合肥大學

HEFEI UNIVERSITY

Frequency Fitness Assignment

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: 2025-08-11

Outline

Introduction

Introduction to Optimization

- Optimization means finding superlatives.

biggest ...

with the least energy...

...best trade-offs between ....

...highest quality ...longest possible duration

most efficient ... most precise ... cheapest ... fastest...

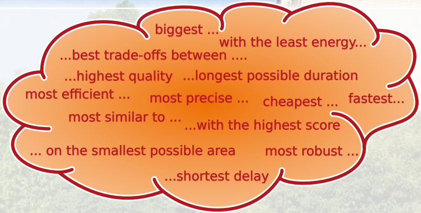
most similar to ... ...with the highest score

... on the smallest possible area most robust ...

...shortest delay

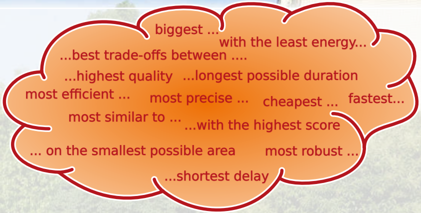
Introduction to Optimization

- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.



Introduction to Optimization

- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.
- Find the shortest route through n cities.

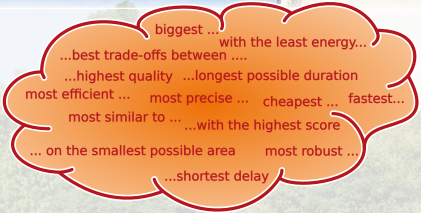


Introduction to Optimization

biggest ... with the least energy...
...best trade-offs between ....
...highest quality ...longest possible duration
most efficient ... most precise ... cheapest ... fastest...
most similar to ... ...with the highest score
... on the smallest possible area most robust ...
...shortest delay

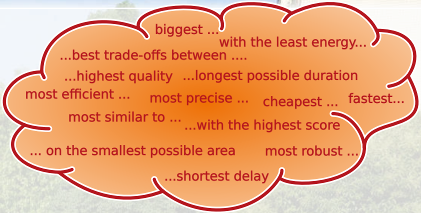
- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.
- Find the shortest route through n cities.
- Set the pricing for these apples such that we can get the largest revenue when selling them.

Introduction to Optimization



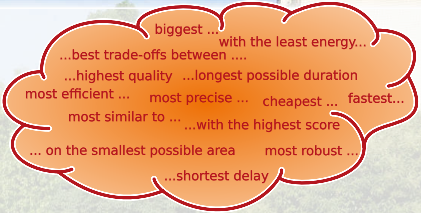
- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.
- Find the shortest route through n cities.
- Set the pricing for these apples such that we can get the largest revenue when selling them.
- Place all these chips on a circuit board so that they occupy the smallest area while we can still properly connect and cool them.

Introduction to Optimization



- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.
- Find the shortest route through n cities.
- Set the pricing for these apples such that we can get the largest revenue when selling them.
- Place all these chips on a circuit board so that they occupy the smallest area while we can still properly connect and cool them.
- Find the cheapest way to transport these goods from Hefei to Wellington.

Introduction to Optimization



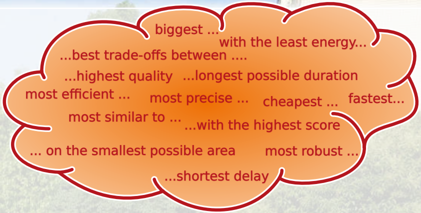
- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.
- Find the shortest route through n cities.
- Set the pricing for these apples such that we can get the largest revenue when selling them.
- Place all these chips on a circuit board so that they occupy the smallest area while we can still properly connect and cool them.
- Find the cheapest way to transport these goods from Hefei to Wellington.
- Design an airplane wing with the least aerodynamic drag.

Introduction to Optimization

biggest ...
with the least energy...
...best trade-offs between ....
...highest quality ...longest possible duration
most efficient ... most precise ... cheapest ... fastest...
most similar to ... ...with the highest score
... on the smallest possible area most robust ...
...shortest delay

- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.
- Find the shortest route through n cities.
- Set the pricing for these apples such that we can get the largest revenue when selling them.
- Place all these chips on a circuit board so that they occupy the smallest area while we can still properly connect and cool them.
- Find the cheapest way to transport these goods from Hefei to Wellington.
- Design an airplane wing with the least aerodynamic drag.
- Find a strategy to manage the power of the nodes in this sensor network so that full coverage is guaranteed for the longest possible duration.

Introduction to Optimization



- Optimization means finding superlatives.
- Find the fastest way to get from Hefei to Beijing.
- Find the shortest route through n cities.
- Set the pricing for these apples such that we can get the largest revenue when selling them.
- Place all these chips on a circuit board so that they occupy the smallest area while we can still properly connect and cool them.
- Find the cheapest way to transport these goods from Hefei to Wellington.
- Design an airplane wing with the least aerodynamic drag.
- Find a strategy to manage the power of the nodes in this sensor network so that full coverage is guaranteed for the longest possible duration.
- And so on.

