合肥大學
HEFEI UNIVERSITY

Frequency Fitness Assignment as a Research Direction

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO) | 应用优化研究所
School of Artificial Intelligence and Big Data | 人工智能与大数据学院
Hefei University | 合肥大学
Hefei, Anhui, China | 中国安徽省合肥市

Version: 2026-01-29

Outline

1. Introduction
2. Research Field: Optimization
3. Research Direction: Frequency Fitness Assignment
4. Our Results
5. Future Works
6. Summary
7. Advertisement

Introduction

Introduction

- Our team has two general research directions: Optimization[47] and Artificial Intelligence (AI).

Introduction

- Our team has two general research directions: Optimization[47] and Artificial Intelligence (AI).
- My students and I are members of the Optimization Direction.

Introduction

- Our team has two general research directions: Optimization[47] and Artificial Intelligence (AI).
- My students and I are members of the Optimization Direction.
- Today, I therefore want to first give a short introduction into the field of optimization, before delving into one of our concrete research topics, namely Frequency Fitness Assignment (FFA).

Research Field: Optimization

What is Optimization?

- There are two ways to look at optimization.

What is Optimization?

- The economic view.

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.

What is Optimization?

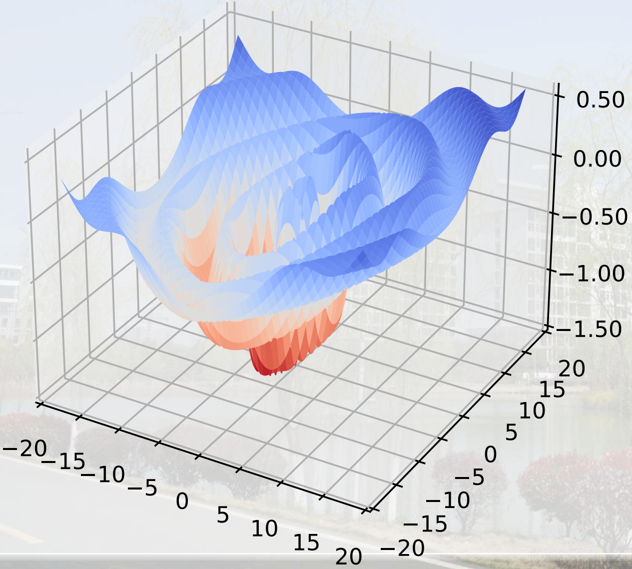
- The mathematical view.

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.

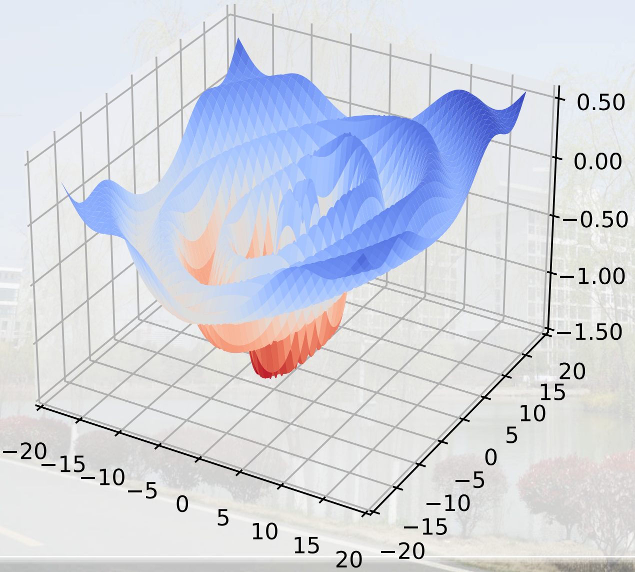
Solving an optimization problem requires finding an input element $x^\star$ within a set $\mathbb{X}$ of allowed elements for which a mathematical function $f: \mathbb{X} \mapsto \mathbb{R}$ takes on the smallest possible value.

Example: Function Optimization



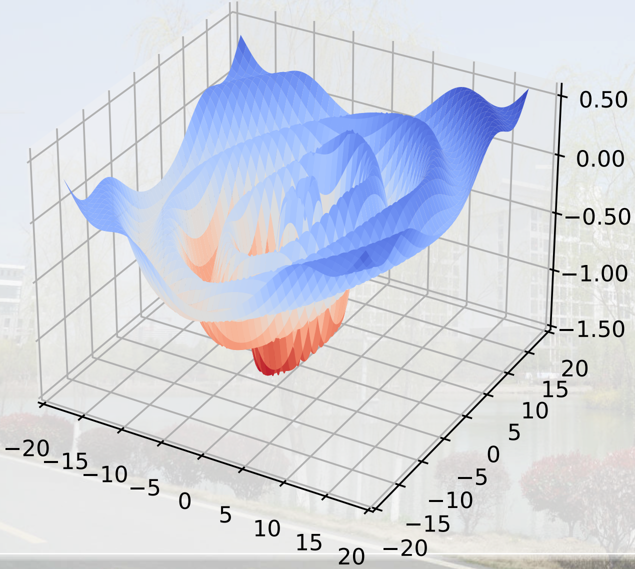
- In the field of continuous optimization, we could literally try to find the minimum of a mathematical function $f : \mathbb{R}^n \mapsto \mathbb{R}$.

Example: Function Optimization



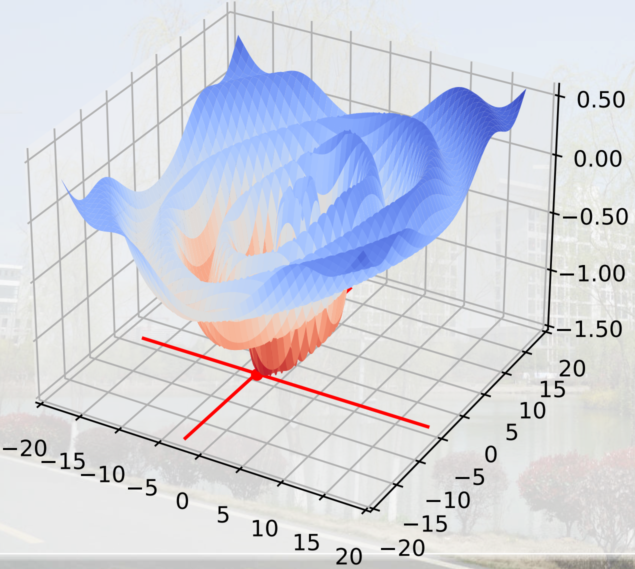
- In the field of continuous optimization, we could literally try to find the minimum of a mathematical function $f : \mathbb{R}^n \mapsto \mathbb{R}$.
- The search space $\mathbb{X}$ would then be the n-dimensional real vectors, i.e., $\mathbb{X} = \mathbb{R}^n$.

Example: Function Optimization



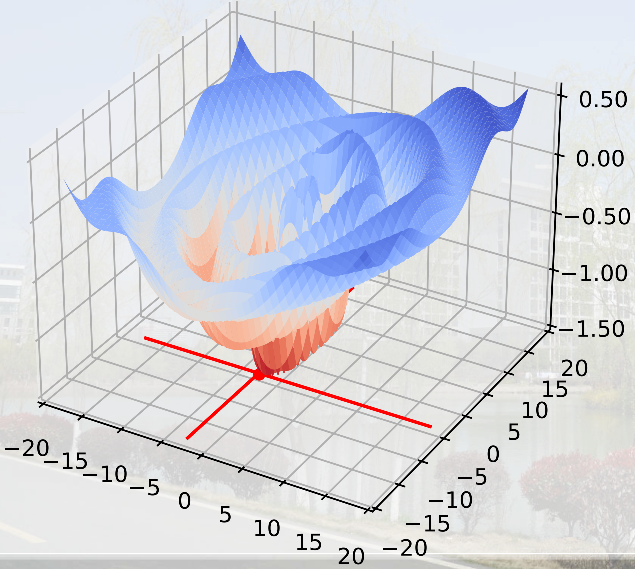
- In the field of continuous optimization, we could literally try to find the minimum of a mathematical function $f : \mathbb{R}^n \mapsto \mathbb{R}$.
- The search space $\mathbb{X}$ would then be the n-dimensional real vectors, i.e., $\mathbb{X} = \mathbb{R}^n$.
- The objective function would, well, be the function f.

Example: Function Optimization



- In the field of continuous optimization, we could literally try to find the minimum of a mathematical function $f : \mathbb{R}^n \mapsto \mathbb{R}$.
- The search space $\mathbb{X}$ would then be the n-dimensional real vectors, i.e., $\mathbb{X} = \mathbb{R}^n$.
- The objective function would, well, be the function f.
- The optimal solution $x^\star \in \mathbb{X}$ is the minimum of f.

Example: Function Optimization



- In the field of continuous optimization, we could literally try to find the minimum of a mathematical function $f : \mathbb{R}^n \mapsto \mathbb{R}$.
- The search space $\mathbb{X}$ would then be the n-dimensional real vectors, i.e., $\mathbb{X} = \mathbb{R}^n$.
- The objective function would, well, be the function f.
- The optimal solution $x^\star \in \mathbb{X}$ is the minimum of f.
- This is *not* what I am working on, though.

Example: Traveling Salesperson Problem

- In the Traveling Salesperson Problem (TSP)[1,16,28,30,49,57], the goal is to find the shortest round-trip tour through a set of n cities.

Example: Traveling Salesperson Problem

- In the Traveling Salesperson Problem (TSP)[1,16,28,30,49,57], the goal is to find the shortest round-trip tour through a set of n cities.
- The search space $\mathbb{X}$ thus is the set of all possible round-trip tours through these n cities, usually specified as permutations of the first n natural numbers.

