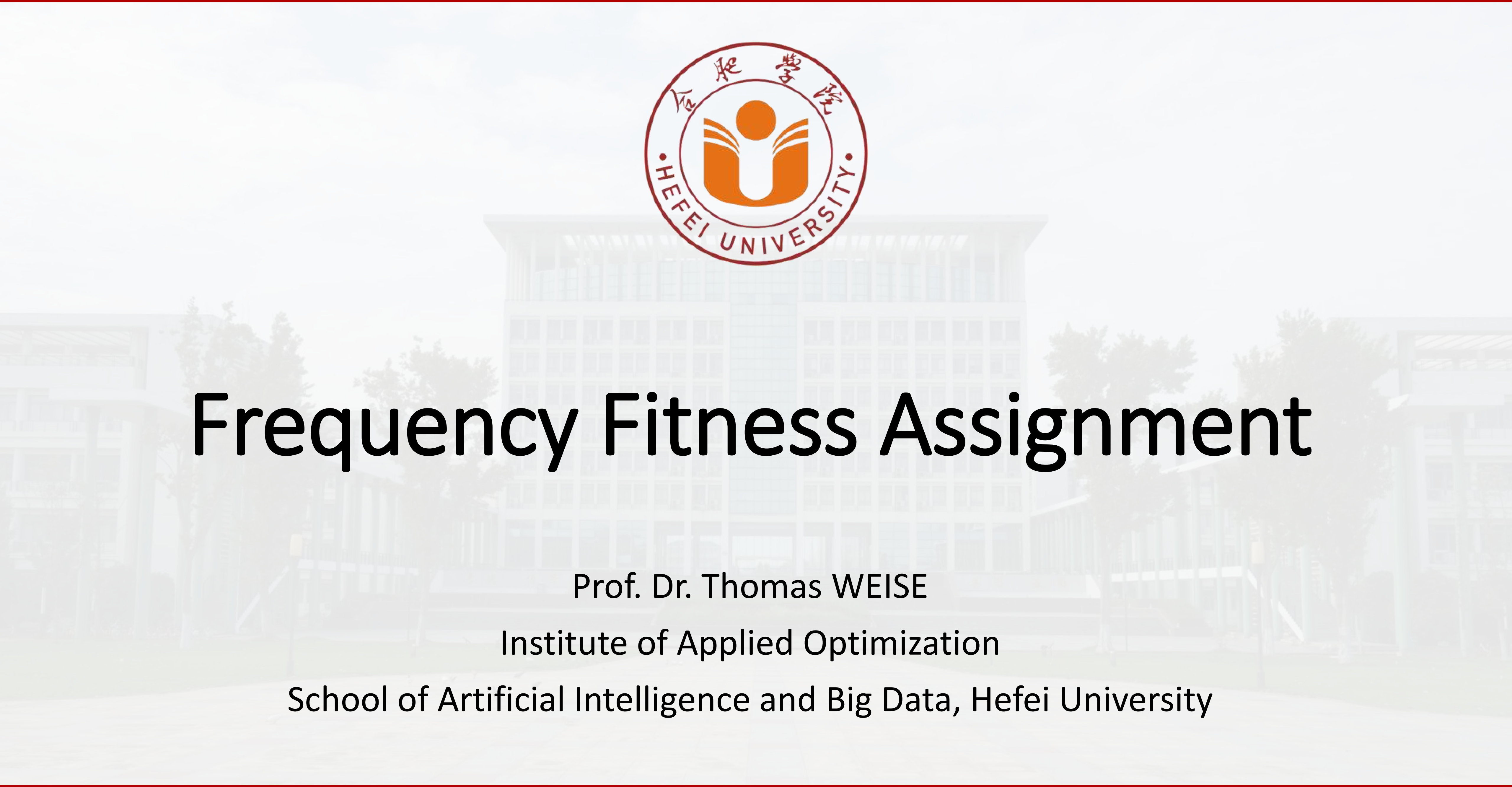


Frequency Fitness Assignment

Prof. Dr. Thomas WEISE

Institute of Applied Optimization

School of Artificial Intelligence and Big Data, Hefei University

Frequency Fitness Assignment

Slides

Video

1. Introduction into Optimization

Slides

Video

Introduction into Optimization

- Optimization means finding “superlatives”

biggest …

with the least energy…

…best trade-offs between ….

…highest quality

…longest possible duration

most efficient …

most precise …

cheapest …

fastest…

most similar to …

…with the highest score

… on the smallest possible area

most robust …

…shortest delay

Introduction: Optimization Problem

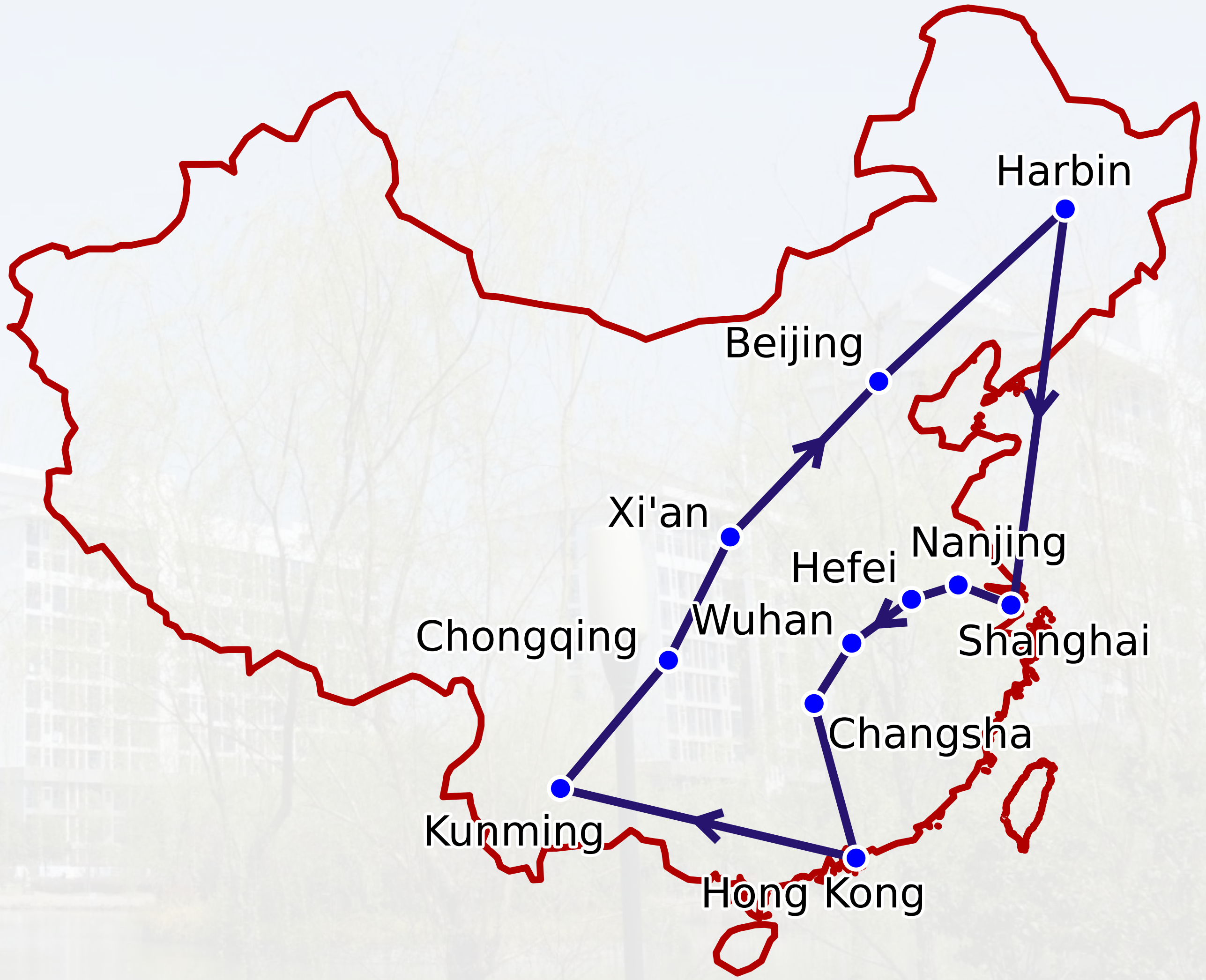
Optimization

An optimization problem is a situation which requires deciding for one choice from a set of possible alternatives in order to reach a predefined or required benefit at minimal costs.

Solving an optimization problem requires finding an input element $x^\star$ within a set $\mathbb{X}$ of allowed elements for which a mathematical function $f: \mathbb{X} \mapsto \mathbb{R}$ takes on the smallest possible value.

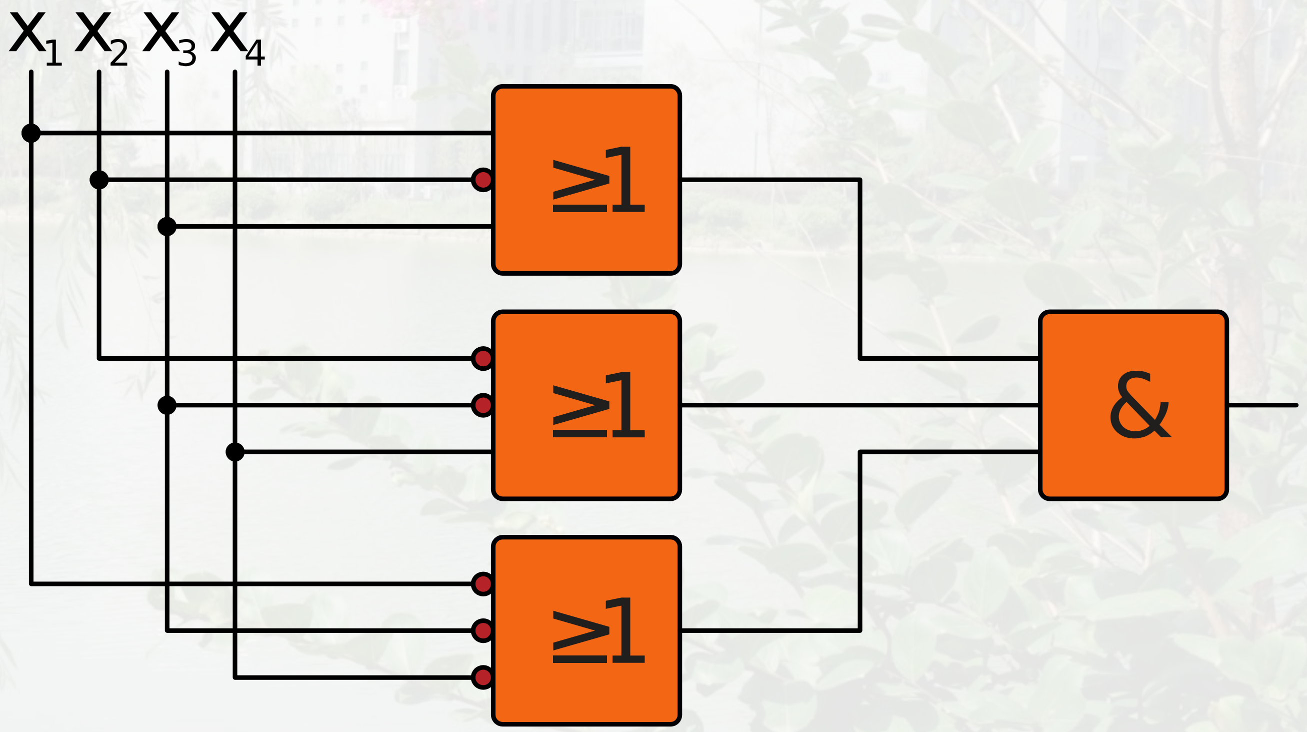
Introduction: Optimization Examples – TSP

- Traveling Salesperson Problem (TSP): $\mathbb{X}$ = the set of all possible round-trip tours through n given cities
- $f : \mathbb{X} \mapsto \mathbb{R}$: length of the tour
- optimal solution $x^{\star}$ = shortest possible tour



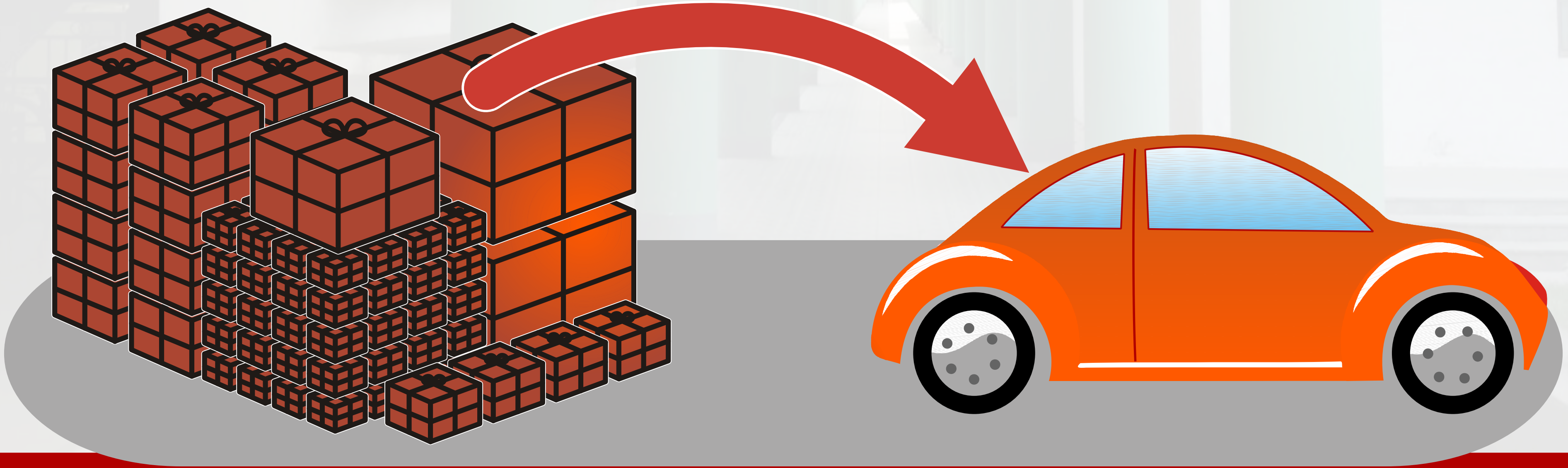
Introduction: Optimization Examples – MaxSat

- Maximum Satisfiability Problem (MaxSat): $\mathbb{X}$ = set of all possible bit strings of length n
- $f: \mathbb{X} \mapsto \mathbb{R}$: number of OR-clauses left unsatisfied
- optimal solution $x^{\star}$ = bit string that satisfies all OR clauses (and, hence, makes the AND clause become TRUE)



Introduction: Optimization Examples – Packing

- 1-D Bin Packing Problem: $\mathbb{X} =$ all possible orders to pack n objects into bins of a given size
- $f : \mathbb{X} \mapsto \mathbb{R}$: number of bins needed
- optimal solution $x^{\star} =$ the packing needing the fewest bins



Introduction: Optimization is Hard!

- Finding the globally optimal solution $x^\star$ from the set of all possible solutions $\mathbb{X}$ is often an $\mathcal{NP}$-hard problem.
- Currently, there is no algorithm that can **guarantee** to find the optimal solution of **every instance** of a given $\mathcal{NP}$-hard problem in a runtime that is not longer than polynomial in the size of the problem (i.e., existing algorithms may need exponential runtime in the **worst case**).
- In other words, if we want to guarantee to find the best possible solution $x^\star$ for all possible instances of a problem, we often cannot really be much faster than testing all possible candidate solutions $x \in \mathbb{X}$ in the worst case.

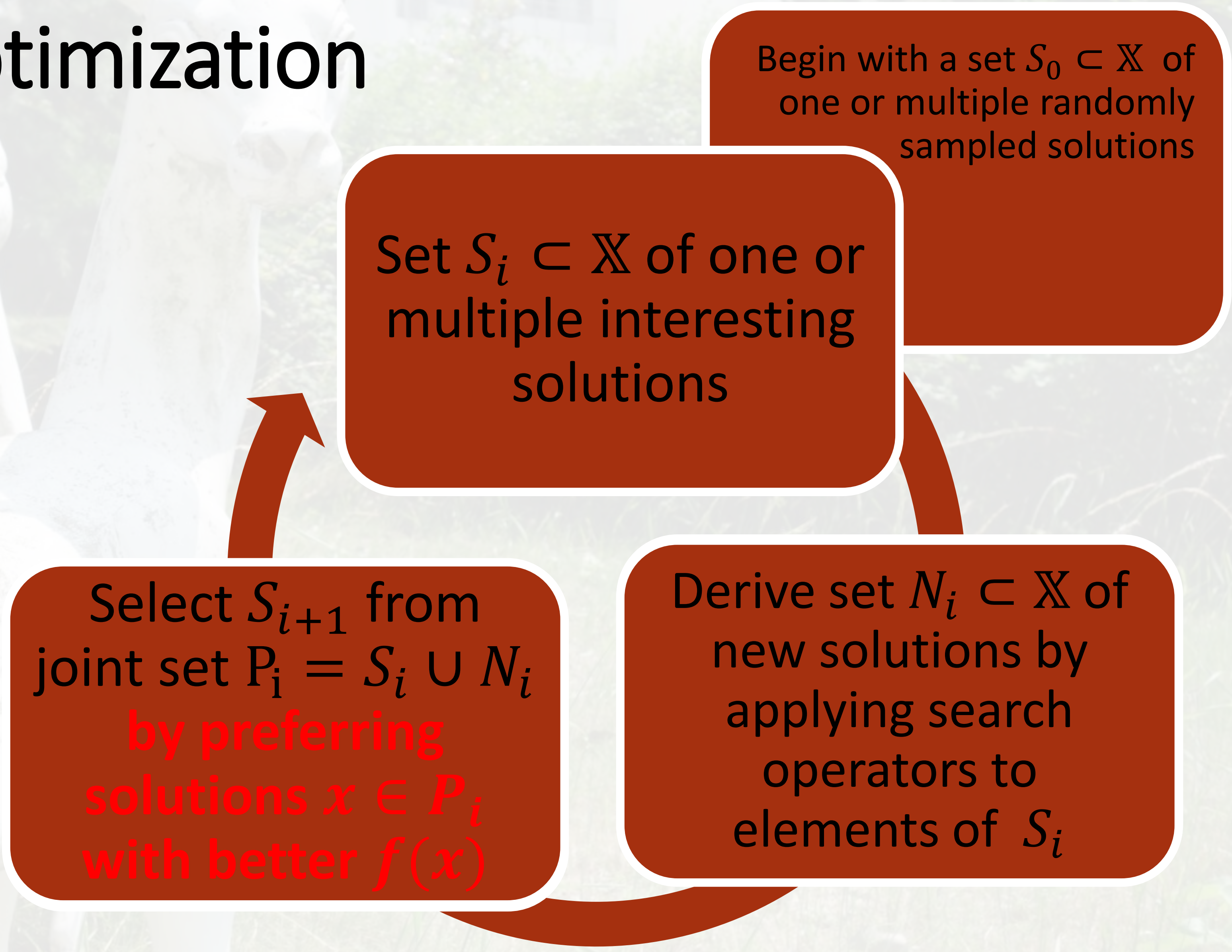
2. Metaheuristic Optimization

Slides

Video

Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error Idea of iterative improvement
- **Drop** the guarantee to find the optimal solution.
- Find good solution within **a feasible runtime**.



