合肥大學
HEFEI UNIVERSITY

Frequency Fitness Assignment
频率适应度分配

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

人工智能与大数据学院
合肥大学
中国安徽省合肥市

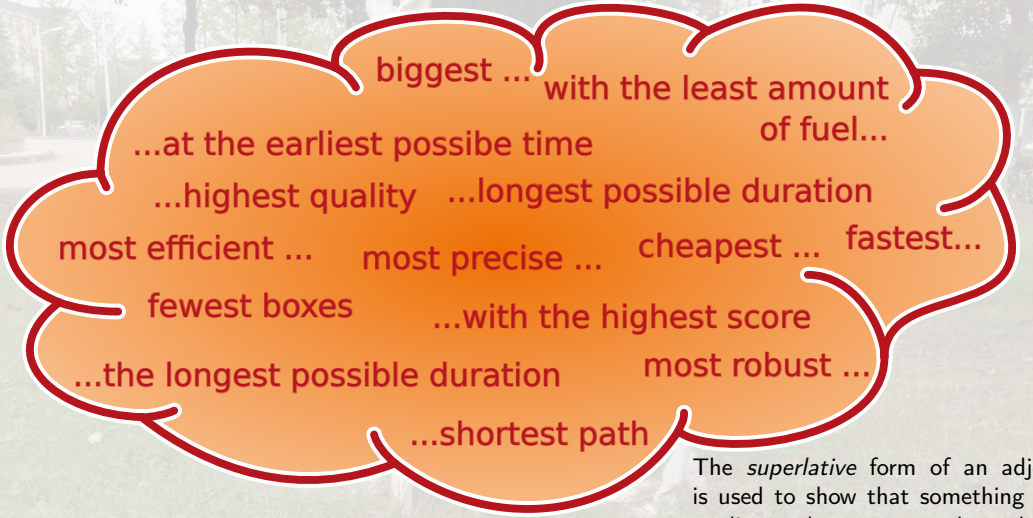
Version: 2026-02-20

Outline

Introduction

Introduction to Optimization

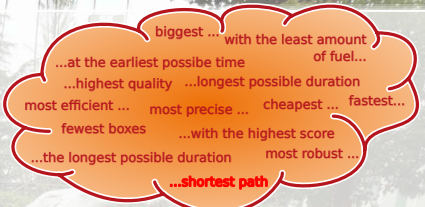
- Optimization means finding superlatives.



The *superlative* form of an adjective is used to show that something has a quality to the greatest or least degree.

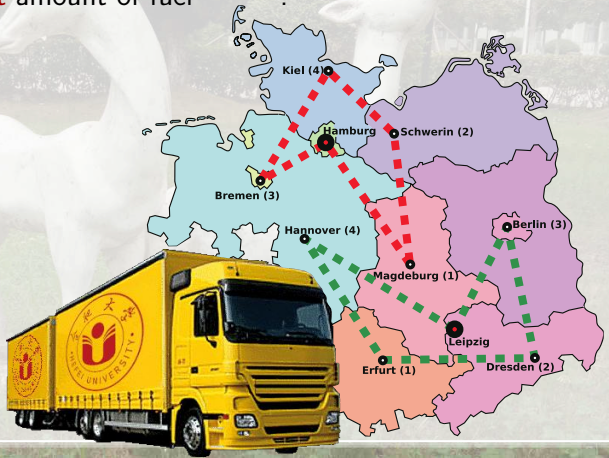
Introduction to Optimization

- Optimization means finding superlatives.
- Find the shortest path from start to goal[26,55].



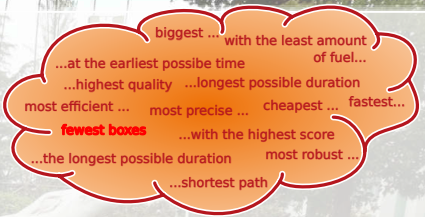
Introduction to Optimization

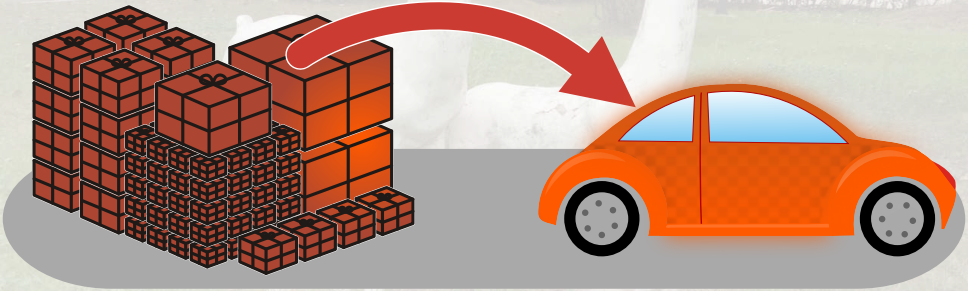
- Optimization means finding superlatives.
- Find the shortest path from start to goal[26,55].
- Pick up and deliver packages from different places to customers using the least amount of fuel[57,74–76].



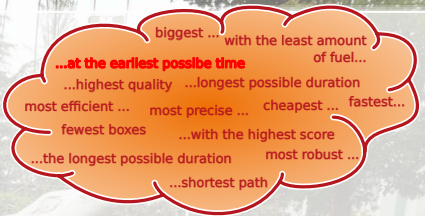
Introduction to Optimization

- Optimization means finding superlatives.
- Find the shortest path from start to goal[26,55].
- Pick up and deliver packages from different places to customers using the least amount of fuel[57,74–76].
- Pack a set of things into the fewest boxes[88,89].

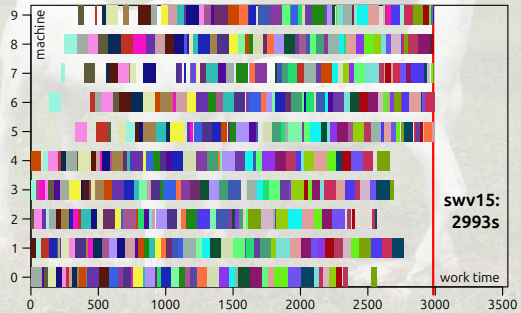




Introduction to Optimization

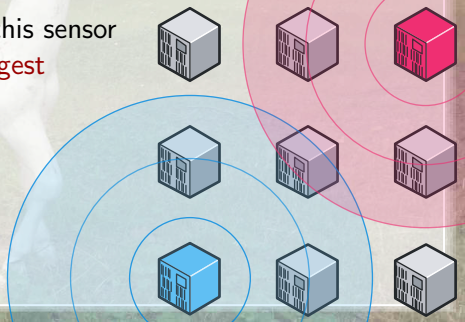


- Optimization means finding superlatives.
- Find the shortest path from start to goal[26,55].
- Pick up and deliver packages from different places to customers using the least amount of fuel[57,74–76].
- Pack a set of things into the fewest boxes[88,89].
- Assign tasks to machines such that we can finish our work the at earliest possible time.

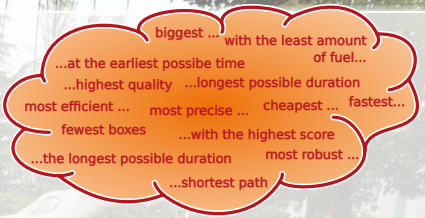


Introduction to Optimization

- Optimization means finding superlatives.
- Find the shortest path from start to goal[26,55].
- Pick up and deliver packages from different places to customers using the least amount of fuel[57,74–76].
- Pack a set of things into the fewest boxes[88,89].
- Assign tasks to machines such that we can finish our work the at earliest possible time.
- 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 shortest path from start to goal[26,55].
- Pick up and deliver packages from different places to customers using the least amount of fuel[57,74–76].
- Pack a set of things into the fewest boxes[88,89].
- Assign tasks to machines such that we can finish our work the at earliest possible time.
- 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 $f\colon \mathbb{X} \mapsto \mathbb{R}$ takes on the smallest possible value.

Example: Traveling Salesperson Problem

- In the Traveling Salesperson Problem (TSP)[3,24,40,42,71,83], 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)[3,24,40,42,71,83], 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

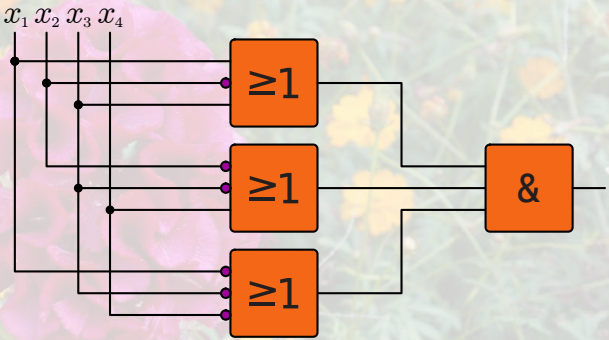
- In the Traveling Salesperson Problem (TSP)[3,24,40,42,71,83], the goal is to find the shortest round-trip tour through a set of n cities.
- The search space $\mathbb{X}$ thus is the set of all possible round-trip tours through these n cities, usually specified as permutations of the first n natural 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)[3,24,40,42,71,83], 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.

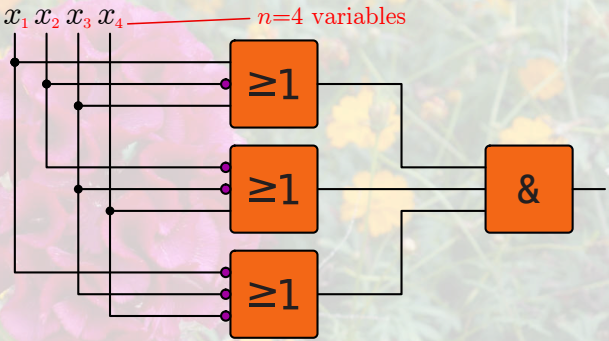
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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.



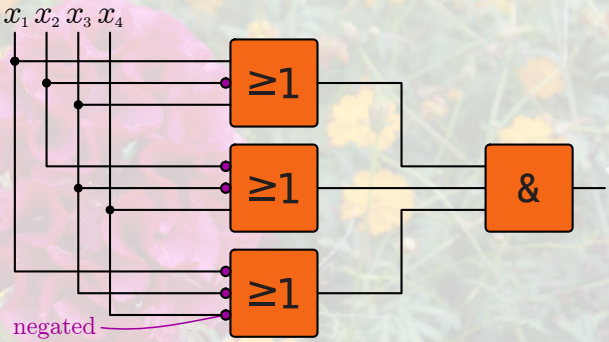
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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.



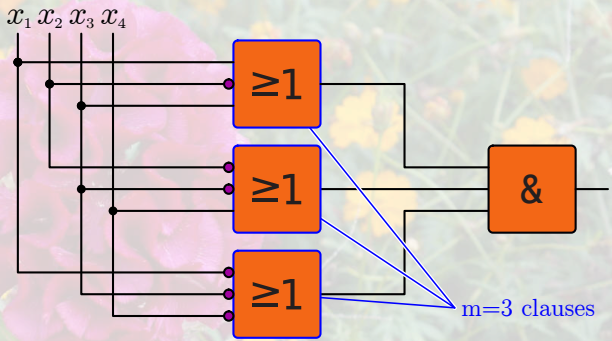
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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.



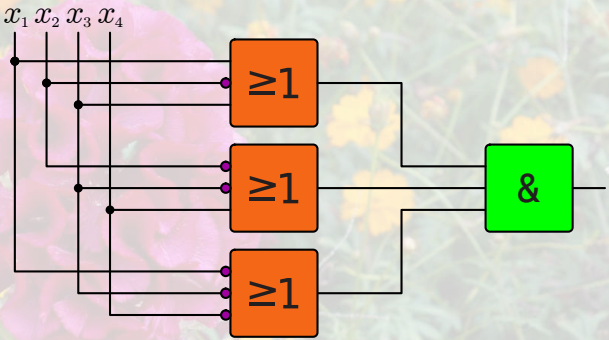
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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.



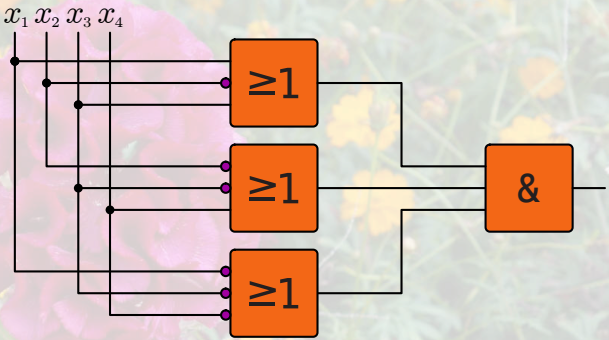
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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.



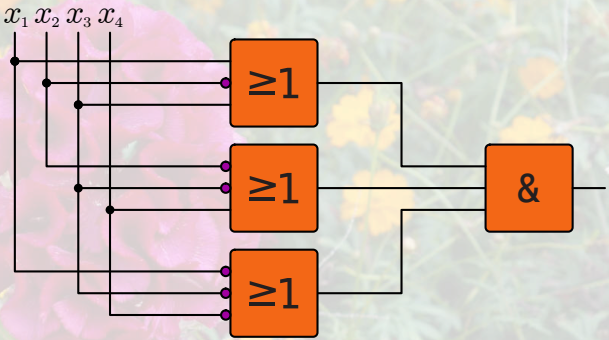
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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.



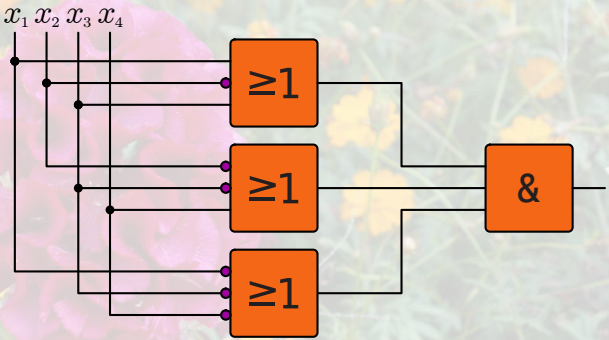
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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.



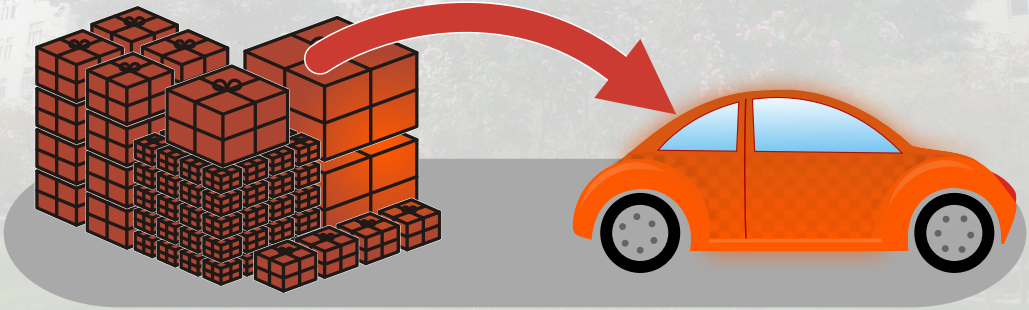
Example: Maximum Satisfiability Problem

- The goal of the Maximum Satisfiability (MaxSAT)[13,28] 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., $F(x^\star) = \text{True}$.



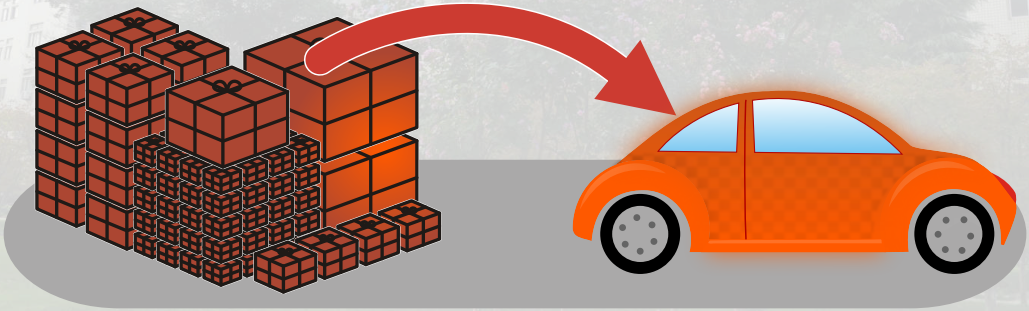
Example: Bin Packing Problem

- 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[88,89].



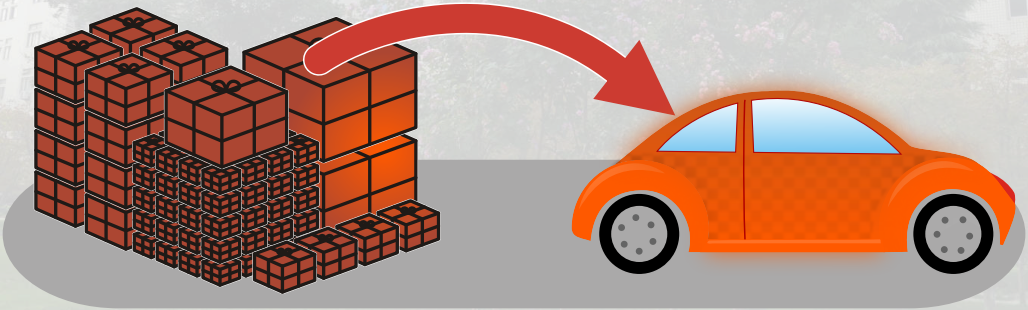
Example: Bin Packing Problem

- 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[88,89].
- The $\mathbb{X}$ comprises all possible packing orders of the n objects.



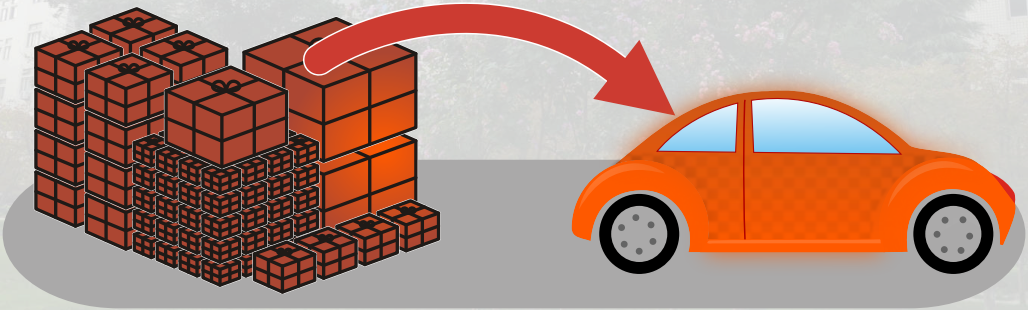
Example: Bin Packing Problem

- 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[88,89].
- 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[88,89].
- 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

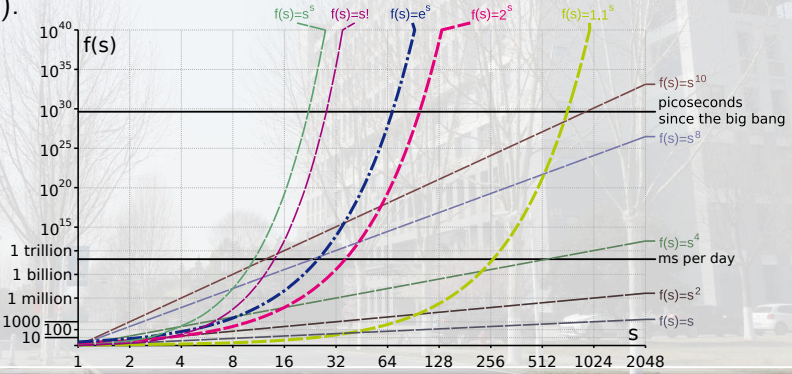
- Finding the globally optimal solution $x^\star$ from the set of all possible solutions $\mathbb{X}$ is often an $\mathcal{NP}$-hard problem.