Example: Traveling Salesperson Problem

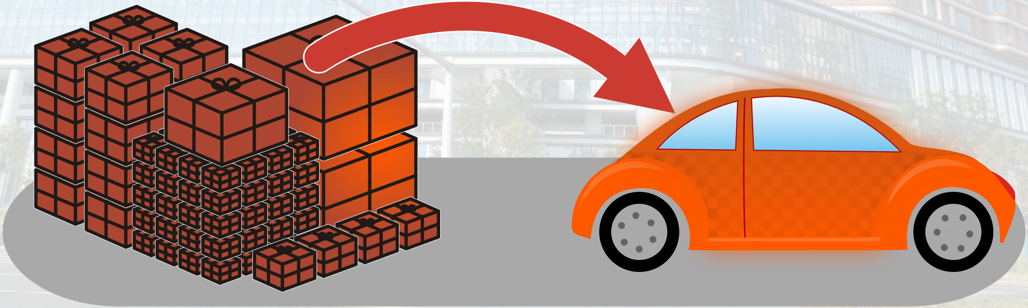
- In the Traveling Salesperson Problem (TSP)[1,16,28,30,49,57], the goal is to find the shortest round-trip tour through a set of n cities.
- The search space $\mathbb{X}$ thus is the set of all possible round-trip tours through these n cities, usually specified as permutations of the first n natural numbers.
- The objective function $f : \mathbb{X} \mapsto \mathbb{R}$, subject to minimization, is the length of the tour.

Example: Traveling Salesperson Problem

- In the Traveling Salesperson Problem (TSP)[1,16,28,30,49,57], the goal is to find the shortest round-trip tour through a set of n cities.
- The search space $\mathbb{X}$ thus is the set of all possible round-trip tours through these n cities, usually specified as permutations of the first n natural numbers.
- The objective function $f : \mathbb{X} \mapsto \mathbb{R}$, subject to minimization, is the length of the tour.
- The optimal solution $x^{\star} \in \mathbb{X}$ is the shortest possible tour.

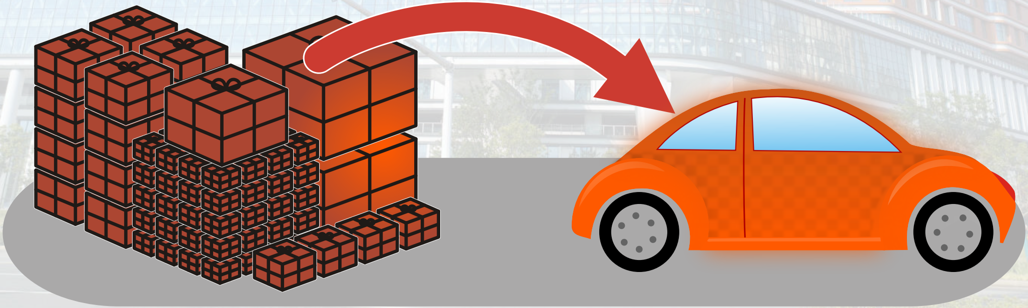
Example: Bin Packing Problem

- The goal of the Bin Packing Problem is to pack n objects, each having a specific size, into as few bins (also of a given size) as possible[60,61].



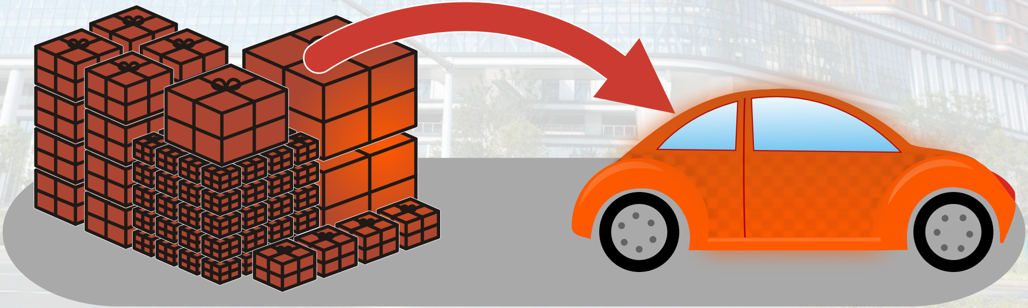
Example: Bin Packing Problem

- The goal of the Bin Packing Problem is to pack n objects, each having a specific size, into as few bins (also of a given size) as possible[60,61].
- The $\mathbb{X}$ comprises all possible packing orders of the n objects.



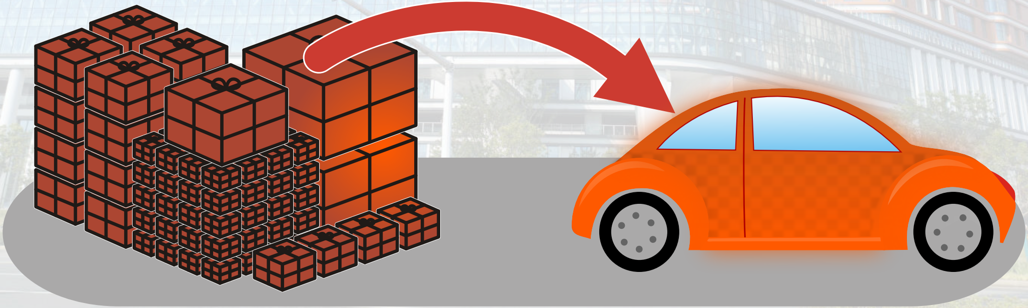
Example: Bin Packing Problem

- The goal of the Bin Packing Problem is to pack n objects, each having a specific size, into as few bins (also of a given size) as possible[60,61].
- The $\mathbb{X}$ comprises all possible packing orders of the n objects.
- The objective function f is the number of bins needed by a given packing order.



Example: Bin Packing Problem

- The goal of the Bin Packing Problem is to pack n objects, each having a specific size, into as few bins (also of a given size) as possible[60,61].
- The $\mathbb{X}$ comprises all possible packing orders of the n objects.
- The objective function f is the number of bins needed by a given packing order.
- The optimum $x^{\star}$ is the packing order requiring the fewest bins.



Optimization is Hard

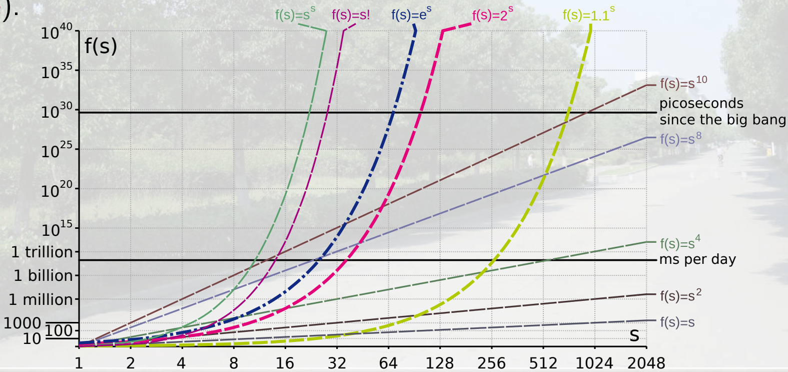
- Finding the globally optimal solution $x^\star$ from the set of all possible solutions $\mathbb{X}$ is often an $\mathcal{NP}$-hard problem.

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).

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).



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 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.

Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.

Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.

Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.

Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.
- They (usually) start with random solutions.

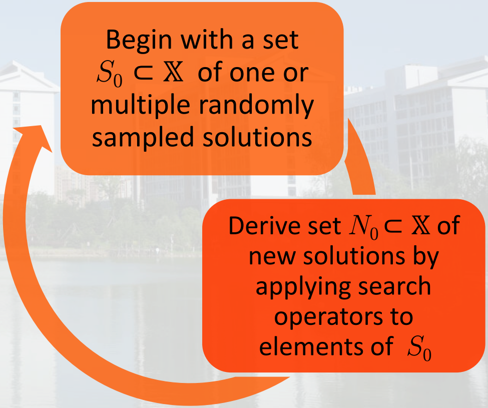
Begin with a set $S_0 \subset \mathbb{X}$ of one or multiple randomly sampled solutions

Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.
- They (usually) start with random solutions.
- And then roughly follow this cycle.

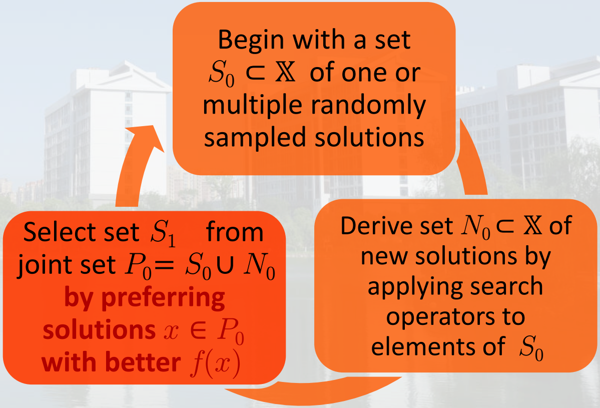
Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.
- They (usually) start with random solutions.
- And then roughly follow this cycle.



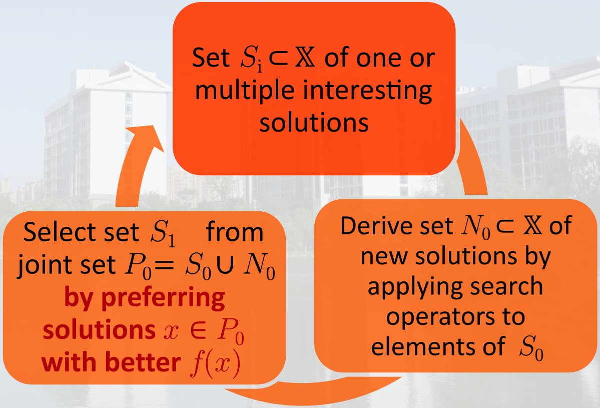
Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.
- They (usually) start with random solutions.
- And then roughly follow this cycle.



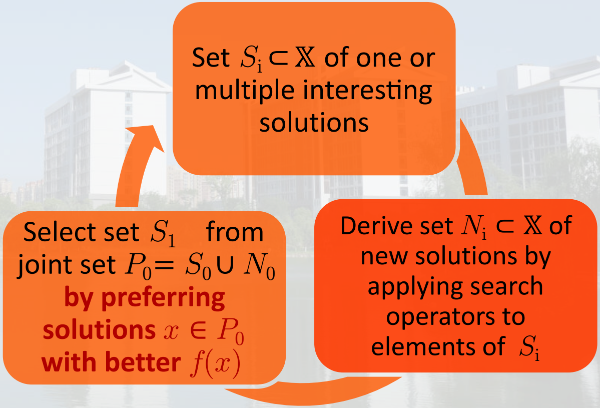
Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.
- They (usually) start with random solutions.
- And then roughly follow this cycle.