Begin with a set $S_0 \subset \mathbb{X}$ of one or multiple randomly sampled solutions
Set $S_i \subset \mathbb{X}$ of one or multiple interesting solutions
Derive set $N_i \subset \mathbb{X}$ of new solutions by applying search operators to elements of S_i
Select S_{i+1} from joint set $\mathrm{P_i} = S_i \cup N_i$ **by preferring solutions $x \in P_i$ with better $f(x)$**

Examples of Metaheuristics: (1+1) EA a.k.a. RLS

- Local Search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea
- accepts new solutions if better or equally good as current one

```
**procedure** $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$
    randomly sample $x_c$ from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
    **while** $\neg$ terminate **do**
        $x_n \leftarrow \mathtt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
        **if** $y_n \leq y_c$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    **return** $x_c, y_c$
```

Examples of Metaheuristics: Simulated Annealing

- SA is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference quality
- Probability regulated by temperature schedule with parameters T_0 and ε

```
procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, $T_0$, $\varepsilon$)
    randomly sample $x_c$ from $\mathbb{X}$;  $y_c \leftarrow f(x_c)$;
    $x_B \leftarrow x_c$;  $y_B \leftarrow y_c$;                    ▷ preserve best!
    $\tau \leftarrow 0$;                                  ▷ $\tau$ is iteration counter
    while ¬ terminate do
        $x_n \leftarrow \texttt{move}(x_c)$;  $y_n \leftarrow f(x_n)$;
        $\tau \leftarrow \tau + 1$;
        $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$;             ▷ $T$ decreases over time
        if $\mathfrak{R}_0^1 < e^{\frac{y_c-y_n}{T}}$ then      ▷ always true if $y_n \leq y_c$
            $x_c \leftarrow x_n$;  $y_c \leftarrow y_n$;
            if $y_c < y_B$ then $x_B \leftarrow x_c$;  $y_B \leftarrow y_c$;
    return $x_B$, $y_B$
```

Examples of Metaheuristics: Standard Genetic Alg.

- Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) for maximization
- Uses a population of size ps and unary and binary operator (with crossover rate cr)

```
procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, $ps$, $cr$)                      ▷ for maximization!
    $x_B \leftarrow \emptyset$;  $y_B \leftarrow -\infty$;                              ▷ best-so-far solution
    for $j \in 1 \dots ps$ do                                                  ▷ random initial population
        randomly sample $S_0[j].x$ from $\mathbb{X}$;  $S_0[j].y \leftarrow f(S_0[j].x)$;
        if $S_0[j].y > y_B$ then $x_B \leftarrow S_0[j].x$;  $y_B \leftarrow S_0[j].y$;
    for $i \in 0 \dots \infty$ do                                               ▷ iterate "generations"
        for $j \in 1 \dots ps$ do              ▷ new pop. via mutation and crossover
            if $\mathfrak{R}_0^1 < cr$ then $N_i[j].x \leftarrow$ binary($S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x$, $S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x$);
            else $N_i[j].x \leftarrow$ move($S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x$);
            $N_i[j].y \leftarrow f(N_i[j].x)$;
            if $N_i[j].y > y_B$ then $x_B \leftarrow N_i[j].x$;  $y_B \leftarrow N_i[j].y$;
        $S_{i+1} \leftarrow$ Roulette Wheel: select $ps$ records from $P_i = S_i \cup N_i$
                such that, for each of the $ps$ slots, the probability
                of $P_i[j]$ to be chosen is proportional to $P_i[j].y$.
    return $x_B$, $y_B$
```

Metaheuristic Optimization

- Different metaheuristics realize the trial-and-error scheme differently
- They all prefer better solutions over worse ones.
- If they would always and only accept the better solutions, they could get trapped in local optima.
- So they *sometimes* accept worse solutions, but the probability to choose a better solution is always higher in average.

This is the most fundamental concept of metaheuristic optimization:

If you keep good solutions and modify them, you are likely to get better solutions.

If you keep accepting better and better solutions, you will get really good solutions eventually.

3. Invariance Properties

Slides

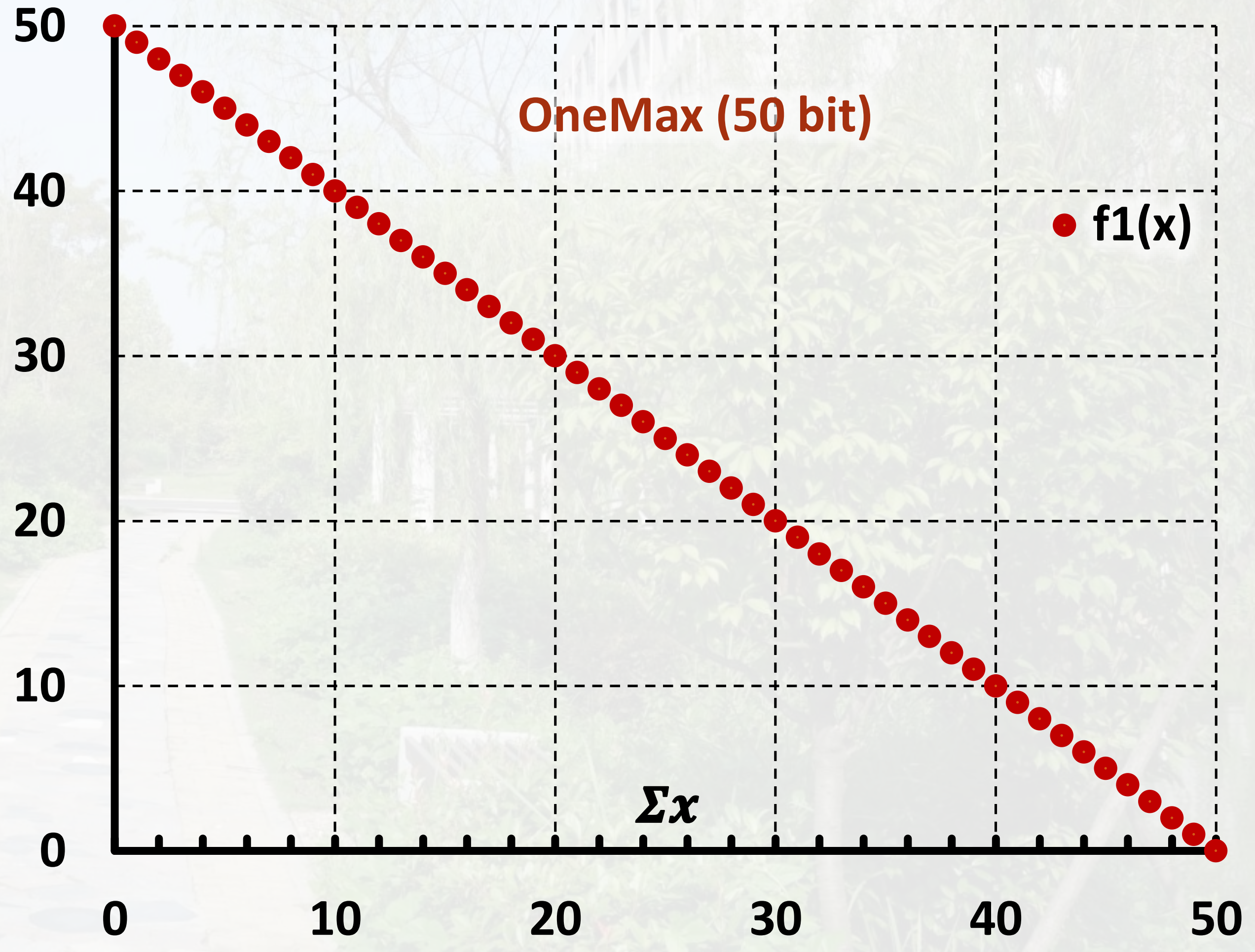
Video

Invariance Properties

- Research in optimization, Machine Learning, and Artificial Intelligence often use simple problems to try out and benchmark algorithms.
- These allow for many experiments in a short time.
- We often know the optimal solutions or bounds for their quality, we can understand the results well.
- What we want is that algorithms perform similar to our benchmarking results also on actual, real-world problems.
- We want **invariance properties**.

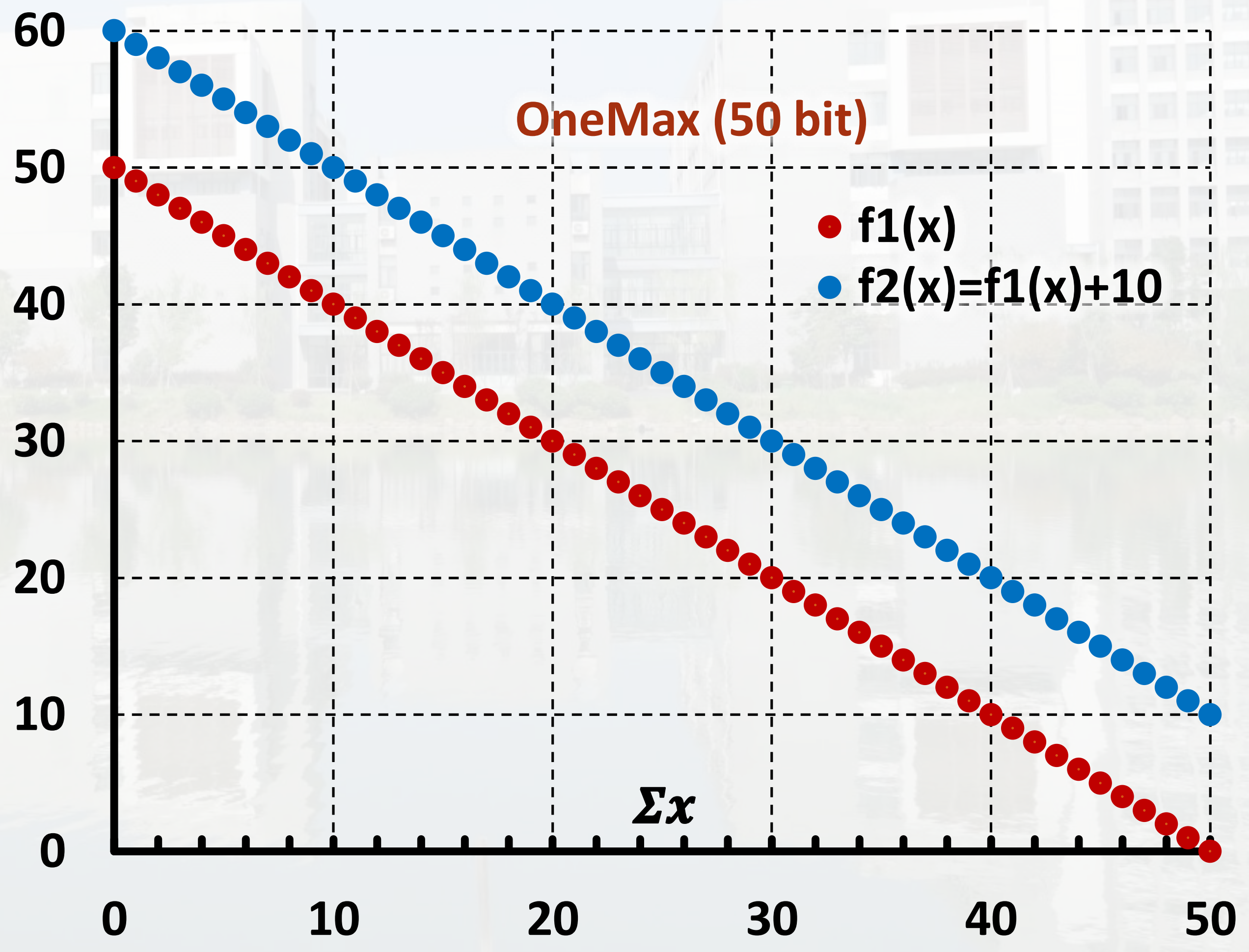
Invariance Properties: Example OneMax

- OneMax is simplest benchmark problem in discrete optimization.
- It is defined over $X = \{0,1\}^n$, i.e., bit strings of length n.
- "Find the bit string with all ones."
- "Maximize the number of ones."
- $f_1(x) = n - \sum x$



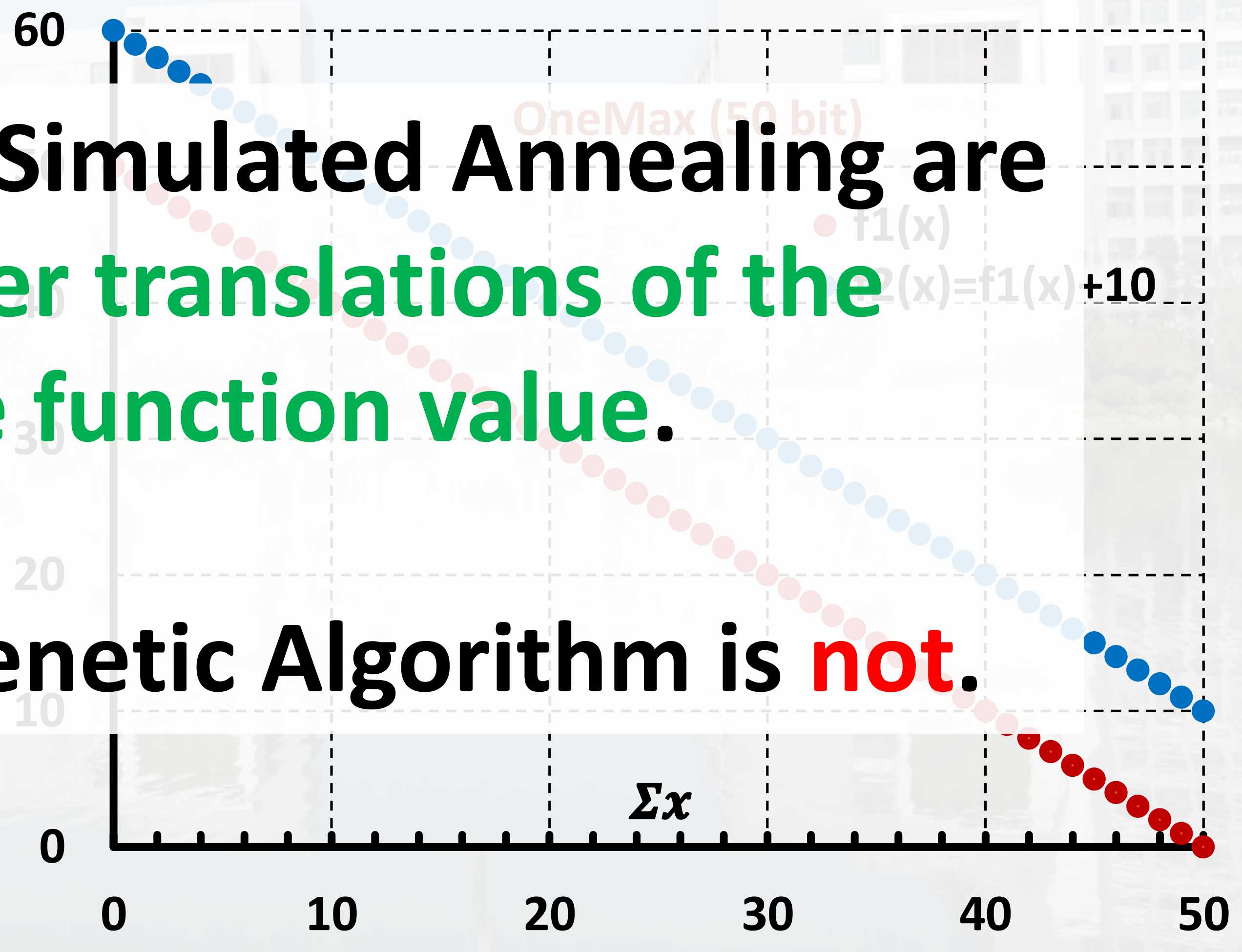
Invariance Properties: Example OneMax

- Now I create a modified version of this problem.
- $f_2(x) = f_1(x) + 10$
- **Expectation:** Any reasonable algorithm should perform exactly the same on f_1 and f_2.
- (1+1)-EA: acceptance decision based on $f(x_1) \leq f(x_2)$ ✓
- SA: acceptance decision based on $f(x_1) - f(x_2)$ ✓
- SGA: acceptance decision based on ratio of $f(x_1)$ to $f(x_2)$ ✘



Invariance Properties: Example OneMax

- Now I create a modified version of this problem.
- $f_2(x) = f_1(x) + 10$
- **Expectation:** Any reasonable algorithm should perform exactly the same on f_1 and f_2
- (1+1)-EA: acceptance decision based on $f(x_1) \leq f(x_2)$ ✓
- SA: acceptance decision based on $f(x_1) - f(x_2)$ ✓
- SGA: acceptance decision based on ratio of $f(x_1)$ to $f(x_2)$ ✗

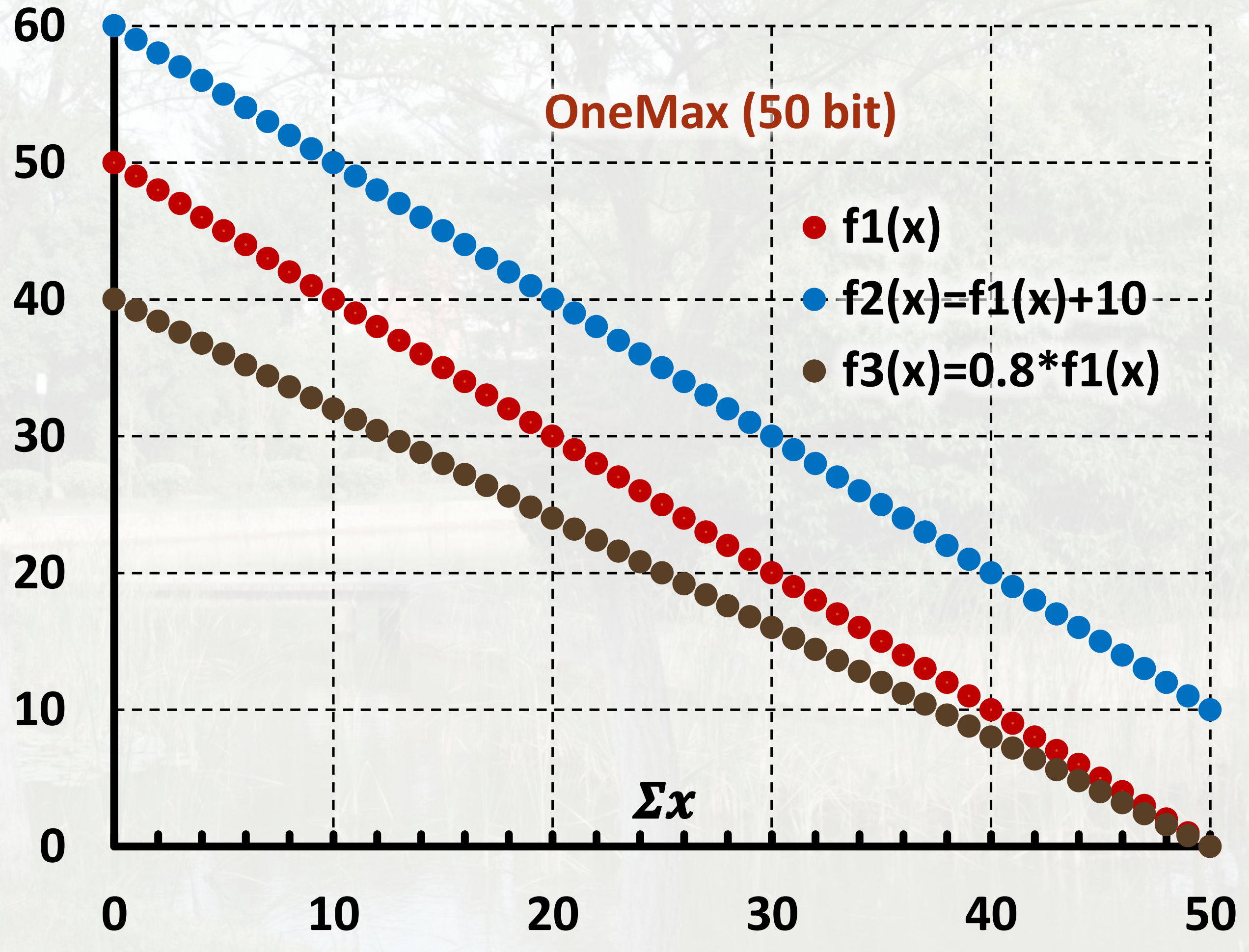


The (1+1) EA and Simulated Annealing are invariant under translations of the objective function value.