Views on Optimization

- There are two ways to look at optimization.

Views on 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.

Views on 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: Traveling Salesperson Problem

- In the Traveling Salesperson Problem (TSP)[1,12,19,30], the goal is to find the shortest round-ttrip tour through a set of n cities.

Example: Traveling Salesperson Problem

- In the Traveling Salesperson Problem (TSP)[1,12,19,30], the goal is to find the shortest round-ttrip 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

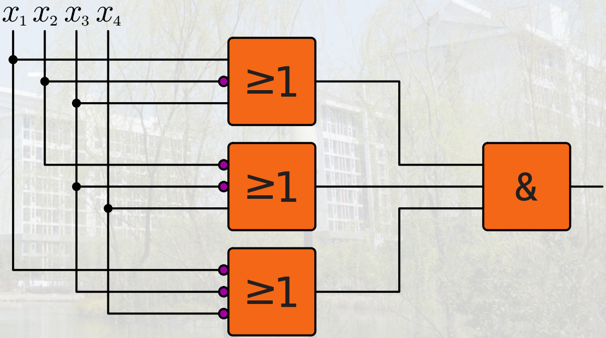
- In the Traveling Salesperson Problem (TSP)[1,12,19,30], the goal is to find the shortest round-ttrip tour through a set of n cities.
- The search space $\mathbb{X}$ thus is the set of all possible round-trip tours through these n cities, usually specified as permutations of the first n natural numbers.
- 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)[1,12,19,30], the goal is to find the shortest round-ttrip 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.

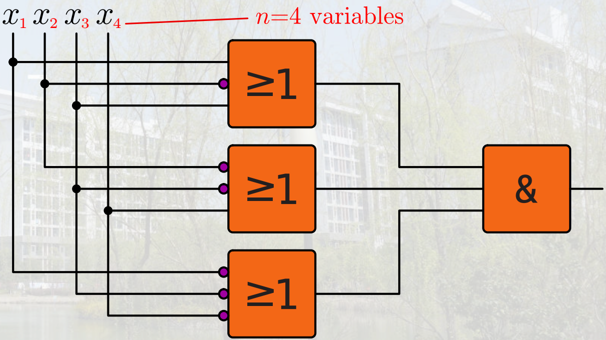
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



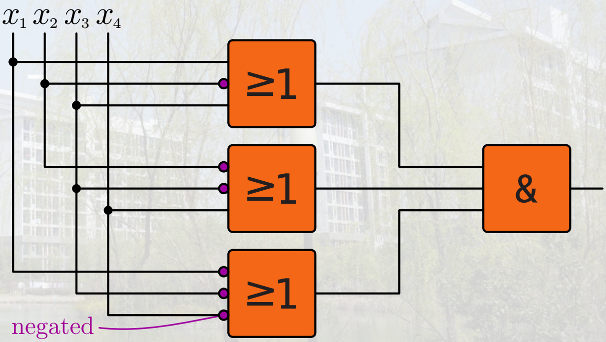
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



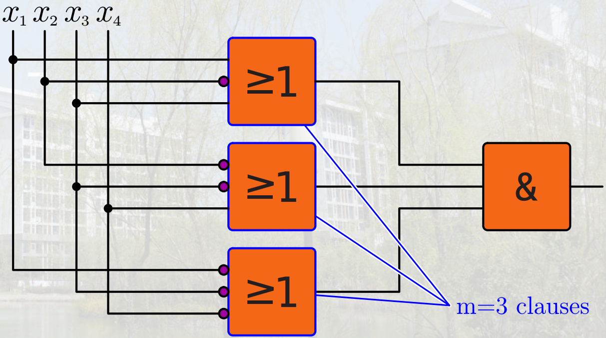
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



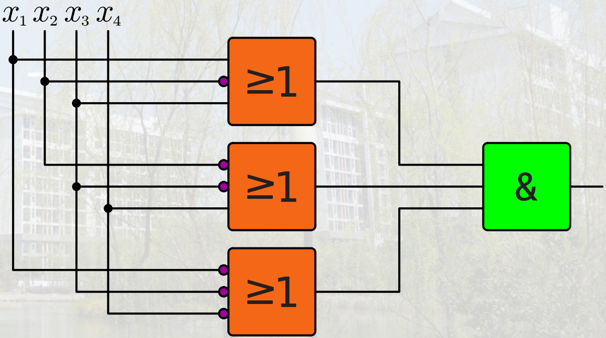
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



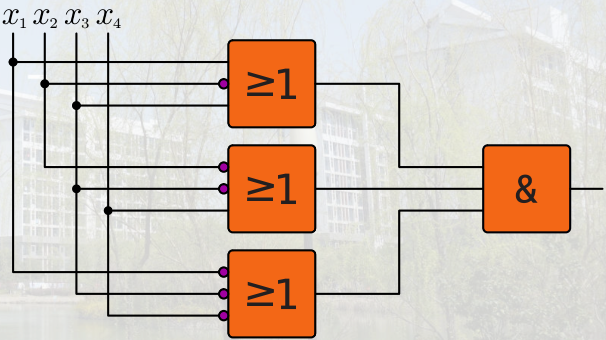
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