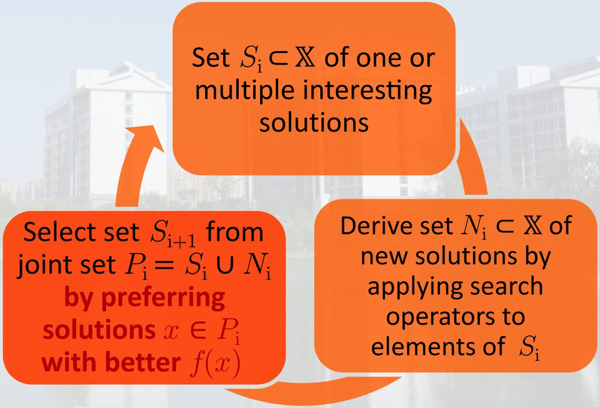
Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.
- They (usually) start with random solutions.
- And then roughly follow this cycle.



Metaheuristic Optimization

- Because of this hardness of optimization, metaheuristic algorithms that follow the Trial-and-Error idea of iterative improvement have emerged.
- They drop the guarantee to find the optimal solution.
- They try to find good solutions within a feasible runtime.
- They (usually) start with random solutions.
- And then roughly follow this cycle.



The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator $move$)[6,12].

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator $move$)[6,12].

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$

  randomly sample x_c from $\mathbb{X}$;

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator $move$)[6,12].

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$

randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator $move$)[6,12].

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**

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator $move$)[6,12].

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**
    $x_n \leftarrow \texttt{move}(x_c)$;

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator *move*)[6,12].

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**
        $x_n \leftarrow \mathtt{move}(x_c)$; $y_n \leftarrow f(x_n)$;

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator $move$)[6,12].
- They accept the new solution if it is better or equally good compared to the current solution.

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**

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator *move*)[6,12].
- They accept the new solution if it is better or equally good compared to the current solution.

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** $x_c \leftarrow x_n$;

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator *move*)[6,12].
- They accept the new solution if it is better or equally good compared to the current solution.

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** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

The $(1+1)$ EA and RLS

- Local search with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according the same pattern (and differ only in their unary search operator *move*)[6,12].
- They accept the new solution if it is better or equally good compared to the current solution.

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

Research Field

- My students and I together work on solving discrete or combinatorial problems with metaheuristic algorithms.

Research Field

- My students and I together work on solving discrete or combinatorial problems with metaheuristic algorithms.
- We investigate classical hard problems that have discrete search spaces, such as bit strings or permutations.

Research Field

- My students and I together work on solving discrete or combinatorial problems with metaheuristic algorithms.
- We investigate classical hard problems that have discrete search spaces, such as bit strings or permutations.
- Here, the objective functions take on only natural numbers in $\mathbb{N}_0$.

Research Field

- My students and I together work on solving discrete or combinatorial problems with metaheuristic algorithms.
- We investigate classical hard problems that have discrete search spaces, such as bit strings or permutations.
- Here, the objective functions take on only natural numbers in $\mathbb{N}_0$.
- We try to enable metaheuristic algorithms to find better solutions for these problems.

Research Direction: Frequency Fitness Assignment

Metaheuristic Optimization Algorithms

- The most fundamental concept in metaheuristic optimization is

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

Metaheuristic Optimization Algorithms

- The most fundamental concept in metaheuristic optimization is

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.

Metaheuristic Optimization Algorithms

- The most fundamental concept in metaheuristic optimization is

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.

- Algorithms like random sampling or exhaustive enumeration that do not at least statistically prefer better solutions have extremely bad performance.

Metaheuristic Optimization Algorithms

- The most fundamental concept in metaheuristic optimization is

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.

- Algorithms like random sampling or exhaustive enumeration that do not at least statistically prefer better solutions have extremely bad performance.
- We challenge this principle.

Metaheuristic Optimization Algorithms

- The most fundamental concept in metaheuristic optimization is

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.

- Algorithms like random sampling or exhaustive enumeration that do not at least statistically prefer better solutions have extremely bad performance.
- We challenge this principle.
- Our Frequency Fitness Assignment (FFA) does **not** prefer better solutions ... yet it can improve the performance of existing algorithms in several cases!

FFA: Idea

- Frequency Fitness Assignment (FFA) is a module that can be plugged into different existing algorithms.

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 sets $P_i = S_i \cup N_i$.

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 sets $P_i = S_i \cup N_i$.
- It therefore maintains a table H with the encounter frequency of each objective value in the selection decisions.

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 sets $P_i = S_i \cup N_i$.
- 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.

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 sets $P_i = S_i \cup N_i$.
- 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, $H[f(P_i[j])]$ for all $j \in 1..|P_i|$ is incremented by 1.

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 sets $P_i = S_i \cup N_i$.
- 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, $H[f(P_i[j])]$ for all $j \in 1..|P_i|$ is incremented by 1.
- Then, the frequencies $H[f(P_i[j])]$ replace the objective values $f(P_i[j])$ in the actual selection decisions.

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

- Let's plug FFA into the $(1+1)$ EA and obtain the $(1+1)$ EA with FFA ($(1+1)$ 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 $\boldsymbol{y_n \leq y_c}$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
  return x_c, y_c

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

- We start with the $(1+1)$ EA.

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 $\boldsymbol{y_n \leq y_c}$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
  return x_c, y_c

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{N})$

  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 $\boldsymbol{y_n \leq y_c}$ **then**
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

  return x_c, y_c

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

- We begin by initializing the frequency table H with all zeros.

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 $\boldsymbol{y_n \leq y_c}$ **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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

  while $\neg$ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;

    if $\boldsymbol{y_n \leq y_c}$ **then**
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

  return x_c, y_c

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

- *Before* the selection decision, we increment the frequency values of the objective values of all current solutions.

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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

  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 $y_n \leq y_c$ **then**
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

  return x_c, y_c

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

- Now the frequency values replace the objective values in the selection decisions.

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 $\boldsymbol{y_n \leq y_c}$ **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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

  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**
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

  return x_c, y_c

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

- Since we may now lose the best-so-far solution, we need to track it in additional variables.

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 $\boldsymbol{y_n \leq y_c}$ **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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
    $x_{\mathrm{B}} \leftarrow x_c$; $y_{\mathrm{B}} \leftarrow y_c$;
    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**
            $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

    return x_c, y_c

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

- Since we may now lose the best-so-far solution, we need to track it in additional variables.

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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
  $x_\mathrm{B} \leftarrow x_c$; $y_\mathrm{B} \leftarrow y_c$;
  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 $y_c < y_\mathrm{B}$ **then** $x_\mathrm{B} \leftarrow x_c$; $y_\mathrm{B} \leftarrow y_c$;
  return x_c, y_c

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

- Since we may now lose the best-so-far solution, we need to track it in additional variables.
- . . . which are then the return values of the $(1+1)$ 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 $\boldsymbol{y_n \leq y_c}$ **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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
    $x_{\mathrm{B}} \leftarrow x_c$; $y_{\mathrm{B}} \leftarrow y_c$;
    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 $y_c < y_{\mathrm{B}}$ **then** $x_{\mathrm{B}} \leftarrow x_c$; $y_{\mathrm{B}} \leftarrow y_c$;
    return x_{B}, y_{B}

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on how often solutions x' with $f(x') = f(x)$ have previously been seen in the optimization process.

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on how often solutions x' with $f(x') = f(x)$ have previously been seen in the optimization process.
- Static optimization problems become dynamic, because frequency fitness H changes over time.

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on how often solutions x' with $f(x') = f(x)$ have previously been seen in the optimization process.
- Static optimization problems become dynamic, because frequency fitness H changes over time.
- Solutions get less attractive the more often their corresponding objective values have been seen. This also holds for local optima...

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on how often solutions x' with $f(x') = f(x)$ have previously been seen in the optimization process.
- Static optimization problems become dynamic, because frequency fitness H 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.

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on how often solutions x' with $f(x') = f(x)$ have previously been seen in the optimization process.
- Static optimization problems become dynamic, because frequency fitness H 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.

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on how often solutions x' with $f(x') = f(x)$ have previously been seen in the optimization process.
- Static optimization problems become dynamic, because frequency fitness H 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.
- Algorithms using FFA are invariant under all injective transformations of the objective function value.

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on how often solutions x' with $f(x') = f(x)$ have previously been seen in the optimization process.
- Static optimization problems become dynamic, because frequency fitness H 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.
- Algorithms using FFA are invariant under all injective transformations of the objective function value.
- They are less likely to get stuck at local optima, which is a problem for, e.g., the $(1+1)$ EA.

Our Results

Three Core Works

snippets of [53, 55, 56]

Frequency Fitness Assignment

Thomas Weise, *Member, IEEE,* Mingxu Wan, Pu Wang, *Member, IEEE,* Ke Tang, *Senior Member, IEEE,* Alexandre Devert, and Xin Yao, *Fellow, IEEE*

***Abstract*—Metaheuristic optimization procedures such as evolutionary algorithms are usually driven by an objective function that rates the quality of a candidate solution. However, it is not clear in practice whether an objective function adequately rewards intermediate solutions on the path to the global optimum and it may exhibit deceptiveness, epistasis, neutrality, ruggedness, and a lack of causality. In this paper, we introduce the frequency fitness H, subject to minimization, which rates how often solutions with the same objective value have been discovered so far. The ideas behind this method are that good solutions are difficult to find and that if an algorithm gets stuck at a local optimum, the frequency of the objective values of the surrounding solutions will increase over time, which will eventually allow it to leave that region again. We substitute a frequency fitness assignment process (FFA) for the objective function into several different optimization algorithms. We conduct a comprehensive set of experiments: the synthesis of algorithms with genetic programming (GP), the solution of MAX-3SAT problems with genetic algorithms, classification with Memetic Genetic Programming, and numerical optimization with a (1+1) Evolution Strategy, to verify the utility of FFA. Given that they have no access to the original objective function at all, it is surprising that for some problems (e.g., the algorithm synthesis task) the FFA-based algorithm variants perform significantly better. However, this cannot be guaranteed for all tested problems. Thus, we also analyze scenarios where algorithms using FFA do not perform better or perform even worse than with the original objective functions.**

***Index Terms*—Combinatorial optimization, diversity, fitness assignment, frequency, genetic programming (GP), numerical optimization.**

I. Introduction

SINGLE-OBJE… VE optimi… ion is … process with the goal of findi… (id… all… i.e. … nal) sol… tions x from within a space $\mathbb{X}$ of possible solutions. An objective function f serves as quality measure guiding the search. Black-box metaheuristic approaches are methods that only require such an objective function and search operations to solve an optimization problem without any further insight into their structure. The most prominent family of these methods are evolutionary algorithms, which have wide applications ranging from engineering, planning and scheduling, numerical optimization, to even program synthesis.