The Standard Genetic Algorithm is not.

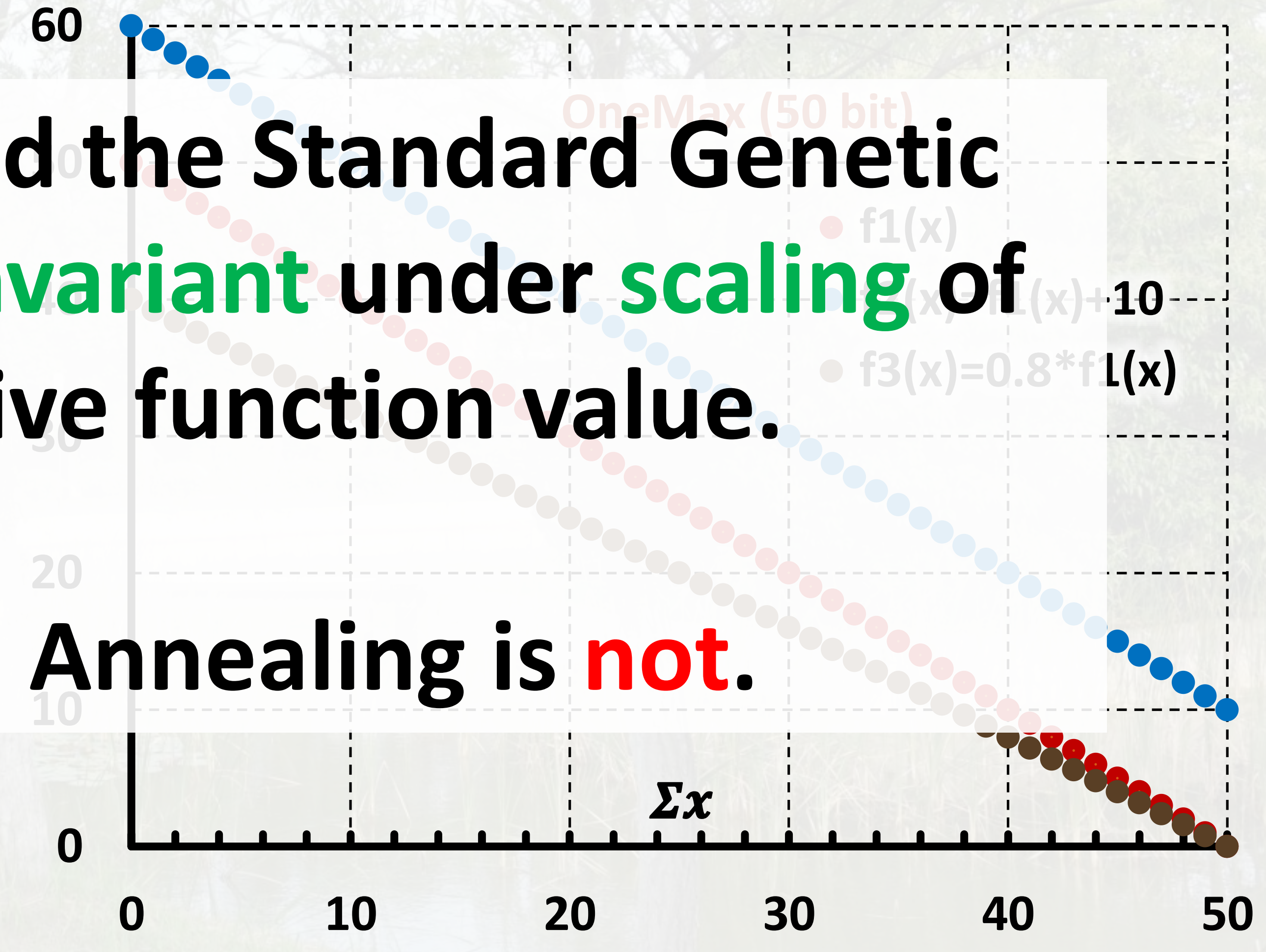
Invariance Properties: Example OneMax

- Now I create another modified version of this problem.
- $f_3(x) = 0.8 * f_1(x)$
- **Expectation:** A reasonable algorithm should perform exactly the same on f_1 and f_3.
- (1+1)-EA: acceptance decision based on $f(x_1) \leq f(x_2)$ ✓
- SA: acceptance decision based on $f(x_1) - f(x_2)$ ✘
- SGA: acceptance decision based on ratio of $f(x_1)$ to $f(x_2)$ ✓



Invariance Properties: Example OneMax

- Now I create another modified version of this problem.
- $f_3(x) = 0.8 * f_1(x)$
- **Expectation:** A reasonable algorithm should perform exactly the same on f_1 and f_3.
- (1+1)-EA: acceptance decision based on $f(x_1) \leq f(x_2)$ ✓
- SA: acceptance decision based on $f(x_1) - f(x_2)$ ✗
- SGA: acceptance decision based on ratio of $f(x_1)$ to $f(x_2)$ ✓

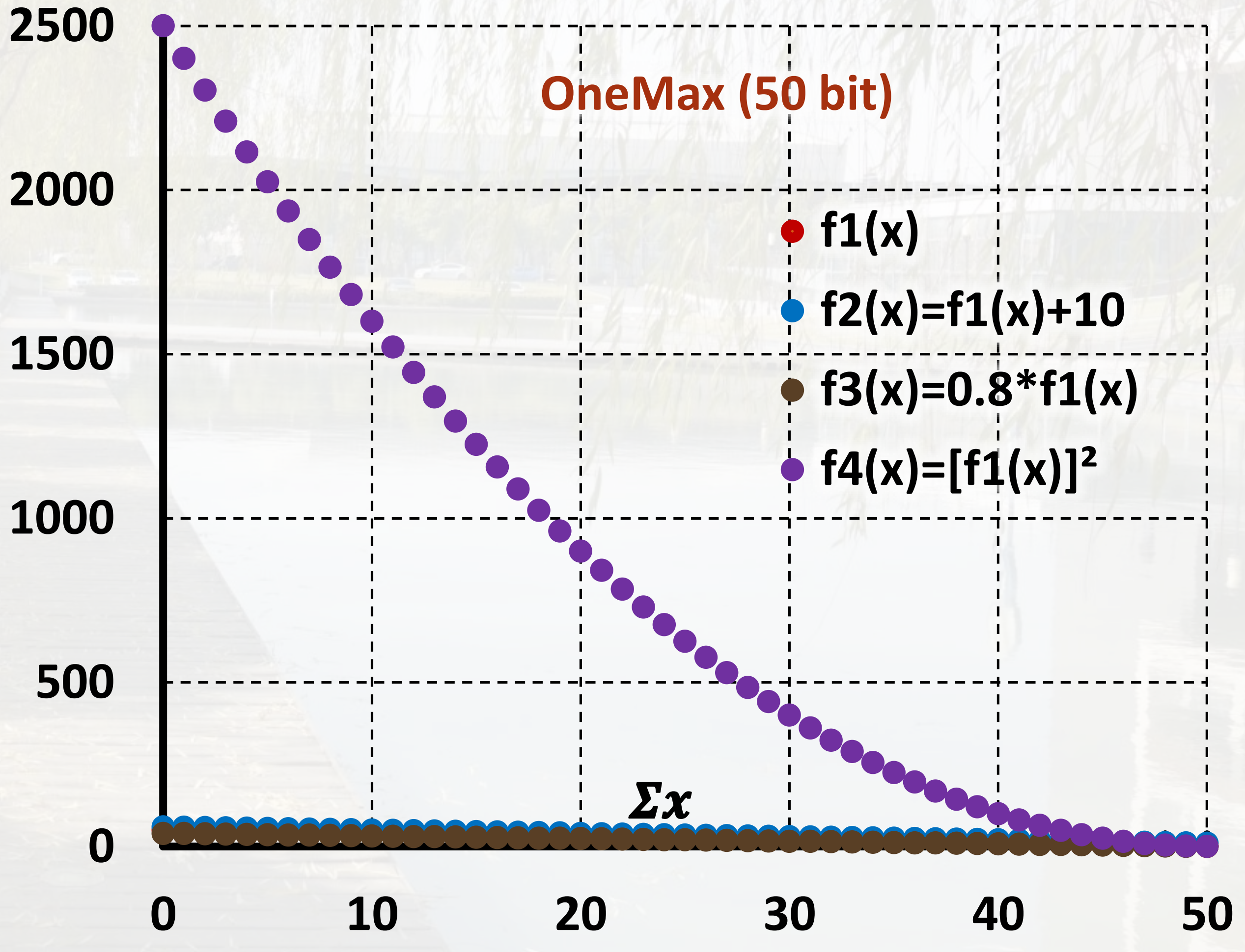


The (1+1) EA and the Standard Genetic Algorithm are invariant under scaling of the objective function value.

Simulated Annealing is not.

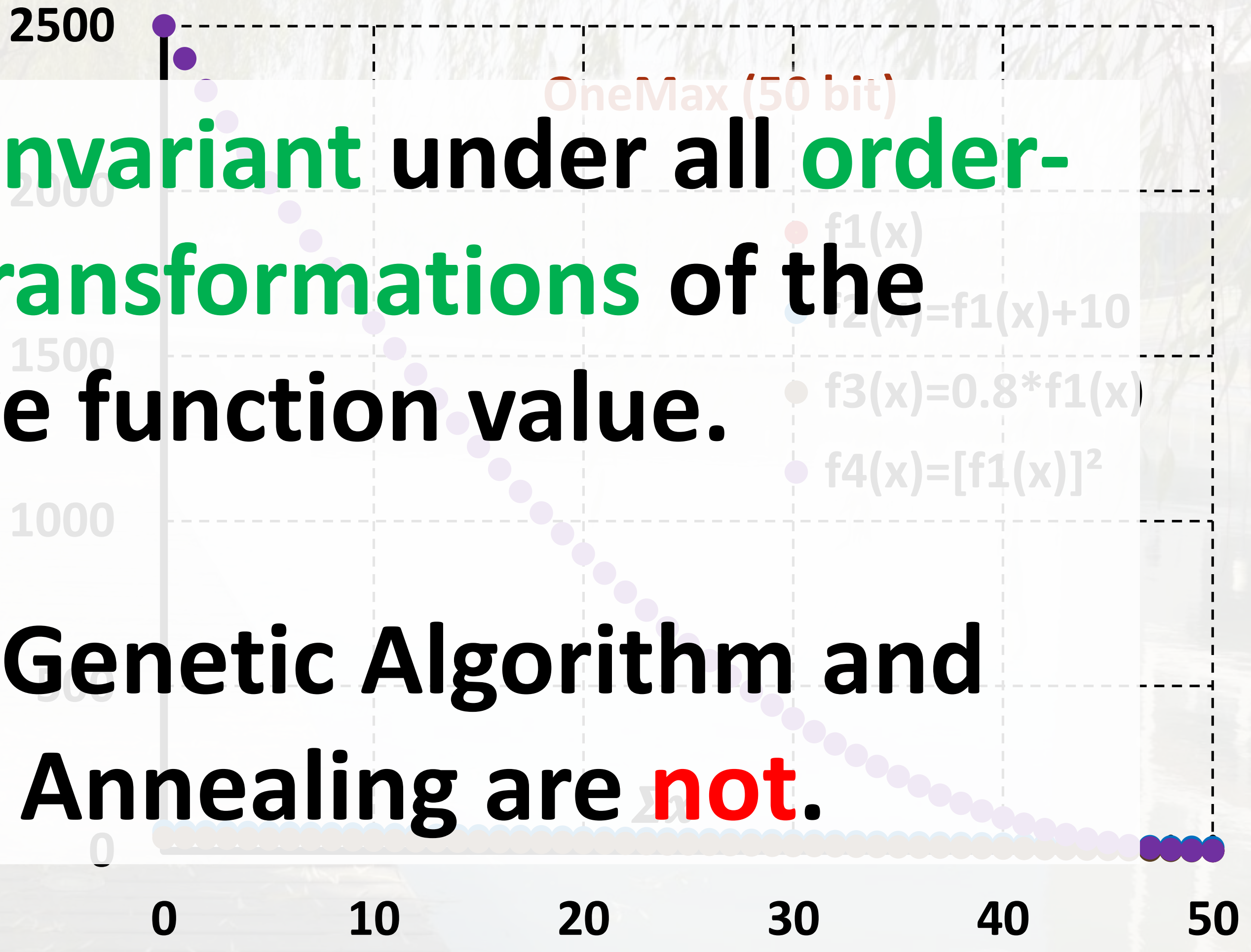
Invariance Properties: Example OneMax

- Now I create another modified version of this problem.
- $f_4(x) = [f_1(x)]^2$
- **Expectation:** A nice algorithm should perform exactly the same on f_1 and f_4.
- (1+1)-EA: acceptance decision based on $f(x_1) \leq f(x_2)$ ✓
- SA: acceptance decision based on $f(x_1) - f(x_2)$ ✗
- SGA: acceptance decision based on ratio of $f(x_1)$ to $f(x_2)$ ✗



Invariance Properties: Example OneMax

- Now I create another modified version of this problem
- $f_4(x) = [f_1(x)]^2$
- **Expectation: A** [illegible] should perform exactly the same on f_1 and f_4.
- (1+1)-EA: acceptance decision based on $f(x_1) \leq f(x_2)$ ✓
- SA: acceptance decision based on $f(x_1) - f(x_2)$ ✗
- SGA: acceptance decision based on ratio of $f(x_1)$ to $f(x_2)$ ✗



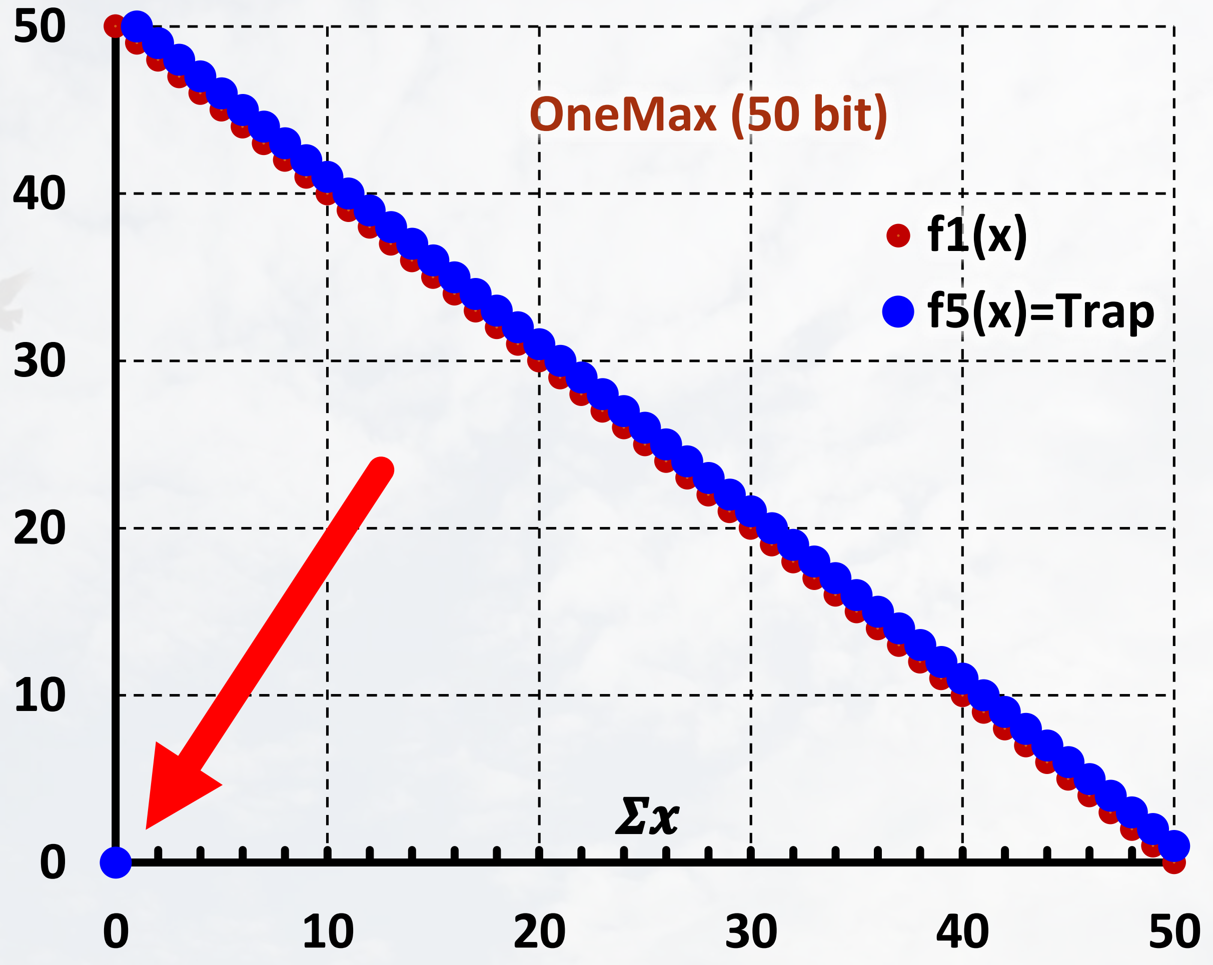
The (1+1) EA is invariant under all order-preserving transformations of the objective function value.