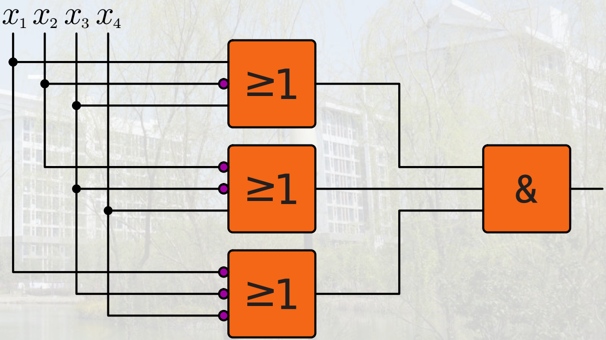
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.
- $\mathbb{X}$ is the set of all possible bit strings of length n.



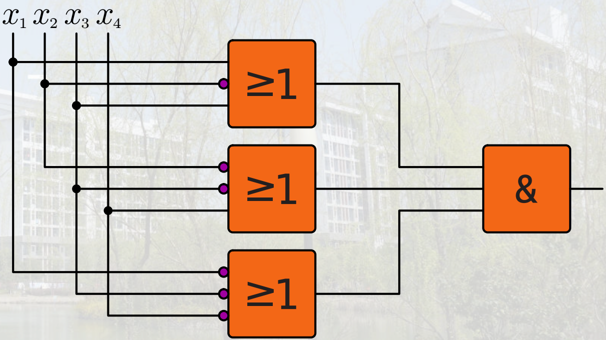
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.
- $\mathbb{X}$ is the set of all possible bit strings of length n.
- The objective function $f : \mathbb{X} \mapsto \mathbb{R}$ is the number of unsatisfied OR-clauses.



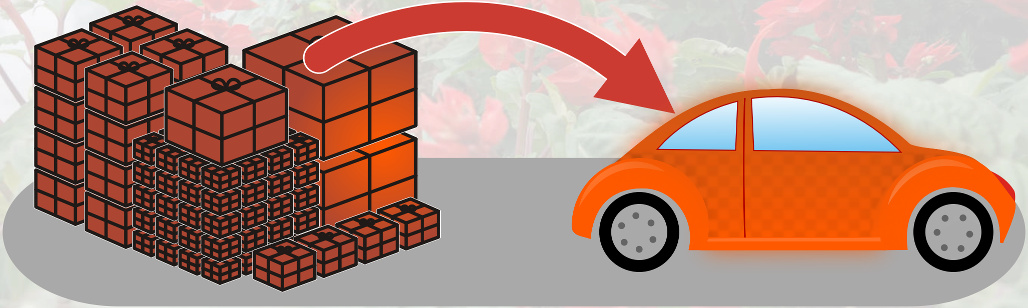
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[7,13] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.
- $\mathbb{X}$ is the set of all possible bit strings of length n.
- The objective function $f : \mathbb{X} \mapsto \mathbb{R}$ is the number of unsatisfied OR-clauses.
- The optimum $x^{\star} \in \mathbb{X}$ has $f(x^{\star}) = 0$, i.e., all clauses satisfied, i.e., $F(x^{\star}) = $ True.



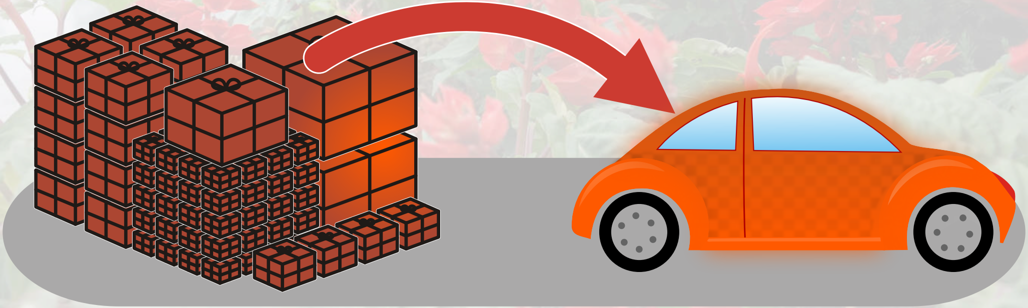
Example: Bin Packing Problem

- 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.



