radius

- In discrete optimization, e.g., [Kern/Müller/Hansen/Büche/Ocenasek/Koumoutsakos04, Auger09]:
  - When success rate \( \approx 1/5 \), parameter value should be stable
  - In our algorithm:
    \[
    \text{If } f(y) \geq f(x): p \leftarrow \min \left\{ Ap, \frac{1}{2} \right\}
    \text{else } p \leftarrow \max \{bp, 1/n^2\}
    \]
  - \( A = \left(\frac{1}{b}\right)^{1/4} \) since \( Ab^4 = 1 \)
  - \( b = \frac{1}{A^4} \)
Heatmaps for OneMax

(a) **OneMax with** $n = 500$

(b) **OneMax with** $n = 1500$
Summary: when can this actually help you?

- This is all about greedily running into a local optimum!

Benefits:
- If your problem has a OneMax/LeadingOnes component, or several, then do this
- if RLS is a Local Search in your big approach, give this a try.

Further components that might prove helpful (not considered by us here):
- Ways to get out of a local optimum (see relationship to Simulated Annealing)
- Diversity
- Multi-objective approaches
- ...
Research Opportunities

- Performance on other test functions
- Real-world problems?
  → you are all cordially invited to collaborate on this!

- Use the scheme for other parameters, e.g., in combination with heavy-tailed mutation
  - Doerr GECCO 2017:
    don’t do 1/n, but draw k from a heavy-tailed distribution (has a parameter!)
  - Friedrich, Quinzan, Wagner GECCO 2018:
    flip 1 with 1/c
    flip k with (1-p)/(n-1)
  - Friedrich, Goebel, Quinzan, Wagner PPSN 2018:
    flip based on a mirrored, heavy-tailed distribution
Escaping Large Deceptive Basins of Attraction with Heavy-Tailed Mutation Operators
How to mutate?

- Many packages do this: if $n$ is the length of a solution, then perform mutation with probability $1/n$.
- Often found in theory: if $n$ is the bitstring of length $n$, then flip each bit with $1/n$. 
How to mutate?

- Many packages do this: if $n$ is the length of a solution, then perform mutation with probability $1/n$.
- Often found in theory: if $n$ is the bitstring of length $n$, then flip each bit with $1/n$
How to mutate?

- Many packages do this: if $n$ is the length of a solution, then perform mutation with probability $1/n$.
- Often found in theory: if $n$ is the bitstring of length $n$, then flip each bit with $1/n$.

**GECCO’17**: theoretical study, where the number of flipped bits is drawn from a “power law” distribution.

**Goal**: escape local optima.
How to mutate?

- Many packages do this: if \( n \) is the length of a solution, then perform mutation with probability \( 1/n \).
- Often found in theory: if \( n \) is the bitstring of length \( n \), then flip each bit with \( 1/n \).

GECCO’17: theoretical study, where the number of flipped bits is drawn from a “power law” distribution

Goal: escape local optima

This work: simpler operator, theory, experiments on minimum vertex cover + maximum cut
**Preliminaries**

**Algorithm 1:** General framework for the (1+1) EA

Choose initial solution $x \in \{0, 1\}^n$ u.a.r.;

while convergence criterion not met do

\[
y \leftarrow \text{Mutation}(x) \text{ for given mutation operator;}
\]

if $f(y) \geq f(x)$ then

\[
x \leftarrow y;
\]

return $x$;

return $y$;

**Algorithm 2:** The mutation operator $\text{fmut}_\beta(x)$

\[
y \leftarrow x;
\]

choose $k \in [1, \ldots, n/2]$ according to distribution $D^\beta_{n/2}$;

for $j = 1, \ldots, n$ do

\[
\text{if random}([0, 1])n \leq k \text{ then}
\]

\[
y[j] \leftarrow 1 - y[j];
\]

return $y$;

**Algorithm 3:** The mutation operator $\text{cMut}_p(x)$

\[
y \leftarrow x, k \leftarrow 1;
\]

if $\text{random}([0, 1]) > p$ then

\[
\text{choose } k \in \{2, \ldots, n\} \text{ u.a.r.;}
\]

flip $k$-bits of $y$ chosen u.a.r.;

return $y$;

---

Intuitively: probability to perform a $k$-bit mutation is $\sim k^\beta$
There is Theory…

OneMax\( (x_1, \ldots, x_n) = |x|_1 = \sum_{j=1}^{n} x_j \)

\[ \Rightarrow \quad O \left( \frac{n}{p} \log n \right) \]

**LEMMA 3.1.**

Jump\( (m, n)(x) = \begin{cases} 
  m + |x|_1 & \text{if } |x|_1 \leq n - m \text{ or } |x|_1 = n; \\
  n - |x|_1 & \text{otherwise};
\end{cases} \)

\[ \text{n=50} \quad \text{m=20} \quad \rightarrow \text{20-flip mutation needed!} \]

if m is constant \[ T_p(f) = \Theta(nT_\beta(f)) \]

if \( \ldots \leq m \leq n/2 \) \[ T_p(f) \leq \frac{c}{H(\beta)} T_\beta(f) \]

if n-m is constant \[ T_\beta(f) = 2^{\Omega(n)} \quad T_p(f) = n^{\Theta(1)} \]

More results on minimum vertex cover, max cut, submodular functions, …
Psst.... Let me tell you a secret... we evolved the distribution!