The Standard Genetic Algorithm and Simulated Annealing are not.

Now let’s enter eerie territory.

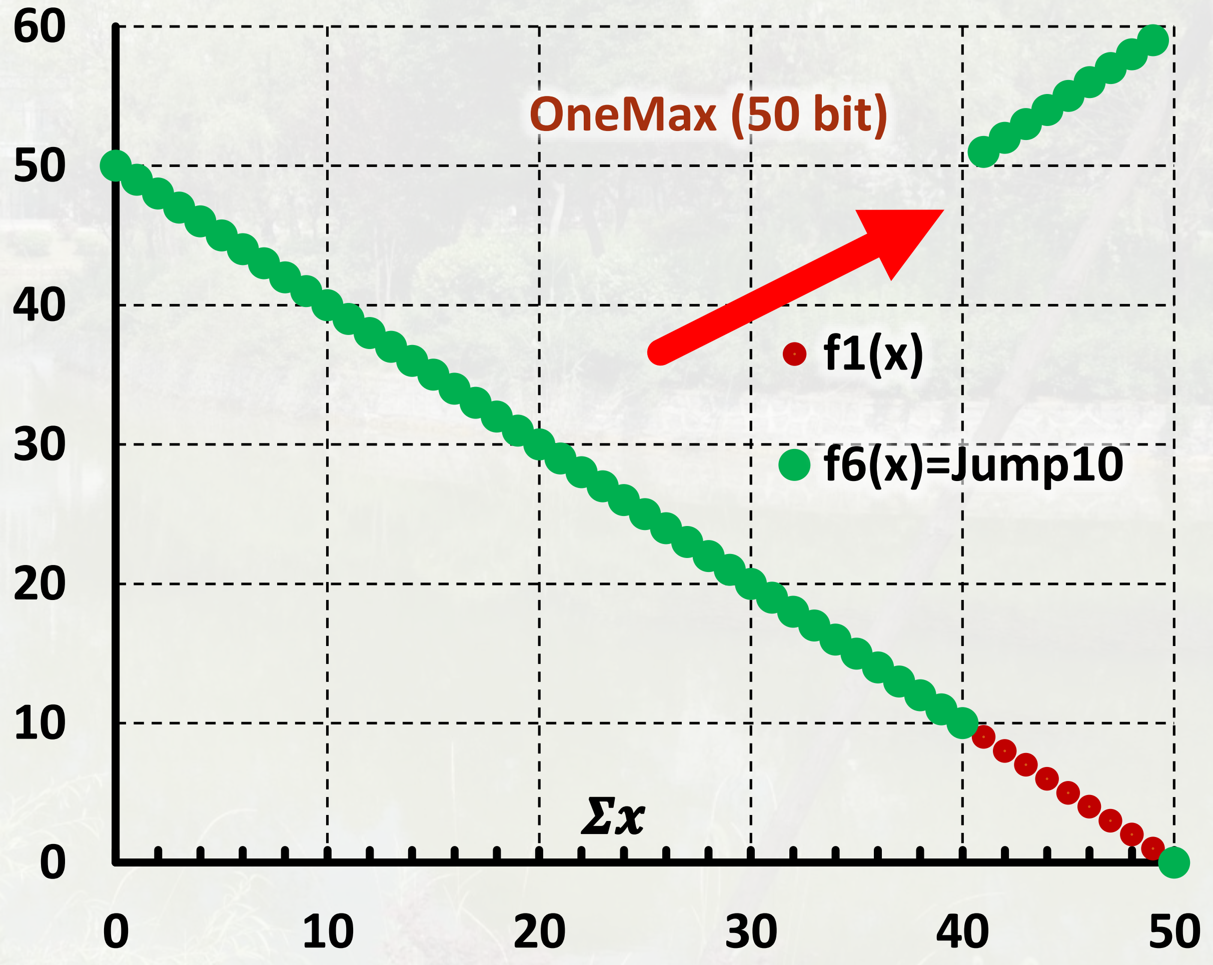
Invariance Properties: Example OneMax

- Now I create another modified version of this problem: a **trap**.
- $f_5(x) = \begin{cases} 0 & \text{if } f_1(x) = 50 \\ 1 + f_1(x) & \text{else} \end{cases}$
- **Expectation:** Algorithm performance on $f_1(x)$ probably does not carry over to $f_5(x)$.
- Neither the (1+1) EA, SA, nor SGA can deal with this.
- The (1+1) EA has exponential runtime on traps.



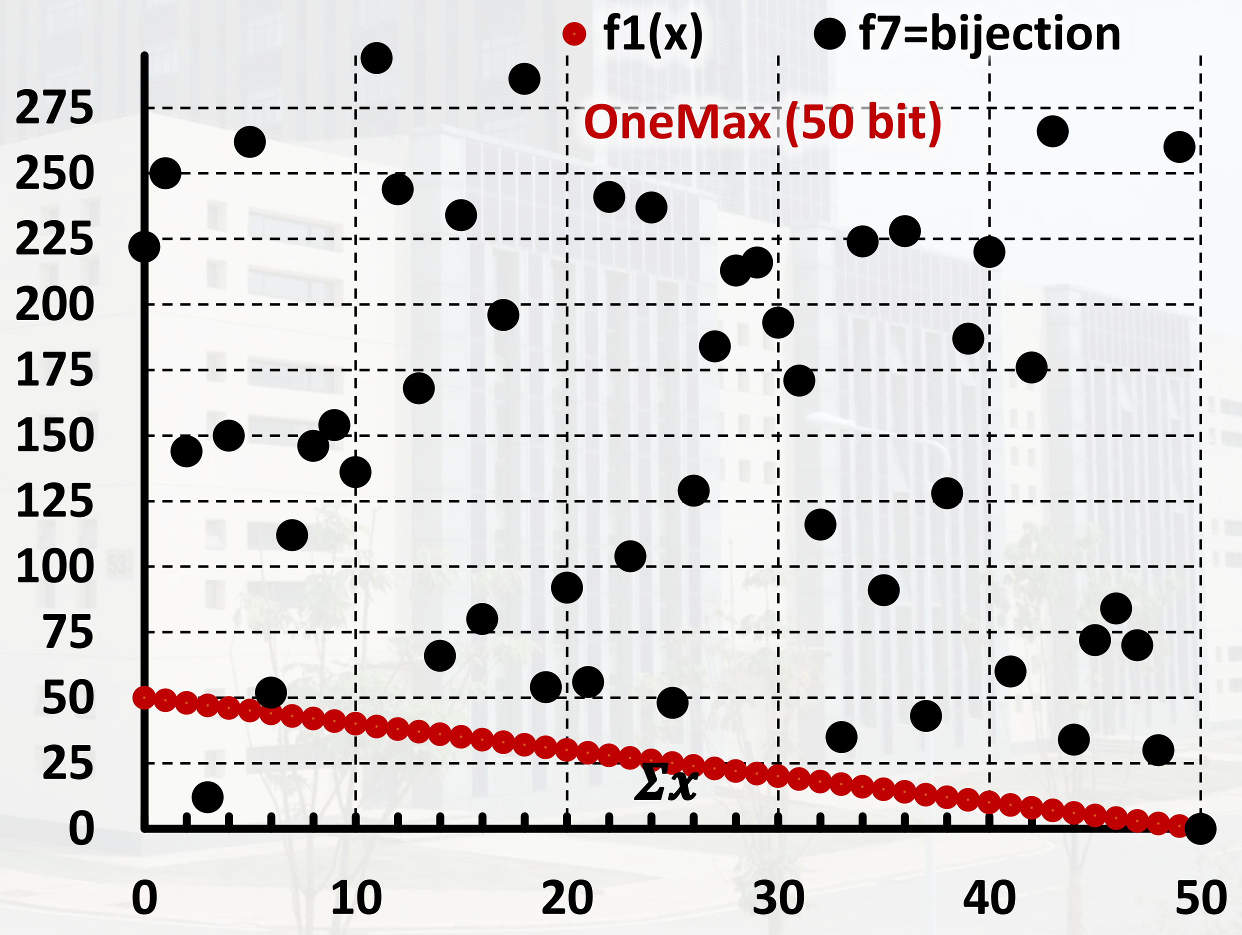
Invariance Properties: Example OneMax

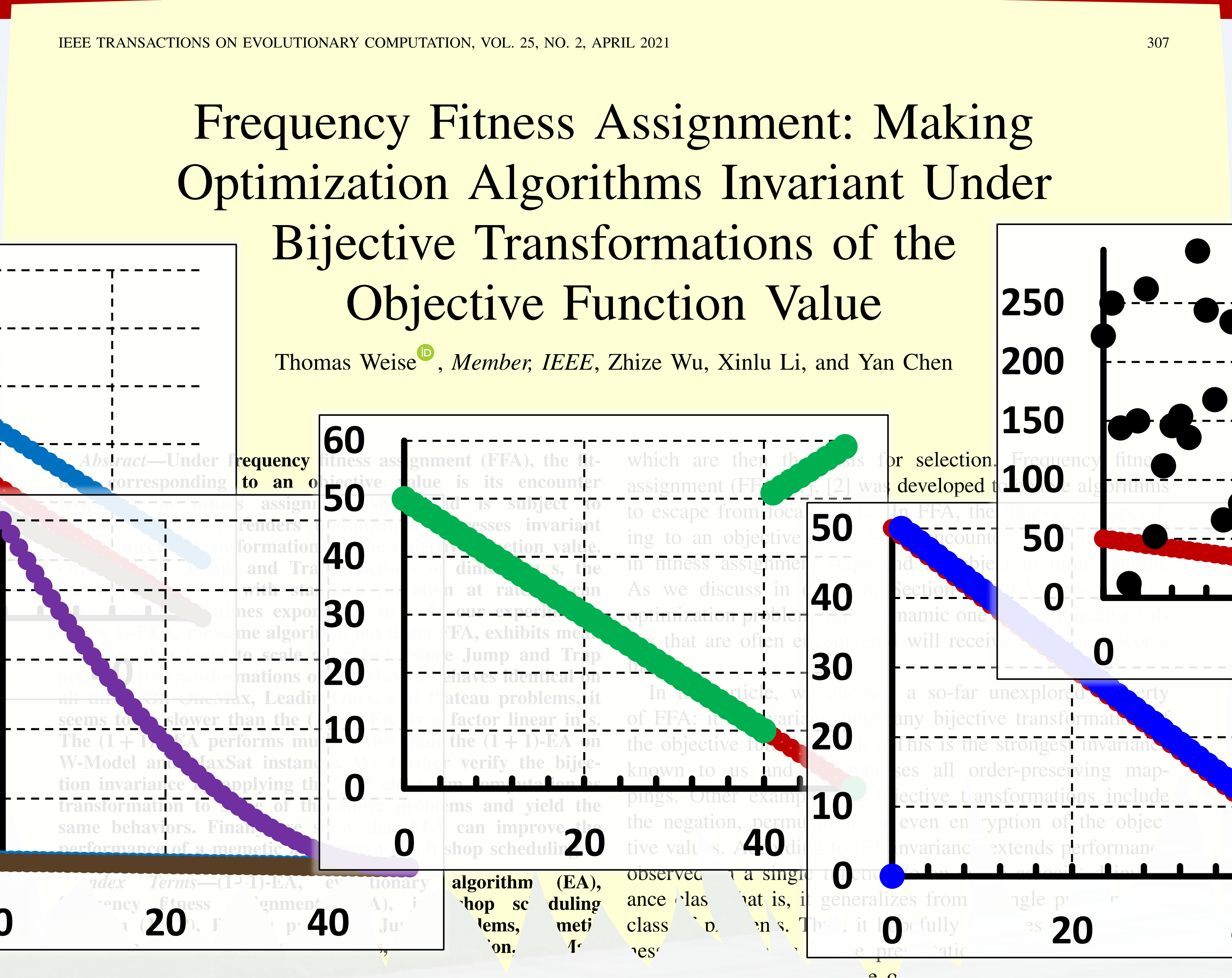
- Now I create another modified version: a $w = \mathbf{10}$ **jump**.
- Insert a deceptive area of length $w - 1 = \mathbf{10}$ before optimum
- **Expectation:** Algorithm performance on $f_1(x)$ probably does not carry over to $f_6(x)$.
- Neither the (1+1) EA, SA, nor SGA can deal with this well.
- The (1+1) EA a runtime exponential in w on jumps.



Invariance Properties: Example OneMax

- How about I apply an arbitrary bijection g that preserves the optimum to $f_1(x)$ and get $f_7(x) = g(f_1(x))$?
- **Expectation:** Algorithm performance on $f_1(x)$ probably does not carry over to $f_7(x)$.
- Neither the (1+1) EA, SA, nor SGA can deal with this well.
- Indeed, there is no method that can deal with this well.

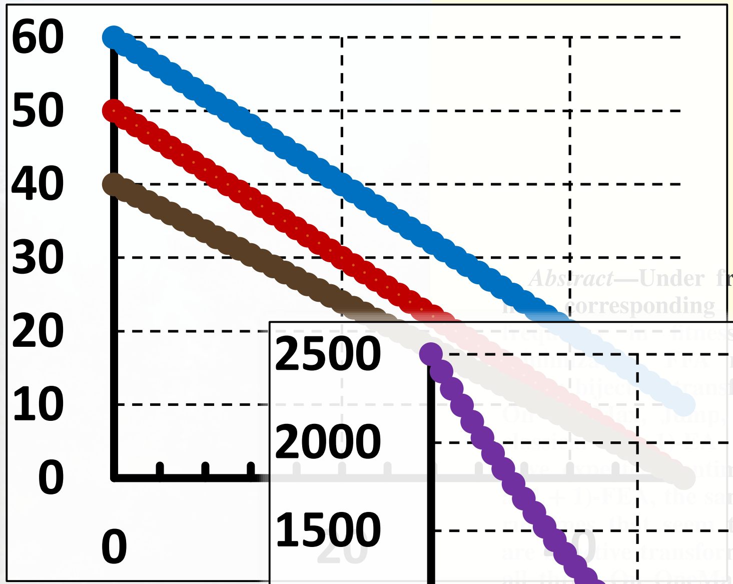


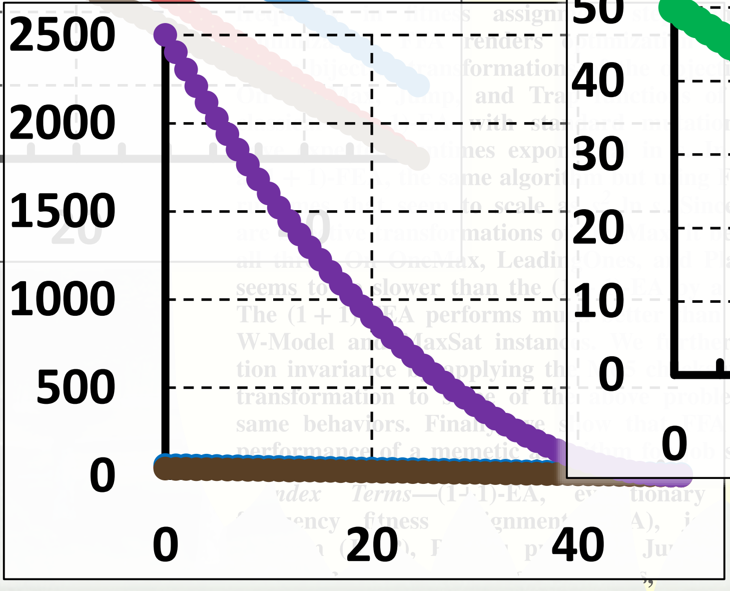


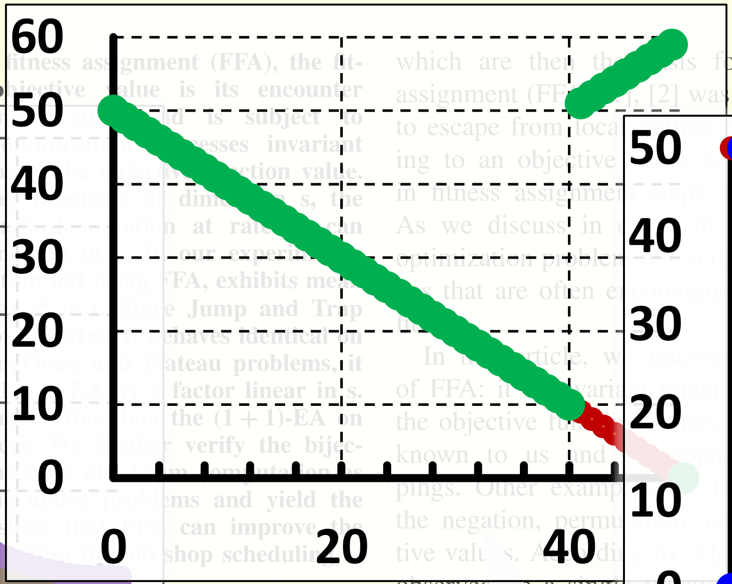
IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 25, NO. 2, APRIL 2021 307

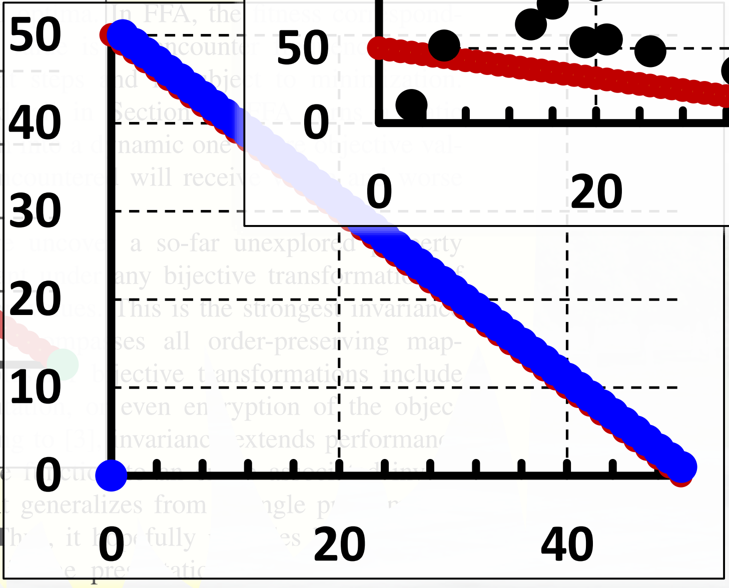
Frequency Fitness Assignment: Making Optimization Algorithms Invariant Under Bijective Transformations of the Objective Function Value

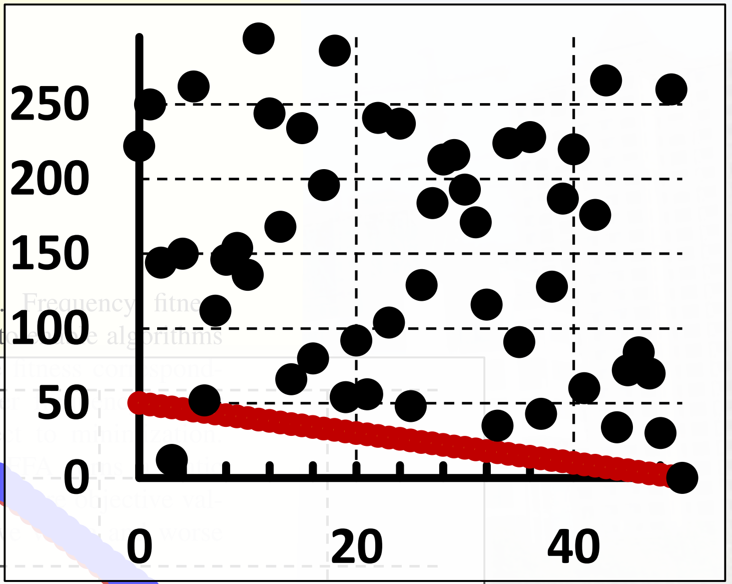
Thomas Weise, *Member, IEEE*, Zhize Wu, Xinlu Li, and Yan Chen











T Weise, Z Wu, X Li, and Y Chen. Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value. *IEEE Transactions on Evolutionary Computation* 25(2):307–319. 2021.