Most metaheuristic optimization methods start with a randomly generated set of candidate solutions. New points in the search space are derived by modifying or combining promising existing solutions. Promising here means having a better objective value than the other points visited so far, maybe combined with some considerations about diversity. The rationale is that in the ideal case, solutions that have better objective values should be closer to the global optimum or, at least, may have even better solutions in their vicinity.

The principle of tending to choose areas of the solution space for sampling where points with better objective values have previously been discovered is one of the most universally applied ideas in black-box optimization. Lehman and Stanley [1] argued that increasing fitness does not always reveal the best path through the search space. Building on their work, we believe that there is at least one other fun… ental principle inherent to non-trivial optimiz… tion problems … be exploited to solve them: good solut… ons (and henc… objective … lues) are hard to find.

If … e … nsi… er optimi… ation as sa… of th… pace … v… uld exp… freq…

Three Core Works

snippets of [53, 55, 56]

226 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 18, NO. 2, APRIL 2014

Frequency Fitness A...

Thomas Weise, *Member, IEEE,* Mingxu Wan, Pu Wang, *Me...*
Alexandre Devert, and Xin Ya...

***Abstract*—Metaheuristic optimization procedures such as evolutionary algorithms are usually driven by an objective function that rates the quality of a candidate solution. However, it is not clear in practice whether an objective function adequately rewards intermediate solutions on the path to the global optimum and it may exhibit deceptiveness, epistasis, neutrality, ruggedness, and a lack of causality. In this paper, we introduce the frequency fitness H, subject to minimization, which rates how often solutions with the same objective value have been discovered so far. The ideas behind this method are that good solutions are difficult to find and that if an algorithm gets stuck at a local optimum, the frequency of the objective values of the surrounding solutions will increase over time, which will eventually allow it to leave that region again. We substitute a frequency fitness assignment process (FFA) for the objective function into several different optimization algorithms. We conduct a comprehensive set of experiments: the synthesis of algorithms with genetic programming (GP), the solution of MAX-3SAT problems with genetic algorithms, classification with Memetic Genetic Programming, and numerical optimization with a (1+1) Evolution Strategy, to verify the utility of FFA. Given that they have no access to the original objective function at all, it is surprising that for some problems (e.g., the algorithm synthesis task) the FFA-based algorithm variants perform significantly better. However, this cannot be guaranteed for all tested problems. Thus, we also analyze scenarios where algorithms using FFA do not perform better or perform even worse than with the original objective functions.**

***Index Terms*—Combinatorial optimization, diversity, fitness assignment, frequency, genetic programming (GP), numerical optimization.**

I. INTRODUCTION

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

***Abstract*—Under frequency fitness assignment (FFA), the fitness corresponding to an objective value is its encounter frequency in fitness assignment steps and is subject to minimization. FFA renders optimization processes invariant under bijective transformations of the objective function value. On TwoMax, Jump, and Trap functions of dimension s, the classical (1 + 1)-EA with standard mutation at rate 1/s can have expected runtimes exponential in s. In our experiments, a (1 + 1)-FEA, the same algorithm but using FFA, exhibits mean runtimes that seem to scale as $s^2 \ln s$. Since Jump and Trap are bijective transformations of OneMax, it behaves identical on all three. On OneMax, LeadingOnes, and Plateau problems, it seems to be slower than the (1 + 1)-EA by a factor linear in s. The (1 + 1)-FEA performs much better than the (1 + 1)-EA on W-Model and MaxSat instances. We further verify the bijection invariance by applying the Md5 checksum computation as transformation to some of the above problems and yield the same behaviors. Finally, we show that FFA can improve the performance of a memetic algorithm for job shop scheduling.**

which are then the basis for selection. Frequency fitness assignment (FFA) [1], [2] was developed to enable algorithms to escape from local optima. In FFA, the fitness corresponding to an objective value is its encounter frequency so far in fitness assignment steps and is subject to minimization. As we discuss in detail in Section II, FFA turns a static optimization problem into a dynamic one where objective values that are often encountered will receive worse and worse fitness.

In this article, we uncover a so-far unexplored property of FFA: it is invariant under any bijective transformation of the objective function values. This is the strongest invariance known to us and encompasses all order-preserving mappings. Other examples for bijective transformations include the negation, permutation, or even encryption of the objective values. According to [3], invariance extends performance observed ...

Three Core Works

snippets of [53, 55, 56]

Frequency Fitness A

Thomas Weise, *Member, IEEE,* Mingxu Wan, Pu Wang, *Me*
Alexandre Devert, and Xin Ya

Abstract—**Metaheuristic optimization procedures such as evolutionary algorithms are usually driven by an objective function that rates the quality of a candidate solution. However, it is not clear in practice whether an objective function adequately rewards intermediate solutions on the path to the global optimum and it may exhibit deceptiveness, epistasis, neutrality, ruggedness, and a lack of causality. In this paper, we introduce the frequency fitness H, subject to minimization, which rates how often solutions with the same objective value have been discovered so far. The ideas behind this method are that good solutions are difficult to find and that if an algorithm gets stuck at a local optimum, the frequency of the objective values of the surrounding solutions will increase over time, which will eventually allow it to leave that region again. We substitute a frequency fitness assignment process (FFA) for the objective function into several different optimization algorithms. We conduct a comprehensive set of experiments: the synthesis of algorithms with genetic programming (GP), the solution of MAX-3SAT problems with genetic algorithms, classification with Memetic Genetic Programming, and numerical optimization with a (1+1) Evolution Strategy, to verify the utility of FFA. Given that they have no access to the original objective function at all, it is surprising that for some problems (e.g., the algorithm synthesis task) the FFA-based algorithm variants perform significantly better. However, this cannot be guaranteed for all tested problems. Thus, we also analyze scenarios where algorithms using FFA do not perform better or perform even worse than with the original objective functions.**

Index Terms—**Combinatorial optimization, diversity, fitness assignment, frequency, genetic programming (GP), numerical optimization.**

I. Introduction

SINGLE-OBJE… VE optimi… ion is … process with the goal of findi… (id… all… i.e. … nal) sol… tions

Frequency Fitness A… Optimization Algorit… Bijective Transfo… Objective Fu…

Thomas Weise, *Member, IEEE,* Z

Abstract—**Under frequency fitness assignment (FFA), the fitness corresponding to an objective value is its encounter frequency in fitness assignment steps and is subject to minimization. FFA renders optimization processes invariant under bijective transformations of the objective function value. On TwoMax, Jump, and Trap functions of dimension s, the classical $(1+1)$-EA with standard mutation at rate 1/s can have expected runtimes exponential in s. In our experiments, a $(1+1)$-FEA, the same algorithm but using FFA, exhibits mean runtimes that seem to scale as $s^2 \ln s$. Since Jump and Trap are bijective transformations of OneMax, it behaves identical on all three. On OneMax, LeadingOnes, and Plateau problems, it seems to be slower than the $(1+1)$-EA by a factor linear in s. The $(1+1)$-FEA performs much better than the $(1+1)$-EA on W-Model and MaxSat instances. We further verify the bijection invariance by applying the Md5 checksum computation as transformation to some of the above problems and yield the same behaviors. Finally, we show that FFA can improve the performance of a memetic algorithm for job shop scheduling.**

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 toward 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 evolutionary algorithms, the Greedy (2 + 1) GA and the self-adjusting $(1+(\lambda,\lambda))$ 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—**Evolutionary algorithm (EA), FEA, frequency fitness assignment (FFA), Ising problems, jump problems, linear harmonic functions, MaxSat problem, *N*-queens problem, OneMax, Plateau problems, satisfiability, Trap function, TwoMax, *W*-model benchmark.**

fitness assignment (FFA), the fitness of a candidate solution is the absolute encounter frequency of its objective value so far during the optimization process [1]. Being subject to minimization, FFA drives the search away from already-discovered objective values and toward solutions with new qualities.

FFA breaks with one of the most fundamental concepts of heuristic optimization: FFA-based algorithms are not biased toward better solutions [2], i.e., they do not prefer better solutions over worse ones. They also are invariant under all injective transformations of the objective function value, which is the strongest invariance property of any nontrivial 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 [1], [4] and on a benchmark problem simulating such landscapes [5]. While the $(1+1)$ EA has exp… ntial expected runtime on problems, such as Jum… TwoMax … d Trap, the $(1+1)$ FEA, th… same al… using FFA … hibits mean runtimes that …

INTRODUCTION

ITNESS … many evo…

Discrete Benchmark Functions

- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.

Discrete Benchmark Functions

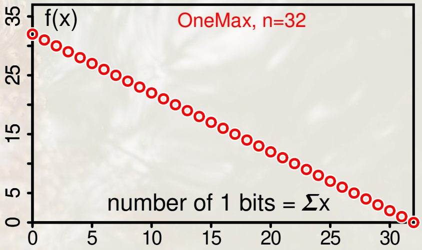
- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.
- They are usually tested on a variety of benchmark problems.

Discrete Benchmark Functions

- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.
- They are usually tested on a variety of benchmark problems.
- We plug FFA into the $(1+1)$ EA and obtain the $(1+1)$ FEA.

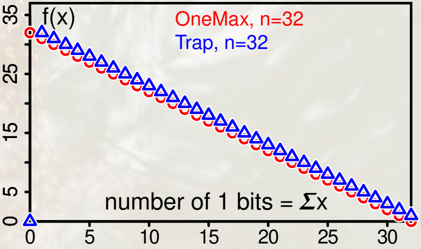
Discrete Benchmark Functions

- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.
- They are usually tested on a variety of benchmark problems.
- We plug FFA into the $(1+1)$ EA and obtain the $(1+1)$ FEA.
- The $(1+1)$ EA can easily solve the OneMax problem, where the goal is to find a bit string of all 1s[36].