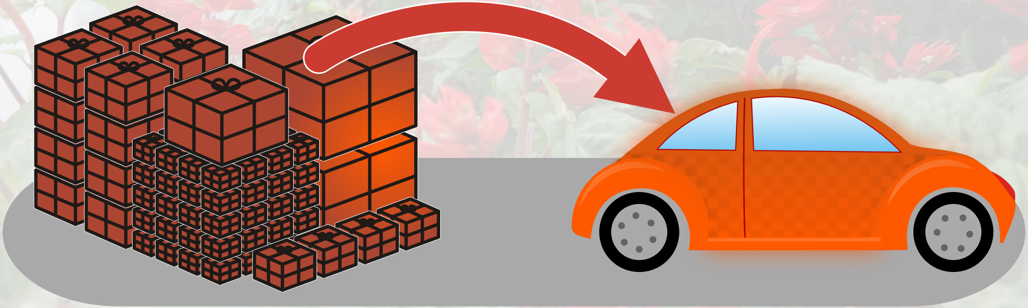
Example: Bin Packing Problem

- 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.
- The $\mathbb{X}$ comprises all possible packing orders of the n objects.



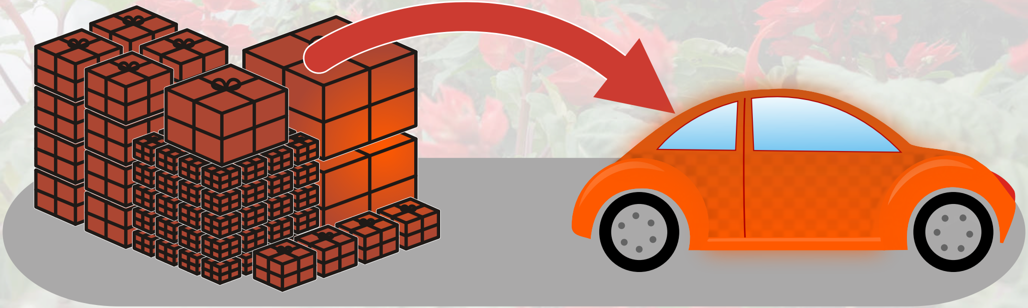
Example: Bin Packing Problem

- 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.
- 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.
- 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).
- 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

Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.

Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.

Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.

Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They start with random solutions.

Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They start with random solutions.
- And then roughly follow this cycle.

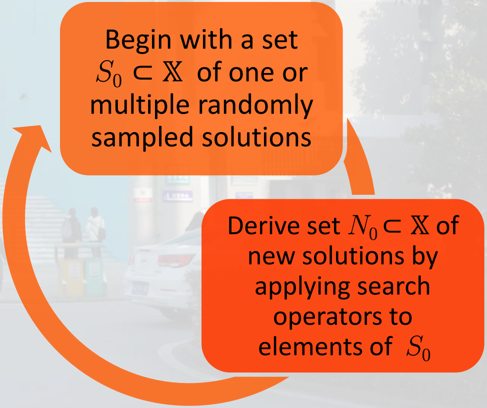
Begin with a set $S_0 \subset \mathbb{X}$ of one or multiple randomly sampled solutions

Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They start with random solutions.
- And then roughly follow this cycle.

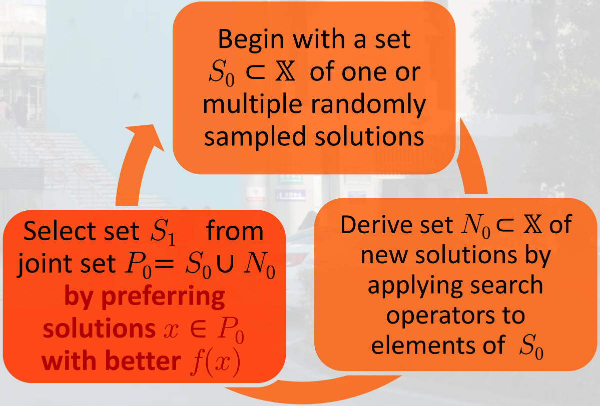
Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They start with random solutions.
- And then roughly follow this cycle.



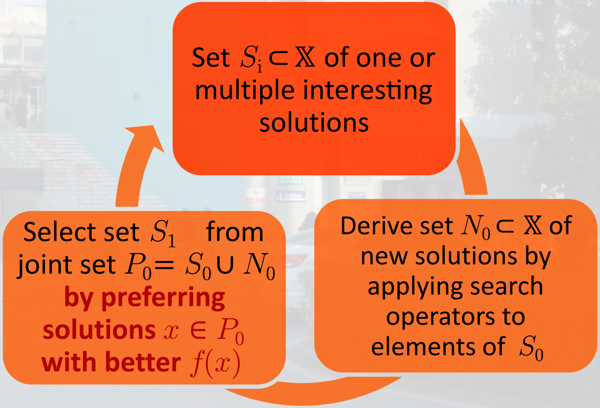
Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They start with random solutions.
- And then roughly follow this cycle.