4. Frequency Fitness Assignment

Slides

Video

FFA: Idea

- Frequency Fitness Assignment (FFA) is a module that can be plugged into different *existing* algorithms.
- It changes the way the algorithm selects the interesting solutions S_{i+1} from the set $P_i = S_i \cup N_i$.
- It therefore maintains a table H with the encounter frequency of each objective value in the selection decisions.
- The table H is initially filled with zeros.
- *Before* the selection step of the algorithm, $H[f(P_i[j])]\ \forall j \in 1..|P_i|$ is incremented by 1.
- Then, $\mathrm{H}[f(P_i[j])]$ replaces $f(P_i[j])$ in the actual selection decisions.

FFA: (1+1) EA and (1+1) FEA

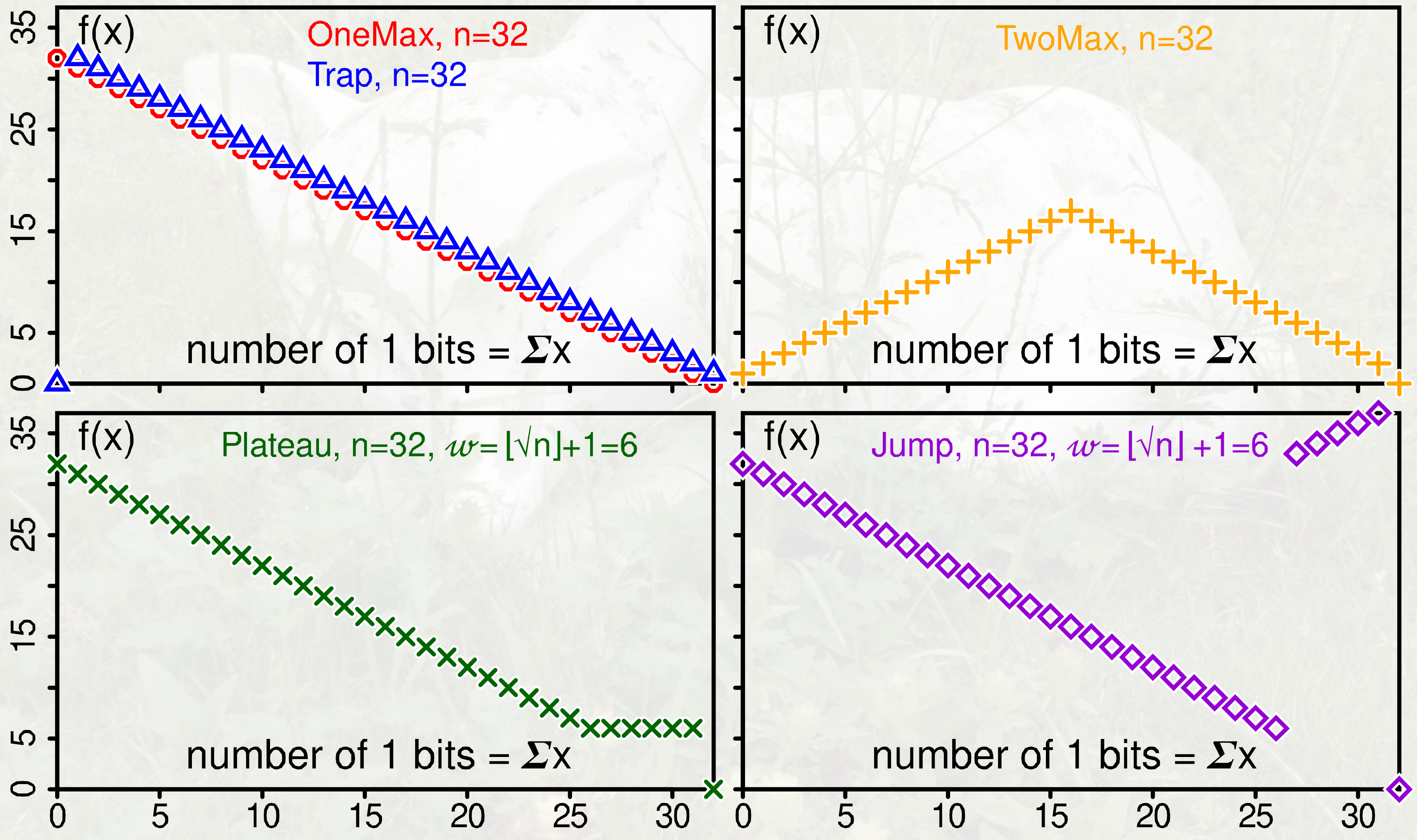
```
procedure (1 + 1) EA($f : \mathbb{X} \mapsto \mathbb{R}$)
    randomly sample $x_c$ from $\mathbb{X}$;  $y_c \leftarrow f(x_c)$;
    while $\neg$ terminate do
        $x_n \leftarrow \texttt{move}(x_c)$;  $y_n \leftarrow f(x_n)$;
        if $y_n \leq y_c$ then  $x_c \leftarrow x_n$;  $y_c \leftarrow y_n$;
    return $x_c$, $y_c$
```

```
procedure (1 + 1) FEA($f : \mathbb{X} \mapsto \mathbb{N}$)
    $H \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathbb{X}$;  $y_c \leftarrow f(x_c)$;
    $x_B \leftarrow x_c$;  $y_B \leftarrow y_c$;
    while $\neg$ terminate do
        $x_n \leftarrow \texttt{move}(x_c)$;  $y_n \leftarrow f(x_n)$;
        $H[y_c] \leftarrow H[y_c] + 1$;  $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then
            $x_c \leftarrow x_n$;  $y_c \leftarrow y_n$;
            if $y_c < y_B$ then $x_B \leftarrow x_c$;  $y_B \leftarrow y_c$;
    return $x_B$, $y_B$
```

FFA: What does this do?

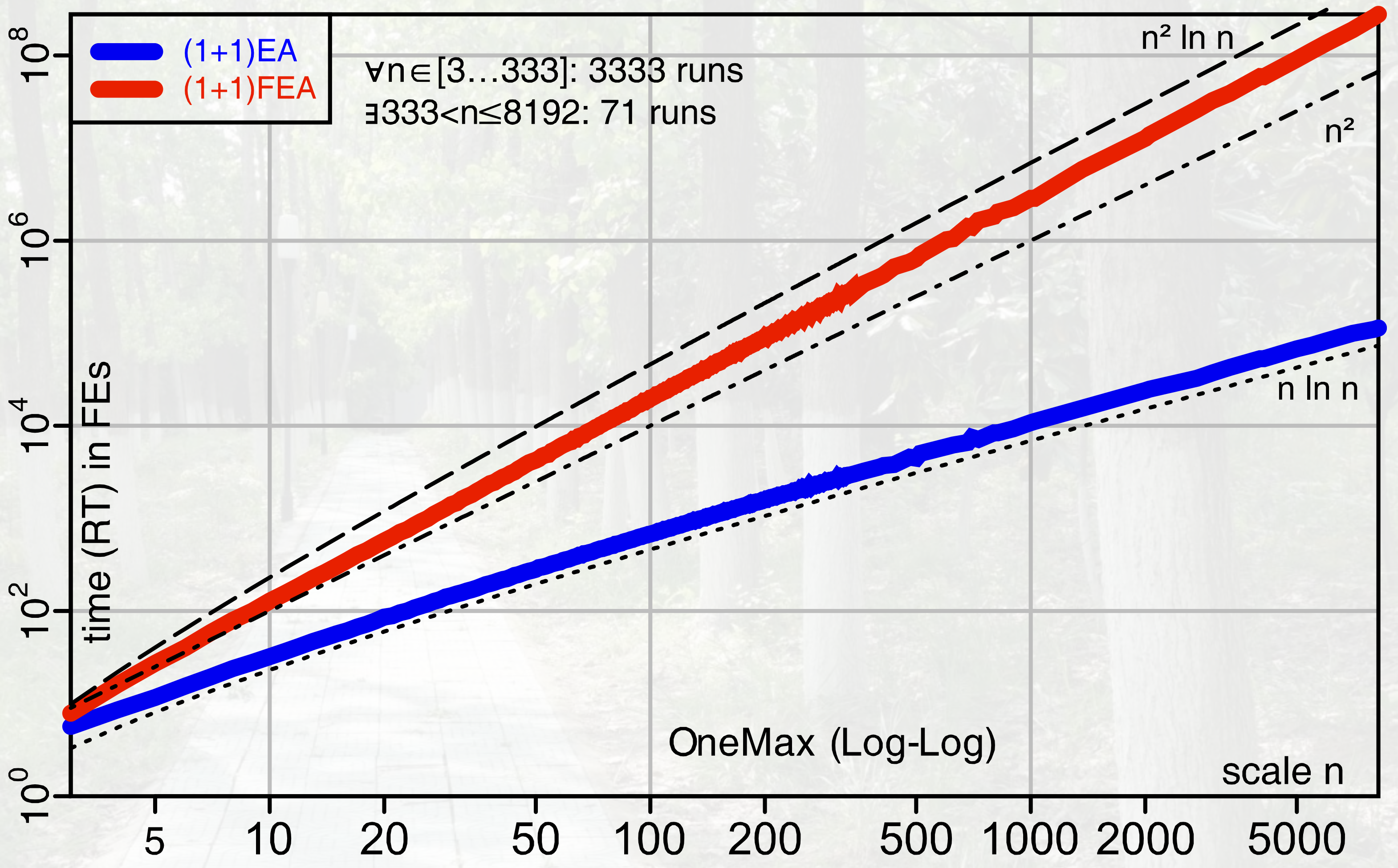
- Static optimization problems become dynamic, because frequency fitness changes over time.
- Solutions get less attractive the more often their corresponding objective values have been seen. This also holds for local optima…
- **Solutions with better objective values are no longer preferred over such with worse objective value.**
- Instead, solutions with less-frequent objective values are preferred.
- An algorithm using FFA is **invariant under all injective transformations** of the objective function value.
- They will perform identical on ALL of the OneMax-based functions from before!

FFA: Discrete Optimization Theory Benchmarks



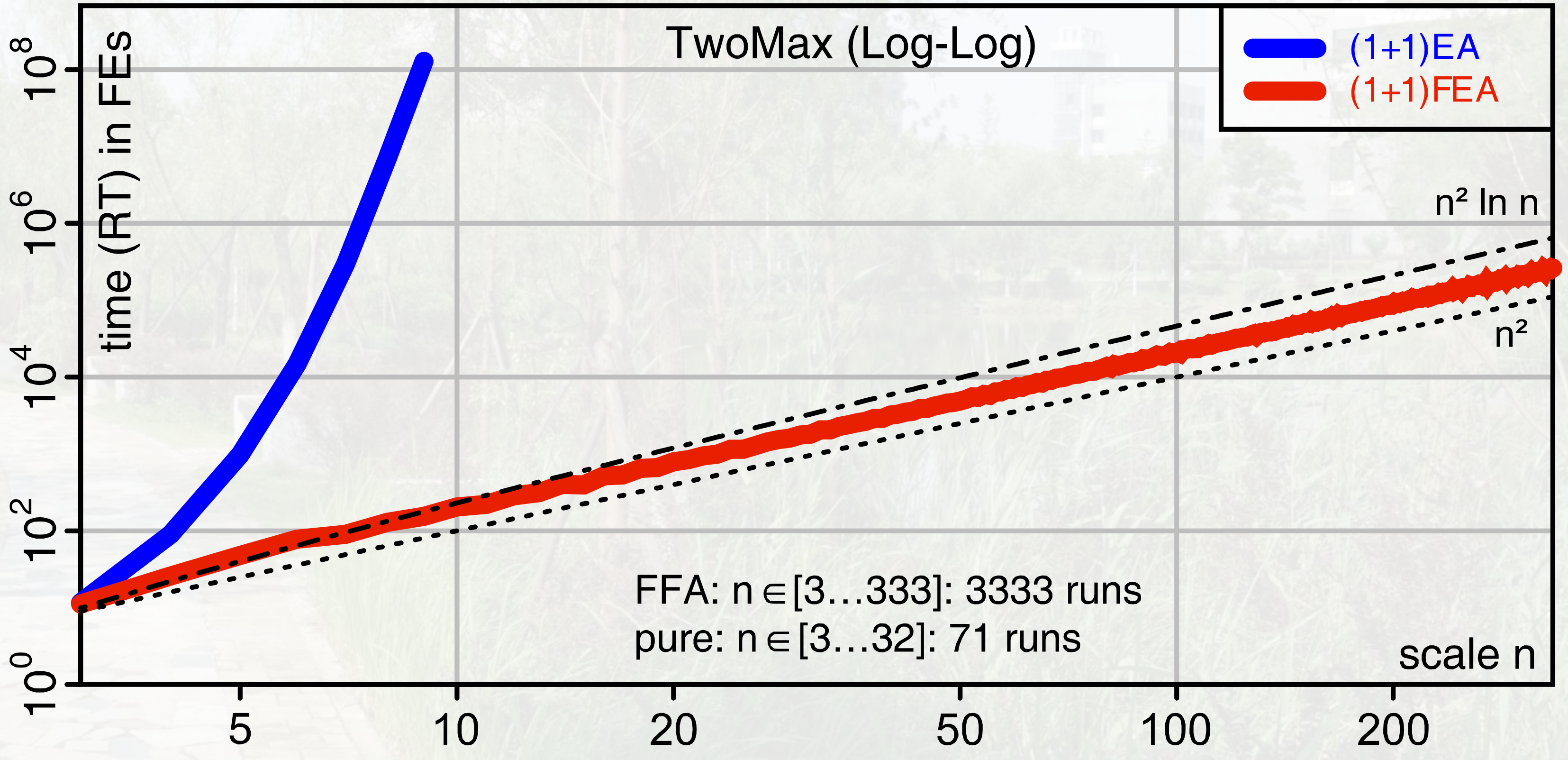
FFA: (1+1) FEA on OneMax

- Average runtime of (1+1) EA on OneMax is in $\mathcal{O}(n \ln n)$
- Average runtime of (1+1) FEA on OneMax seems to be slower by factor proportional in n, i.e., seems to be in $\mathcal{O}(n^2 \ln n)$.



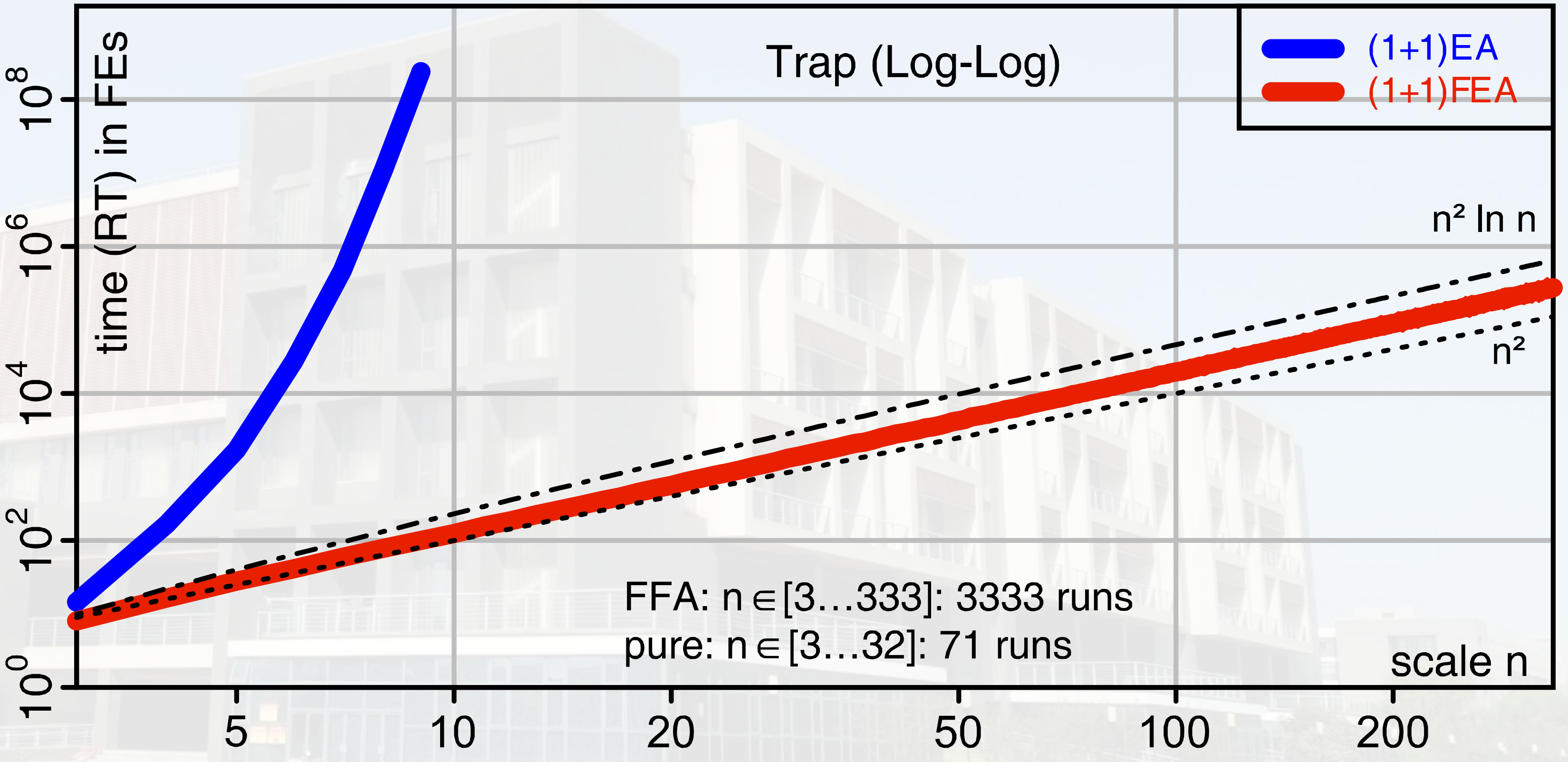
FFA: (1+1) FEA on TwoMax

- One local and opposite global optimum of almost same size
- Average runtime of (1+1) EA on TwoMax is exponential
- (1+1) FEA has a mean runtime that seems to be in $\mathcal{O}(n^2 \ln n)$.