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 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

- 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.

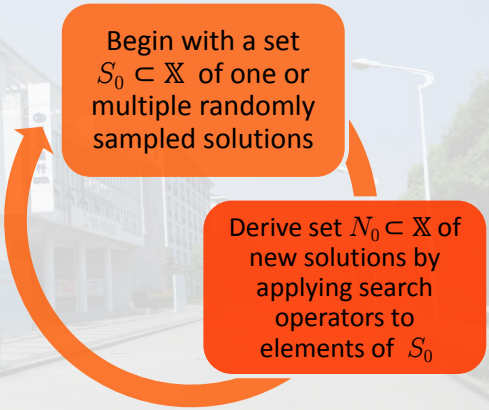
Begin with a set $S_0 \subset \mathbb{X}$ of one or multiple randomly sampled solutions

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.

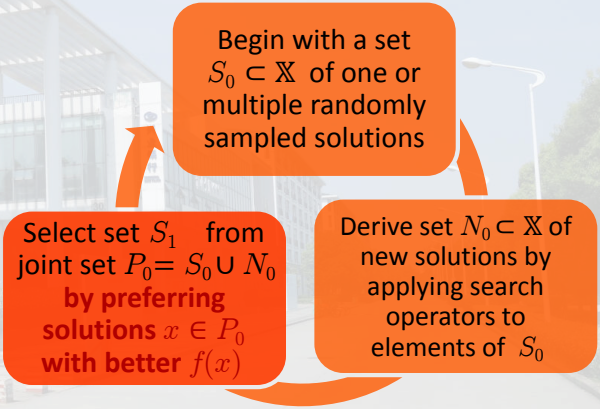
Metaheuristic Optimization

- 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.



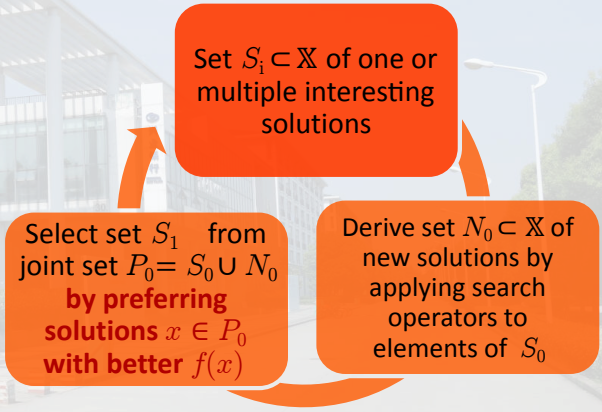
Metaheuristic Optimization

- 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.



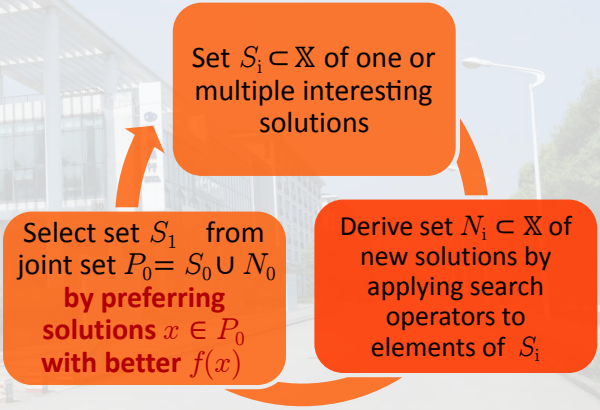
Metaheuristic Optimization

- 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.



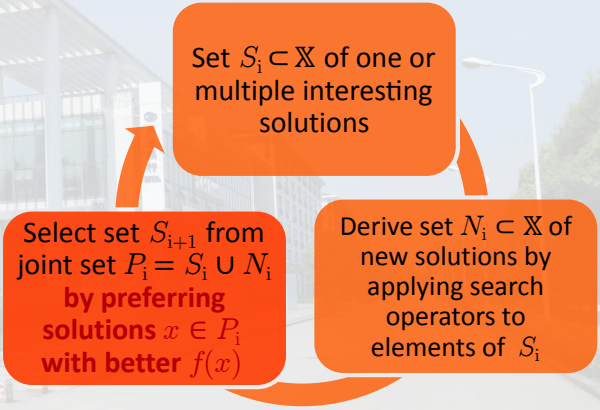
Metaheuristic Optimization

- 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 $move$)[9,17].

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 $move$)[9,17].

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 $move$)[9,17].

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 $move$)[9,17].

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 $move$)[9,17].

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 $move$)[9,17].

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 $move$)[9,17].
- 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 $move$)[9,17].
- 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 $move$)[9,17].
- 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** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

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 $move$)[9,17].
- 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

Simulated Annealing

- Simulated Annealing (SA)[33] 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)[33] 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)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.

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 $\mathfrak{R}_0^1 < e^{y_c - y_n}$ **then**
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

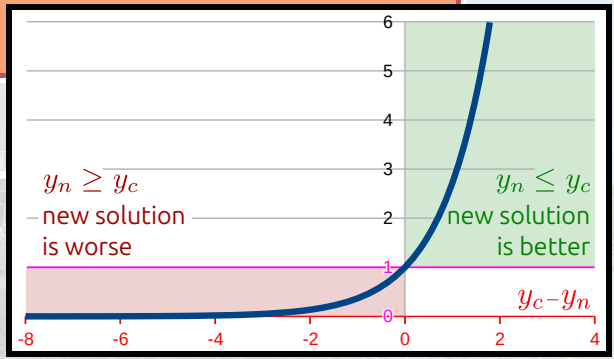
Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.

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 $\mathfrak{R}_0^1 < e^{y_c - y_n}$ **then**



Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.

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

Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.

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

Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P 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 $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P 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**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;

    $T \leftarrow T_0$;
    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)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P 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 $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P 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 ¬ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P 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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
  $x_B \leftarrow x_c$; $y_B \leftarrow y_c$; ▷ preserve best!
  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while ¬ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

- SA also remembers best-so-far solution x_B and its objective value y_B, because it could get lost.

Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P 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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
  $x_B \leftarrow x_c$; $y_B \leftarrow y_c$; ▷ preserve best!
  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while ¬ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
      if $y_c < y_B$ **then** $x_B \leftarrow x_c$; $y_B \leftarrow y_c$;

- SA also remembers best-so-far solution x_B and its objective value y_B, because it could get lost.

Simulated Annealing

- Simulated Annealing (SA)[33] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P 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 $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
  $x_B \leftarrow x_c$; $y_B \leftarrow y_c$; ▷ preserve best!
  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while ¬ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
      if $y_c < y_B$ **then** $x_B \leftarrow x_c$; $y_B \leftarrow y_c$;
  return x_B, y_B

- SA also remembers best-so-far solution x_B and its objective value y_B, because it could get lost.

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[5,15,22,48,49,69].

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[5,15,22,48,49,69].
- 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[5,15,22,48,49,69].
- 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[5,15,22,48,49,69].
- 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[5,15,22,48,49,69].
- 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 $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[5,15,22,48,49,69].
- 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 $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

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

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[5,15,22,48,49,69].
- 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 $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

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

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[5,15,22,48,49,69].
- 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 $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

for $i \in 0 \ldots \infty$ **do** ▷ iterate "generations"
for $j \in 1 \ldots ps$ **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[5,15,22,48,49,69].
- 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 $j \in 1 \ldots ps$ **do** ▷ random initial population
randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

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

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[5,15,22,48,49,69].
- 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 $j \in 1 \ldots ps$ **do** ▷ random initial population
randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

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

Standard Genetic Algorithm with Roulette Wheel Selection

- The Standard Genetic Algorithm (SGA) with Fitness Proportionate Selection (Roulette Wheel) is for maximization[5,15,22,48,49,69].
- 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 $j \in 1 \ldots ps$ **do** ▷ random initial population
randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

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

$S_{i+1} \leftarrow$ *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[5,15,22,48,49,69].
- 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 $j \in 1 \ldots ps$ **do** ▷ random initial population
    randomly sample $S_0[j].x$ from $\mathbb{X}$; $S_0[j].y \leftarrow f(S_0[j].x)$;

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

    $S_{i+1} \leftarrow$ *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[5,15,22,48,49,69].
- 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 $j \in 1 \ldots ps$ **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 $i \in 0 \ldots \infty$ **do** ▷ iterate "generations"
    for $j \in 1 \ldots ps$ **do** ▷ new pop. via mutation and crossover
      if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x, S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x)$;
      else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x)$;
      $N_i[j].y \leftarrow f(N_i[j].x)$;

    $S_{i+1} \leftarrow$ *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[5,15,22,48,49,69].
- 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 $j \in 1 \ldots ps$ **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 $i \in 0 \ldots \infty$ **do** ▷ iterate "generations"
    for $j \in 1 \ldots ps$ **do** ▷ new pop. via mutation and crossover
      if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x, S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x)$;
      else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x)$;
      $N_i[j].y \leftarrow f(N_i[j].x)$;
      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[5,15,22,48,49,69].
- 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 $j \in 1 \ldots ps$ **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 $i \in 0 \ldots \infty$ **do** ▷ iterate "generations"
    for $j \in 1 \ldots ps$ **do** ▷ new pop. via mutation and crossover
      if $\mathfrak{R}_0^1 < cr$ **then** $N_i[j].x \leftarrow \texttt{binary}(S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x, S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x)$;
      else $N_i[j].x \leftarrow \texttt{move}(S_i[\lfloor \mathfrak{R}_i^{ps} \rfloor].x)$;
      $N_i[j].y \leftarrow f(N_i[j].x)$;
      if $N_i[j].y > y_B$ **then** $x_B \leftarrow N_i[j].x$; $y_B \leftarrow N_i[j].y$;
    $S_{i+1} \leftarrow$ *Roulette Wheel:* select ps records from $P_i = S_i \cup N_i$ such that, for each of the ps slots, the probability of $P_i[j]$ to be chosen is **proportional to $P_i[j].y$.**
  return x_B, y_B

Metaheuristic Optimization

- What we have learned is that different metaheuristics realize the trial-and-error scheme differently.

Metaheuristic Optimization

- What we have learned is that different metaheuristics realize the trial-and-error scheme differently.
- During their selection steps, they all prefer better solutions over worse ones.

Metaheuristic Optimization

- What we have learned is that different metaheuristics realize the trial-and-error scheme differently.
- During their selection steps, they all prefer better solutions over worse ones.
- If they would always and only accept the better solutions, they could get trapped in local optima.

Metaheuristic Optimization

- What we have learned is that different metaheuristics realize the trial-and-error scheme differently.
- During their selection steps, they all prefer better solutions over worse ones.
- If they would always and only accept the better solutions, they could get trapped in local optima.
- So some of them sometimes accept worse solutions, but the probability to choose a better solution is always higher in average.

Metaheuristic Optimization

- What we have learned is that different metaheuristics realize the trial-and-error scheme differently.
- During their selection steps, they all prefer better solutions over worse ones.
- If they would always and only accept the better solutions, they could get trapped in local optima.
- So some of them sometimes accept worse solutions, but the probability to choose a better solution is always higher in average.
- This is the most fundamental concept of metaheuristic optimization:

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

Metaheuristic Optimization

- What we have learned is that different metaheuristics realize the trial-and-error scheme differently.
- During their selection steps, they all prefer better solutions over worse ones.
- If they would always and only accept the better solutions, they could get trapped in local optima.
- So some of them sometimes accept worse solutions, but the probability to choose a better solution is always higher in average.
- This is the most fundamental concept of metaheuristic optimization:

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

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

Invariance Properties

Invariance Properties

- Research in optimization, Machine Learning (ML), and Artificial Intelligence (AI) often use simple problems to test and benchmark algorithms.

Invariance Properties

- Research in optimization, Machine Learning (ML), and Artificial Intelligence (AI) often use simple problems to test and benchmark algorithms.
- These allow for many experiments in a short time.

Invariance Properties

- Research in optimization, Machine Learning (ML), and Artificial Intelligence (AI) often use simple problems to test and benchmark algorithms.
- These allow for many experiments in a short time.
- We often know the optimal solutions or at lower bounds for the optimal objective values.

Invariance Properties

- Research in optimization, Machine Learning (ML), and Artificial Intelligence (AI) often use simple problems to test and benchmark algorithms.
- These allow for many experiments in a short time.
- We often know the optimal solutions or at lower bounds for the optimal objective values.
- We can understand the results well.

Invariance Properties

- Research in optimization, Machine Learning (ML), and Artificial Intelligence (AI) often use simple problems to test and benchmark algorithms.
- These allow for many experiments in a short time.
- We often know the optimal solutions or at lower bounds for the optimal objective values.
- We can understand the results well.
- What we want is that algorithms perform similar to our benchmarking results also on actual, real-world problems.

Invariance Properties

- Research in optimization, Machine Learning (ML), and Artificial Intelligence (AI) often use simple problems to test and benchmark algorithms.
- These allow for many experiments in a short time.
- We often know the optimal solutions or at lower bounds for the optimal objective values.
- We can understand the results well.
- What we want is that algorithms perform similar to our benchmarking results also on actual, real-world problems.
- We want invariance properties.[25,32,52]

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c.

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c and we look at the probability P to accept x_n.

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c and we look at the probability P to accept x_n.
- For the $(1+1)$ EA, $P = 1$ if $f(x_n) \leq f(x_c)$ and 0 otherwise.

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c and we look at the probability P to accept x_n.
- For the $(1+1)$ EA, $P = 1$ if $f(x_n) \leq f(x_c)$ and 0 otherwise.
- In the SA, P depends on the difference $\Delta = f(x_c) - f(x_n)$ and is 1 if x_n is better than x_c.

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c and we look at the probability P to accept x_n.
- For the $(1+1)$ EA, $P = 1$ if $f(x_n) \leq f(x_c)$ and 0 otherwise.
- In the SA, P depends on the difference $\Delta = f(x_c) - f(x_n)$ and is 1 if x_n is better than x_c.
- In an SGA for maximization, P is $f(x_n)/(f(x_c) + f(x_n))$.

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c and we look at the probability P to accept x_n.
- For the $(1+1)$ EA, $P = 1$ if $f(x_n) \leq f(x_c)$ and 0 otherwise.
- In the SA, P depends on the difference $\Delta = f(x_c) - f(x_n)$ and is 1 if x_n is better than x_c.
- In an SGA for maximization, P is $f(x_n)/(f(x_c) + f(x_n))$. If we wanted to do minimization, maybe we could use $1 - f(x_n)/(f(x_c) + f(x_n))$ instead.

Acceptance Criteria

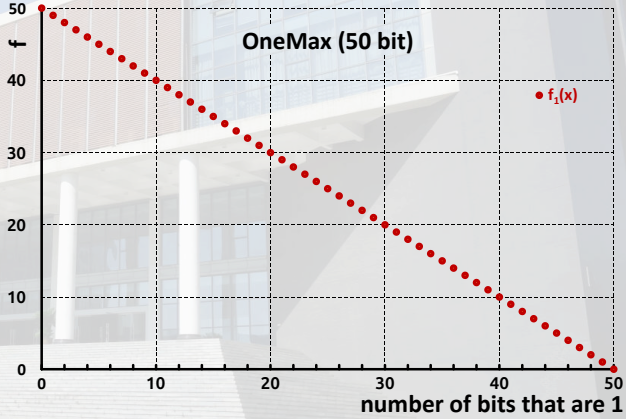
- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c and we look at the probability P to accept x_n.
- For the $(1+1)$ EA, $P = 1$ if $f(x_n) \leq f(x_c)$ and 0 otherwise.
- In the SA, P depends on the difference $\Delta = f(x_c) - f(x_n)$ and is 1 if x_n is better than x_c.
- In an SGA for maximization, P is $f(x_n)/(f(x_c) + f(x_n))$. If we wanted to do minimization, maybe we could use $1 - f(x_n)/(f(x_c) + f(x_n))$ instead.
- $(1+1)$ EA and SA always accept x_n if it is better than x_c, while in the SGA, a better x_n at least has a $P > 0.5$.

Acceptance Criteria

- We have now three example algorithms: $(1+1)$ EA, SA, and SGA.
- Assume that they all work on a population of two solutions, where the new solution be x_n and the old/parent solution be x_c and we look at the probability P to accept x_n.
- For the $(1+1)$ EA, $P = 1$ if $f(x_n) \leq f(x_c)$ and 0 otherwise.
- In the SA, P depends on the difference $\Delta = f(x_c) - f(x_n)$ and is 1 if x_n is better than x_c.
- In an SGA for maximization, P is $f(x_n)/(f(x_c) + f(x_n))$. If we wanted to do minimization, maybe we could use $1 - f(x_n)/(f(x_c) + f(x_n))$ instead.
- $(1+1)$ EA and SA always accept x_n if it is better than x_c, while in the SGA, a better x_n at least has a $P > 0.5$.
- Let's keep these things in mind.

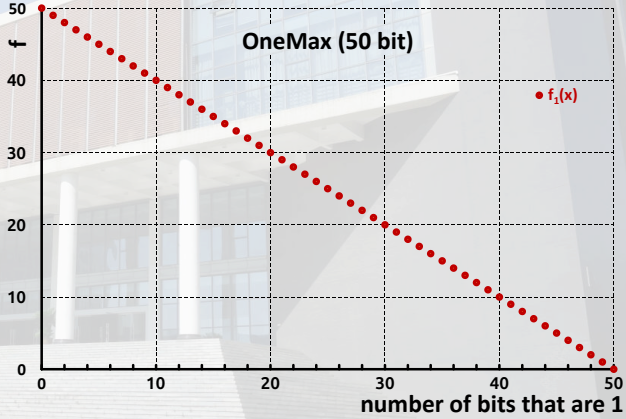
Invariance Properties: OneMax Example

- OneMax is the simplest benchmark problem in discrete optimization[50].