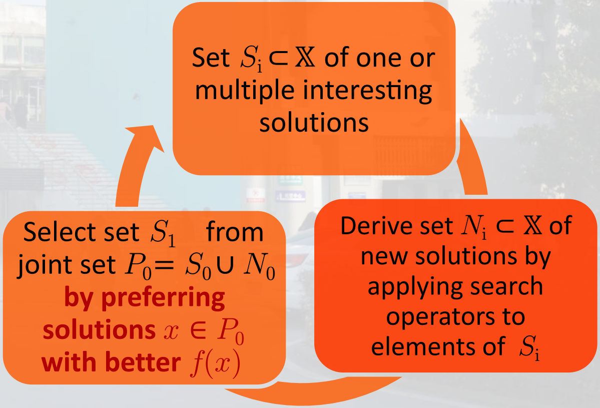
Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They start with random solutions.
- And then roughly follow this cycle.



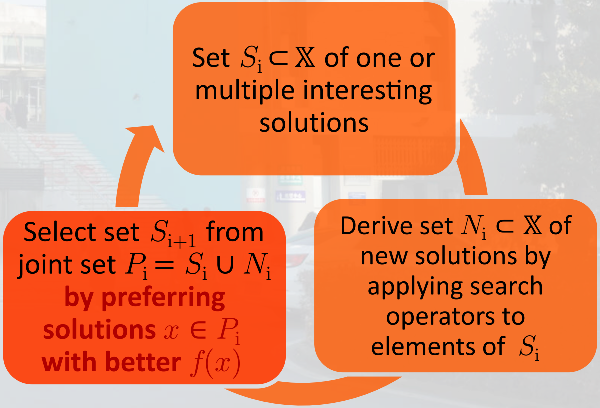
Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They start with random solutions.
- And then roughly follow this cycle.



Metaheuristic Optimization

- Metaheuristics follow the Trial-and-Error idea of iterative improvement.
- They drop the guarantee to find the optimal solution.
- They try to find good solution within a feasible runtime.
- They 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 $move$)[4,9].

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*)[4,9].

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*)[4,9].

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 $move$)[4,9].

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*)[4,9].

```
procedure (1 + 1) EA(f : 𝕏 ↦ ℝ)
    randomly sample x_c from 𝕏;  y_c ← f(x_c);
    while ¬ terminate do
        x_n ← 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 $move$)[4,9].

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

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 $move$)[4,9].
- 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 $move$)[4,9].
- 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** $x_c \leftarrow x_n$;

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 $move$)[4,9].
- They accept the new solution if it is better or equally good compared to the current solution.

```
procedure (1+1) EA(f : 𝕏 ↦ ℝ)
    randomly sample x_c from 𝕏;  y_c ← f(x_c);
    while ¬ terminate do
        x_n ← move(x_c);  y_n ← f(x_n);
        if y_n ≤ y_c then  x_c ← x_n;  y_c ← 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 $move$)[4,9].
- 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** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
  return x_c, y_c

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  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**
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)

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

  while $\neg$ terminate **do**

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

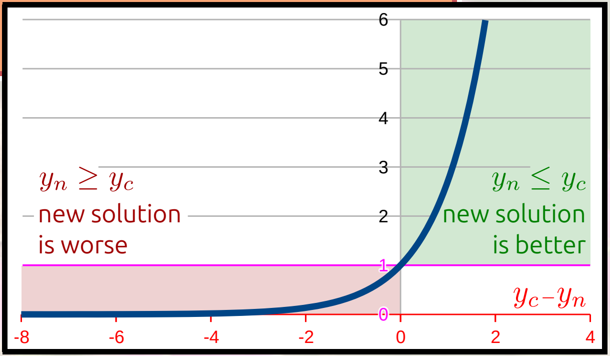
    if $\Re_0^1 < e^{y_c - y_n}$ **then**

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

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
  while $\neg$ terminate **do**
    $x_n \leftarrow$ `move`(x_c); $y_n \leftarrow f(x_n)$;
    if $\Re_0^1 < e^{y_c - y_n}$ **then**



Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  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 $\Re_0^1 < e^{y_c - y_n}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  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 $\Re_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.
- The probability is regulated by temperature schedule with parameter T_0.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
    randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

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

        $T \leftarrow T_0$;
        if $\Re_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
            $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.
- The probability is regulated by temperature schedule with parameter T_0.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while $\neg$ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;

    $T \leftarrow T_0$;
    if $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.
- The probability is regulated by temperature schedule with parameter T_0.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)

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

  $\tau \leftarrow 0$; ▷ τ is iteration counter

  while ¬ terminate **do**

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

    $\tau \leftarrow \tau + 1$;