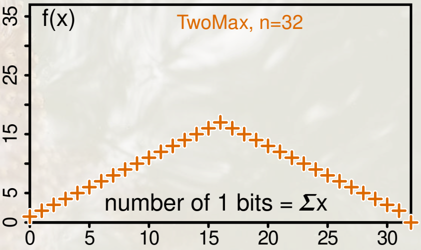
Discrete Benchmark Functions

- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.
- They are usually tested on a variety of benchmark problems.
- We plug FFA into the $(1+1)$ EA and obtain the $(1+1)$ FEA.
- The $(1+1)$ EA can easily solve the OneMax problem, where the goal is to find a bit string of all 1s[36].
- It needs $\Theta(n^n)$ on traps[12], where the optimum and the worst solution are exchanged compared to OneMax.



Discrete Benchmark Functions

- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.
- They are usually tested on a variety of benchmark problems.
- We plug FFA into the $(1+1)$ EA and obtain the $(1+1)$ FEA.
- The $(1+1)$ EA can easily solve the OneMax problem, where the goal is to find a bit string of all 1s[36].
- It needs $\Theta(n^n)$ on traps[12], where the optimum and the worst solution are exchanged compared to OneMax, as well as on TwoMax[14,44], which has one local and one global optimum.

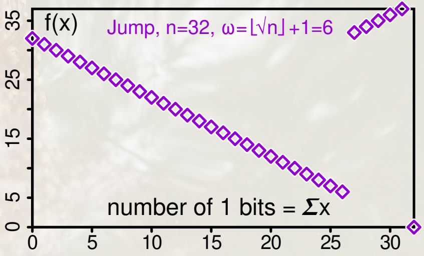


Discrete Benchmark Functions

- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.
- They are usually tested on a variety of benchmark problems.
- We plug FFA into the $(1+1)$ EA and obtain the $(1+1)$ FEA.
- The $(1+1)$ EA can easily solve the OneMax problem, where the goal is to find a bit string of all 1s[36].
- It needs $\mathbf{\Theta}(n^n)$ on traps[12], where the optimum and the worst solution are exchanged compared to OneMax, as well as on TwoMax[14,44], which has one local and one global optimum.
- The $(1+1)$ FEA solves all three problems in polynomial time!

Discrete Benchmark Functions

- Discrete optimization algorithms work on bit strings of length n as search space and use only the objective values of the solutions they sample to solve a problem.
- They are usually tested on a variety of benchmark problems.
- We plug FFA into the $(1+1)$ EA and obtain the $(1+1)$ FEA.
- The $(1+1)$ EA can easily solve the OneMax problem, where the goal is to find a bit string of all 1s[36].
- It needs $\boldsymbol{\Theta}(n^n)$ on traps[12], where the optimum and the worst solution are exchanged compared to OneMax, as well as on TwoMax[14,44], which has one local and one global optimum.



- The $(1+1)$ FEA solves all three problems in polynomial time as well as the Jump problem, where the $(1+1)$ EA also needs exponential runtime!

Maximum Satisfiability Problem

- FFA on the Maximum Satisfiability (MaxSAT) Problem, one of the most famous discrete optimization problems [53, 55, 56].

Maximum Satisfiability Problem

- FFA on the Maximum Satisfiability (MaxSAT) Problem, one of the most famous discrete optimization problems [53, 55, 56].

> From Table I, we can see that **the highest number of failed runs at scale $s = 250$ of *any* algorithm using FFA is lower than the lowest number of failed runs of *any* pure algorithm at $s = 50$.** From Table II, we find that no FFA-based algorithm has a higher ERT at scale $s = 250$ than its pure variant on $s = 50$. On the scales $s \leq 75$, the FFA-based algorithms have a mean runtime which is between three and four orders of magnitude smaller that the ERT of the pure algorithms.

- Snippet of page 10 of [56] (copyright IEEE).
- Several different EAs with and without FFA

Traveling Salesperson Problem

- FFA on the Traveling Salesperson Problem (TSP): [30–32].

Traveling Salesperson Problem

- FFA on the Traveling Salesperson Problem (TSP): [30–32].
- Snippet of page 12 of [30] (copyright Springer).
- $(1+1)$ EA, $(1+1)$ FEA, Simulated Annealing (SA) w/o FFA, hybrids

In this work, we explored both the EA and SA on 56 symmetric instances from the benchmark set TSPLib (Reinelt 1991, 1995). The EA is unsuitable for this problem, but SA can find the optimum on many small and mid-sized instances. We then plug FFA into both algorithms and obtain the FEA and the FSA, respectively, which both exhibit very similar performance. **The FEA solves 27 of the instances in all of its runs, which SA can only achieve for 19 instances.** Plugging FFA into either the EA or SA thus substantially improved the number of runs in which the algorithms can find the optimum, however, both FFA-based variants suffer when the number of unique objective values is high.

• • •

Both types of **hybridization** methods significantly improve the average result quality compared to both the objective-guided and FFA-based FEA variants. **The SAFEA$_A$ discovers the optimal solutions in more runs than any other algorithm setup in our study.** It also finds the optimum most often in most instances and delivers the best approximation quality on most instances, compared to the other algorithms.

Quadratic Assignment Problem

- FFA on the Quadratic Assignment Problem (QAP): [9, 43].

Quadratic Assignment Problem

- FFA on the Quadratic Assignment Problem (QAP): [9, 43].
- Snippet of page 7 of [9] (copyright SciTePress).
- RLS w/o FFA

We find that the **FFA-based** randomized local search **FRLS does not just find better solutions than the objective-guided RLS algorithm on the vast majority of the QAPLIB instances**, it also keeps improving its current best solution for the complete computational budget of 10^8 FEs that we assigned to the runs. With this budget, **it can discover the optimal solutions of over 58% of the QAPLIB instances**. Had we assigned a larger budget – (Liang et al., 2022; Liang et al., 2024; Weise et al., 2021b; Weise et al., 2023) use 10^{10} FEs – we would likely have seen even more instances solved.

We furthermore confirm the remarkable ability of FFA to discover very diverse solutions (at least from the perspective of the objective function). It is known that on the QAP, many solutions tend to have the same objective values (Tayarani-N. and Prügel-Bennett, 2015). Yet, on some of the instances, more than half of the objective values discovered by FRLS were unique.

Job Shop Scheduling Problem

- FFA on the Job Shop Scheduling Problem (JSSP) Problem: [11, 50, 55].

Job Shop Scheduling Problem

- FFA on the Job Shop Scheduling Problem (JSSP) Problem: [11, 50, 55].

The **end result quality delivered by (1+1)-FEA is better in average on the abz*, ft*, la*, orb*, and yn4* instance sets, both in terms of *best* and *mean*.** On swv*, the average for *mean* is better for (1+1)-FEA, while (1+1)-EA has a slight lead in *best*. The (1+1)-EA performs better on the dmu* and ta* instances. Since these two sets are larger (holding 160 out of the 242 instances), the (1+1)-EA comes out ahead in the overall averages, but with no more than a 1.5% advantage.

- Snippet of page 10 of [50] (copyright ACM).

Algorithm Synthesis and Genetic Programming

- FFA for algorithm synthesis and Genetic Programming (GP): [52, 53].

Algorithm Synthesis and Genetic Programming

- FFA for algorithm synthesis and Genetic Programming (GP): [52, 53].

> Fourth, we were able to confirm with great significance that **FFA** has a tremendous positive impact on the performance of GP, as it **can increase the success rate by 40%.**

- Snippet of page 7 of [52] (copyright IEEE).

Future Works

Future Works

- Apply FFA to other discrete and combinatorial problems.

Future Works

- Apply FFA to other discrete and combinatorial problems:
    1. Where can FFA improve the solution quality?

Future Works

- Apply FFA to other discrete and combinatorial problems:
  1. Where can FFA improve the solution quality?
  2. Where can it not do that?

Future Works

- Apply FFA to other discrete and combinatorial problems:
    1. Where can FFA improve the solution quality?
    2. Where can it not do that?
    3. Why?

Future Works

- Apply FFA to other discrete and combinatorial problems:
    1. Where can FFA improve the solution quality?
    2. Where can it not do that?
    3. Why?
- Plug FFA into more optimization algorithms.

Future Works

- Apply FFA to other discrete and combinatorial problems:
  1. Where can FFA improve the solution quality?
  2. Where can it not do that?
  3. Why?
- Plug FFA into more optimization algorithms.
- Combine "traditional" and FFA-based optimization, i.e., create hybrid algorithms, to reap the best of both worlds.

Summary

Summary

- Our students and I work on the big field of optimization.

Summary

- Our students and I work on the big field of optimization.
- We try to improve the quality of solutions that metaheuristics can deliver on discrete or combinatorial problems.

Summary

- Our students and I work on the big field of optimization.
- We try to improve the quality of solutions that metaheuristics can deliver on discrete or combinatorial problems.
- We have a very interesting and novel technique called Frequency Fitness Assignment (FFA).

Summary

- Our students and I work on the big field of optimization.
- We try to improve the quality of solutions that metaheuristics can deliver on discrete or combinatorial problems.
- We have a very interesting and novel technique called Frequency Fitness Assignment (FFA).
- We are figuring out where it really can give us better results.

Summary

- Our students and I work on the big field of optimization.
- We try to improve the quality of solutions that metaheuristics can deliver on discrete or combinatorial problems.
- We have a very interesting and novel technique called Frequency Fitness Assignment (FFA).
- We are figuring out where it really can give us better results.
- But we are not picky.

Summary

- Our students and I work on the big field of optimization.
- We try to improve the quality of solutions that metaheuristics can deliver on discrete or combinatorial problems.
- We have a very interesting and novel technique called Frequency Fitness Assignment (FFA).
- We are figuring out where it really can give us better results.
- But we are not picky. For example,
    1. on the two-dimensional bin packing problem, we found out that a simple local search can actually outperform the complex state-of-the-art metaheuristics and FFA could not improve the performance . . . so we published this surprising result and the student graduated with it[60,61].