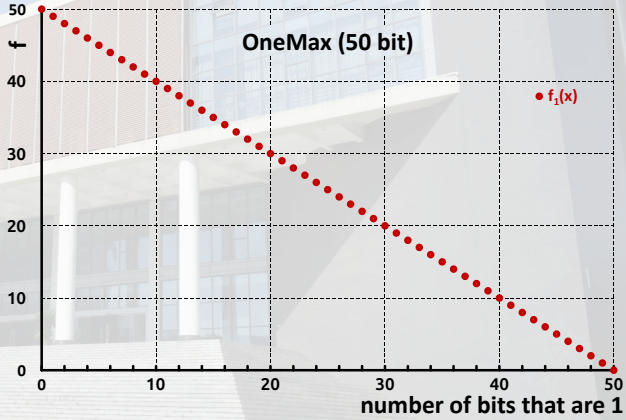
Invariance Properties: OneMax Example

- OneMax is the simplest benchmark problem in discrete optimization[50].
- It is defined over $\mathbb{X} = \{0,1\}^n$, i.e., the bit strings of length n.



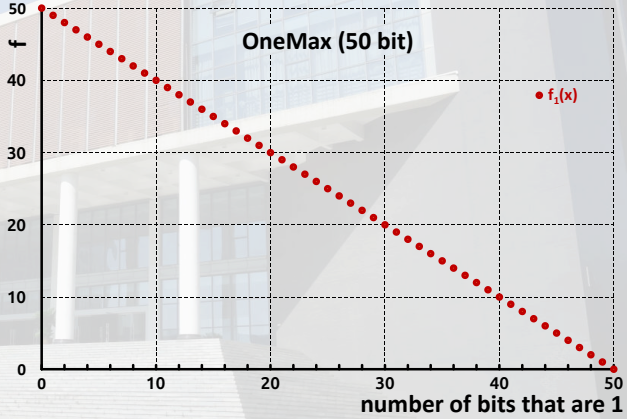
Invariance Properties: OneMax Example

- OneMax is the simplest benchmark problem in discrete optimization[50].
- It is defined over $\mathbb{X} = \{0,1\}^n$, i.e., the bit strings of length n.
- Goal: "Find the bit string of all 1s!"



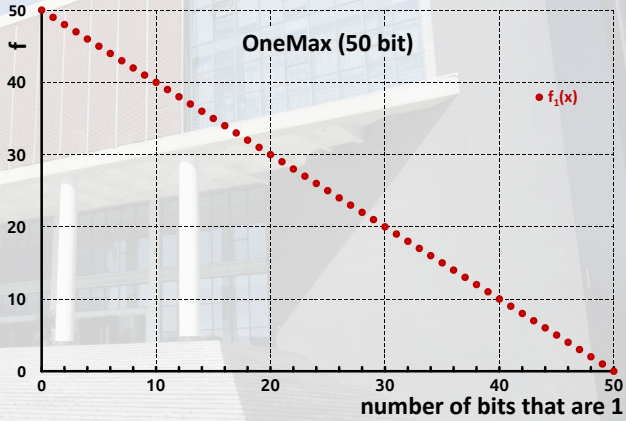
Invariance Properties: OneMax Example

- OneMax is the simplest benchmark problem in discrete optimization[50].
- It is defined over $\mathbb{X} = \{0,1\}^n$, i.e., the bit strings of length n.
- Goal: "Find the bit string of all 1s!"
- "Maximize the number of 1s!"



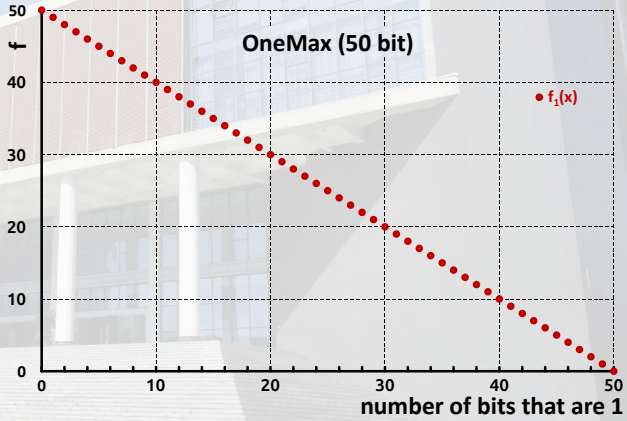
Invariance Properties: OneMax Example

- OneMax is the simplest benchmark problem in discrete optimization[50].
- It is defined over $\mathbb{X} = \{0,1\}^n$, i.e., the bit strings of length n.
- Goal: "Find the bit string of all 1s!"
- "Maximize the number of 1s!"
- $f_1(x) = n - \sum x$ (for minimization).



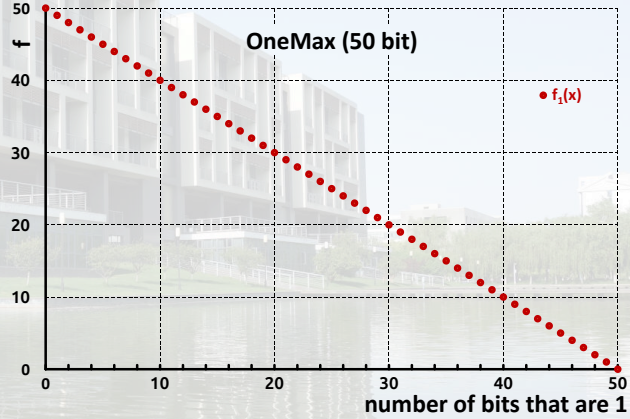
Invariance Properties: OneMax Example

- OneMax is the simplest benchmark problem in discrete optimization[50].
- It is defined over $\mathbb{X} = \{0,1\}^n$, i.e., the bit strings of length n.
- Goal: "Find the bit string of all 1s!"
- "Maximize the number of 1s!"
- $f_1(x) = n - \sum x$ (for minimization).
- Every reasonable algorithm can solve this problem.



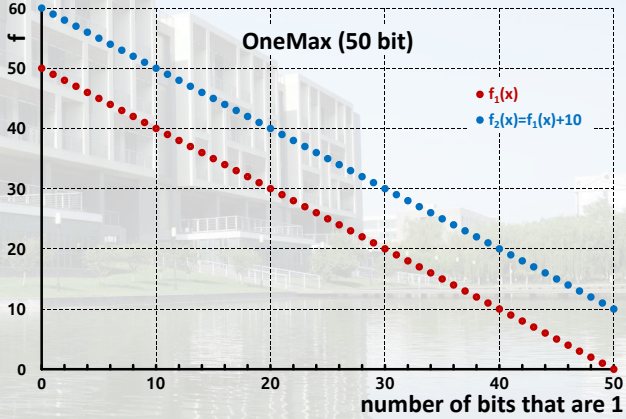
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.



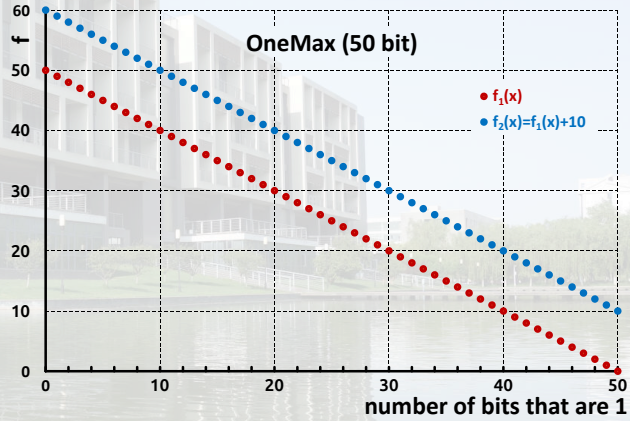
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.



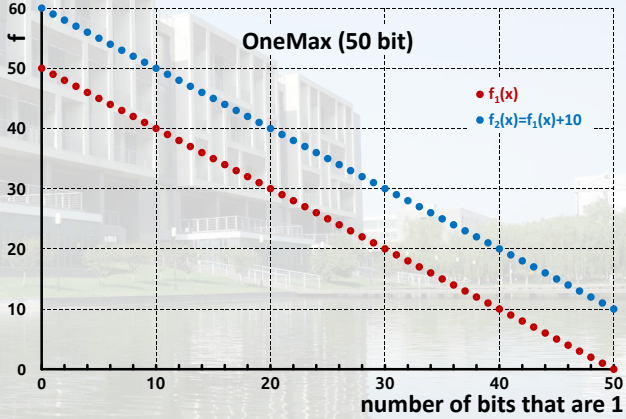
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2.



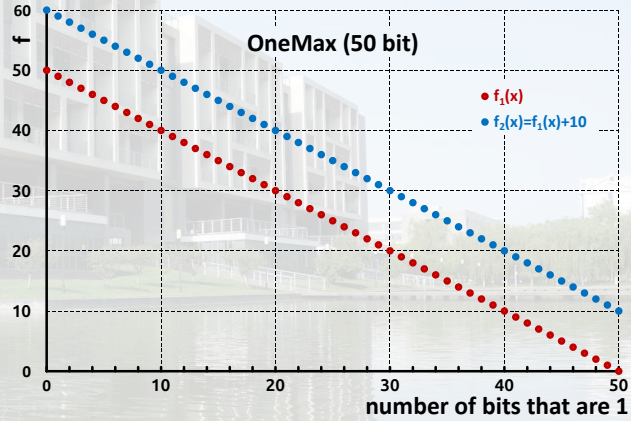
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions.



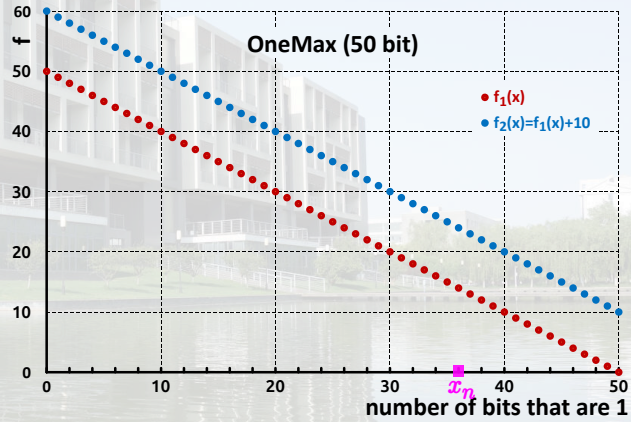
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.



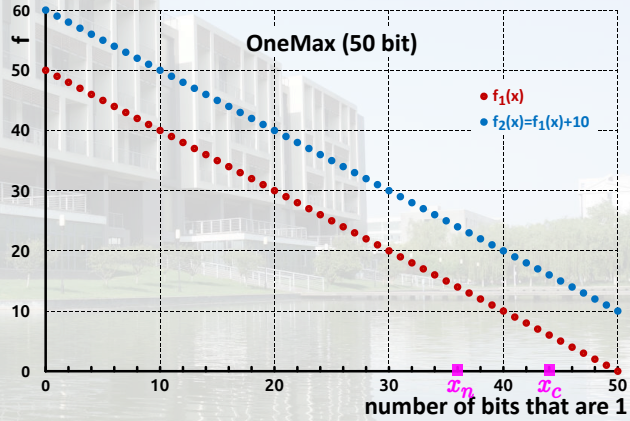
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.



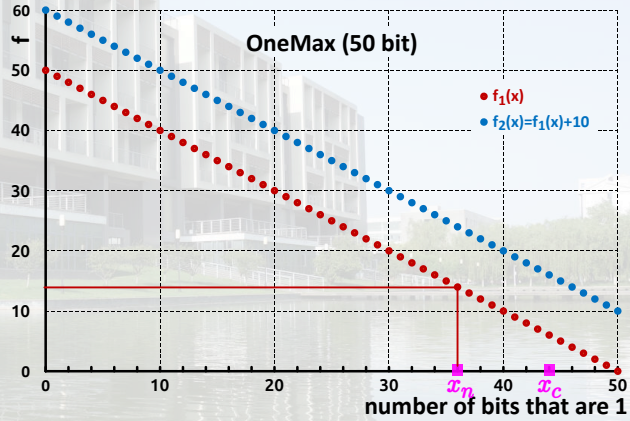
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.



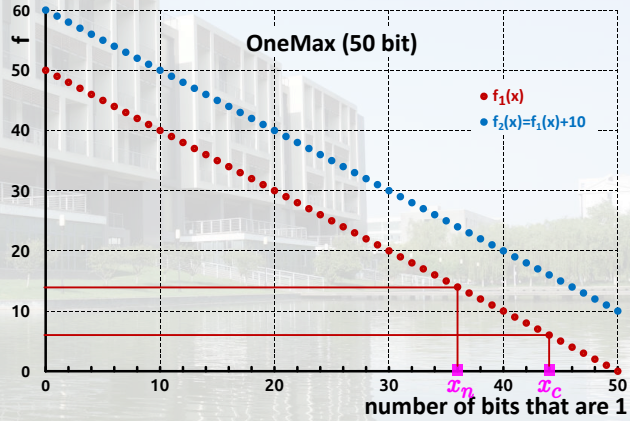
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$



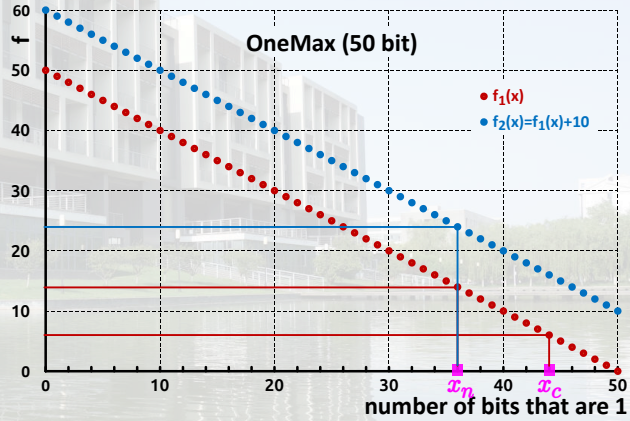
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$



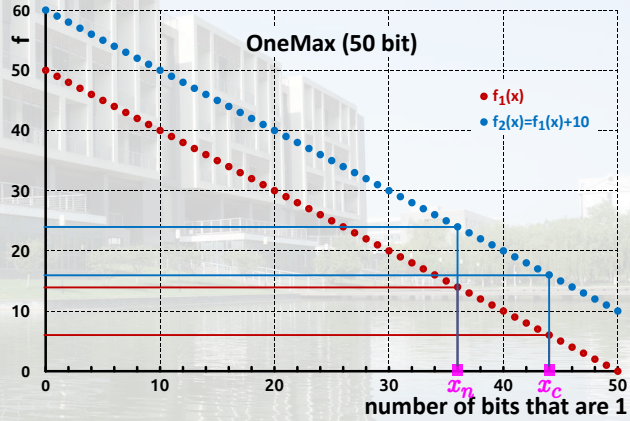
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_2(x_n) = 24$



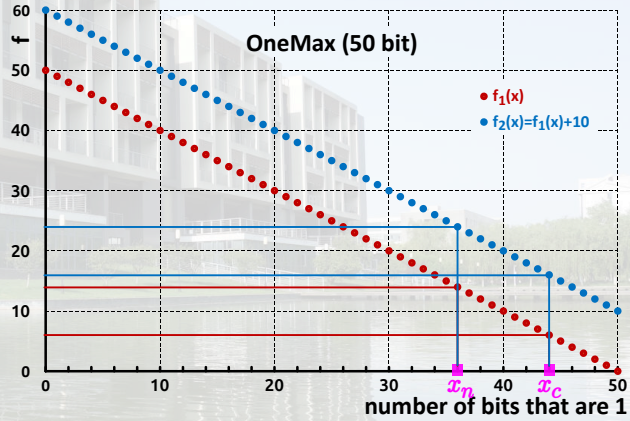
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_2(x_n) = 24$, $f_2(x_c) = 16$



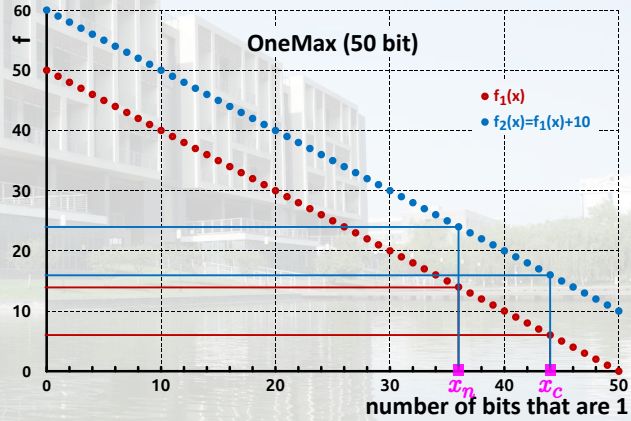
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_2(x_n) = 24$, $f_2(x_c) = 16$
- $(1+1)$ EA: $14 \leq 6 = 24 \leq 16$ ✓



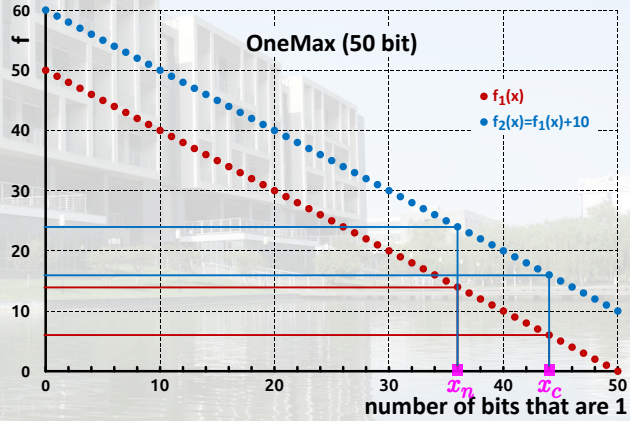
Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_2(x_n) = 24$, $f_2(x_c) = 16$
- $(1+1)$ EA: $14 \leq 6 = 24 \leq 16$ ✓
- SA: $6 - 14 = 16 - 24$ ✓



Invariance Properties: OneMax Example – Translation

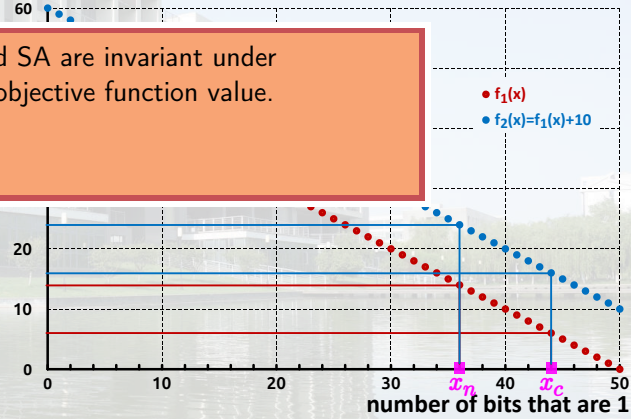
- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_2, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14,\ f_1(x_c) = 6,$ $f_2(x_n) = 24,\ f_2(x_c) = 16$
- $(1+1)$ EA: $14 \leq 6 = 24 \leq 16$ ✓
- SA: $6 - 14 = 16 - 24$ ✓
- SGA: $1-14/(14+6) = 0.3 \neq 1-24/(24+16) = 0.4$ ✗



Invariance Properties: OneMax Example – Translation

- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: algorithm sh on f_1 and f_2 same decisio random seed
- $f_1(x_n) = 14,\ f_1(x_c) = 6,$ $f_2(x_n) = 24,\ f_2(x_c) = 16$
- $(1+1)$ EA: $14 \leq 6 = 24 \leq 16$ ✓
- SA: $6 - 14 = 16 - 24$ ✓
- SGA: $1-14/(14+6) = 0.3 \neq 1-24/(24+16) = 0.4$ ✗

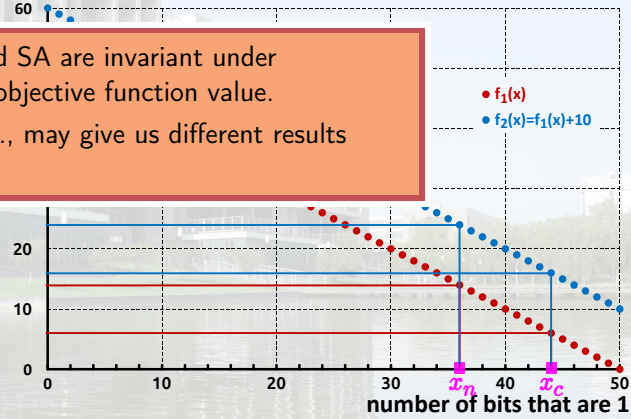
- The $(1+1)$ EA and SA are invariant under translations of the objective function value.