    $T \leftarrow T_0$;

    if $\Re_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$

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

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.
- The probability is regulated by temperature schedule with parameters T_0 and ϵ.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while $\neg$ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\Re_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.
- The probability is regulated by temperature schedule with parameters T_0 and ϵ.
- It also remembers best-so-far solution x_B and its objective value y_B, because it could get lost.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
  $x_B \leftarrow x_c$; $y_B \leftarrow y_c$; ▷ preserve best!
  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while $\neg$ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\Re_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.
- The probability is regulated by temperature schedule with parameters T_0 and ϵ.
- It also remembers best-so-far solution x_B and its objective value y_B, because it could get lost.

```
procedure SA(f : 𝕏 ↦ ℝ, T₀, ε)
    randomly sample x_c from 𝕏;  y_c ← f(x_c);
    x_B ← x_c;  y_B ← y_c;                          ▷ preserve best!
    τ ← 0;                                          ▷ τ is iteration counter
    while ¬ terminate do
        x_n ← move(x_c);  y_n ← f(x_n);
        τ ← τ + 1;
        T ← T₀(1 − ε)^(τ−1);                        ▷ T decreases over time
        if ℜ₀¹ < e^((y_c − y_n)/T) then             ▷ always true if y_n ≤ y_c
            x_c ← x_n;  y_c ← y_n;
            if y_c < y_B then x_B ← x_c;  y_B ← y_c;
```

Simulated Annealing

- Simulated Annealing (SA)[5,16,17,24] is a local search that accepts also worsening moves, but with a probability that decreases over time AND with the difference in solution quality.
- The probability is regulated by temperature schedule with parameters T_0 and ϵ.
- It also remembers best-so-far solution x_B and its objective value y_B, because it could get lost.

procedure SA($f : \mathbb{X} \mapsto \mathbb{R}$, T_0, ε)
  randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
  $x_\mathrm{B} \leftarrow x_c$; $y_\mathrm{B} \leftarrow y_c$; ▷ preserve best!
  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while $\neg$ terminate **do**
    $x_n \leftarrow \mathtt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\Re_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
      if $y_c < y_\mathrm{B}$ **then** $x_\mathrm{B} \leftarrow x_c$; $y_\mathrm{B} \leftarrow y_c$;
  return x_B, y_B

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary operator.

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary operator.

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

  for $j \in 1 \ldots ps$ **do** ▷ random initial population

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary operator.

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

for $j \in 1 \ldots ps$ **do** ▷ random initial population
randomly sample $S_0[j].x$ from $\mathbb{X}$;

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary operator.

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

for $j \in 1 \dots ps$ **do** ▷ random initial population
  randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary operator.

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

for $j \in 1 \dots ps$ **do** ▷ random initial population
  randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary operator.

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

for $j \in 1 \dots ps$ **do** ▷ random initial population
randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"
for $j \in 1 \dots ps$ **do** ▷ new pop

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary operator.

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

for $j \in 1 \dots ps$ **do** ▷ random initial population
  randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"
  for $j \in 1 \dots ps$ **do** ▷ new pop. via mutation

    $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x)$;

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary and binary operator (with crossover rate cr).

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

    for $j \in 1 \dots ps$ **do** ▷ random initial population
        randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

    for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"
        for $j \in 1 \dots ps$ **do** ▷ new pop. via mutation and crossover
            if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x, S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x)$;
            else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x)$;

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary and binary operator (with crossover rate cr).

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

  for $j \in 1 \dots ps$ **do** ▷ random initial population

    randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

  for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"

    for $j \in 1 \dots ps$ **do** ▷ new pop. via mutation and crossover

      if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x, S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;

      else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;

      $N_i[j].y \leftarrow f(N_i[j].x)$;

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary and binary operator (with crossover rate cr).

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

for $j \in 1 \dots ps$ **do** ▷ random initial population
randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"
for $j \in 1 \dots ps$ **do** ▷ new pop. via mutation and crossover
if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x, S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;
else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;
$N_i[j].y \leftarrow f(N_i[j].x)$;

$S_{i+1} \leftarrow$ *Roulette Wheel:* select ps records from $P_i = S_i \cup N_i$ such that, for each of the ps slots, the probability of $P_i[j]$ to be chosen is **proportional to $P_i[j].y$.**

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary and binary operator (with crossover rate cr).

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**

  $x_B \leftarrow \emptyset$; $y_B \leftarrow -\infty$; ▷ best-so-far solution

  for $j \in 1 \dots ps$ **do** ▷ random initial population

    randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

  for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"

    for $j \in 1 \dots ps$ **do** ▷ new pop. via mutation and crossover

      if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x, S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;

      else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;

      $N_i[j].y \leftarrow f(N_i[j].x)$;

    $S_{i+1} \leftarrow$ *Roulette Wheel:* select ps records from $P_i = S_i \cup N_i$ such that, for each of the ps slots, the probability of $P_i[j]$ to be chosen is **proportional to $P_i[j].y$.**

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary and binary operator (with crossover rate cr).