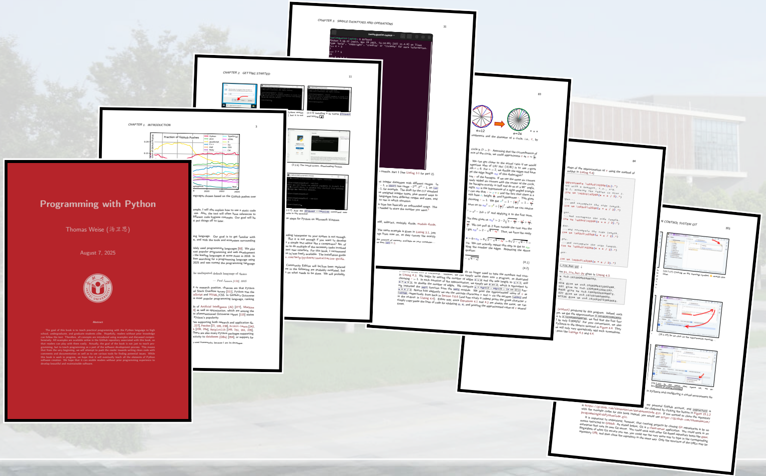
Summary

- Our students and I work on the big field of optimization.
- We try to improve the quality of solutions that metaheuristics can deliver on discrete or combinatorial problems.
- We have a very interesting and novel technique called Frequency Fitness Assignment (FFA).
- We are figuring out where it really can give us better results.
- But we are not picky. For example,
  1. on the two-dimensional bin packing problem, we found out that a simple local search can actually outperform the complex state-of-the-art metaheuristics and FFA could not improve the performance ... so we published this surprising result and the student graduated with it[60,61], and
  2. on the Traveling Tournament Problem (TTP), we also got surprisingly good results with RLS[58] and SA.

Advertisement

Programming with Python

We have a freely available course book on *Programming with Python* at https://thomasweise.github.io/programmingWithPython, with focus on practical software development using the Python ecosystem of tools[48].

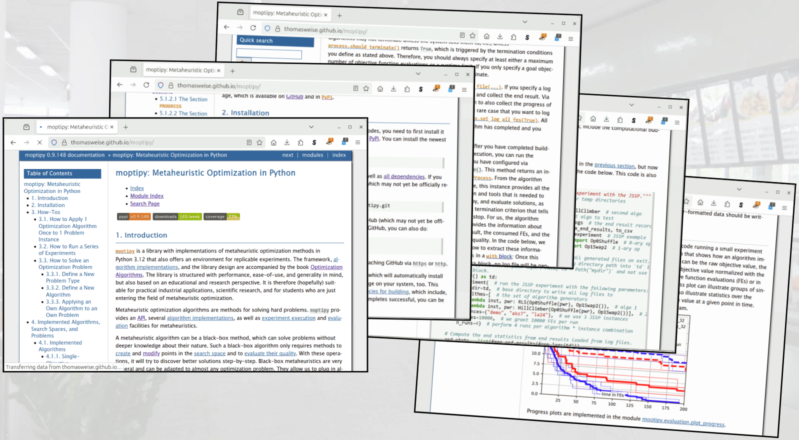


Databases

We have a freely available course book on *Databases* at https://thomasweise.github.io/databases, with actual practical examples using a real database management system (DBMS)[46].

Metaheuristic Optimization in Python: moptipy

We offer moptipy[54] a mature open source Python package for metaheuristic optimization, which implements several algorithms, can run self-documenting experiments in parallel and in a distributed fashion, and offers statistical evaluation tools.



谢谢您们！

Thank you!

Vielen Dank!

References I

[1] David Lee Applegate, Robert E. Bixby, Vašek Chvátal, and William John Cook. *The Traveling Salesman Problem: A Computational Study*. 2nd ed. Vol. 17 of Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 978-0-691-12993-8 (cit. on pp. 16–19, 133).

[2] Paul Gustav Heinrich Bachmann. *Die Analytische Zahlentheorie / Dargestellt von Paul Bachmann*. Vol. Zweiter Theil of Zahlentheorie: Versuch einer Gesamtdarstellung dieser Wissenschaft in ihren Haupttheilen. Leipzig, Sachsen, Germany: B. G. Teubner, 1894. ISBN: 978-1-4181-6963-3. URL: http://gallica.bnf.fr/ark:/12148/bpt6k994750 (visited on 2023-12-13) (cit. on p. 134).

[3] Thomas Bäck, David B. Fogel, and Zbigniew "Zbyszek" Michalewicz, eds. *Handbook of Evolutionary Computation*. Bristol, England, UK: IOP Publishing Ltd and Oxford, Oxfordshire, England, UK: Oxford University Press, 1997. ISBN: 978-0-7503-0392-7 (cit. on p. 131).

[4] Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. "The Job Shop Scheduling Problem: Conventional and New Solution Techniques". *European Journal of Operational Research* 93(1):1–33, Aug. 1996. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(95)00362-2 (cit. on p. 132).

[5] Rainer E. Burkard, Eranda Çela, Panos Miltiades Pardalos, and Leonidas S. Pitsoulis. "The Quadratic Assignment Problem". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1713–1809. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_27 (cit. on p. 132).

[6] Eduardo Carvalho Pinto and Carola Doerr. *Towards a More Practice-Aware Runtime Analysis of Evolutionary Algorithms*. arXiv.org: Computing Research Repository (CoRR) abs/1812.00493. Ithaca, NY, USA: Cornell Universiy Library, Dec. 3, 2018. doi:10.48550/arXiv.1812.00493. URL: https://arxiv.org/abs/1812.00493 (visited on 2025-08-08). arXiv:1812.00493v1 [cs.NE] 3 Dec 2018 (cit. on pp. 38–48, 131).

[7] Vladimír Černý. "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm". *Journal of Optimization Theory and Applications* 45(1):41–51, Jan. 1985. New York, NY, USA: Springer Science+Business Media, LLC. ISSN: 0022-3239. doi:10.1007/BF00940812 (cit. on p. 133).

[8] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. "A Review of Machine Scheduling: Complexity, Algorithms and Approximability". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1493–1641. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_25. See also pages 21–169 in volume 3/3 by Norwell, MA, USA: Kluwer Academic Publishers. (Cit. on pp. 132, 134).

References II

[9] Jiayang Chen (陈嘉阳), Zhize Wu (吴志泽), Sarah Louise Thomson, and Thomas Weise (汤卫思). "Frequency Fitness Assignment: Optimization Without Bias for Good Solution Outperforms Randomized Local Search on the Quadratic Assignment Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 27–37. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888600003837 (cit. on pp. 96, 97, 131, 132).

[10] Stephen Arthur Cook. "The Complexity of Theorem-Proving Procedures". In: *Third Annual ACM Symposium on Theory of Computing (STOC'1971)*. May 3–5, 1971, Shaker Heights, OH, USA. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. New York, NY, USA: Association for Computing Machinery (ACM), 1971, pp. 151–158. ISBN: 978-1-4503-7464-4. doi:10.1145/800157.805047 (cit. on pp. 132, 134).

[11] Ege de Bruin, Sarah Louise Thomson, and Daan van den Berg. "Frequency Fitness Assignment on JSSP: A Critical Review". In: *26th European Conference on Applications of Evolutionary Computation (EvoApplications'2023), Held as Part of EvoStar'2023*. Apr. 12–14, 2023, Brno, Czech Republic. Ed. by João Correia, Stephen Smith, and Raneem Qaddoura. Vol. 13989 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2023, pp. 351–363. ISSN: 0302-9743. ISBN: 978-3-031-30228-2. doi:10.1007/978-3-031-30229-9_23. Independent reproduction of[50] (cit. on pp. 98, 99).

[12] Stefan Droste, Thomas Jansen, and Ingo Wegener. "On the Analysis of the $(1+1)$ Evolutionary Algorithm". *Theoretical Computer Science* 276(1-2):51–81, Apr. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/S0304-3975(01)00182-7 (cit. on pp. 38–48, 84–91, 131).

[13] Kelly Easton, George L. Nemhauser, and Michael A. Trick. "The Traveling Tournament Problem Description and Benchmarks". In: *7th International Conference on Principles and Practice of Constraint Programming (CP'01)*. Nov. 26–Dec. 1, 2001, Paphos, Cyprus. Ed. by Toby Walsh. Vol. 2239 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2001, pp. 580–584. ISSN: 0302-9743. ISBN: 978-3-540-42863-3. doi:10.1007/3-540-45578-7_43 (cit. on p. 133).

[14] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. "Escaping Large Deceptive Basins of Attraction with Heavy-Tailed Mutation Operators". In: *Genetic and Evolutionary Computation Conference (GECCO'2018)*. July 15–19, 2018, Kyoto, Japan. Ed. by Hernán E. Aguirre and Keiki Takadama. New York, NY, USA: Association for Computing Machinery (ACM), 2018, pp. 293–300. ISBN: 978-1-4503-5618-3. doi:10.1145/3205455.3205515 (cit. on pp. 84–91).

References III

[15] Michael T. Goodrich. *A Gentle Introduction to NP-Completeness*. Irvine, CA, USA: University of California, Irvine, Apr. 2022. URL: https://ics.uci.edu/~goodrich/teach/cs165/notes/NPComplete.pdf (visited on 2025-08-01) (cit. on pp. 133, 134).

[16] Gregory Z. Gutin and Abraham P. Punnen, eds. *The Traveling Salesman Problem and its Variations*. Vol. 12 of Combinatorial Optimization (COOP). New York, NY, USA: Springer New York, May 2002. ISSN: 1388-3011. doi:10.1007/b101971 (cit. on pp. 16–19, 133).

[17] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Amsterdam, The Netherlands: Elsevier B.V., 2004. ISBN: 978-1-55860-872-6. doi:10.1016/B978-1-55860-872-6.X5016-1 (cit. on p. 132).

[18] John Hunt. *A Beginners Guide to Python 3 Programming*. 2nd ed. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (cit. on p. 132).

[19] Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. "Monte Carlo Techniques in Code Optimization". In: *15th Annual Workshop on Microprogramming (MICRO 15)*. Oct. 5–7, 1982. Ed. by Joseph Allen Fisher, William J. Tracz, and Bill Hopkins. Palo Alto, CA, USA: Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE) and New York, NY, USA: Association for Computing Machinery (ACM), Oct. 1982, pp. 143–148. doi:10.5555/800036.800944. See[20] (cit. on p. 124).

[20] Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. "Monte Carlo Techniques in Code Optimization". *ACM SIGMICRO Newsletter* 13(4):143–148, Dec. 1982. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 1050-916X. doi:10.1145/1014194.800944. See[19] (cit. on pp. 124, 133).