Invariance Properties: OneMax Example – Translation

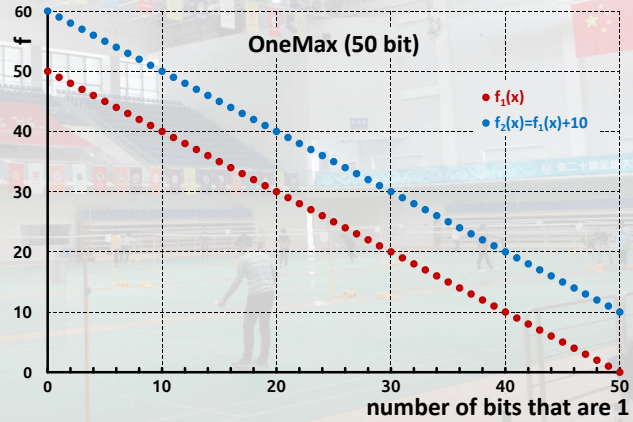
- Now I create a slightly modified variant of this problem.
- $f_2(x) = f_1(x) + 10$.
- Expectation: algorithm sh… on f_1 and f_2 same decisio… random seed…
- $f_1(x_n) = 14,\ f_1(x_c) = 6,$ $f_2(x_n) = 24,\ f_2(x_c) = 16$
- $(1+1)$ EA: $14 \leq 6 = 24 \leq 16$ ✓
- SA: $6 - 14 = 16 - 24$ ✓
- SGA: $1-14/(14+6) = 0.3 \neq 1-24/(24+16) = 0.4$ ✗

- The $(1+1)$ EA and SA are invariant under translations of the objective function value.
- The SGA is not, i.e., may give us different results for f_1 and f_2.



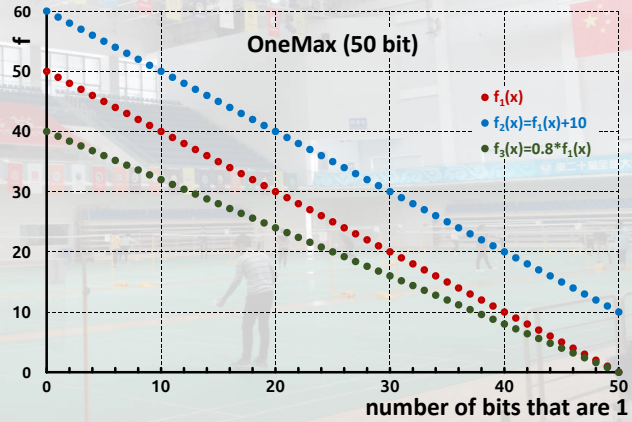
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.



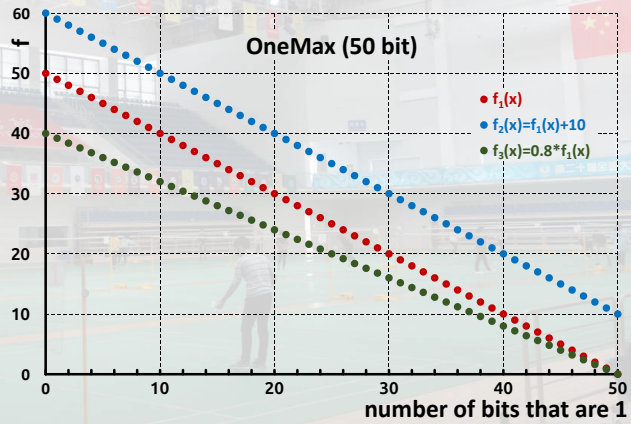
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.



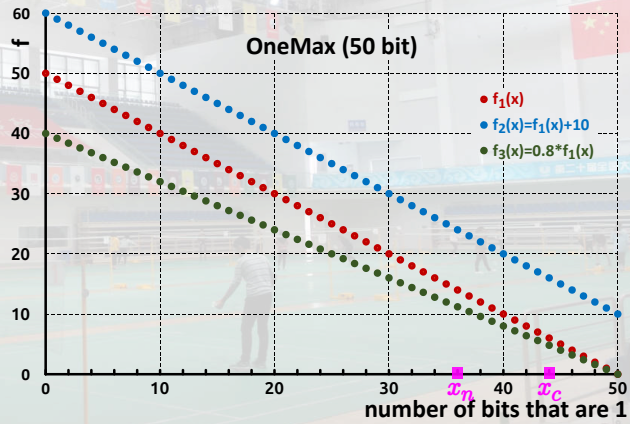
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_3, i.e., make the exact same decisions given the same random seeds.



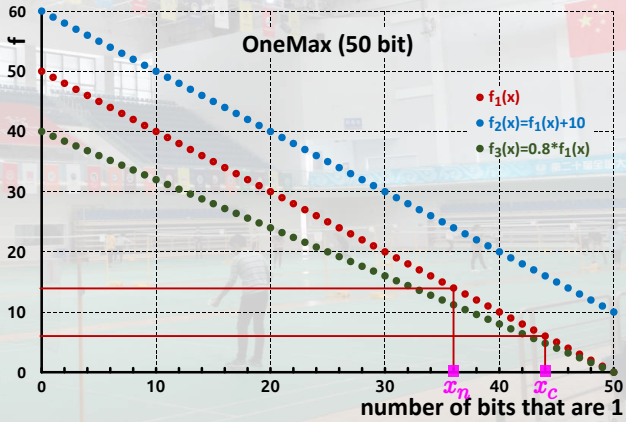
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_3, i.e., make the exact same decisions given the same random seeds.



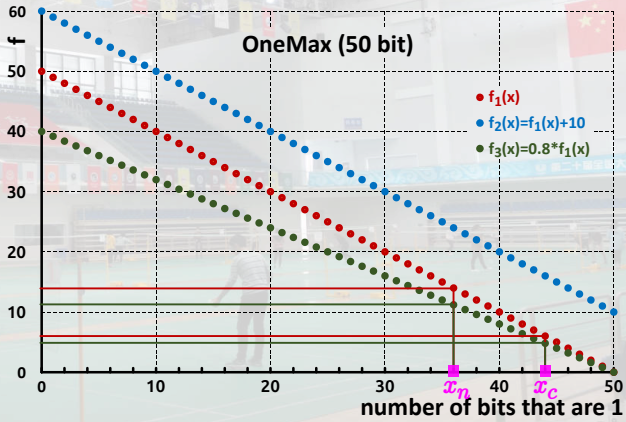
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_3, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$



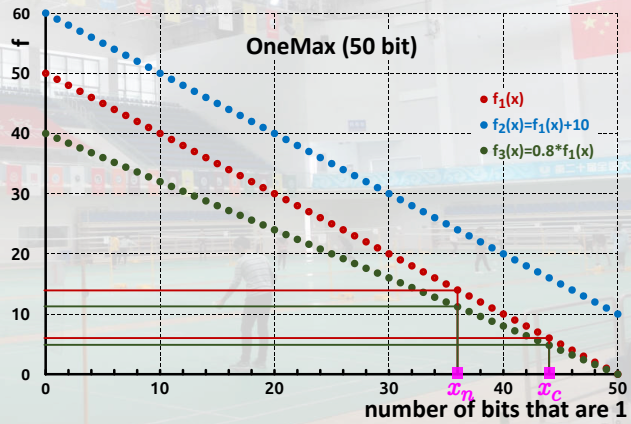
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_3, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_3(x_n) = 11.2$, $f_3(x_c) = 4.8$



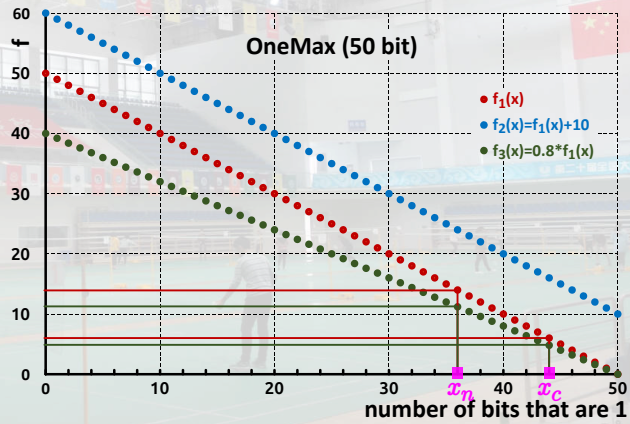
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_3, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_3(x_n) = 11.2$, $f_3(x_c) = 4.8$
- $(1+1)$ EA: $14 \leq 6 = 11.2 \leq 4.8$ ✓



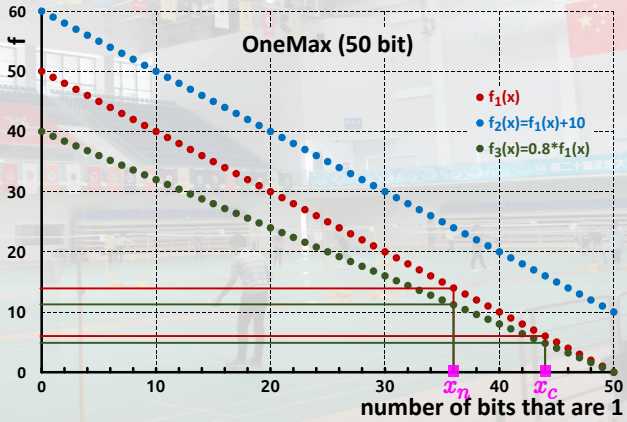
Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_3, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_3(x_n) = 11.2$, $f_3(x_c) = 4.8$
- $(1+1)$ EA: $14 \leq 6 = 11.2 \leq 4.8$ ✓
- SA: $6 - 14 \neq 4.8 - 11.2$ ✗



Invariance Properties: OneMax Example – Scaling

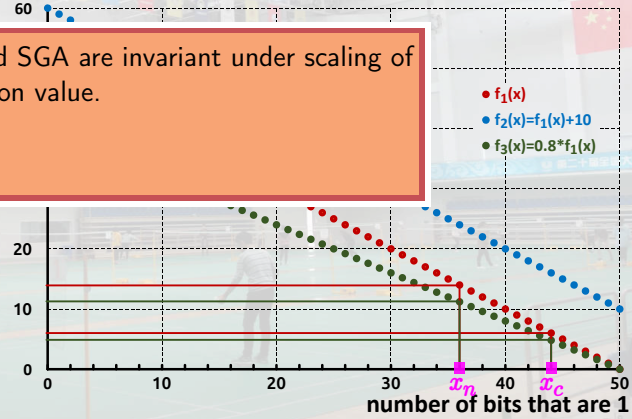
- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: Any reasonable algorithm should perform the same on f_1 and f_3, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_3(x_n) = 11.2$, $f_3(x_c) = 4.8$
- $(1+1)$ EA: $14 \leq 6 = 11.2 \leq 4.8$ ✓
- SA: $6 - 14 \neq 4.8 - 11.2$ ✗
- SGA: $1 - 14/(14+6) = 0.3 = 1 - 11.2/(11.2+4.8) = 0.3$ ✓



Invariance Properties: OneMax Example – Scaling

- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: algorithm sh… on f_1 and f_3 same decisio… random seed…
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_3(x_n) = 11.2$, $f_3(x_c) = 4.8$
- $(1+1)$ EA: $14 \leq 6 = 11.2 \leq 4.8$ ✓
- SA: $6 - 14 \neq 4.8 - 11.2$ ✗
- SGA: $1 - 14/(14+6) = 0.3 = 1 - 11.2/(11.2+4.8) = 0.3$ ✓

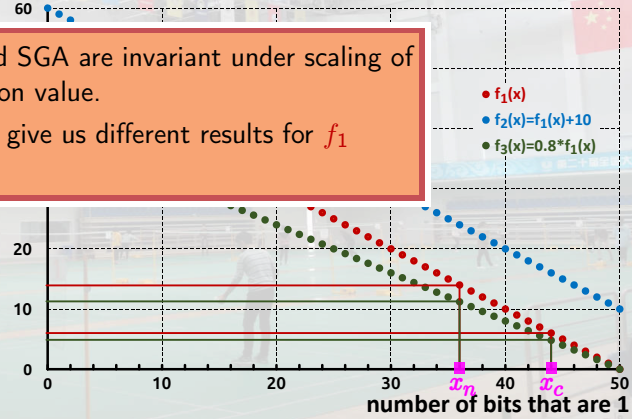
- The $(1+1)$ EA and SGA are invariant under scaling of the objective function value.



Invariance Properties: OneMax Example – Scaling

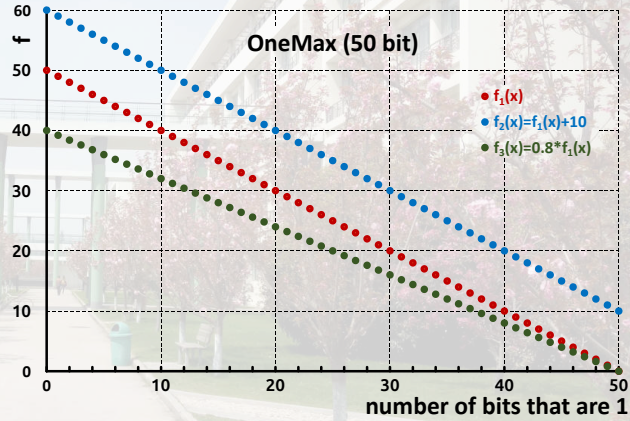
- Now I create another slightly modified variant of this problem.
- $f_3(x) = 0.8 f_1(x)$.
- Expectation: algorithm sh… on f_1 and f_3 same decisio… random seed…
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_3(x_n) = 11.2$, $f_3(x_c) = 4.8$
- $(1+1)$ EA: $14 \leq 6 = 11.2 \leq 4.8$ ✓
- SA: $6 - 14 \neq 4.8 - 11.2$ ✗
- SGA: $1 - 14/(14+6) = 0.3 = 1 - 11.2/(11.2+4.8) = 0.3$ ✓

- The $(1+1)$ EA and SGA are invariant under scaling of the objective function value.
- SA is not, i.e., may give us different results for f_1 and f_3.



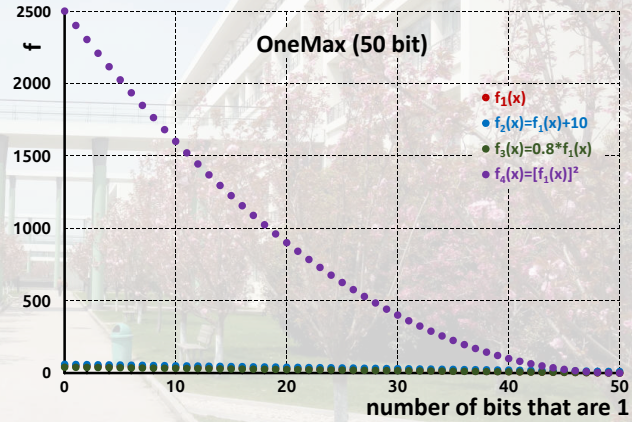
Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.



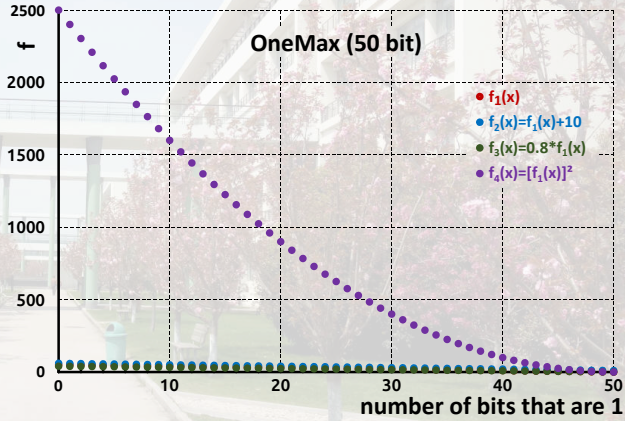
Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.



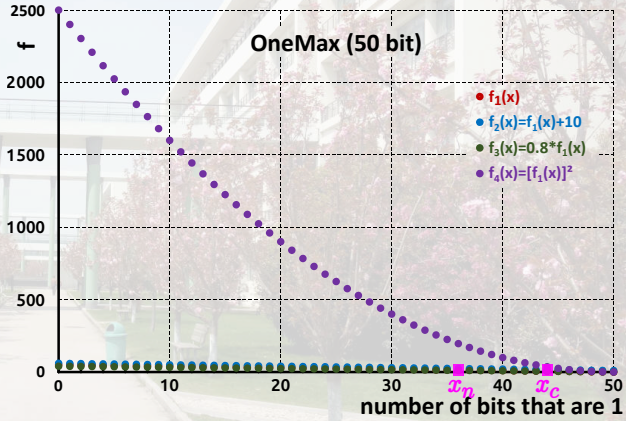
Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: A nice algorithm should perform the same on f_1 and f_4, i.e., make the exact same decisions given the same random seeds.