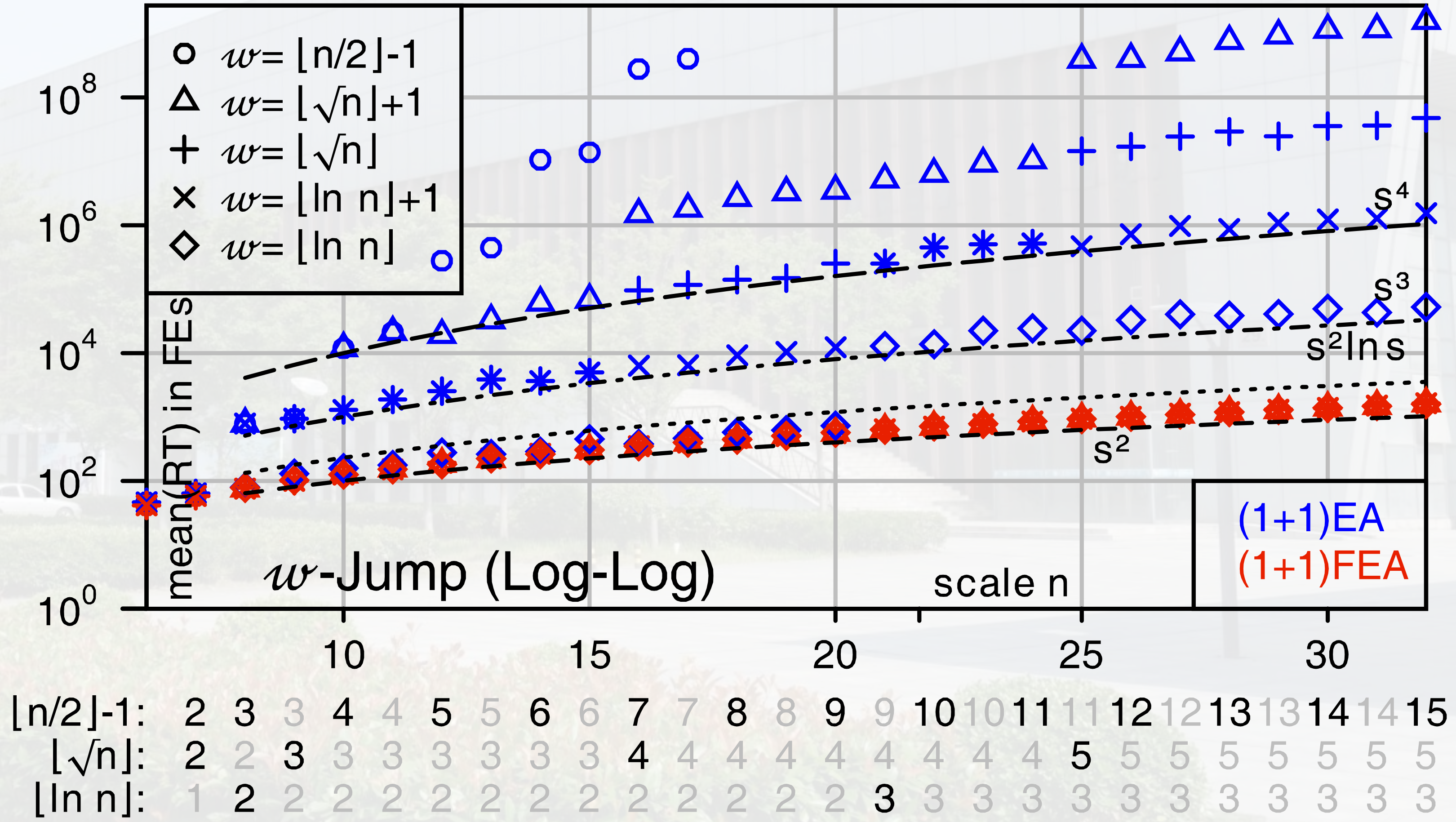
FFA: (1+1) FEA on Trap

- Average runtime of (1+1) EA on Trap is exponential
- (1+1) FEA behaves the same as on OneMax, i.e., has polynomial mean runtime



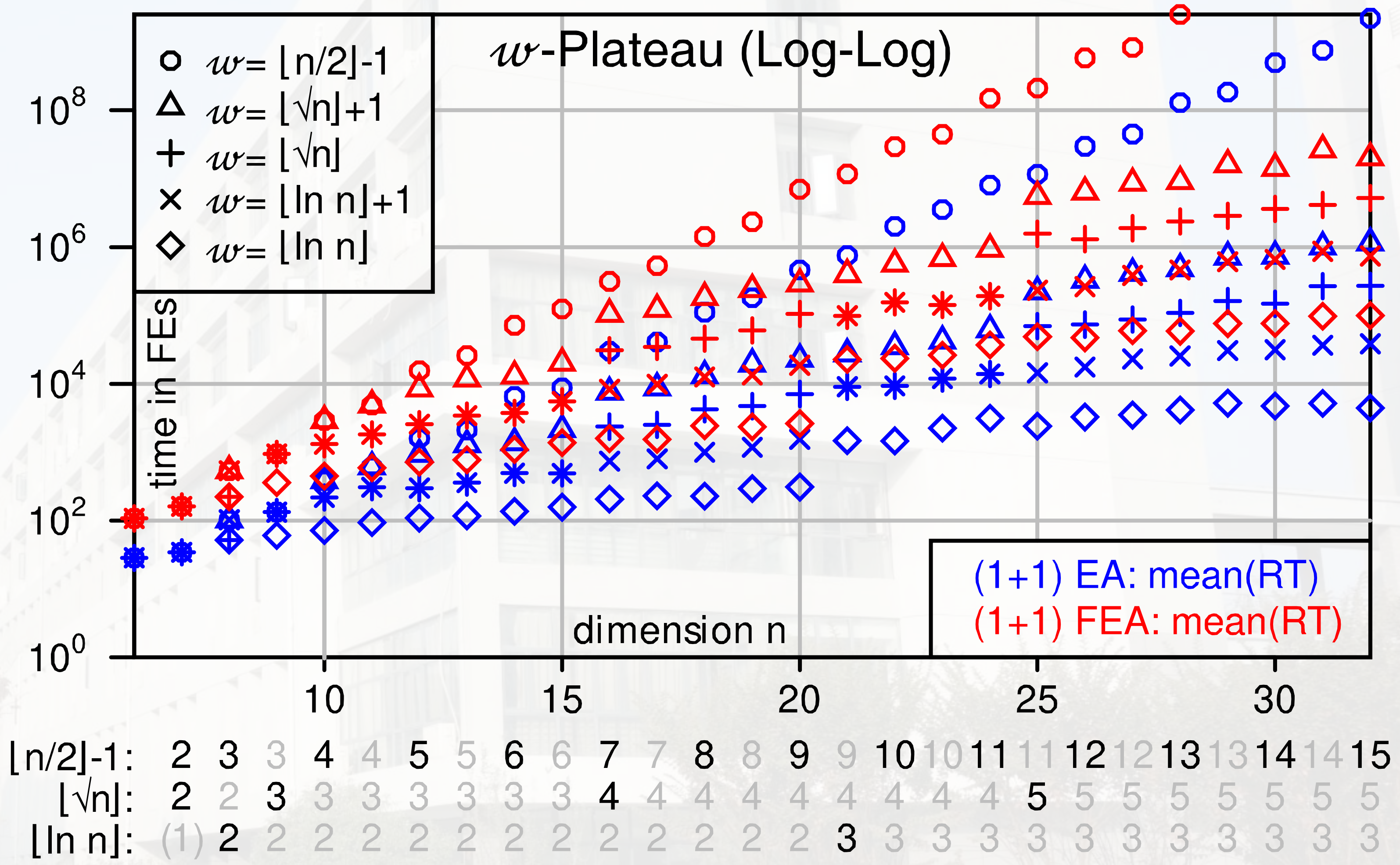
FFA: (1+1) FEA on Jump

- Average runtime of (1+1) EA on Jump is exponential in jump width w
- (1+1) FEA behaves the same as on OneMax for all jump widths w, i.e., has polynomial mean runtime



FFA: (1+1) FEA on Plateau

- Average runtime of (1+1) EA on Plateaus is exponential in plateau width w
- (1+1) FEA is a bit slower, probably proportional to a factor linear in n
- Plateaus remain plateaus under FFA



FFA: (1+1) FEA on MaxSat

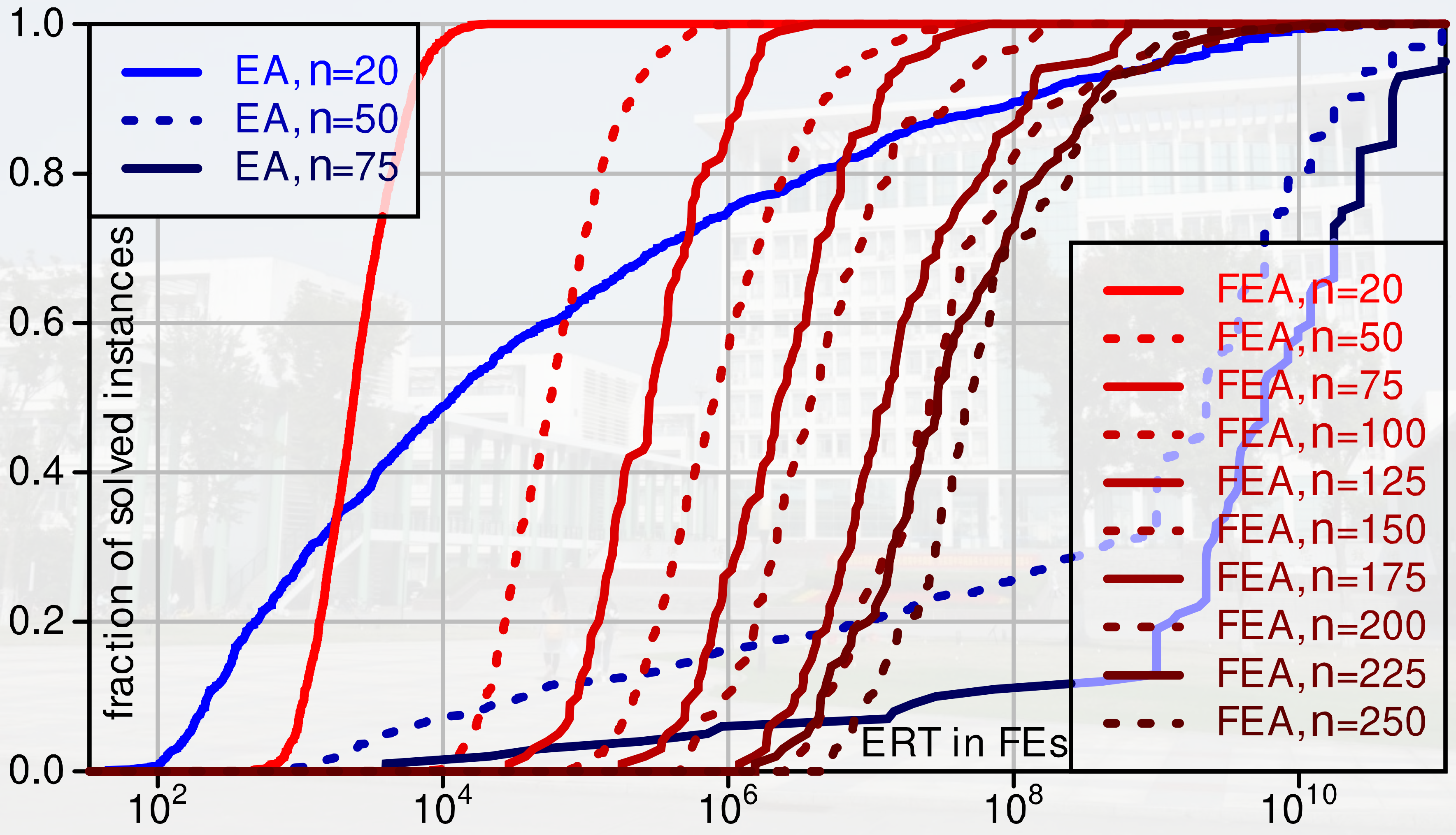
- The Maximum Satisfiability Problem (MaxSat) is $\mathcal{NP}$-hard
- SatLib provides satisfiable benchmark instances from the phase transition region (i.e., the hardest type of instances) for different scales $n \in \{20\} \cup \{25i \; \forall i \in 2..10\}$.
- We conduct 11’000 runs with the (1+1) FEA on each instance scale.
- The (1+1) EA is very much slower than the (1+1) FEA, so we can use it only on smaller scales.
- Our computational budget is always 10^{10} FEs.

FFA: (1+1) FEA on MaxSat

- The FEA is better on problems with 250 variables than the EA on problems with 50.

instance type	(1+1) EA			(1+1) FEA		
	success rate	ERT	mean y_c	success rate	ERT	mean y_B
uf20_*	0.985	$1.91 * 10^8$	0.015	1	3′091	0
uf50_*	0.748	$3.56 * 10^9$	0.299	1	93′459	0
uf75_*	0.583	$7.41 * 10^9$	0.528	1	490′166	0
uf100_*				1	$2.14 * 10^6$	0
uf125_*				1	$5.27 * 10^6$	0
uf150_*				1	$1.40 * 10^7$	0
uf175_*				1	$5.78 * 10^7$	0
uf200_*				0.991	$2.44 * 10^8$	0.00945
uf225_*				0.994	$2.43 * 10^8$	0.00555
uf250_*				0.992	$2.43 * 10^8$	0.00782

FFA: (1+1) FEA on MaxSat – ERT-ECDF



FFA: What does this do?

- FFA makes the simple (1+1) EA slower on problems that it can easily solve.
- The slowdown is roughly proportional to the number of possible objective values.
- On some non-$\mathcal{NP}$-hard problems for which the (1+1) EA needs exponential runtime, the (1+1) FEA needs polynomial mean runtime
- Plateaus of the objective are still plateaus under FFA
- FFA very significantly speeds up the (1+1) EA on the $\mathcal{NP}$-hard MaxSat problem

FFA: Now something weird…

- Let’s say you have an optimization problem with objective function $f(x)$
- You encrypt the objective values and do not tell them to the algorithm
- Let’s say you apply AES, RSA, or the Cesar cypher as a function $g: \mathbb{N} \mapsto \mathbb{N}$, i.e., do $g\big(f(x)\big)$
- Encryption removes any order, correlation or causality information, i.e., $g\big(f(x)\big)$ does not correlate with $f(x)$ in any way
- If the (1+1) FEA can find the optimum of $f(x)$…
- …then it will find exactly the same solution in exactly the same runtime even if you apply it to the encrypted problem $g\big(f(x)\big)$
- …because encryption is a bijective transformation.

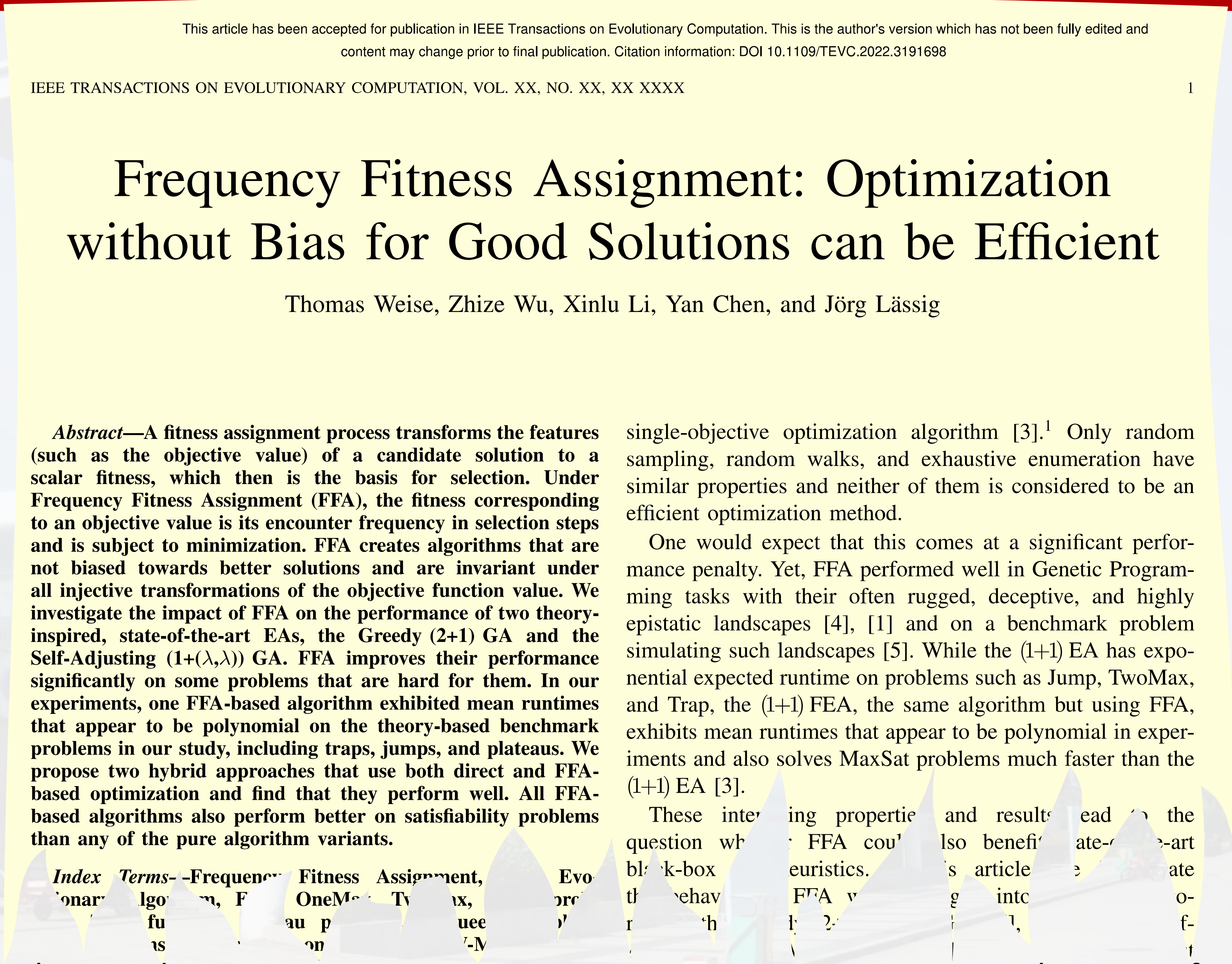
5. Summary

Slides

Video

Summary

- Frequency Fitness Assignment (FFA) is an algorithm module that can be plugged into existing algorithms.
- It renders algorithms invariant under all injective transformations of the objective function value.
- It makes them optimize without bias for good solutions.
- It slows them down on easy problems.
- It can speed them up on hard problems.
- It is limited to objective functions that cannot take on too many different objective values.