[21] Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. "Optimization by Simulated Annealing". *Science Magazine* 220(4598):671–680, May 13, 1983. Washington, D.C., USA: American Association for the Advancement of Science (AAAS). ISSN: 0036-8075. doi:10.1126/science.220.4598.671. URL: https://www.researchgate.net/publication/6026283 (visited on 2025-08-08) (cit. on p. 133).

[22] Donald Ervin Knuth. "Big Omicron and Big Omega and Big Theta". *ACM SIGACT News* 8(2):18–24, Apr.–June 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0163-5700. doi:10.1145/1008328.1008329 (cit. on p. 134).

References IV

[23] Donald Ervin Knuth. *Fundamental Algorithms*. 3rd ed. Vol. 1 of The Art of Computer Programming. Reading, MA, USA: Addison-Wesley Professional, 1997. ISBN: 978-0-201-89683-1 (cit. on p. 134).

[24] Tjalling C. Koopmans and Martin Beckmann. "Assignment Problems and the Location of Economic Activities". *Econometrica* 25(1):53–76, 1957. New Haven, CT, USA: The Econometric Society and Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0012-9682. doi:10.2307/1907742 (cit. on p. 132).

[25] John R. Koza. *Genetic Programming – On the Programming of Computers by Means of Natural Selection*. Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, 1993. ISBN: 978-0-262-11170-6 (cit. on p. 132).

[26] Edmund Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. Leipzig, Sachsen, Germany: B. G. Teubner, 1909. ISBN: 978-0-8218-2650-8 (cit. on p. 134).

[27] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. "Sequencing and Scheduling: Algorithms and Complexity". In: *Production Planning and Inventory*. Ed. by Stephen C. Graves, Alexander Hendrik George Rinnooy Kan, and Paul H. Zipkin. Vol. IV of Handbooks of Operations Research and Management Science. Amsterdam, The Netherlands: Elsevier B.V., 1993. Chap. 9, pp. 445–522. ISSN: 0927-0507. ISBN: 978-0-444-87472-6. doi:10.1016/S0927-0507(05)80189-6. URL: http://alexandria.tue.nl/repository/books/339776.pdf (visited on 2023-12-06) (cit. on pp. 132, 134).

[28] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, England, UK: Wiley Interscience, Sept. 1985. ISSN: 0277-2698. ISBN: 978-0-471-90413-7 (cit. on pp. 16–19, 133).

[29] Kent D. Lee and Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (cit. on p. 132).

[30] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, Sarah Louise Thomson, and Thomas Weise (汤卫思). "Addressing the Traveling Salesperson Problem with Frequency Fitness Assignment and Hybrid Algorithms". *Soft Computing* 28(17-18):9495–9508, July 2024. London, England, UK: Springer Nature Limited. ISSN: 1432-7643. doi:10.1007/S00500-024-09718-8 (cit. on pp. 16–19, 94, 95, 132, 133).

References V

[31] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, and Thomas Weise (汤卫思). "Solving the Traveling Salesperson Problem using Frequency Fitness Assignment". In: *IEEE Symposium Series on Computational Intelligence (SSCI'2022)*. Dec. 4–7, 2022, Singapore. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2022. ISBN: 978-1-6654-8769-6. doi:10.1109/SSCI51031.2022.10022296 (cit. on pp. 94, 95, 132).

[32] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Matthias Thürer, Markus Wagner, and Thomas Weise (汤卫思). "Generating Small Instances with Interesting Features for the Traveling Salesperson Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 173–180. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888800003837 (cit. on pp. 94, 95, 132).

[33] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter M. Hahn, and Tania Querido. "A Survey for the Quadratic Assignment Problem". *European Journal of Operational Research* 176(2):657–690, 2007. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2005.09.032 (cit. on p. 132).

[34] Mark Lutz. *Learning Python*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2025. ISBN: 978-1-0981-7130-8 (cit. on p. 132).

[35] Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe, eds. *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0000195000003837.

[36] Heinz Mühlenbein. "How Genetic Algorithms Really Work: Mutation and Hillclimbing". In: *Parallel Problem Solving from Nature 2 (PPSN-II)*. Sept. 28–30, 1992, Brussels, Belgium. Ed. by Reinhard Männer and Bernard Manderick. Amsterdam, The Netherlands: Elsevier B.V., 1992, pp. 15–26. URL: https://www.researchgate.net/publication/284805798 (visited on 2025-08-15) (cit. on pp. 84–91).

[37] Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham, eds. *Handbook of Combinatorial Optimization*. 1st ed. Boston, MA, USA: Springer, 1998. ISBN: 978-1-4613-7987-4.

[38] Martin Pincus. "Letter to the Editor – A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems". *Operations Research* 18(6):1225–1228, Nov.–Dec. 1970. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0030-364X. doi:10.1287/opre.18.6.1225 (cit. on p. 133).

References VI

[39] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. lulu.com, 2008. ISBN: 978-1-4092-0073-4. URL: https://www.researchgate.net/publication/216301261 (visited on 2025-08-18) (cit. on p. 132).

[40] Sanatan Rai and George Vairaktarakis. "NP-Complete Problems and Proof Methodology". In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos Miltiades Pardalos. 2nd ed. Boston, MA, USA: Springer, Sept. 2008, pp. 2675–2682. ISBN: 978-0-387-74758-3. doi:10.1007/978-0-387-74759-0_462 (cit. on p. 134).

[41] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (AIMA)*. 4th ed. Hoboken, NJ, USA: Pearson Education, Inc. ISBN: 978-1-292-40113-3. URL: https://aima.cs.berkeley.edu (visited on 2024-06-27) (cit. on p. 131).

[42] Sartaj Sahni and Teofilo Gonzalez. "NP-complete Approximation Problems". *Journal of the ACM (JACM)* 23(3):555–565, 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0004-5411. doi:10.1145/321958.321975 (cit. on p. 132).

[43] Sarah Louise Thomson, Gabriela Ochoa, Daan van den Berg, Tianyu Liang (梁天宇), and Thomas Weise (汤卫思). "Entropy, Search Trajectories, and Explainability for Frequency Fitness Assignment". In: *Parallel Problem Solving from Nature (PPSN XVIII)*. Vol. 1. Sept. 14–18, 2024, Hagenberg, Mühlkreis, Austria. Ed. by Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tušar, Penousal Machado, and Thomas Bäck. Vol. 15148 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer. ISSN: 0302-9743. ISBN: 978-3-031-70054-5. doi:10.1007/978-3-031-70055-2_23 (cit. on pp. 96, 97, 132).

[44] Clarissa Van Hoyweghen, David Edward Goldberg, and Bart Naudts. "From Twomax To The Ising Model: Easy And Hard Symmetrical Problems". In: *Genetic and Evolutionary Computation Conf. (GECCO'02)*. July 9–13, 2002, New York, NY, USA. Ed. by William B. Langdon, Erick Cantú-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant G. Honavar, Günter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, Alan C. Schultz, Julian F. Miller, Edmund K. Burke, and Natasa Jonoska. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 2002, pp. 626–633. ISBN: 978-1-55860-878-8 (cit. on pp. 84–91).

[45] Kristian Verduin, Sarah Louise Thomson, and Daan van den Berg. "Too Constrained for Genetic Algorithms. Too Hard for Evolutionary Computing. The Traveling Tournament Problem". In: *15th International Joint Conference on Computational Intelligence (IJCCI'23)*. Nov. 13–15, 2023, Rome, Italy. Ed. by Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2023, pp. 246–257. ISSN: 2184-3236. ISBN: 978-989-758-674-3. doi:10.5220/0012192100003595 (cit. on p. 133).

References VII

[46] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2025. URL: https://thomasweise.github.io/databases (visited on 2025-01-05) (cit. on pp. 119, 131).

[47] Thomas Weise (汤卫思). *Global Optimization Algorithms – Theory and Application*. self-published, 2009. URL: https://www.researchgate.net/publication/200622167 (visited on 2025-07-25) (cit. on pp. 4–6, 131, 132).

[48] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (visited on 2025-01-05) (cit. on pp. 118, 132).

[49] Thomas Weise (汤卫思), Raymond Chiong, Jörg Lässig, Ke Tang (唐珂), Shigeyoshi Tsutsui, Wenxiang Chen (陈文祥), Zbigniew "Zbyszek" Michalewicz, and Xin Yao (姚新). "Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem". *IEEE Computational Intelligence Magazine (CIM)* 9(3):40–52, Aug. 2014. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1556-603X. doi:10.1109/MCI.2014.2326101 (cit. on pp. 16–19, 133).

[50] Thomas Weise (汤卫思), Xinlu Li (李新路), Yan Chen (陈岩), and Zhize Wu (吴志泽). "Solving Job Shop Scheduling Problems without using a Bias for Good Solutions". In: *Genetic and Evolutionary Computation Conference, Companion Volume*. July 10–14, 2021, Lille, France. Ed. by Krzysztof Krawiec. New York, NY, USA: Association for Computing Machinery (ACM), 2021, pp. 1459–1466. ISBN: 978-1-4503-8351-6. doi:10.1145/3449726.3463124 (cit. on pp. 98, 99, 123).

[51] Thomas Weise (汤卫思) and Ke Tang (唐珂). "Evolving Distributed Algorithms with Genetic Programming". *IEEE Transactions on Evolutionary Computation* 16(2):242–265, Apr. 2012. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2011.2112666 (cit. on p. 132).

[52] Thomas Weise (汤卫思), Mingxu Wan (万明绪), Ke Tang (唐珂), and Xin Yao (姚新). "Evolving Exact Integer Algorithms with Genetic Programming". In: *IEEE Congress on Evolutionary Computation (CEC'2014)*. July 6–11, 2014, China, Beijing (中国北京市). Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2014, pp. 1816–1823. ISBN: 978-1-4799-1488-3. doi:10.1109/CEC.2014.6900292 (cit. on pp. 100, 101).

References VIII

[53] Thomas Weise (汤卫思), Mingxu Wan (万明绪), Pu Wang (王璞), Ke Tang (唐珂), Alexandre Devert, and Xin Yao (姚新). "Frequency Fitness Assignment". *IEEE Transactions on Evolutionary Computation* 18(2):226–243, Apr. 2014. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2013.2251885 (cit. on pp. 81–83, 92, 93, 100, 101).