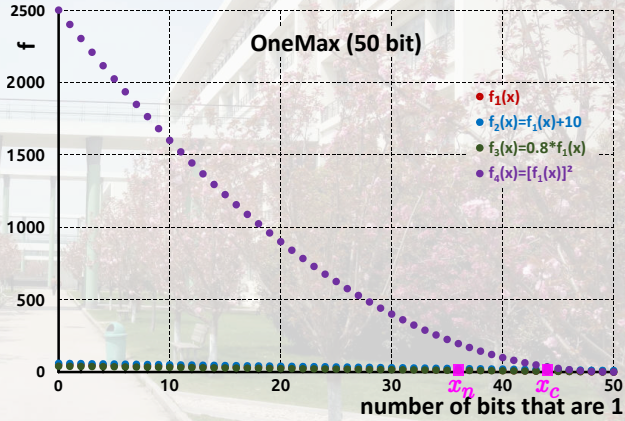
Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: A nice algorithm should perform the same on f_1 and f_4, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$



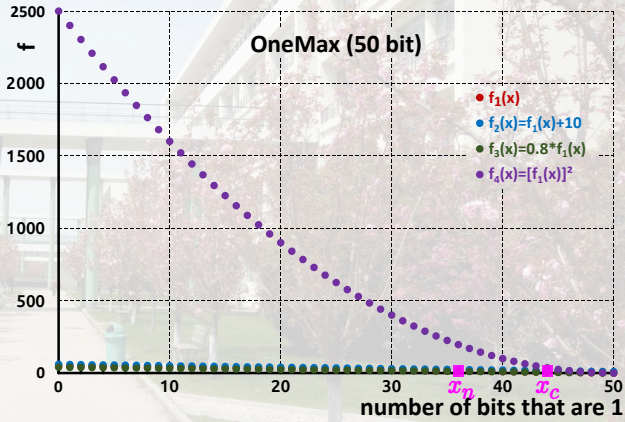
Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: A nice algorithm should perform the same on f_1 and f_4, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_4(x_n) = 196$, $f_4(x_c) = 36$



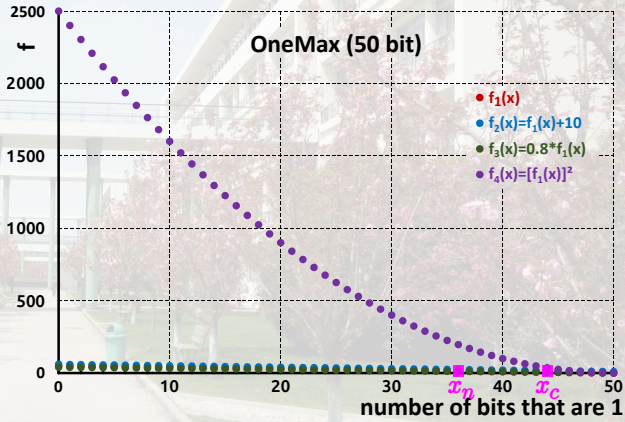
Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: A nice algorithm should perform the same on f_1 and f_4, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_4(x_n) = 196$, $f_4(x_c) = 36$
- $(1+1)$ EA: $14 \leq 6 = 196 \leq 36$ ✓



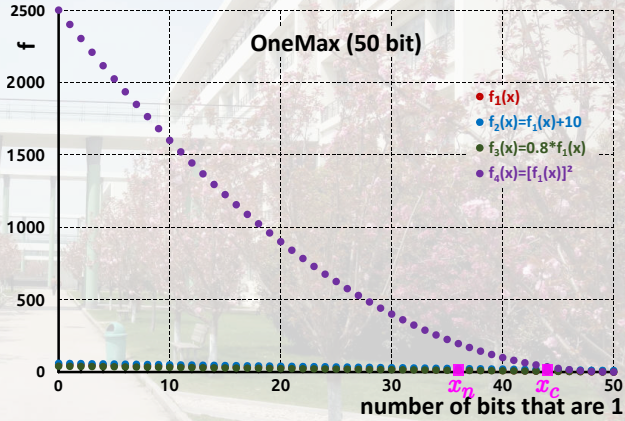
Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: A nice algorithm should perform the same on f_1 and f_4, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_4(x_n) = 196$, $f_4(x_c) = 36$
- $(1+1)$ EA: $14 \leq 6 = 196 \leq 36$ ✓
- SA: $6 - 14 \neq 36 - 196$ ✗



Invariance Properties: OneMax Example – Squaring

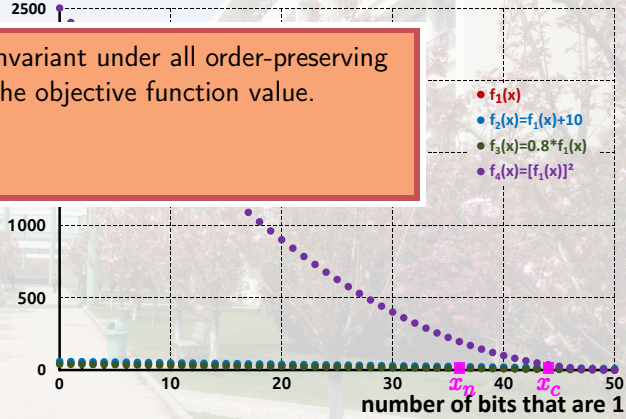
- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: A nice algorithm should perform the same on f_1 and f_4, i.e., make the exact same decisions given the same random seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_4(x_n) = 196$, $f_4(x_c) = 36$
- $(1+1)$ EA: $14 \leq 6 = 196 \leq 36$ ✓
- SA: $6 - 14 \neq 36 - 196$ ✗
- SGA: $1-14/(14+6) = 0.3 \neq 1-196/(196+36) \approx 0.156$ ✗



Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: should perfor and f_4, i.e., decisions give seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_4(x_n) = 196$, $f_4(x_c) = 36$
- $(1+1)$ EA: $14 \leq 6 = 196 \leq 36$ ✓
- SA: $6 - 14 \neq 36 - 196$ ✗
- SGA: $1-14/(14+6) = 0.3 \neq 1-196/(196+36) \approx 0.156$ ✗

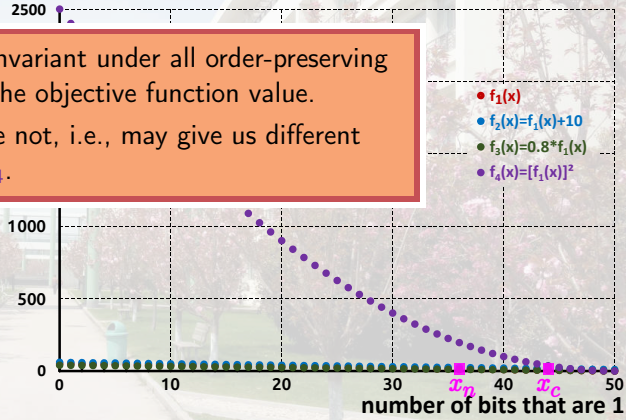
- The $(1+1)$ EA is invariant under all order-preserving transformations of the objective function value.



Invariance Properties: OneMax Example – Squaring

- Now I create another slightly modified variant of this problem.
- $f_4(x) = [f_1(x)]^2$.
- Expectation: should perfor and f_4, i.e., decisions give seeds.
- $f_1(x_n) = 14$, $f_1(x_c) = 6$, $f_4(x_n) = 196$, $f_4(x_c) = 36$
- $(1+1)$ EA: $14 \leq 6 = 196 \leq 36$ ✓
- SA: $6 - 14 \neq 36 - 196$ ✗
- SGA: $1-14/(14+6) = 0.3 \neq 1-196/(196+36) \approx 0.156$ ✗

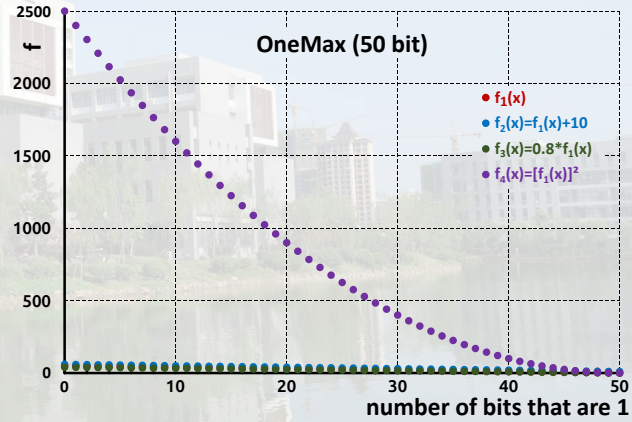
- The $(1+1)$ EA is invariant under all order-preserving transformations of the objective function value.
- SA and the SGA are not, i.e., may give us different results for f_1 and f_4.




Now let's enter eerie territory...

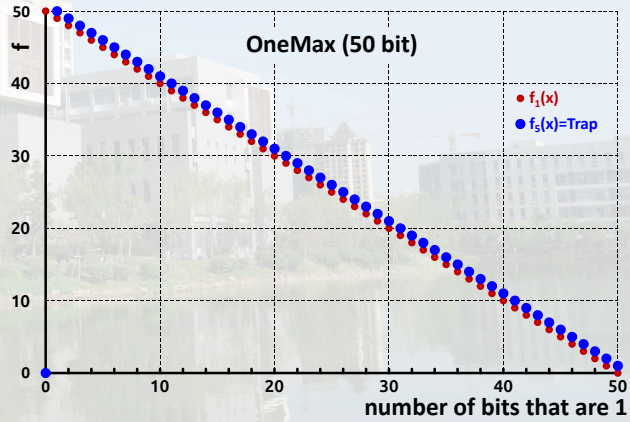
Invariance Properties: OneMax Example – Trap

- Now I create another modified variant of this problem: a trap[17,51].



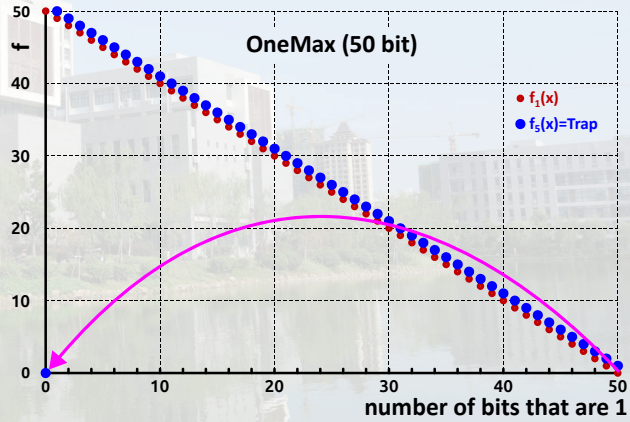
Invariance Properties: OneMax Example – Trap

- Now I create another modified variant of this problem: a trap[17,51].
- $f_5(x) = \begin{cases} 0 & \text{if } f_1(x) = 50 \\ 1 + f_1(x) & \text{otherwise} \end{cases}$



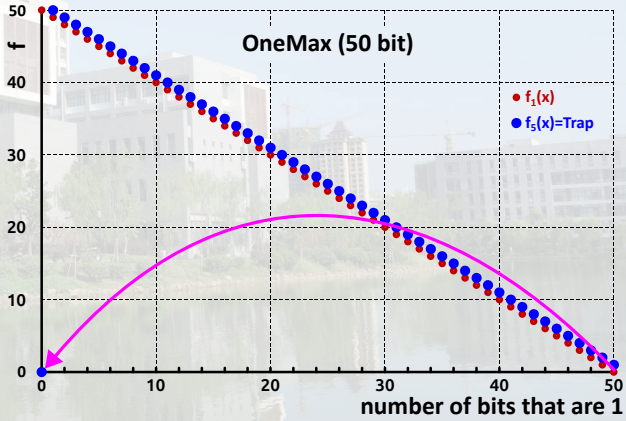
Invariance Properties: OneMax Example – Trap

- Now I create another modified variant of this problem: a trap[17,51].
- $f_5(x) = \begin{cases} 0 & \text{if } f_1(x) = 50 \\ 1 + f_1(x) & \text{otherwise} \end{cases}$



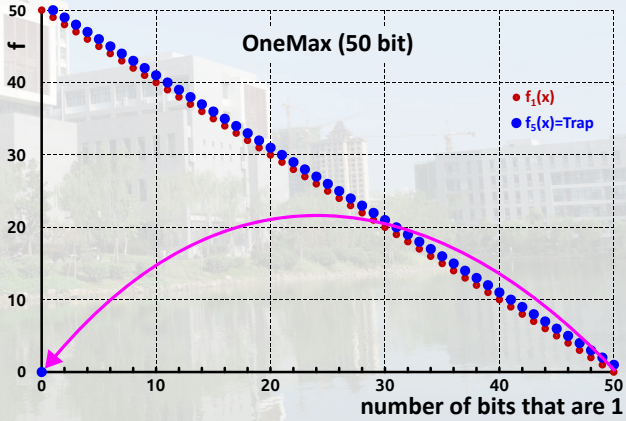
Invariance Properties: OneMax Example – Trap

- Now I create another modified variant of this problem: a trap[17,51].
- $f_5(x) = \begin{cases} 0 & \text{if } f_1(x) = 50 \\ 1 + f_1(x) & \text{otherwise} \end{cases}$
- Expectation: Algorithm performance on f_1 does not carry over to f_5 .



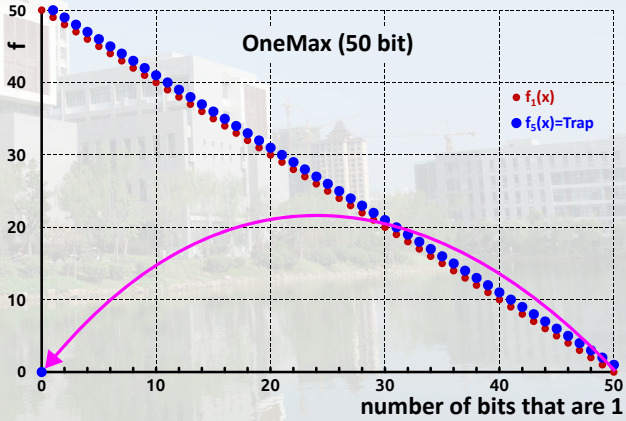
Invariance Properties: OneMax Example – Trap

- Now I create another modified variant of this problem: a trap[17,51].
- $f_5(x) = \begin{cases} 0 & \text{if } f_1(x) = 50 \\ 1 + f_1(x) & \text{otherwise} \end{cases}$
- Expectation: Algorithm performance on f_1 does not carry over to f_5 .
- Neither the $(1+1)$ EA, SA, nor the SGA can deal with this well.



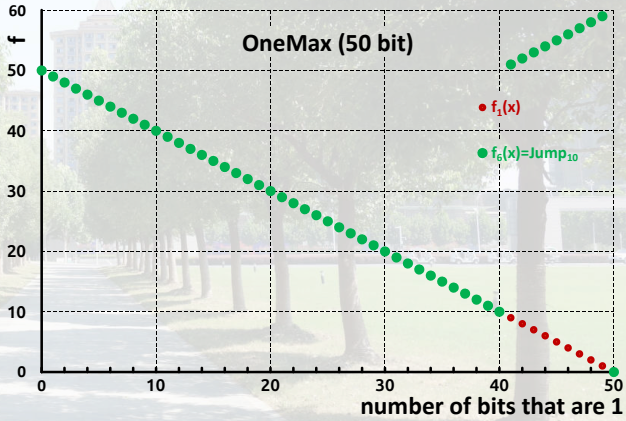
Invariance Properties: OneMax Example – Trap

- Now I create another modified variant of this problem: a trap[17,51].
- $f_5(x) = \begin{cases} 0 & \text{if } f_1(x) = 50 \\ 1 + f_1(x) & \text{otherwise} \end{cases}$
- Expectation: Algorithm performance on f_1 does not carry over to f_5 .
- Neither the $(1+1)$ EA, SA, nor the SGA can deal with this well.
- The $(1+1)$ EA has expected runtime $\Theta(n^n)$ on traps[17].



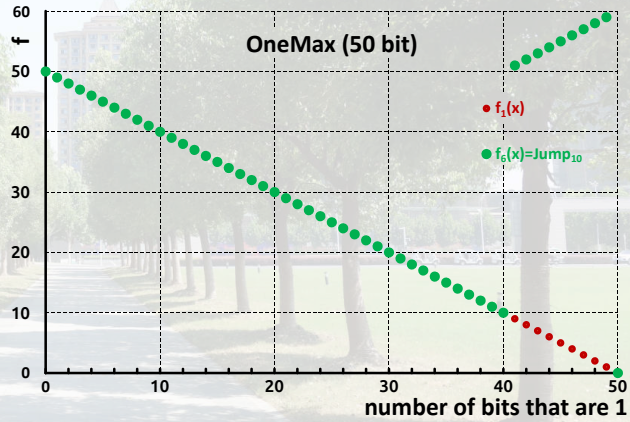
Invariance Properties: OneMax Example – Jump

- Now I create another modified variant of this problem: a jump of width $\omega = 10$ bits.



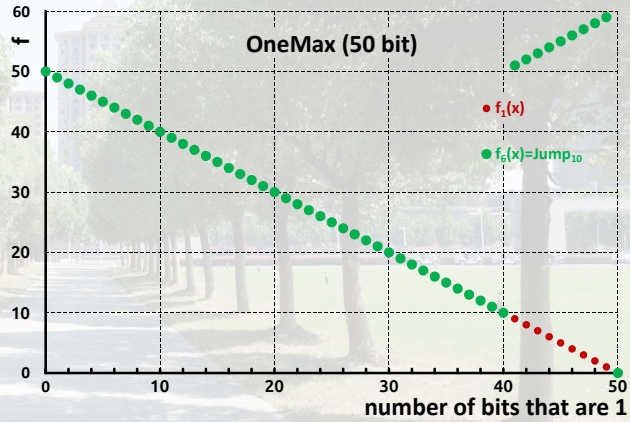
Invariance Properties: OneMax Example – Jump

- Now I create another modified variant of this problem: a jump of width $\omega = 10$ bits.
- $f_6(x)$ has a deceptive area of $\omega - 1$ bits before the optimum.



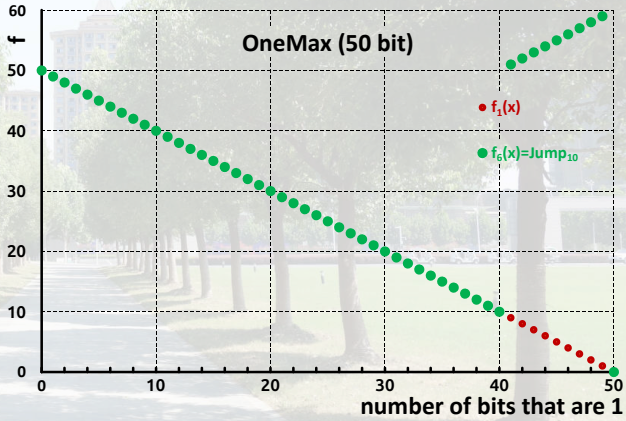
Invariance Properties: OneMax Example – Jump

- Now I create another modified variant of this problem: a jump of width $\omega = 10$ bits.
- $f_6(x)$ has a deceptive area of $\omega - 1$ bits before the optimum.
- Expectation: Algorithm performance on f_1 does not carry over to f_6 .