Frequency Fitness Assignment: Optimization without Bias for Good Solutions can be Efficient

Thomas Weise, Zhize Wu, Xinlu Li, Yan Chen, and Jörg Lässig

***Abstract*—A fitness assignment process transforms the features (such as the objective value) of a candidate solution to a scalar fitness, which then is the basis for selection. Under Frequency Fitness Assignment (FFA), the fitness corresponding to an objective value is its encounter frequency in selection steps and is subject to minimization. FFA creates algorithms that are not biased towards better solutions and are invariant under all injective transformations of the objective function value. We investigate the impact of FFA on the performance of two theory-inspired, state-of-the-art EAs, the Greedy (2+1) GA and the Self-Adjusting (1+(λ,λ)) GA. FFA improves their performance significantly on some problems that are hard for them. In our experiments, one FFA-based algorithm exhibited mean runtimes that appear to be polynomial on the theory-based benchmark problems in our study, including traps, jumps, and plateaus. We propose two hybrid approaches that use both direct and FFA-based optimization and find that they perform well. All FFA-based algorithms also perform better on satisfiability problems than any of the pure algorithm variants.**

Index Terms—Frequenc Fitness Assignment, Evo- ionar lgo m, F OneM Tw ax, pro fu au p uee

single-objective optimization algorithm [3].[1] Only random sampling, random walks, and exhaustive enumeration have similar properties and neither of them is considered to be an efficient optimization method.

One would expect that this comes at a significant performance penalty. Yet, FFA performed well in Genetic Programming tasks with their often rugged, deceptive, and highly epistatic landscapes [4], [1] and on a benchmark problem simulating such landscapes [5]. While the (1+1) EA has exponential expected runtime on problems such as Jump, TwoMax, and Trap, the (1+1) FEA, the same algorithm but using FFA, exhibits mean runtimes that appear to be polynomial in experiments and also solves MaxSat problems much faster than the (1+1) EA [3].

These inter ing propertie and results ead o the question wh FFA cou lso benefi ate-o e-art bl k-box euristics. s article ate th ehav FFA v g into o-

T Weise, Z Wu, X Li, Y Chen, and J Lässig. Frequency Fitness Assignment: Optimization without Bias for Good Solutions can be Efficient. *IEEE Transactions on Evolutionary Computation*. Early Access 2022.

Applied Soft Computing Journal 92 (2020) 106269

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms

Thomas Weise [*], Yan Chen, Xinlu Li, Zhize Wu

Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University, Jinxiu Dadao 99, Hefei, Anhui, 230601, China

ARTICLE INFO

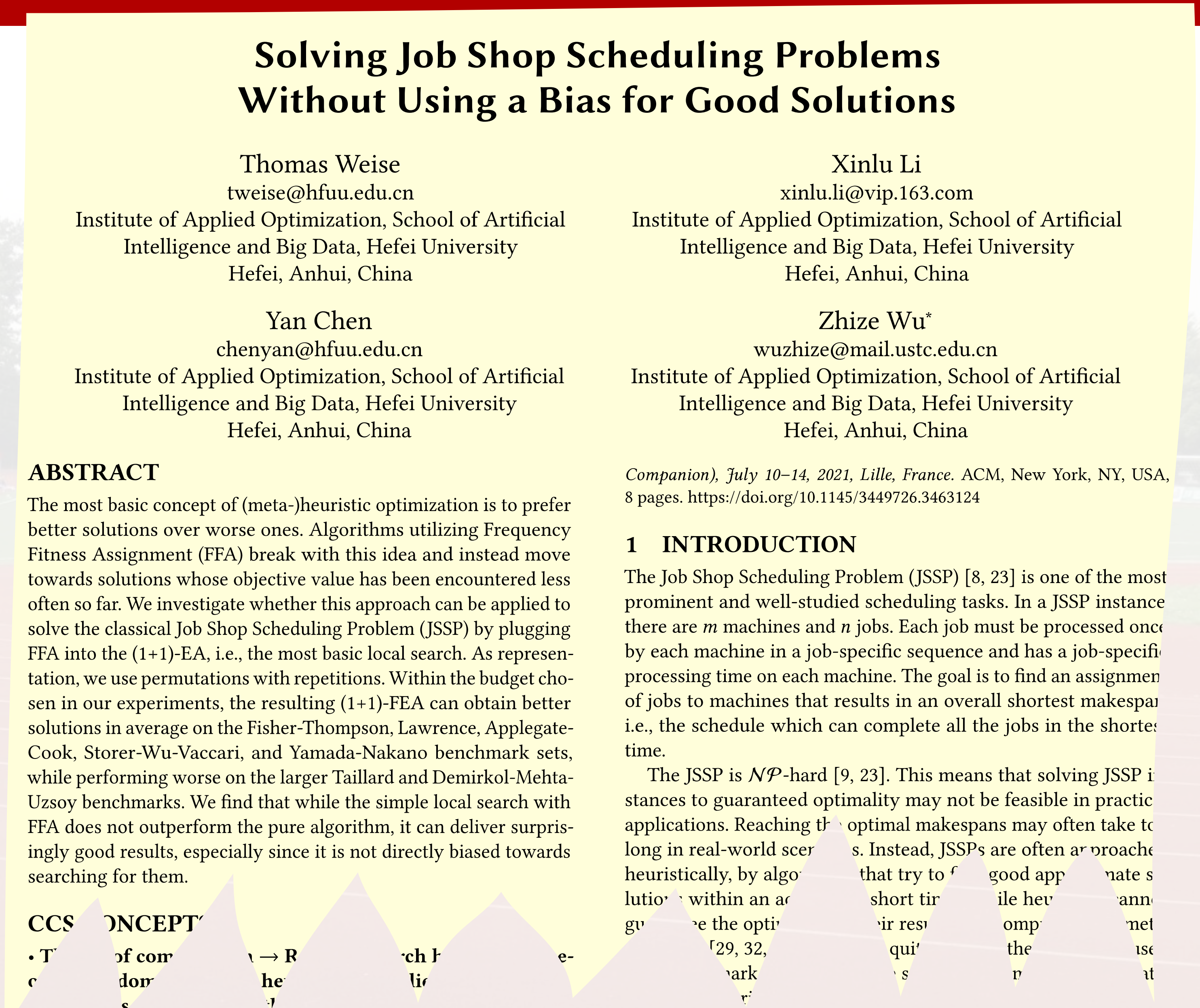
Article history:
Received 15 August 2019
Received in revised form 6 March 2020
Accepted 28 March 2020
Available online 6 April 2020

Keywords:
Experimentation
Benchmarking
Optimization
Runtime behavior
Black-box optimization
Discrete optimization

ABSTRACT

As the number of practical applications of discrete black-box metaheuristics is growing faster and faster, the benchmarking of these algorithms is rapidly gaining importance. While new algorithms are often introduced for specific problem domains, researchers are also interested in which general problem characteristics are hard for which type of algorithm. The *W-Model* is a benchmark function for discrete black-box optimization, which allows for the easy, fast, and reproducible generation of problem instances exhibiting characteristics such as ruggedness, deceptiveness, epistasis, and neutrality in a tunable way. We conduct the first large-scale study with the *W-Model* in its fixed-length single-objective form, investigating 17 algorithm configurations (including Evolutionary Algorithms and local searches) and 8372 problem instances. We develop and apply a machine learning methodology to automatically discover several clusters of optimization process runtime behaviors as well as their reasons grounded in the algorithm and model parameters. Both a detailed statistical evaluation and our methodology confirm that the different model parameters allow us to generate problem instances of different hardness, but also find that the investigated algorithms struggle with different problem characteristics. With our methodology, we select a set of 19 diverse problem instances with which researchers can conduct a fast but still in-depth analysis of algorithm performance. The best-performing algorithms in our experiment were Evolutionary Algorithms applying Frequency Fitness Assignment, which turned out to be robust over a wide range of problem settings and solved more instances than the other tested algorithms.

T Weise, Y Chen, X Li, and Z Wu. Selecting a diverse set of benchmark instances from a tunable model problem for black-box discrete optimization algorithms. *Applied Soft Computing Journal* 92:106269, June 2020.



Solving Job Shop Scheduling Problems Without Using a Bias for Good Solutions

Thomas Weise
tweise@hfuu.edu.cn
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

Xinlu Li
xinlu.li@vip.163.com
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

Yan Chen
chenyan@hfuu.edu.cn
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

Zhize Wu*
wuzhize@mail.ustc.edu.cn
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

ABSTRACT

The most basic concept of (meta-)heuristic optimization is to prefer better solutions over worse ones. Algorithms utilizing Frequency Fitness Assignment (FFA) break with this idea and instead move towards solutions whose objective value has been encountered less often so far. We investigate whether this approach can be applied to solve the classical Job Shop Scheduling Problem (JSSP) by plugging FFA into the (1+1)-EA, i.e., the most basic local search. As representation, we use permutations with repetitions. Within the budget chosen in our experiments, the resulting (1+1)-FEA can obtain better solutions in average on the Fisher-Thompson, Lawrence, Applegate-Cook, Storer-Wu-Vaccari, and Yamada-Nakano benchmark sets, while performing worse on the larger Taillard and Demirkol-Mehta-Uzsoy benchmarks. We find that while the simple local search with FFA does not outperform the pure algorithm, it can deliver surprisingly good results, especially since it is not directly biased towards searching for them.

CCS ONCEPT

• T of com → R ch h e-
c dom he li
s

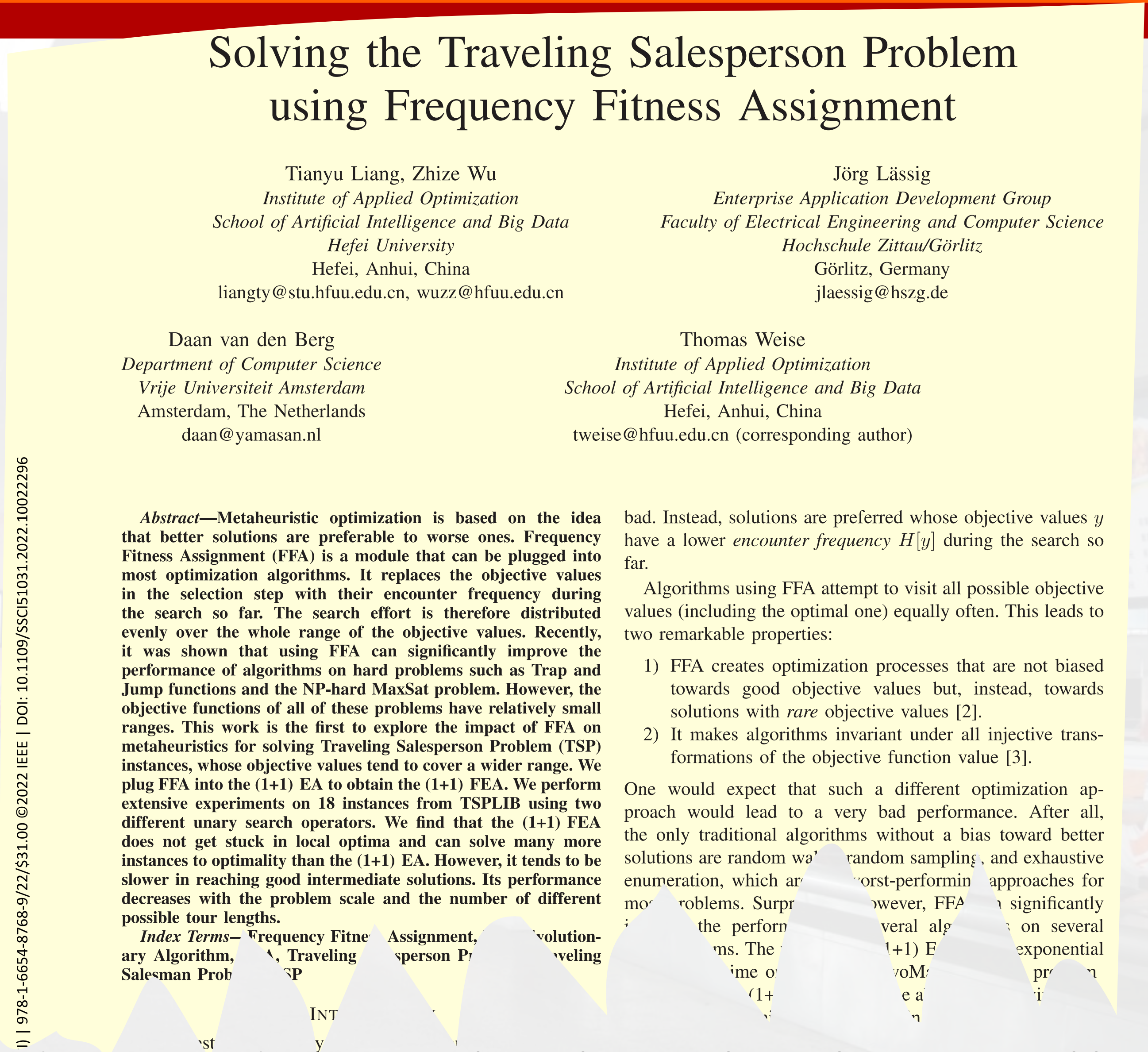
Companion), July 10–14, 2021, Lille, France. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3449726.3463124

1 INTRODUCTION

The Job Shop Scheduling Problem (JSSP) [8, 23] is one of the most prominent and well-studied scheduling tasks. In a JSSP instance there are m machines and n jobs. Each job must be processed once by each machine in a job-specific sequence and has a job-specifi processing time on each machine. The goal is to find an assignmen of jobs to machines that results in an overall shortest makespar i.e., the schedule which can complete all the jobs in the shortes time.

The JSSP is $\mathcal{NP}$-hard [9, 23]. This means that solving JSSP i stances to guaranteed optimality may not be feasible in practic applications. Reaching th optimal makespans may often take to long in real-world scer s. Instead, JSSPs are often ar proache heuristically, by algo that try to good app nate s lutio within an a short tir ile heu ann gu e the opti ir res omp met [29, 32, qui the use ark s n at

T Weise, X Li, Y Chen, and Z Wu. Solving Job Shop Scheduling Problems Without Using a Bias for Good Solutions. In *Genetic and Evolutionary Computation Conference Companion (GECCO'21)*, July 10-14, 2021, Lille, France. ACM.



Solving the Traveling Salesperson Problem using Frequency Fitness Assignment

Tianyu Liang, Zhize Wu
Institute of Applied Optimization
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China
liangty@stu.hfuu.edu.cn, wuzz@hfuu.edu.cn

Jörg Lässig
Enterprise Application Development Group
Faculty of Electrical Engineering and Computer Science
Hochschule Zittau/Görlitz
Görlitz, Germany
jlaessig@hszg.de

Daan van den Berg
Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
daan@yamasan.nl

Thomas Weise
Institute of Applied Optimization
School of Artificial Intelligence and Big Data
Hefei, Anhui, China
tweise@hfuu.edu.cn (corresponding author)

l) | 978-1-6654-8768-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SSCI51031.2022.10022296

***Abstract*—Metaheuristic optimization is based on the idea that better solutions are preferable to worse ones. Frequency Fitness Assignment (FFA) is a module that can be plugged into most optimization algorithms. It replaces the objective values in the selection step with their encounter frequency during the search so far. The search effort is therefore distributed evenly over the whole range of the objective values. Recently, it was shown that using FFA can significantly improve the performance of algorithms on hard problems such as Trap and Jump functions and the NP-hard MaxSat problem. However, the objective functions of all of these problems have relatively small ranges. This work is the first to explore the impact of FFA on metaheuristics for solving Traveling Salesperson Problem (TSP) instances, whose objective values tend to cover a wider range. We plug FFA into the (1+1) EA to obtain the (1+1) FEA. We perform extensive experiments on 18 instances from TSPLIB using two different unary search operators. We find that the (1+1) FEA does not get stuck in local optima and can solve many more instances to optimality than the (1+1) EA. However, it tends to be slower in reaching good intermediate solutions. Its performance decreases with the problem scale and the number of different possible tour lengths.**

***Index Terms*— Frequency Fitne Assignment, volution-ary Algorithm, A, Traveling sperson Pr veling Salesman Prob SP**

INT

st y

bad. Instead, solutions are preferred whose objective values y have a lower *encounter frequency* $H[y]$ during the search so far.

Algorithms using FFA attempt to visit all possible objective values (including the optimal one) equally often. This leads to two remarkable properties:

1) FFA creates optimization processes that are not biased towards good objective values but, instead, towards solutions with *rare* objective values [2].
2) It makes algorithms invariant under all injective transformations of the objective function value [3].