[54] Thomas Weise (汤卫思) and Zhize Wu (吴志泽). "Replicable Self-Documenting Experiments with Arbitrary Search Spaces and Algorithms". In: *Conference on Genetic and Evolutionary Computation (GECCO'2023), Companion Volume*. July 15–19, 2023, Lisbon, Portugal. Ed. by Sara Silva and Luís Paquete. New York, NY, USA: Association for Computing Machinery (ACM), 2023, pp. 1891–1899. ISBN: 979-8-4007-0120-7. doi:10.1145/3583133.3596306 (cit. on pp. 120, 132).

[55] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路), and Yan Chen (陈岩). "Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value". *IEEE Transactions on Evolutionary Computation* 25(2):307–319, Apr. 2021. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2020.3032090 (cit. on pp. 81–83, 92, 93, 98, 99).

[56] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路), Yan Chen (陈岩), and Jörg Lässig. "Frequency Fitness Assignment: Optimization without Bias for Good Solutions can be Efficient". *IEEE Transactions on Evolutionary Computation* 27(4):980–992, 2023. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2022.3191698 (cit. on pp. 81–83, 92, 93).

[57] Thomas Weise (汤卫思), Yuezhong Wu (吴越钟), Raymond Chiong, Ke Tang (唐珂), and Jörg Lässig. "Global versus Local Search: The Impact of Population Sizes on Evolutionary Algorithm Performance". *Journal of Global Optimization* 66(3):511–534, Feb. 2016. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1007/s10898-016-0417-5 (cit. on pp. 16–19, 133).

[58] CAO Xiang (曹翔), Zhize Wu (吴志泽), Daan van den Berg, and Thomas Weise (汤卫思). "Randomized Local Search vs. NSGA-II vs. Frequency Fitness Assignment on The Traveling Tournament Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 38–49. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012891500003837 (cit. on pp. 110–116, 132, 133).

References IX

[59] Kinza Yasar and Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., June 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (visited on 2025-01-11) (cit. on p. 131).

[60] Rui Zhao (赵睿), Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, and Thomas Weise (汤卫思). "Randomized Local Search on the 2D Rectangular Bin Packing Problem with Item Rotation". In: *Genetic and Evolutionary Computation Conference (GECCO'2024)*. July 14–18, 2024, Melbourne, VIC, Australia. Ed. by Xiaodong Li and Julia Handl. New York, NY, USA: Association for Computing Machinery (ACM), 2024, pp. 235–238. ISBN: 979-8-4007-0494-9. doi:10.1145/3638530.3654139 (cit. on pp. 20–23, 110–116, 132).

[61] Rui Zhao (赵睿), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, Tianyu Liang (梁天宇), Ming Tan (檀明), and Thomas Weise (汤卫思). "Randomized Local Search for Two-Dimensional Bin Packing and a Negative Result for Frequency Fitness Assignment". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 15–26. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888500003837 (cit. on pp. 20–23, 110–116, 132).

Glossary I

$(1+1)$ EA The $(1+1)$ EA is a local search algorithm that retains the best solution x_c discovered so far during the search[6,12]. In each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then the $(1+1)$ EA uses a unary search operator that flips each bit independently with probability m/n, where usually $m=1$. This operator is the main difference to randomized local search (RLS). The $(1+1)$ EA is a special case of the $(\mu+\lambda)$ evolutionary algorithm ($(\mu+\lambda)$ EA) where $\mu=\lambda=1$.

$(1+1)$ FEA The $(1+1)$ EA with FFA plugged in.

EA An *evolutionary algorithm* is a metaheuristic optimization method that maintains a population of candidate solutions, which undergo selection (where better solutions are chosen with higher probability) and reproduction (where mutation and recombination create a new candidate solution from one or two existing ones, repectively)[3,47].

$(\mu+\lambda)$ EA The $(\mu+\lambda)$ EA is an evolutionary algorithm (EA) where, in each generation, λ offspring solutions are generated from the current population of μ parent solutions. The offspring and parent populations are merged, yielding $\mu+\lambda$ solutions, from which then the best μ solutions are ratained to form the parent population of the next generation. If the search space is the bit strings of length n, then this algorithm usually applies a mutation operator flipping each bit independently with probability $1/n$.

AI Artificial Intelligence, see, e.g.,[41]

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*[46].

DBMS A *database management system* is the software layer located between the user or application and the database (DB). The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB[59].

FFA *Frequency Fitness Assignment* is a algorithm plugin for optimization methods applied to discrete or combinatorial problems with not-too-many different possible objective values. It replaces the objective values in all comparisons with their absolute encounter frequency so far during the search. FFA has successfully been applied to the QAP[9].

Glossary II

GP Genetic Programming[25,39,47,51] is the application of metaheuristic optimization, usually in form of an EAs, to a search space comprised of tree datastructures. These tree datastructures often represent programs or mathematical expressions.

JSSP The *Job Shop Scheduling Problem*[4,27] is one of the most prominent and well-studied scheduling tasks. In a JSSP instance, there are k machines and m jobs. Each job must be processed once by each machine in a job-specific sequence and has a job-specific processing time on each machine. The goal is to find an assignment of jobs to machines that results in an overall shortest makespan, i.e., the schedule which can complete all the jobs in the shortest time. The JSSP is $\mathcal{NP}$-complete[8,27].

MaxSAT The goal of satisfiability problems is to find an assignment for n Boolean variables that make a given Boolean formula $F : \{0,1\}^n \mapsto \{0,1\}$ become true. In the *Maximum Satisfiability (MaxSAT)* problem[17], F is given in conjunctive normal form, i.e., the variables appear either directly or negated in m "or" clauses, which are all combined into one "and." The objective function $f(x)$, subject to minimization, computes the number of clauses which are false under the variable setting x. If $f(x) = 0$, then all clauses of F are true, which solves the problem. The MaxSAT problem is $\mathcal{NP}$-complete[10].

moptipy is the *Metaheuristic Optimization in Python* library[54]. It has been used in several different research works, including[9,30–32,43,58,60,61]. Learn more at https://thomasweise.github.io/moptipy and https://thomasweise.github.io/moptipyapps.

Python The Python programming language[18,29,34,48], i.e., what you will learn about in our book[48]. Learn more at https://python.org.

QAP The *Quadratic Assignment Problem* is an optimization problem where the goal is to assign a set of n facilities to a set of n locations[5,9,24,33]. Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed. For each QAP, a distance matrix D is given, where D_{pq} specifies the distance from location p to location q, as well as a flow matrix F, where F_{ij} is the amount of material flowing from facility i to facility j. The objective function f then rates a permutation x as $f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} D_{x_i x_j} F_{ij}$. The QAP is $\mathcal{NP}$-complete[42].

Glossary III

RLS Randomized local search retains the best solution x_c discovered so far during the search and, in each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then RLS uses a unary search operator that flips exactly one bit. This operator is the main difference to $(1+1)$ EA.

SA Simulated Annealing is a local search that sometimes accepts a worse solution[7,20,21,38]. The probability to do so decreases over time and with the difference in objective values, i.e.,is the lower the worse the new solution is.

TSP In an instance of the *Traveling Salesperson Problem*, also known as *Traveling Salesman Problem*, a set of n cities or locations as well as the distances between them are defined[1,16,28,30,49,57]. The goal is to find the shortest round-trip tour that starts at one city, visits all the other cities one time each, and returns to the origin. The TSP is one of the most well-known $\mathcal{NP}$-hard combinatorial optimization problems[16].

TTP The *Traveling Tournament Problem* (TTP) is the combinatorial optimization problem of both efficiently and fairly organizing a tournament of n teams that play against each other in a pairwise fashion[13,58]. The efficient part boils down to arranging the games such that the total travel length is short, which is somewhat similar to the classical TSP. Initially, each team is at its home location. On each day, a team needs to travel if its scheduled game is not at its present location. On the last day, each team may need to travel back home unless their last game is a home game. The total travel length sums up the lengths of all travels over all teams. The fair part is represented in several constraints, such as doubleRoundRobin, compactness, maxStreak, and noRepeat. The TTP is $\mathcal{NP}$-hard[45].

$i..j$ with $i, j \in \mathbb{Z}$ and $i \leq j$ is the set that contains all integer numbers in the inclusive range from i to j. For example, 5..9 is equivalent to $\{5, 6, 7, 8, 9\}$

$\mathbb{N}_0$ the set of the natural numbers *including* 0, i.e., 0, 1, 2, 3, and so on. It holds that $\mathbb{N}_0 \subset \mathbb{Z}$.

$\mathcal{NP}$ is the class of computational problems that can be solved in polynomial time by a non-deterministic machine and can be verified in polynomial time by a deterministic machine (such as a normal computer)[15].

Glossary IV

$\mathcal{NP}$-complete A decision problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and all problems in $\mathcal{NP}$ are reducible to it in polynomial time[15,40]. A problem is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard and if it is in $\mathcal{NP}$.

$\mathcal{NP}$-hard Algorithms that guarantee to find the correct solutions of $\mathcal{NP}$-hard problems[8,10,27] need a runtime that is exponential in the problem scale in the worst case. A problem is $\mathcal{NP}$-hard if all problems in $\mathcal{NP}$ are reducible to it in polynomial time[15].

$\mathbf{\Omega}(g(x))$ If $f(x) = \mathbf{\Omega}(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $f(x) \geq c * g(x) \geq 0 \forall x \geq x_0$[22,23]. In other words, $\mathbf{\Omega}(g(x))$ describes a lower bound for function growth.

$\mathcal{O}(g(x))$ If $f(x) = \mathcal{O}(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $0 \leq f(x) \leq c * g(x) \forall x \geq x_0$[2,22,23,26]. In other words, $\mathcal{O}(g(x))$ describes an upper bound for function growth.

$\mathbf{\Theta}(g(x))$ If $f(x) = \mathbf{\Theta}(g(x))$, then $f(x) = \mathcal{O}(g(x))$ and $f(x) = \mathbf{\Omega}(g(x))$[22,23]. In other words, $\mathbf{\Theta}(g(x))$ describes an exact order of function growth.

$\mathbb{R}$ the set of the real numbers.

$\mathbb{R}^+$ the set of the positive real numbers, i.e., $\mathbb{R}^+ = \{x \in \mathbb{R} : x > 0\}$.

$\mathbb{Z}$ the set of the integers numbers including positive and negative numbers and 0, i.e., ..., -3, -2, -1, 0, 1, 2, 3, ..., and so on. It holds that $\mathbb{Z} \subset \mathbb{R}$.