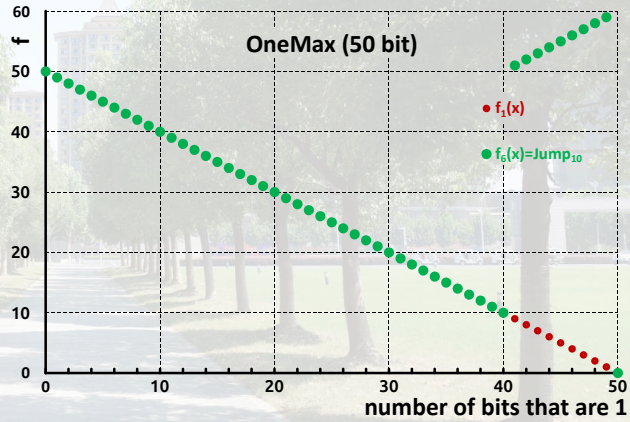
Invariance Properties: OneMax Example – Jump

- Now I create another modified variant of this problem: a jump of width $\omega = 10$ bits.
- $f_6(x)$ has a deceptive area of $\omega - 1$ bits before the optimum.
- Expectation: Algorithm performance on f_1 does not carry over to f_6 .
- Neither the $(1 + 1)$ EA, SA, nor the SGA can deal with this well.



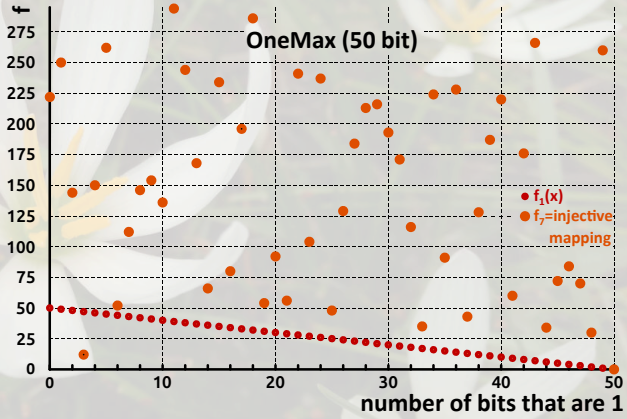
Invariance Properties: OneMax Example – Jump

- Now I create another modified variant of this problem: a jump of width $\omega = 10$ bits.
- $f_6(x)$ has a deceptive area of $\omega - 1$ bits before the optimum.
- Expectation: Algorithm performance on f_1 does not carry over to f_6 .
- Neither the $(1+1)$ EA, SA, nor the SGA can deal with this well.
- The $(1+1)$ EA has expected runtime $\mathbf{\Theta}(n^\omega + n \ln n)$ on jumps[17].



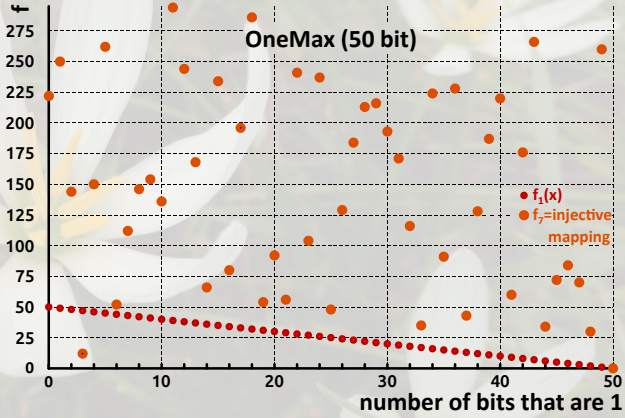
Invariance Properties: OneMax Example – Injection

- How about I apply an arbitrary injection g that preserves the location of optimum to f_1 and get $f_7(x) = g(f_1(x))$?.



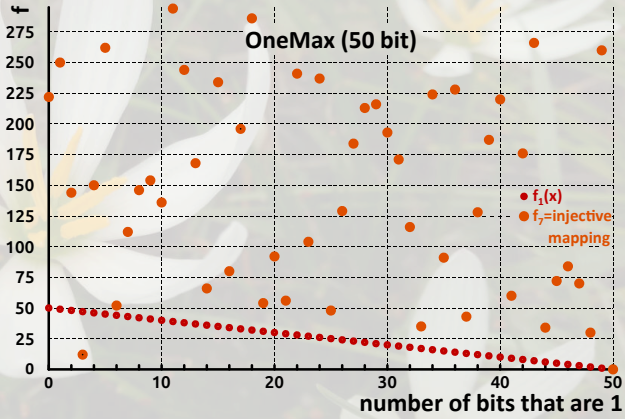
Invariance Properties: OneMax Example – Injection

- How about I apply an arbitrary injection g that preserves the location of optimum to f_1 and get $f_7(x) = g(f_1(x))$?.
- Expectation: Algorithm performance on f_1 is probably unrelated to performance on f_7 .



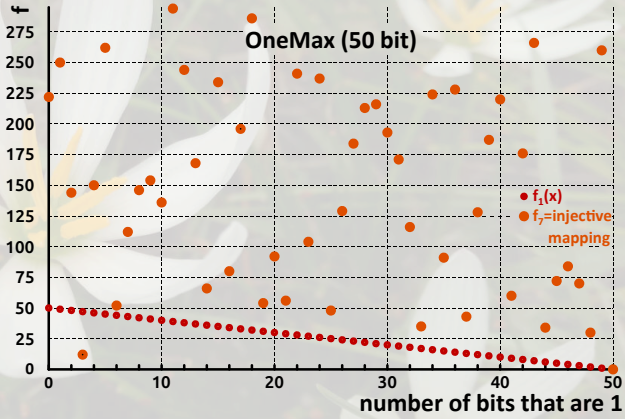
Invariance Properties: OneMax Example – Injection

- How about I apply an arbitrary injection g that preserves the location of optimum to f_1 and get $f_7(x) = g(f_1(x))$?.
- Expectation: Algorithm performance on f_1 is probably unrelated to performance on f_7 .
- Neither the $(1 + 1)$ EA, SA, nor the SGA can deal with this well.



Invariance Properties: OneMax Example – Injection

- How about I apply an arbitrary injection g that preserves the location of optimum to f_1 and get $f_7(x) = g(f_1(x))$?.
- Expectation: Algorithm performance on f_1 is probably unrelated to performance on f_7 .
- Neither the $(1 + 1)$ EA, SA, nor the SGA can deal with this well.
- Indeed, no existing algorithm can deal with this, let alone have some invariance properties with respect to *that*.



Indeed, no existing algorithm can deal with this, let alone have some invariance properties with respect to *that*.

Indeed, no existing algorithm can deal with this, let alone have some invariance properties with respect to *that*.

well...

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

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

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

***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.**

Index Terms—(1+1)-EA, ev... tionary algorithm (EA), ...ency fitness ...gnment ...A), j... shop sc... duling ...n (JS...), ... Ju... ...lems, ...meti...

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 t...ansformations include the negation, permutation, or even en...ryption of the objec... tive valu...s. According to [3], invarianc... extends performan... observed ...n a single function to an en...e associa...d inv... ance clas... ...at is, it generalizes from ...ngle p... class ... p... ...en...s. Th..., it hop...fully ... ...es ... ...es... ...e pre... ...atio...

T Weise, Z Wu, X Li, and Y Chen. "Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value". IEEE Transactions on Evolutionary Computation 25(2):307–319. 2021. [81] [copyright IEEE]

Frequency Fitness Assignment

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

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

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

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on the identity of $f(x)$.

FFA: What does this do?

- The rating $H[f(x)]$ of a solution x depends only on the identity of $f(x)$.
- 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 the identity of $f(x)$.
- 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 the identity of $f(x)$.
- 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 the identity of $f(x)$.
- 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 the identity of $f(x)$.
- 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 the identity of $f(x)$.
- 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 the identity of $f(x)$.
- 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 perform identical on ALL of the OneMax-based functions from before!

Discrete Optimization Theory Benchmarks

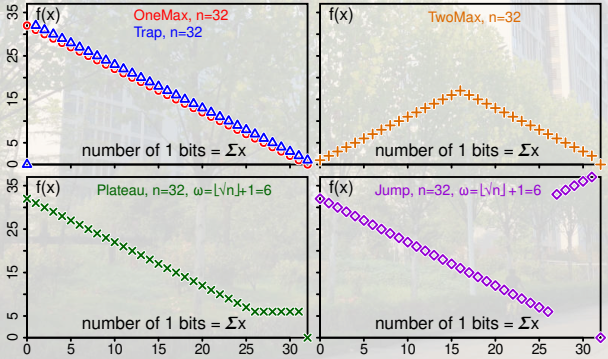
- If we use the encounter frequency of objective values as only guide for optimization ... can this work?

Discrete Optimization Theory Benchmarks

- If we use the encounter frequency of objective values as only guide for optimization . . . can this work?
- Why should this lead to anything but strange random walks?

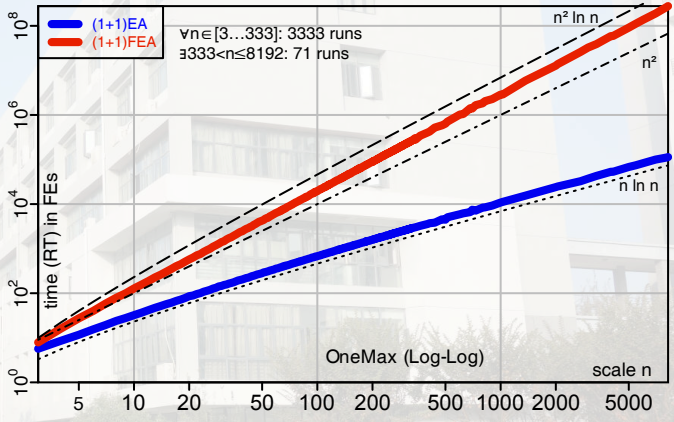
Discrete Optimization Theory Benchmarks

- If we use the encounter frequency of objective values as only guide for optimization . . . can this work?
- Why should this lead to anything but strange random walks?
- So we tested it on discrete optimization benchmarks.



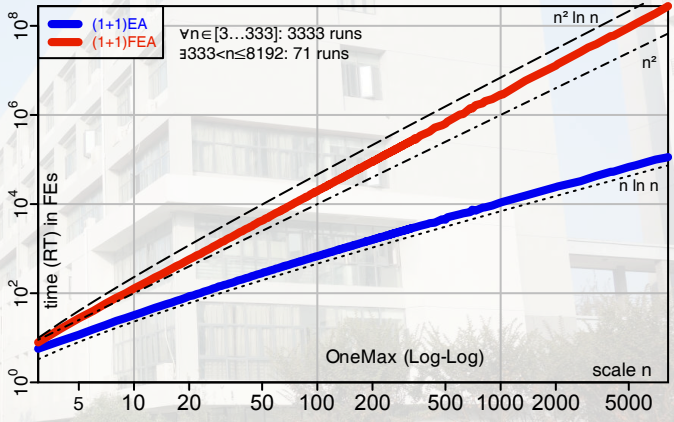
(1+1) FEA on OneMax

- As first example, we again take OneMax[50].



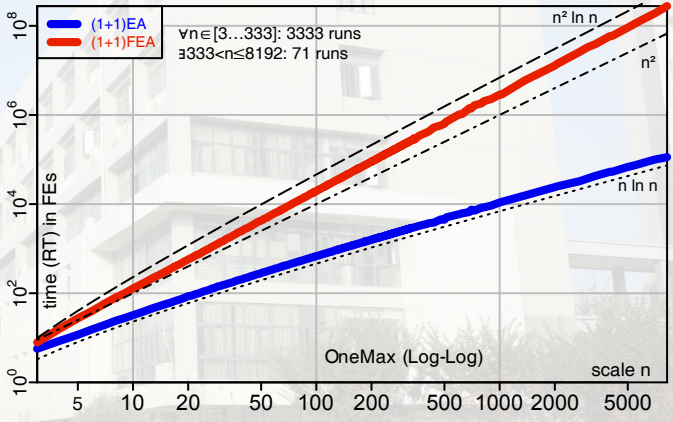
(1+1) FEA on OneMax

- As first example, we again take OneMax[50].
- The expected runtime of $(1+1)$ EA on OneMax is in $\mathbf{\Theta}(n \ln n)$.[16,50]



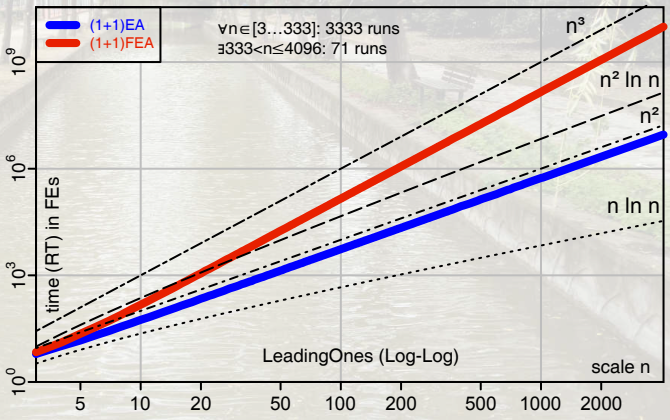
(1+1) FEA on OneMax

- As first example, we again take OneMax[50].
- The expected runtime of $(1+1)$ EA on OneMax is in $\mathbf{\Theta}(n \ln n)$.[16,50]
- The average runtime $(1+1)$ FEA on OneMax seems to be slower by factor proportional to n, i.e., seems to be in $\mathcal{O}(n^2 \ln n)$.



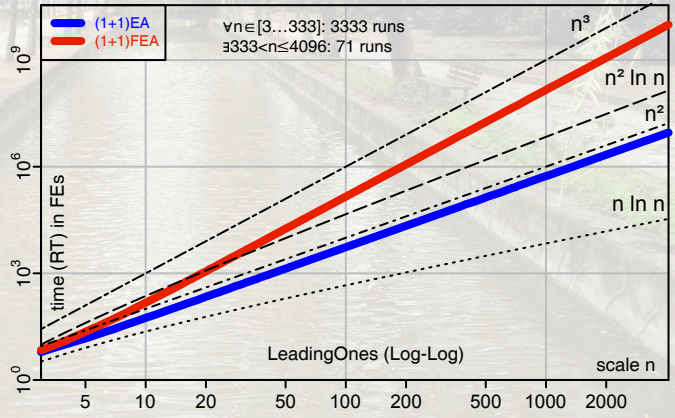
(1+1) FEA on LeadingOnes

- The LeadingOnes problem maximizes the number of leading 1 bits in a bit string[61,84].



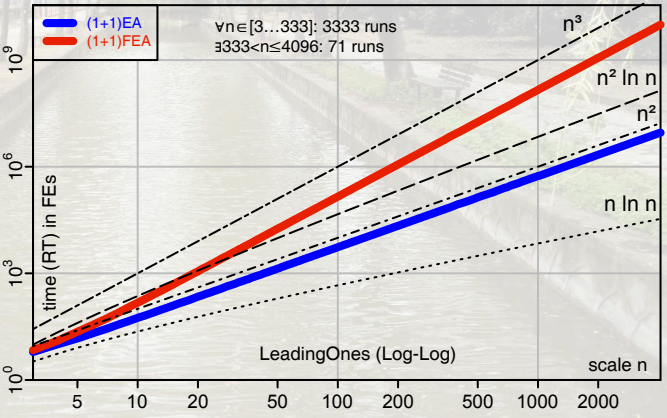
(1+1) FEA on LeadingOnes

- The LeadingOnes problem maximizes the number of leading 1 bits in a bit string[61,84].
- The expected runtime of $(1+1)$ EA on LeadingOnes is in $\mathbf{\Theta}(n^2)$.[17]



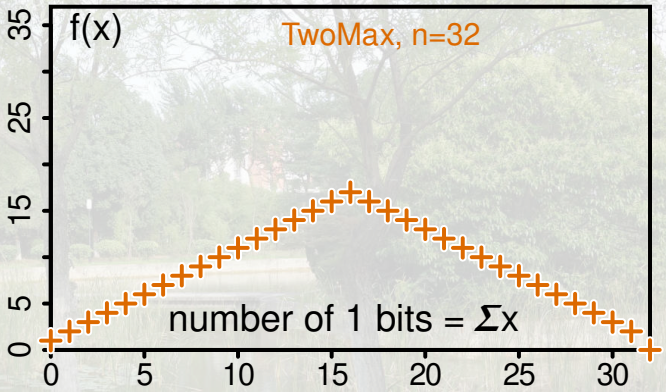
(1+1) FEA on LeadingOnes

- The LeadingOnes problem maximizes the number of leading 1 bits in a bit string[61,84].
- The expected runtime of $(1+1)$ EA on LeadingOnes is in $\mathbf{\Theta}(n^2)$.[17]
- The average runtime $(1+1)$ FEA on LeadingOnes seems to be slower by factor proportional to n, i.e., seems to be in $\mathcal{O}(n^3)$.



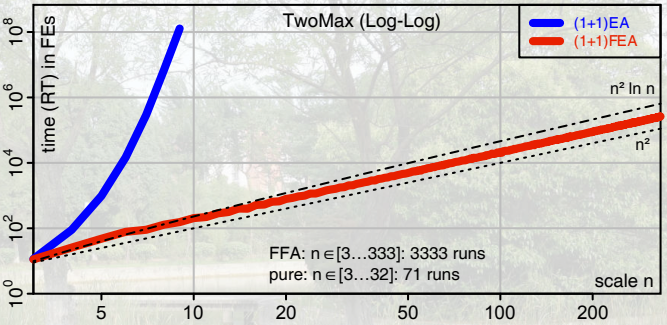
(1+1) FEA on TwoMax

- The TwoMax problem has one local opposite to a global optimum[21,66].