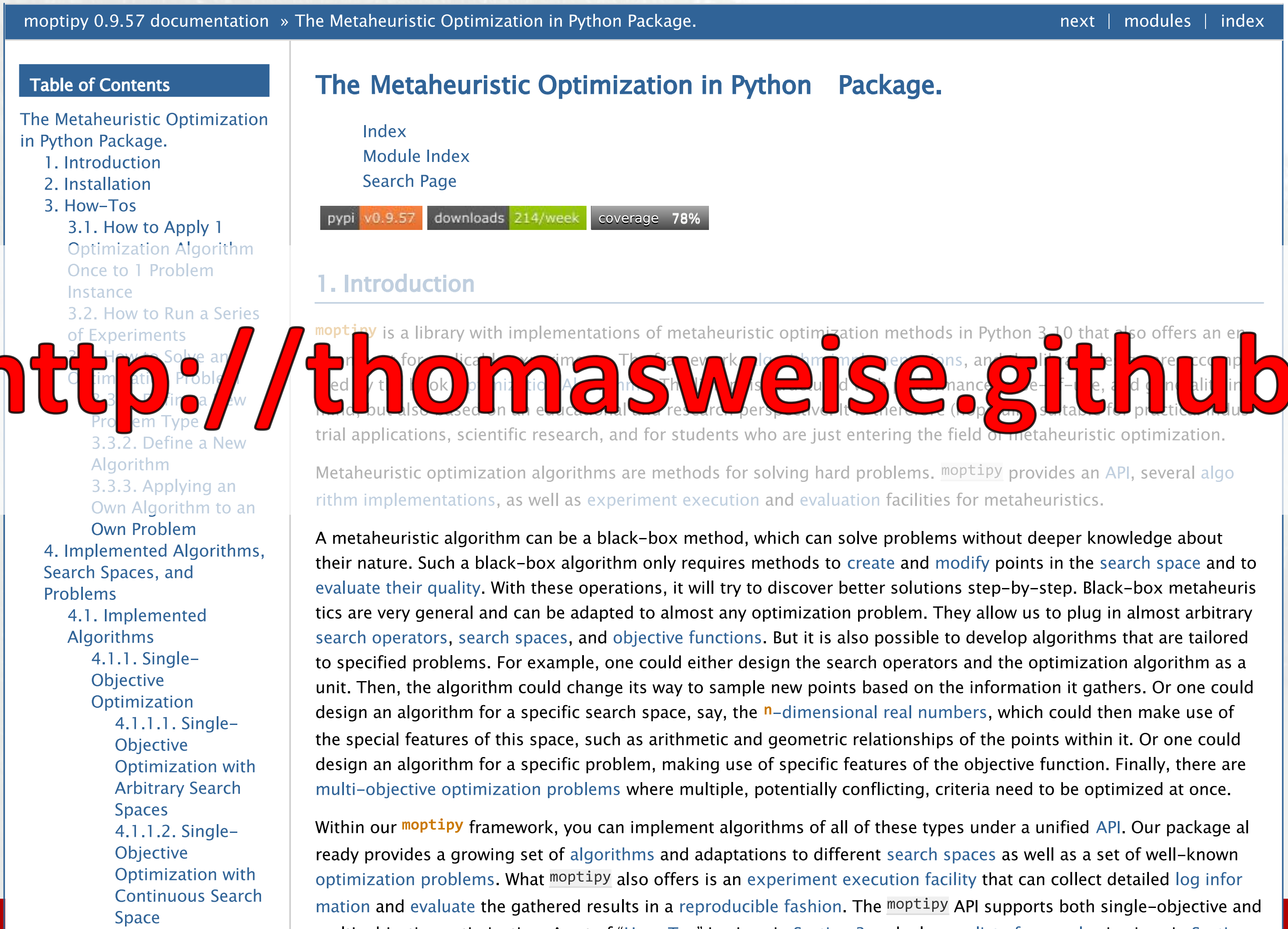
One would expect that such a different optimization approach would lead to a very bad performance. After all, the only traditional algorithms without a bias toward better solutions are random wa random sampling, and exhaustive enumeration, which ar orst-performin approaches for mo roblems. Surpr owever, FFA n significantly the perform veral alg s on several ms. The 1+1) E exponential ime o voM: pr n (1+ e a i

T Liang, Z Wu, J Lässig, D van den Berg, and T Weise. Solving the Traveling Salesperson Problem using Frequency Fitness Assignment. In Proceedings of the *IEEE Symposium on Foundations of Computational Intelligence (IEEE FOCI'22)*, part of the *IEEE Symposium Series on Computational Intelligence (SSCI 2022)*. December 4–7, 2022, Singapore

6. Advertisement

Slides

moptipy – A Python Package for Metaheuristics


moptipy 0.9.57 documentation » The Metaheuristic Optimization in Python Package.

next | modules | index

Table of Contents

The Metaheuristic Optimization in Python Package.

- Index
- Module Index
- Search Page

pypi v0.9.57 | downloads 214/week | coverage 78%

1. Introduction

moptipy is a library with implementations of metaheuristic optimization methods in Python 3.10 that also offers an en[illegible] for replicable experiments. The framework, algorithm implementations, and the library design are accompanied by the book Optimization Algorithms. The library is designed with performance, ease-of-use, and generality in mind, but also based on an educational and research perspective. It is therefore (hopefully) suitable for practical industrial applications, scientific research, and for students who are just entering the field of metaheuristic optimization.

Metaheuristic optimization algorithms are methods for solving hard problems. `moptipy` provides an API, several algorithm implementations, as well as experiment execution and evaluation facilities for metaheuristics.

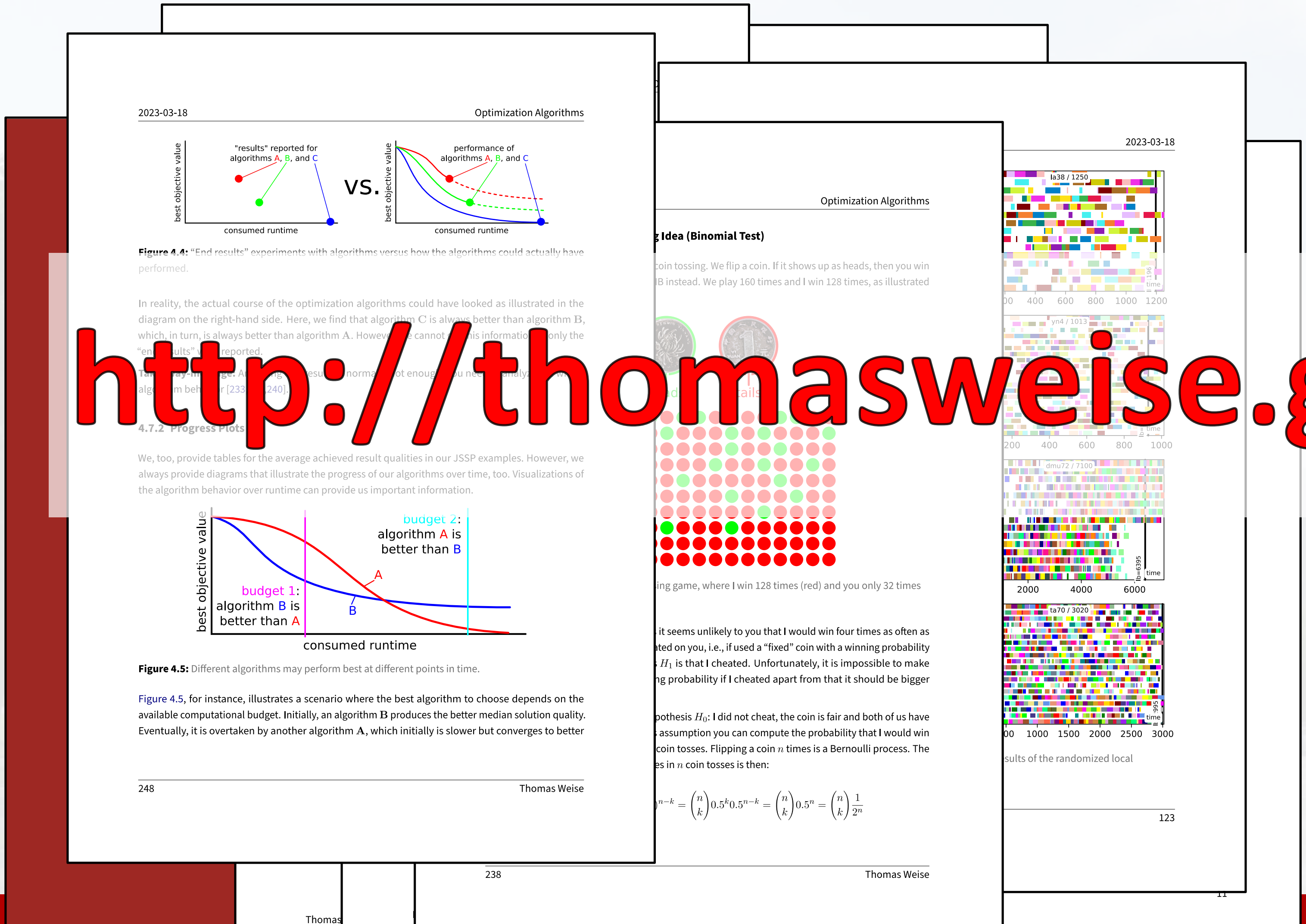
A metaheuristic algorithm can be a black-box method, which can solve problems without deeper knowledge about their nature. Such a black-box algorithm only requires methods to create and modify points in the search space and to evaluate their quality. With these operations, it will try to discover better solutions step-by-step. Black-box metaheuristics are very general and can be adapted to almost any optimization problem. They allow us to plug in almost arbitrary search operators, search spaces, and objective functions. But it is also possible to develop algorithms that are tailored to specified problems. For example, one could either design the search operators and the optimization algorithm as a unit. Then, the algorithm could change its way to sample new points based on the information it gathers. Or one could design an algorithm for a specific search space, say, the n-dimensional real numbers, which could then make use of the special features of this space, such as arithmetic and geometric relationships of the points within it. Or one could design an algorithm for a specific problem, making use of specific features of the objective function. Finally, there are multi-objective optimization problems where multiple, potentially conflicting, criteria need to be optimized at once.

Within our moptipy framework, you can implement algorithms of all of these types under a unified API. Our package already provides a growing set of algorithms and adaptations to different search spaces as well as a set of well-known optimization problems. What `moptipy` also offers is an experiment execution facility that can collect detailed log information and evaluate the gathered results in a reproducible fashion. The `moptipy` API supports both single-objective and

http://thomasweise.github.io/moptipy

- Implementations of several metaheuristics
- Run experiments in a parallel or distributed fashion
- Wrap algorithm implementations from other packages in a unified API
- Evaluate experiments

Optimization Algorithms: Free Online Book



2023-03-18 Optimization Algorithms

"results" reported for algorithms A, B, and C
best objective value
consumed runtime
vs.
performance of algorithms A, B, and C
best objective value
consumed runtime

Figure 4.4: "End results" experiments with algorithms versus how the algorithms could actually have performed.

In reality, the actual course of the optimization algorithms could have looked as illustrated in the diagram on the right-hand side. Here, we find that algorithm $\mathbf{C}$ is always better than algorithm $\mathbf{B}$, which, in turn, is always better than algorithm $\mathbf{A}$. However, we cannot … this information … only the "end results" are reported.

4.7.2 Progress Plots

We, too, provide tables for the average achieved result qualities in our JSSP examples. However, we always provide diagrams that illustrate the progress of our algorithms over time, too. Visualizations of the algorithm behavior over runtime can provide us important information.

best objective value
budget 2: algorithm A is better than B
budget 1: algorithm B is better than A
A
B
consumed runtime

Figure 4.5: Different algorithms may perform best at different points in time.

Figure 4.5, for instance, illustrates a scenario where the best algorithm to choose depends on the available computational budget. Initially, an algorithm $\mathbf{B}$ produces the better median solution quality. Eventually, it is overtaken by another algorithm $\mathbf{A}$, which initially is slower but converges to better

248 Thomas Weise

Optimization Algorithms

Idea (Binomial Test)

coin tossing. We flip a coin. If it shows up as heads, then you win … B instead. We play 160 times and I win 128 times, as illustrated

ing game, where I win 128 times (red) and you only 32 times

it seems unlikely to you that I would win four times as often as … ted on you, i.e., if used a "fixed" coin with a winning probability … H_1 is that I cheated. Unfortunately, it is impossible to make … ng probability if I cheated apart from that it should be bigger

pothesis H_0: I did not cheat, the coin is fair and both of us have … assumption you can compute the probability that I would win … coin tosses. Flipping a coin n times is a Bernoulli process. The … es in n coin tosses is then:

$$\binom{n}{k}0.5^k0.5^{n-k} = \binom{n}{k}0.5^n = \binom{n}{k}\frac{1}{2^n}$$

238 Thomas Weise

2023-03-18

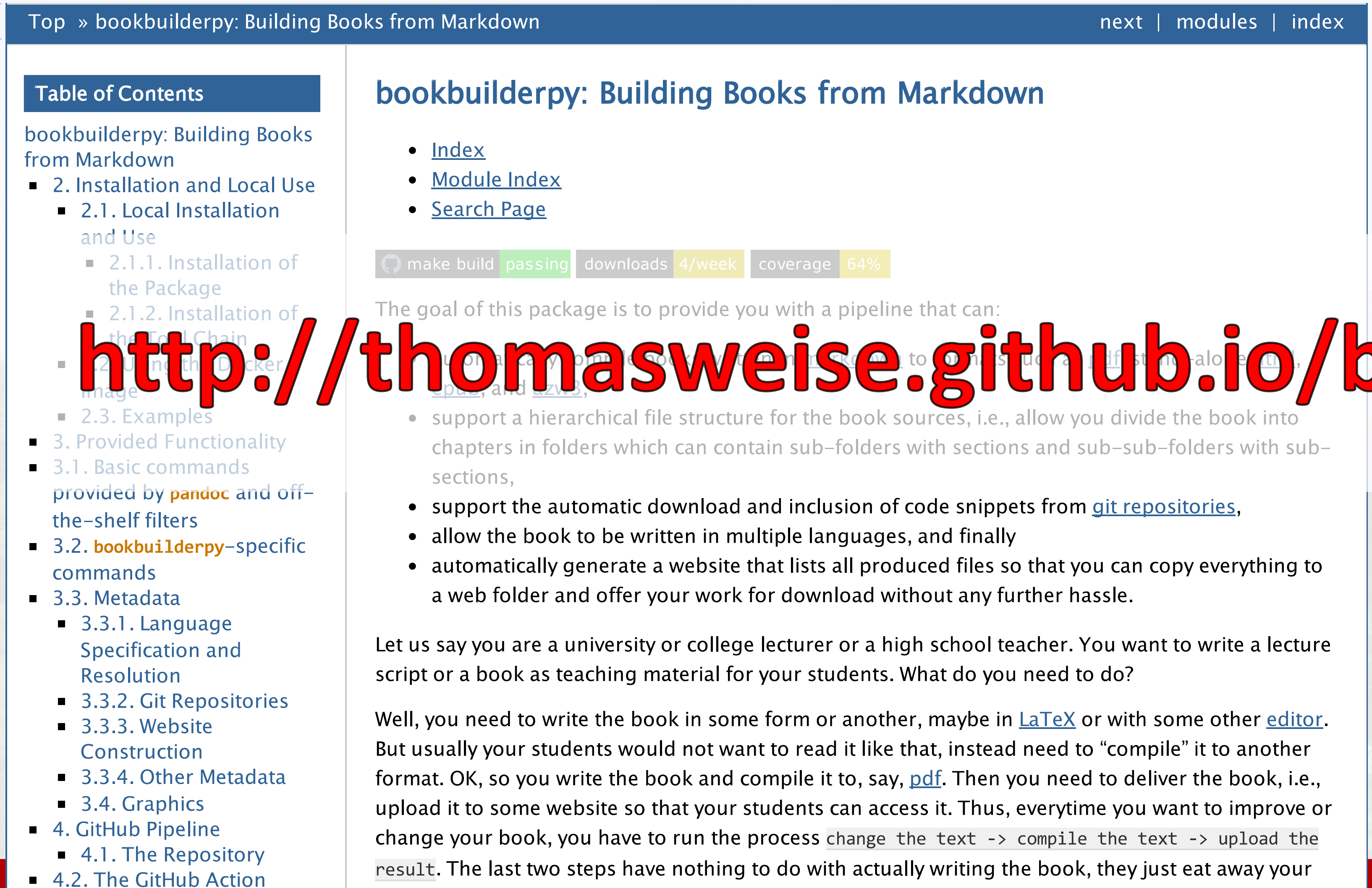
sults of the randomized local

123

http://thomasweise.github.io/oa

- Work in progress
- Several algorithms explained (EA, SA,…)
- Everything is done by experiments and with code examples.
- Goal: Learn how to understand algorithms

bookbuilderpy: Automated Workflow for Books


Top » bookbuilderpy: Building Books from Markdown

next | modules | index

Table of Contents

bookbuilderpy: Building Books from Markdown

- Index
- Module Index
- Search Page

make build passing | downloads 4/week | coverage 64%

The goal of this package is to provide you with a pipeline that can:

- automatically compile books written in Markdown to formats such as pdf, standalone html, epub, and azw3,
- support a hierarchical file structure for the book sources, i.e., allow you divide the book into chapters in folders which can contain sub-folders with sections and sub-sub-folders with sub-sections,
- support the automatic download and inclusion of code snippets from git repositories,
- allow the book to be written in multiple languages, and finally
- automatically generate a website that lists all produced files so that you can copy everything to a web folder and offer your work for download without any further hassle.

Let us say you are a university or college lecturer or a high school teacher. You want to write a lecture script or a book as teaching material for your students. What do you need to do?

Well, you need to write the book in some form or another, maybe in LaTeX or with some other editor. But usually your students would not want to read it like that, instead need to "compile" it to another format. OK, so you write the book and compile it to, say, pdf. Then you need to deliver the book, i.e., upload it to some website so that your students can access it. Thus, everytime you want to improve or change your book, you have to run the process `change the text -> compile the text -> upload the result`. The last two steps have nothing to do with actually writing the book, they just eat away your

http://thomasweise.github.io/bookbuilderpy

- Automated workflow for building pdf/html/epub books from Markdown
- Can be triggered via GitHub actions upon commit and auto-publish book to GitHub pages

Thank you very much.
谢谢您。

Prof. Dr. Thomas Weise (汤卫思)
Institute of Applied Optimization, Director
School of Artificial Intelligence and Big Data
Hefei University, South Campus 2, Building 53, Office 902
Hefei Economic and Technological Development Area
Jinxiu Dadao 99, Shushan, Hefei 230601, Anhui, China

Email: tweise@hfuu.edu.cn, tweise@ustc.edu.cn,
tweise@gmx.de
Web: http://iao.hfuu.edu.cn
Mobile: +8618755122841
Skype: thomas.weise

合肥学院 人工智能与大数据学院 应用优化研究所
中国 安徽省 合肥市 蜀山区 230601 经济技术开发区
南2区/南艳湖校区 锦绣大道99号 53栋 合肥学院综合实验楼

Website

Slides

Video