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^+$, ps, cr) ▷ **for maximization!**
  $x_B \leftarrow \emptyset$; $y_B \leftarrow -\infty$; ▷ best-so-far solution
  for $j \in 1 \dots ps$ **do** ▷ random initial population
    randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;
    if $S_0[j].y > y_B$ **then** $x_B \leftarrow S_0[j].x$; $y_B \leftarrow S_0[j].y$;
  for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"
    for $j \in 1 \dots ps$ **do** ▷ new pop. via mutation and crossover
      if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x, S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x)$;
      else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor \mathfrak{R}_1^{ps} \rfloor].x)$;
      $N_i[j].y \leftarrow f(N_i[j].x)$;

    $S_{i+1} \leftarrow$ *Roulette Wheel:* select ps records from $P_i = S_i \cup N_i$ such that, for each of the ps slots, the probability of $P_i[j]$ to be chosen is **proportional to $P_i[j].y$.**

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary and binary operator (with crossover rate cr).

procedure SGA($f : \mathbb{X} \mapsto \mathbb{R}^{+}$, ps, cr) ▷ **for maximization!**

$x_B \leftarrow \emptyset$; $y_B \leftarrow -\infty$; ▷ best-so-far solution

for $j \in 1 \dots ps$ **do** ▷ random initial population

randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

if $S_0[j].y > y_B$ **then** $x_B \leftarrow S_0[j].x$; $y_B \leftarrow S_0[j].y$;

for $i \in 0 \dots \infty$ **do** ▷ iterate "generations"

for $j \in 1 \dots ps$ **do** ▷ new pop. via mutation and crossover

if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x, S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;

else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor\mathfrak{R}_1^{ps}\rfloor].x)$;

$N_i[j].y \leftarrow f(N_i[j].x)$;

if $N_i[j].y > y_B$ **then** $x_B \leftarrow N_i[j].x$; $y_B \leftarrow N_i[j].y$;

$S_{i+1} \leftarrow$ *Roulette Wheel:* select ps records from $P_i = S_i \cup N_i$ such that, for each of the ps slots, the probability of $P_i[j]$ to be chosen is **proportional to $P_i[j].y$.**

Standard Genetic Algorithm with Roulette Wheel Selection

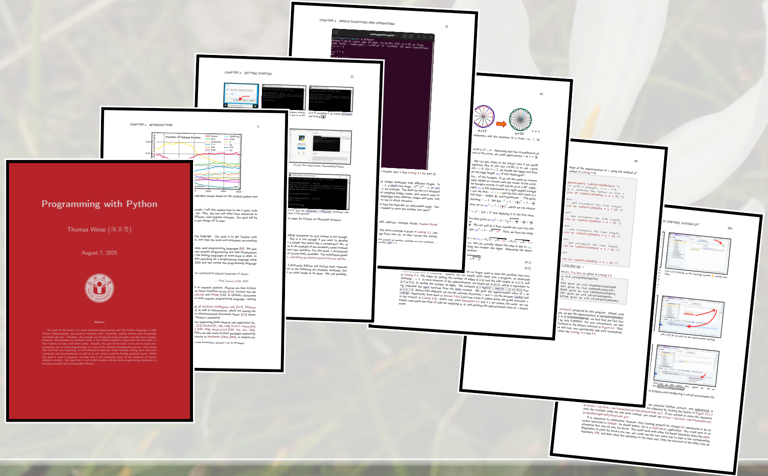
- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[2,8,10,22,23,28].
- It uses a population of size ps as well as a unary and binary operator (with crossover rate cr).

```
procedure SGA(f : 𝕏 ↦ ℝ⁺, ps, cr)                              ▷ for maximization!
    x_B ← ∅;  y_B ← −∞;                                        ▷ best-so-far solution
    for j ∈ 1...ps do                                          ▷ random initial population
        randomly sample S_0[j].x from 𝕏;  S_0[j].y ← f(S_0[j].x);
        if S_0[j].y > y_B then x_B ← S_0[j].x;  y_B ← S_0[j].y;
    for i ∈ 0...∞ do                                           ▷ iterate "generations"
        for j ∈ 1...ps do                    ▷ new pop. via mutation and crossover
            if ℜ_0^1 < cr then N_i[j].x ← binary(S_i[⌊ℜ_1^ps⌋].x, S_i[⌊ℜ_1^ps⌋].x);
            else N_i[j].x ← move(S_i[⌊ℜ_1^ps⌋].x);
            N_i[j].y ← f(N_i[j].x);
            if N_i[j].y > y_B then x_B ← N_i[j].x;  y_B ← N_i[j].y;
        S_{i+1} ← Roulette Wheel: select ps records from P_i = S_i ∪ N_i
                  such that, for each of the ps slots, the probability
                  of P_i[j] to be chosen is proportional to P_i[j].y.
    return x_B, y_B
```

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[29].