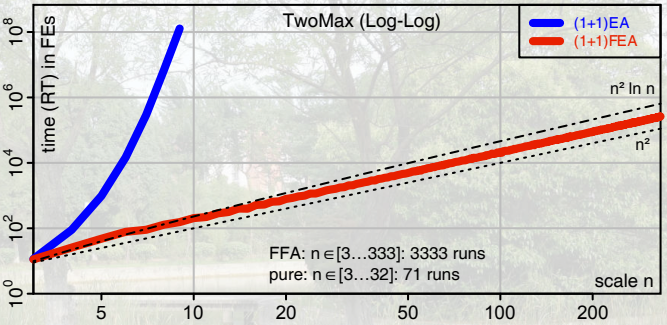
(1+1) FEA on TwoMax

- The TwoMax problem has one local opposite to a global optimum[21,66].
- An $(1+1)$ EA has a 50/50 chance to converge to the local or global optimum and thus has an expected runtime in $\mathbf{\Theta}(n^n)$.[20,21]



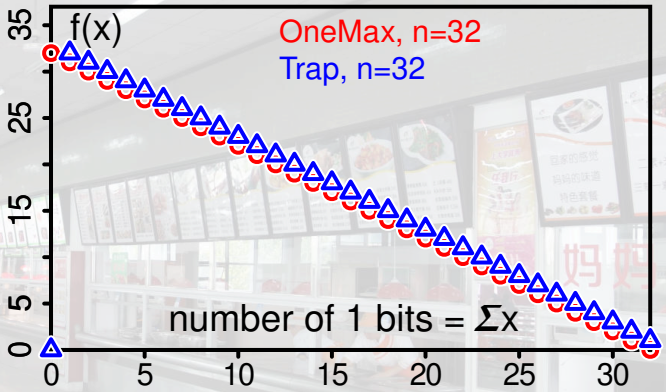
(1+1) FEA on TwoMax

- The TwoMax problem has one local opposite to a global optimum[21,66].
- An $(1+1)$ EA has a 50/50 chance to converge to the local or global optimum and thus has an expected runtime in $\mathbf{\Theta}(n^n)$.[20,21]
- The average runtime $(1+1)$ FEA on TwoMax seems to be in $\mathcal{O}(n^2 \ln n)$.



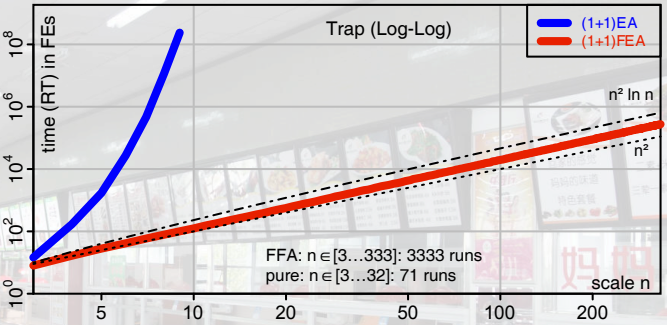
(1+1) FEA on Trap

- The Trap problem basically swaps the global optimum with the worst solution[17,51].



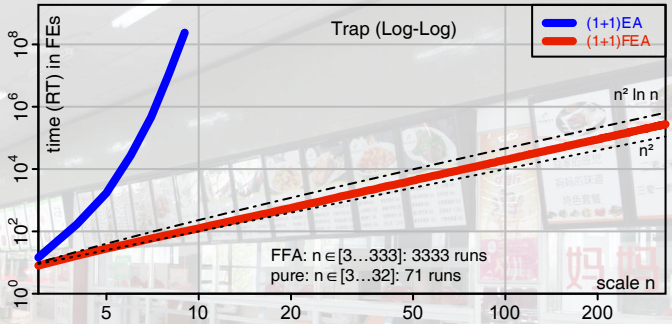
(1+1) FEA on Trap

- The Trap problem basically swaps the global optimum with the worst solution[17,51].
- An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\mathbf{\Theta}(n^n)$.[17]



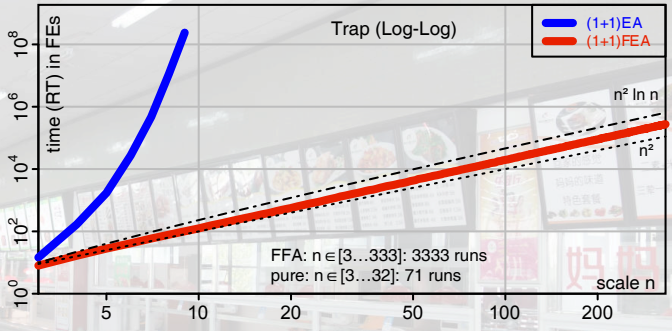
(1+1) FEA on Trap

- The Trap problem basically swaps the global optimum with the worst solution[17,51].
- An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\mathbf{\Theta}(n^n)$.[17]
- The average runtime $(1+1)$ FEA on Trap seems to be in $\mathcal{O}(n^2 \ln n)$.



(1+1) FEA on Trap

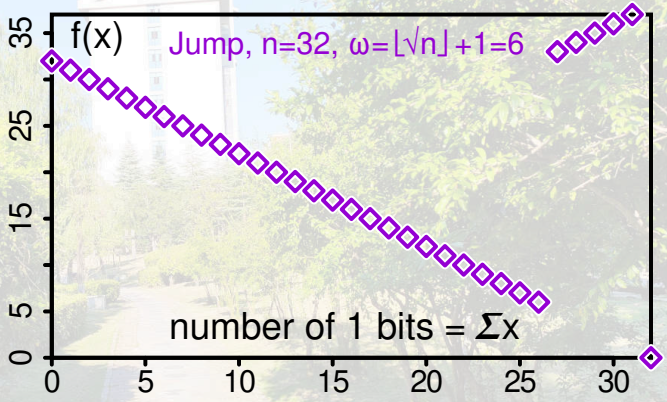
- The Trap problem basically swaps the global optimum with the worst solution[17,51].
- An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\mathbf{\Theta}(n^n)$.[17]
- The average runtime $(1+1)$ FEA on Trap seems to be in $\mathcal{O}(n^2 \ln n)$.



- The expected runtime is the same as on OneMax, because Trap is an injective transformation of OneMax, symmetric, and FFA "converges" into both directions.

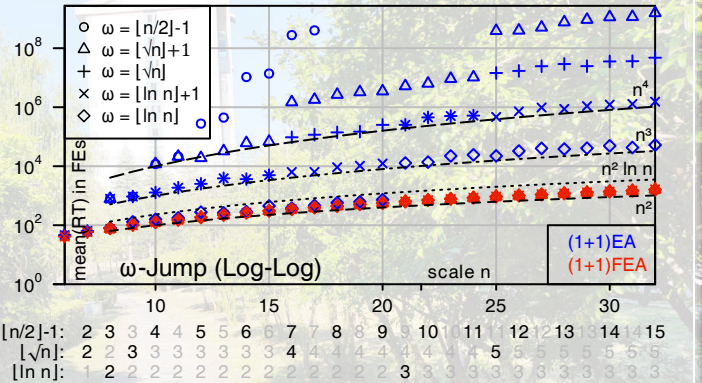
(1+1) FEA on Jumps

- The Jump problem inserts a deceptive region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[17,21].



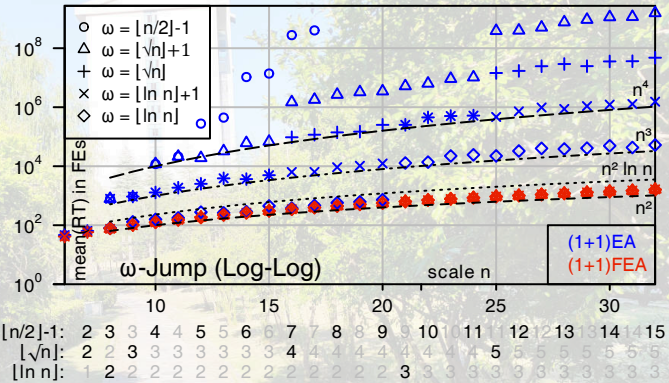
(1+1) FEA on Jumps

- The Jump problem inserts a deceptive region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[17,21].
- An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\mathbf{\Theta}(n^\omega + n \ln n)$.[17]



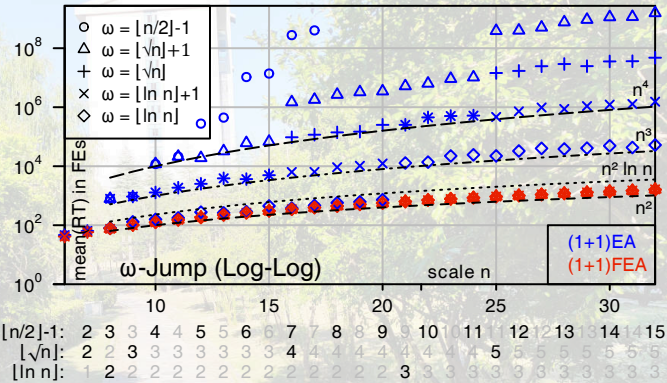
(1+1) FEA on Jumps

- The Jump problem inserts a deceptive region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[17,21].
- An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\mathbf{\Theta}(n^\omega + n \ln n)$.[17]
- The average runtime $(1+1)$ FEA on Jump seems to always be in $\mathcal{O}(n^2 \ln n)$.



(1+1) FEA on Jumps

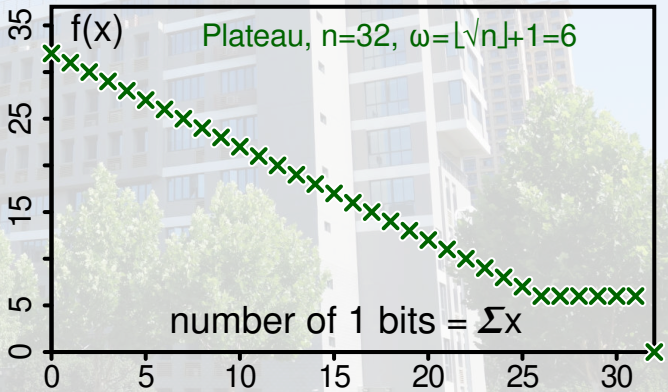
- The Jump problem inserts a deceptive region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[17,21].
- An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\mathbf{\Theta}(n^\omega + n \ln n)$.[17]
- The average runtime $(1+1)$ FEA on Jump seems to always be in $\mathcal{O}(n^2 \ln n)$.



- It is identical to its performance on OneMax regardless of ω, because all Jumps are an injective transformation of OneMax!

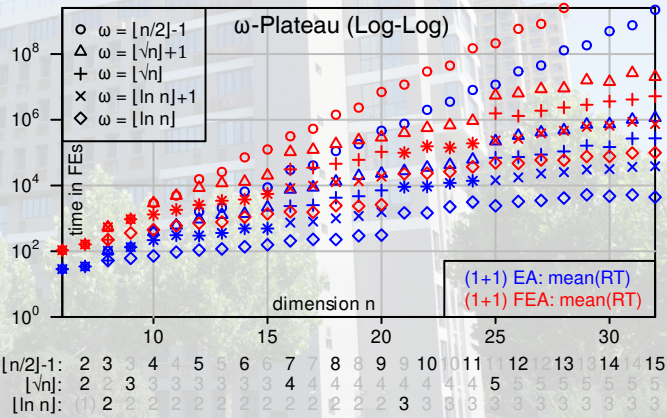
(1+1) FEA on Plateaus

- The Plateau problem inserts a neutral region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[2].



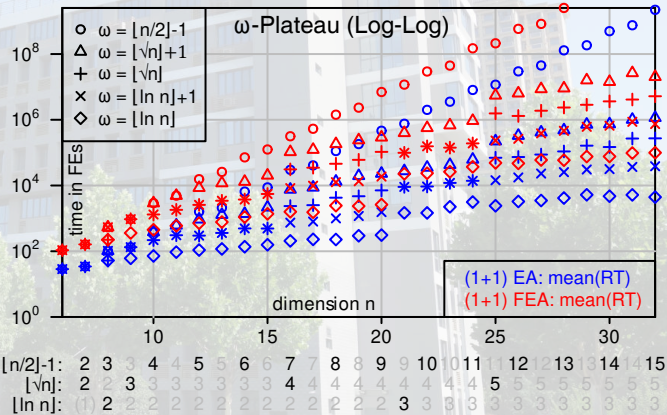
(1+1) FEA on Plateaus

- The Plateau problem inserts a neutral region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[2].
- An $(1+1)$ EA has an expected runtime of $\mathbf{\Theta}(n^\omega)$ on plateaus[2].



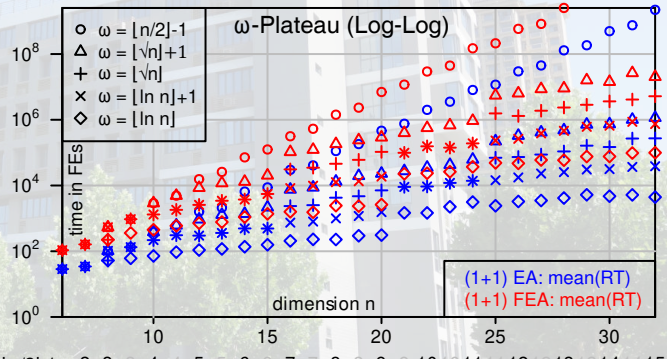
(1+1) FEA on Plateaus

- The Plateau problem inserts a neutral region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[2].
- An $(1+1)$ EA has an expected runtime of $\mathbf{\Theta}(n^\omega)$ on plateaus[2].
- The average runtime $(1+1)$ FEA on Plateaus is slower, again by a factor seemingly proportional in n.



(1+1) FEA on Plateaus

- The Plateau problem inserts a neutral region of $\omega - 1$ bits before the optimum, forcing the algorithm to toggle ω bits at once[2].
- An $(1+1)$ EA has an expected runtime of $\Theta(n^\omega)$ on plateaus[2].
- The average runtime $(1+1)$ FEA on Plateaus is slower, again by a factor seemingly proportional in n.
- Plateaus remain plateaus under the identity-based fitness assignment FFA.



(1+1) FEA on MaxSAT

- The Maximum Satisfiability (MaxSAT)[13,28] problem is $\mathcal{NP}$-complete[13].

(1+1) FEA on MaxSAT

- The Maximum Satisfiability (MaxSAT)[13,28] problem is $\mathcal{NP}$-complete[13].
- SATLIB[27] provides satisfiable benchmark instances from the phase transition region (i.e., the hardest type of instances) for different scales $n \in 20 \cup \{25i \ \forall i \in 2..10\}$.

(1+1) FEA on MaxSAT

- The Maximum Satisfiability (MaxSAT)[13,28] problem is $\mathcal{NP}$-complete[13].
- SATLIB[27] provides satisfiable benchmark instances from the phase transition region (i.e., the hardest type of instances) for different scales $n \in 20 \cup \{25i \ \forall i \in 2..10\}$.
- We conducted 11 000 runs with the $(1 + 1)$ FEA on each instance scale.

(1+1) FEA on MaxSAT

- The Maximum Satisfiability (MaxSAT)[13,28] problem is $\mathcal{NP}$-complete[13].
- SATLIB[27] provides satisfiable benchmark instances from the phase transition region (i.e., the hardest type of instances) for different scales $n \in 20 \cup \{25i \ \forall i \in 2..10\}$.
- We conducted 11 000 runs with the $(1+1)$ FEA on each instance scale.
- The $(1+1)$ EA is very much slower than the $(1+1)$ FEA, so we could use it only on smaller scales.

(1+1) FEA on MaxSAT

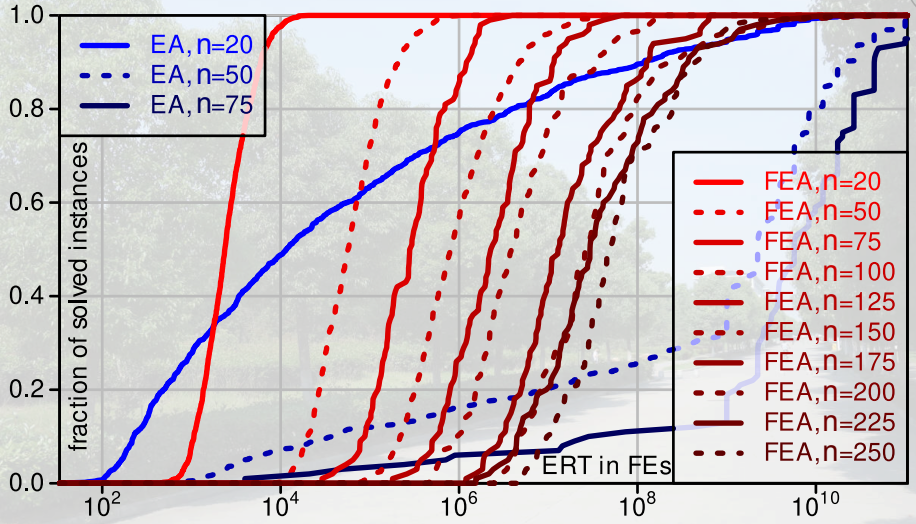
- The Maximum Satisfiability (MaxSAT)[13,28] problem is $\mathcal{NP}$-complete[13].
- SATLIB[27] provides satisfiable benchmark instances from the phase transition region (i.e., the hardest type of instances) for different scales $n \in 20 \cup \{25i \ \forall i \in 2..10\}$.
- We conducted 11 000 runs with the $(1+1)$ FEA on each instance scale.
- The $(1+1)$ EA is very much slower than the $(1+1)$ FEA, so we could use it only on smaller scales.
- Our computational budget was always 10^{10} objective function evaluations (FEs).

(1+1) FEA on MaxSAT

- The $(1+1)$ FEA is better on problems with 250 variables than the $(1+1)$ EA on problems with 50.

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

(1+1) FEA on MaxSAT: ERT-ECDF

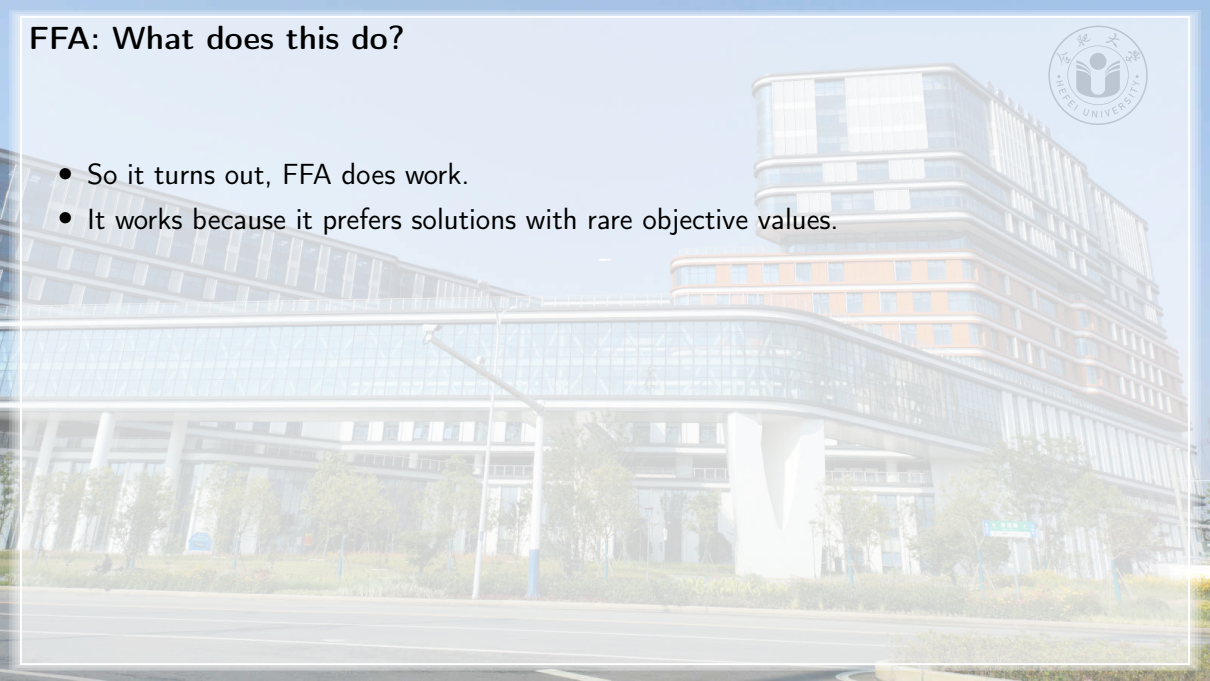


"Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value" [81]

FFA: What does this do?