- Automated algorithm configuration using irace (iterated racing of configurations).
- Result when evolving for the family of Jump functions with \( n=10, \ m=1..5 \):

\[
\begin{array}{cccccccccc}
\text{# ACDT: Elite configurations (first number is the configuration ID):} & \text{p1} & \text{p2} & \text{p3} & \text{p4} & \text{p5} & \text{p6} & \text{p7} & \text{p8} & \text{p9} & \text{p10} \\
5599 & 0.70 & 0.03 & 0.03 & 0.02 & 0.02 & 0.02 & 0.04 & 0.04 & 0.02 & 0.06 \\
8176 & 0.69 & 0.07 & 0.04 & 0.02 & 0.01 & 0.01 & 0.02 & 0.07 & 0.02 & 0.06 \\
6578 & 0.70 & 0.02 & 0.02 & 0.02 & 0.04 & 0.04 & 0.06 & 0.01 & 0.02 & 0.07 \\
8991 & 0.71 & 0.04 & 0.03 & 0.01 & 0.06 & 0.04 & 0.02 & 0.02 & 0.01 & 0.05 \\
9143 & 0.75 & 0.02 & 0.00 & 0.01 & 0.02 & 0.00 & 0.04 & 0.04 & 0.03 & 0.08 \\
\end{array}
\]

- Looks like cmut, with \( p=0.70 \) and the rest is “evenly” distributed.

(we asked experts for the best distributions, and they did not know prior to our experiments)
Psst.... Let me tell you a secret... we evolved the distribution!

- Automated algorithm configuration using irace (irated racing of configurations).
- Result when evolving for the family of Jump functions with n=10, m=1..5:

  # ACDT: Elite configurations (first number is the configuration ID):

<table>
<thead>
<tr>
<th>#</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
<th>p5</th>
<th>p6</th>
<th>p7</th>
<th>p8</th>
<th>p9</th>
<th>p10</th>
</tr>
</thead>
<tbody>
<tr>
<td>5599</td>
<td>0.70</td>
<td>0.03</td>
<td>0.03</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
<td>0.04</td>
<td>0.02</td>
<td>0.06</td>
</tr>
<tr>
<td>8176</td>
<td>0.69</td>
<td>0.07</td>
<td>0.04</td>
<td>0.02</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
<td>0.02</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>6578</td>
<td>0.70</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.04</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>8991</td>
<td>0.71</td>
<td>0.04</td>
<td>0.03</td>
<td>0.01</td>
<td>0.06</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>9143</td>
<td>0.75</td>
<td>0.02</td>
<td>0.00</td>
<td>0.01</td>
<td>0.02</td>
<td>0.00</td>
<td>0.02</td>
<td>0.02</td>
<td>0.06</td>
<td></td>
</tr>
</tbody>
</table>

- Looks like cmut, with p=0.70 and the rest is "evenly" distributed.

(we asked experts for the best distributions, and they did not know prior to our experiments)
Summary: How to mutate?

This work: optimisation provided an insight!

Results:
- simpler operator, theory and experiments show that we beat the state-of-the-art (GECCO 2018, left)
- PPSN 2018 (right): max cut, different characteristics (symmetry) evolved to a different operator (→ specialised “further”!) and again sparked theoretical investigations
WRAP UP!!!

Two slides to go...
Make the best use of what you have!

Learn about the problem:
- Text analysis: 30 CPU years (MSR 2019)
- Restarts paper: 157 million experiments (AAAI 2019)
- Wave energy converters (GECCO 2018)
- Image quality assessment (COSEAL 2019): algorithm selection does not outperform the single best method
- heavy-tailed mutation (GECCO/PPSN 2018)

**DUO**
Step 1) Get into a local optimum, both in the solution space and in the algorithm-configuration space!
Step 2) Inspect these, and learn from them!

Tools: grid search, hill-climbers, parallel coordinate plots, brains, CPUs
Projects....

Other cool technology
- diversity optimisation in the “feature” space (good solutions that differ in characteristics, GECCO 2019 Best Paper Nomination)
- problems with multiple interconnected components (e.g. TSP + scheduling + knapsack)
- smartphones: minimisation of energy consumption (noisy!)
- genetic improvement of software: GIN tool (microframework for Java)
- crypto: minimisation of information leaks (program rewrites at assembly level)

➤ Lots of capable and user-friendly technology out there.
➤ Lots of simple things you can try out at home – hill-climbers are your friends, even grid search can be your friend
➤ @students: pay attention in your algorithms and maths courses 😊

[https://cs.adelaide.edu.au/~markus](https://cs.adelaide.edu.au/~markus) papers+slides online
Markus.Wagner@adelaide.edu.au
Relevant References since 2016

SBSE/DSE/DUO

Theory

Bet-and-run

Benchmarking

Multi-objective optimisation

Cycling
Problems with interdependent components

10. Feature analysis/diversity optimisation

9. Wave energy converters/wind energy