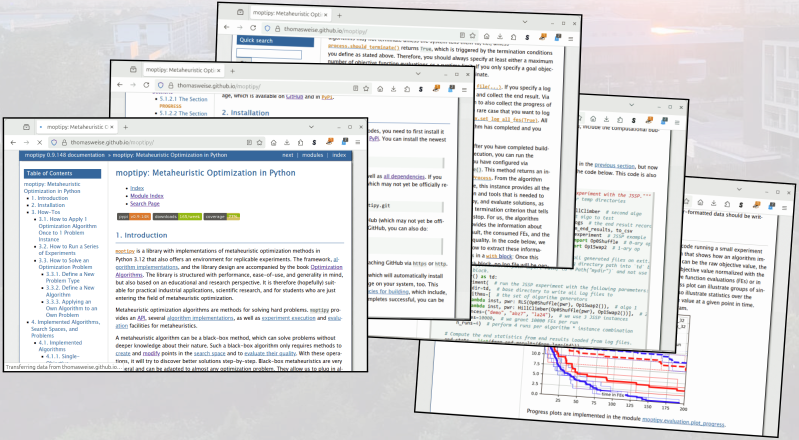


Databases

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)[27].

Metaheuristic Optimization in Python: moptipy

We offer moptipy[31] 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.



谢谢您门！

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. Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 978-0-691-12993-8 (cit. on pp. 16–19, 98).

[2] 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 pp. 71–85, 96, 97).

[3] 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. 96).

[4] 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. 47–57, 96).

[5] 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 pp. 58–70, 97).

[6] 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. 96, 98).

[7] 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. 20–27, 97, 98).

References II

[8] Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*. Vol. 4. Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, 2006. ISBN: 978-0-262-04194-2. URL: https://www.researchgate.net/publication/220740669 (visited on 2025-08-08) (cit. on pp. 71–85, 97).

[9] 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. 47–57, 96).

[10] David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1989. ISBN: 978-0-201-15767-3 (cit. on pp. 71–85, 97).

[11] 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 p. 98).

[12] Gregory Z. Gutin and Abraham P. Punnen, eds. *The Traveling Salesman Problem and its Variations*. Vol. 12. Combinatorial Optimization (COOP). New York, NY, USA: Springer New York, May 2002. ISSN: 1388-3011. doi:10.1007/b101971 (cit. on pp. 16–19, 98).

[13] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. Elsevier B.V., 2004. ISBN: 978-1-55860-872-6 (cit. on pp. 20–27, 97).

[14] 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. 97).

[15] 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[16] (cit. on p. 92).

[16] 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[15] (cit. on pp. 58–70, 92, 97).

References III

[17] 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 pp. 58–70, 97).

[18] 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. 96, 98).

[19] 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, 98).

[20] 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. 97).

[21] Mark Lutz. *Learning Python*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2025. ISBN: 978-1-0981-7130-8 (cit. on p. 97).

[22] Zbigniew "Zbyszek" Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1996. ISBN: 978-3-540-58090-4. doi:10.1007/978-3-662-03315-9 (cit. on pp. 71–85, 97).

[23] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, Feb. 1998. ISBN: 978-0-262-13316-6. URL: http://boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf (visited on 2025-08-08) (cit. on pp. 71–85, 97).

[24] 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 pp. 58–70, 97).

References IV

[25] 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. 98).

[26] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, England, UK: Cambridge University Press & Assessment, July 2014. ISBN: 978-1-107-05713-5. URL: http://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning (visited on 2024-06-27) (cit. on p. 97).

[27] 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. 88, 96).

[28] 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. 71–85, 96, 97).

[29] 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. 87, 97).

[30] 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, 98).

[31] 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. 89, 97).

References V

[32] L. Darrell Whitley. “The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best”. In: *3rd International Conference on Genetic Algorithms (ICGA’1989)*. June 1989, Fairfax, VA, USA: George Mason University. Ed. by J. David Schaffer. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, pp. 116–123. ISBN: 978-1-55860-066-9. URL: https://www.researchgate.net/publication/2527551 (visited on 2025-08-08) (cit. on p. 97).

[33] 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. 96).

Glossary I

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)[2,28].

$(\mu+\lambda)$ 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 $1/n$.

$(1+1)$ EA The $(1+1)$ EA is a local search algorithm that retains the best solution x_c discovered so far during the search[4,9]. 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 $\mu=\lambda=1$.

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*[27].

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[33].

JSSP The *Job Shop Scheduling Problem*[3,18] 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[6,18].

Glossary II

MaxSAT The goal of satisfiablity 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[13], F is given in conjunctive normal form, i.e., the variables appear as literals 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[7].

ML Machine Learning, see, e.g.,[26]

moptipy is the *Metaheuristic Optimization in Python* library[31]. Learn more at https://thomasweise.github.io/moptipy.

Python The Python programming language[14,20,21,29], i.e., what you will learn about in our book[29]. Learn more at https://python.org.

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[5,16,17,24]. 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.

SGA The Standard Genetic Algorithm[2,8,10,22,23,28] was the first population EA. It maintains a population of solutions and applies mutation and crossover to generate offspring solutions. It uses fitness proportionate selection to choose which solutions should "survive" into the next generation, which today is considered a very bad design choice, see, e.g.,[32].

Glossary III

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[1,12,19,30]. 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.

$\mathcal{NP}$ $\mathcal{NP}$ 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)[11].

$\mathcal{NP}$-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[11,25]. 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[6,7,18] 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[11].

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