- So it turns out, FFA does work.

FFA: What does this do?



- So it turns out, FFA does work.
- It works because it prefers solutions with rare objective values.

FFA: What does this do?

- So it turns out, FFA does work.
- It works because it prefers solutions with rare objective values.
- Solutions with very good objective values are also very rare on any problem worth tackling.

FFA: What does this do?

- So it turns out, FFA does work.
- It works because it prefers solutions with rare objective values.
- Solutions with very good objective values are also very rare on any problem worth tackling.
- FFA makes the simple $(1+1)$ EA slower on problems that it can easily solve.

FFA: What does this do?

- So it turns out, FFA does work.
- It works because it prefers solutions with rare objective values.
- Solutions with very good objective values are also very rare on any problem worth tackling.
- FFA makes the simple $(1+1)$ EA slower on problems that it can easily solve.
- The slowdown is roughly proportional to the number of possible different objective values.

FFA: What does this do?

- So it turns out, FFA does work.
- It works because it prefers solutions with rare objective values.
- Solutions with very good objective values are also very rare on any problem worth tackling.
- FFA makes the simple $(1+1)$ EA slower on problems that it can easily solve.
- The slowdown is roughly proportional to the number of possible different objective values.
- On some non-$\mathcal{NP}$-hard problems for which the $(1+1)$ EA needs exponential runtime, the $(1+1)$ FEA needs polynomial mean runtime.

FFA: What does this do?

- So it turns out, FFA does work.
- It works because it prefers solutions with rare objective values.
- Solutions with very good objective values are also very rare on any problem worth tackling.
- FFA makes the simple $(1+1)$ EA slower on problems that it can easily solve.
- The slowdown is roughly proportional to the number of possible different objective values.
- On some non-$\mathcal{NP}$-hard problems for which the $(1+1)$ EA needs exponential runtime, the $(1+1)$ FEA needs polynomial mean runtime.
- But not for all, because plateaus of the objective are still plateaus under FFA.

FFA: What does this do?

- So it turns out, FFA does work.
- It works because it prefers solutions with rare objective values.
- Solutions with very good objective values are also very rare on any problem worth tackling.
- FFA makes the simple $(1+1)$ EA slower on problems that it can easily solve.
- The slowdown is roughly proportional to the number of possible different objective values.
- On some non-$\mathcal{NP}$-hard problems for which the $(1+1)$ EA needs exponential runtime, the $(1+1)$ FEA needs polynomial mean runtime.
- But not for all, because plateaus of the objective are still plateaus under FFA.
- FFA very significantly speeds up the $(1+1)$ EA on the $\mathcal{NP}$-complete MaxSAT problem.

FFA: Let's get weird!

Let's get weird!

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.
- Let's say you apply AES[1,18], RSA[60,87], or the Caesar cipher[53,87] as a function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$ to f, i.e., do $g \circ f$.

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.
- Let's say you apply AES[1,18], RSA[60,87], or the Caesar cipher[53,87] as a function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$ to f, i.e., do $g \circ f$.
- Encryption removes any order, correlation or causality information, i.e., $g \circ f$ does not correlate with f in any way.

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.
- Let's say you apply AES[1,18], RSA[60,87], or the Caesar cipher[53,87] as a function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$ to f, i.e., do $g \circ f$.
- Encryption removes any order, correlation or causality information, i.e., $g \circ f$ does not correlate with f in any way.
- $g(f(x))$ will be totally random.

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.
- Let's say you apply AES[1,18], RSA[60,87], or the Caesar cipher[53,87] as a function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$ to f, i.e., do $g \circ f$.
- Encryption removes any order, correlation or causality information, i.e., $g \circ f$ does not correlate with f in any way.
- $g(f(x))$ will be totally random. No reasonable optimization algorithm could work with that.

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.
- Let's say you apply AES[1,18], RSA[60,87], or the Caesar cipher[53,87] as a function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$ to f, i.e., do $g \circ f$.
- Encryption removes any order, correlation or causality information, i.e., $g \circ f$ does not correlate with f in any way.
- $g(f(x))$ will be totally random. No reasonable optimization algorithm could work with that.
- If the $(1+1)$ FEA can find the optimum of $f(x)\ldots$

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.
- Let's say you apply AES[1,18], RSA[60,87], or the Caesar cipher[53,87] as a function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$ to f, i.e., do $g \circ f$.
- Encryption removes any order, correlation or causality information, i.e., $g \circ f$ does not correlate with f in any way.
- $g(f(x))$ will be totally random. No reasonable optimization algorithm could work with that.
- If the $(1+1)$ FEA can find the optimum of $f(x)\ldots$
- $\ldots$ then it will find exactly the same solution in exactly the same runtime even if you apply it to the encrypted problem $g(f(x))$!

FFA: Let's get weird!

- Let's say you have an optimization problem with objective function $f(x)$.
- You encrypt the objective values and do not tell them to the algorithm.
- Let's say you apply AES[1,18], RSA[60,87], or the Caesar cipher[53,87] as a function $g : \mathbb{N}_0 \mapsto \mathbb{N}_0$ to f, i.e., do $g \circ f$.
- Encryption removes any order, correlation or causality information, i.e., $g \circ f$ does not correlate with f in any way.
- $g(f(x))$ will be totally random. No reasonable optimization algorithm could work with that.
- If the $(1+1)$ FEA can find the optimum of $f(x)\ldots$
- $\ldots$ then it will find exactly the same solution in exactly the same runtime even if you apply it to the encrypted problem $g(f(x))$!
- How incredibly strange is that?

Where did we try FFA?

Applications of FFA

- Often, when some new type of algorithm is designed, one is tempted to pick some special, niche problems as application area.

Applications of FFA

- Often, when some new type of algorithm is designed, one is tempted to pick some special, niche problems as application area.
- Then, the competition is not so high and it's easier to get some good papers out.

Applications of FFA

- Often, when some new type of algorithm is designed, one is tempted to pick some special, niche problems as application area.
- Then, the competition is not so high and it's easier to get some good papers out.
- We did not do this.

Applications of FFA

- Often, when some new type of algorithm is designed, one is tempted to pick some special, niche problems as application area.
- Then, the competition is not so high and it's easier to get some good papers out.
- We did not do this.
- We picked some of the most popular $\mathcal{NP}$-hard problems as testbeds.

Applications of FFA

- Often, when some new type of algorithm is designed, one is tempted to pick some special, niche problems as application area.
- Then, the competition is not so high and it's easier to get some good papers out.
- We did not do this.
- We picked some of the most popular $\mathcal{NP}$-hard problems as testbeds.
- We could not (yet) win against the state-of-the-art that matured over decades.

Applications of FFA

- Often, when some new type of algorithm is designed, one is tempted to pick some special, niche problems as application area.
- Then, the competition is not so high and it's easier to get some good papers out.
- We did not do this.
- We picked some of the most popular $\mathcal{NP}$-hard problems as testbeds.
- We could not (yet) win against the state-of-the-art that matured over decades.
- But we did get some interesting results on real, hard, and well-known problems.

Maximum Satisfiability Problem

- FFA on the Maximum Satisfiability (MaxSAT) Problem: [79, 81, 82].

Maximum Satisfiability Problem

- FFA on the Maximum Satisfiability (MaxSAT) Problem: [79, 81, 82].

> 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 [82] (copyright IEEE).
- Several different EAs with and without FFA
- Budget: 10^{10} FEs

Traveling Salesperson Problem

- FFA on the Traveling Salesperson Problem (TSP): [42–44].

Traveling Salesperson Problem

- FFA on the Traveling Salesperson Problem (TSP): [42–44].
- Snippet of page 12 of [42] (copyright Springer).
- $(1+1)$ EA, $(1+1)$ FEA, SA w/o FFA, hybrids
- Budget: 10^{10} FEs

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): [12, 65].

Quadratic Assignment Problem

- FFA on the Quadratic Assignment Problem (QAP): [12, 65].
- Snippet of page 7 of [12] (copyright SciTePress).
- RLS w/o FFA
- Budget: 10^8 FEs

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: [14, 72, 81].

Job Shop Scheduling Problem

- FFA on the Job Shop Scheduling Problem (JSSP) Problem: [14, 72, 81].

> 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 [72] (copyright ACM).
- Budget: $2^{30} \approx 10^9$ FEs

Algorithm Synthesis and Genetic Programming

- FFA for algorithm synthesis and Genetic Programming (GP): [78, 79].

Algorithm Synthesis and Genetic Programming

- FFA for algorithm synthesis and Genetic Programming (GP): [78, 79].

> 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 [78] (copyright IEEE).
- Budget: $\approx 100\,000$ FEs

Where FFA did not work

- FFA cannot work well on problems where local search already gives good results.

Where FFA did not work

- FFA cannot work well on problems where local search already gives good results.
- This includes, surprisingly, also some computationally hard problems.

Where FFA did not work

- FFA cannot work well on problems where local search already gives good results.
- This includes, surprisingly, also some computationally hard problems:
  - the Traveling Tournament Problem (TTP), where RLS can find surprisingly good results: [85].

Where FFA did not work

- FFA cannot work well on problems where local search already gives good results.
- This includes, surprisingly, also some computationally hard problems:
  - the Traveling Tournament Problem (TTP), where RLS can find surprisingly good results: [85].
  - the two-dimensional bin packing problem, where RLS can actually outperform several complex state-of-the-art metaheuristics [88, 89].

Summary

Summary

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

Summary

- Frequency Fitness Assignment (FFA) is an algorithm module that can be plugged into existing algorithms.
- It renders algorithms invariant under all injective transformations of the objective function value.

Summary

- Frequency Fitness Assignment (FFA) is an algorithm module that can be plugged into existing algorithms.
- It renders algorithms invariant under all injective transformations of the objective function value.
- It makes them optimize without bias for good solutions.

Summary

- Frequency Fitness Assignment (FFA) is an algorithm module that can be plugged into existing algorithms.
- It renders algorithms invariant under all injective transformations of the objective function value.
- It makes them optimize without bias for good solutions.
- It slows them down on easy problems.

Summary

- Frequency Fitness Assignment (FFA) is an algorithm module that can be plugged into existing algorithms.
- It renders algorithms invariant under all injective transformations of the objective function value.
- It makes them optimize without bias for good solutions.
- It slows them down on easy problems.
- It can speed them up on hard problems or lead them to solution qualities they otherwise cannot reach.

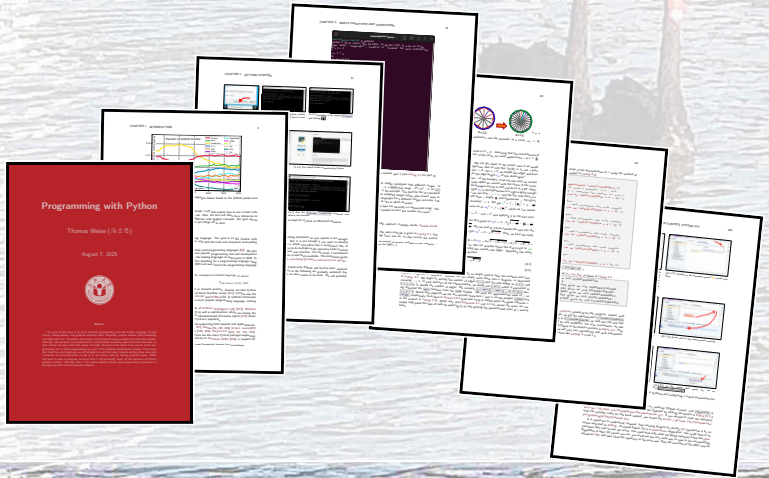
Summary

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

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[70].

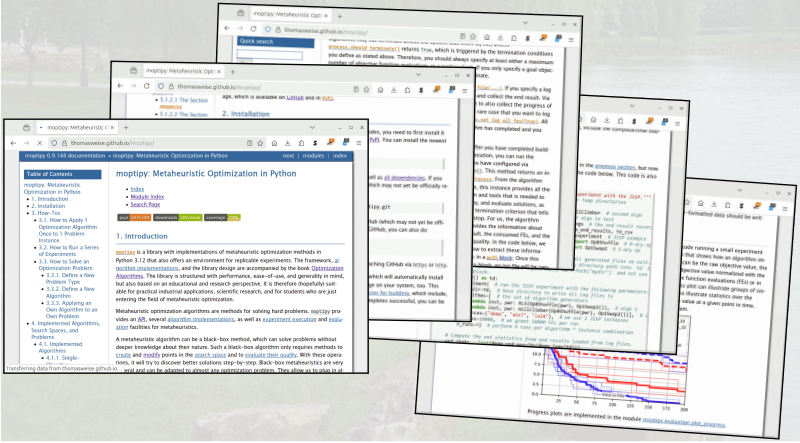


Databases

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)[68].

Metaheuristic Optimization in Python: moptipy

We offer moptipy[80] 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] *Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication (FiPS PUB) 197-upd1. Gaithersburg, MD, USA: U.S. Department of Commerce, National Institute of Standards and Technology (NIST), May 9, 2023. doi:10.6028/NIST.FIPS.197-upd1. URL: https://csrc.nist.gov/pubs/fips/197/final (visited on 2025-08-17) (cit. on pp. 230–239, 275).

[2] Denis Antipov and Benjamin Doerr. "Precise Runtime Analysis for Plateaus". In: *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Vol. 2. Sept. 8–12, 2018, Coimbra, Portugal. Ed. by Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and L. Darrell Whitley. Vol. 11102 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018, pp. 117–128. ISSN: 0302-9743. ISBN: 978-3-319-99258-7. doi:10.1007/978-3-319-99259-4_10 (cit. on pp. 211–214).

[3] 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. 14–17, 289).

[4] 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. 290).

[5] 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. 69–83, 286, 288).

[6] Thomas Bartz-Beielstein, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, William La Cava, Manuel López-Ibáñez, Katherine Mary Malan, Jason Hall Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise (汤卫思). "Benchmarking in Optimization: Best Practice and Open Issues". (abs/2007.03488), Dec. 18, 2020. doi:10.48550/arXiv.2007.03488. URL: https://arxiv.org/abs/2007.03488 (visited on 2025-07-25). arXiv:2007.03488v2 [cs.NE] 16 Dec 2020 (cit. on p. 286).

[7] 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. 287).

References II

[8] 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. 288).

[9] 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. 45–55, 286).

[10] 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. 288).

[11] 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. 287, 289).

[12] 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. 251, 252, 287, 288).

[13] 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. 18–25, 215–219, 287, 289).

References III

[14] 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[72] (cit. on pp. 253, 254).

[15] Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*. Vol. 4 of 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. 69–83, 288).

[16] Benjamin Doerr, Carola Doerr, and Franziska Ebel. "From Black-Box Complexity to Designing New Genetic Algorithms". *Theoretical Computer Science* 567:87–104, Feb. 2015. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/J.TCS.2014.11.028 (cit. on pp. 194–196).

[17] 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. 45–55, 150–160, 197–199, 203–210, 286).

[18] Morris J. Dworkin, Elaine Barker, James R. Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, and James F. Dray Jr. *Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication (FiPS PUB) 197. Gaithersburg, MD, USA: U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Nov. 26, 2001. doi:10.6028/NIST.FIPS.197. URL: https://www.nist.gov/publications/advanced-encryption-standard-aes (visited on 2025-08-17). Updated/replaced by[1] (cit. on pp. 230–239).

[19] 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. 289).

References IV

[20] Tobias Friedrich, Pietro Simone Oliveto, Dirk Sudholt, and Carsten Witt. "Analysis of Diversity-Preserving Mechanisms for Global Exploration". *Evolutionary Computation* 17(4):455–476, Win. 2009. Cambridge, MA, USA: MIT Press. ISSN: 1063-6560. doi:10.1162/evco.2009.17.4.17401 (cit. on pp. 200–202).

[21] 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. 200–202, 207–210).

[22] 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. 69–83, 288).

[23] 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. 289).

[24] 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. 14–17, 289).

[25] Nikolaus Hansen and Anne Auger. "Principled Design of Continuous Stochastic Search: From Theory to Practice". In: *Theory and Principled Methods for the Design of Metaheuristics*. Ed. by Yossi Borenstein and Alberto Moraglio. Natural Computing Series. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2014, pp. 145–180. ISSN: 1619-7127. ISBN: 978-3-642-33205-0. doi:10.1007/978-3-642-33206-7_8. URL: https://inria.hal.science/hal-00808450/file/hansen2013principled.pdf (visited on 2025-08-09) (cit. on pp. 91–96).

[26] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* 4:100–107, July 31, 1968. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 0536-1567. doi:10.1109/TSSC.1968.300136 (cit. on pp. 4–10).

References V

[27] Holger H. Hoos and Thomas Stützle. "SATLIB: An Online Resource for Research on SAT". In: *SAT2000 – Highlights of Satisfiability Research in the Year 2000*. Ed. by Ian Philip Gent, Toby Walsh, and Hans van Maaren. Vol. 63 of Frontiers in Artificial Intelligence and Applications. Amsterdam, The Netherlands: IOS Press BV, Jan. 2000, pp. 283–292. ISSN: 0922-6389. ISBN: 978-1-58603-061-2 (cit. on pp. 215–219, 288).

[28] 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 pp. 18–25, 215–219, 287).

[29] 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. 287).

[30] 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[31] (cit. on p. 277).

[31] 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[30] (cit. on pp. 277, 288).

[32] Mohamed Jebalia and Anne Auger. "Log-Linear Convergence of the Scale-Invariant $(\mu/\mu_w, \lambda)$-ES and Optimal μ for Intermediate Recombination for Large Population Sizes". In: *11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*. Vol. 1. Sept. 11–15, 2010, Kraków, Poland. Ed. by Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph. Vol. 6238 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2010, pp. 52–62. ISSN: 0302-9743. ISBN: 978-3-642-15843-8. doi:10.1007/978-3-642-15844-5_6 (cit. on pp. 91–96).

[33] 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. 56–68, 288).

References VI

[34] 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 pp. 289, 290).

[35] 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 pp. 289, 290).

[36] 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. 288).

[37] 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. 287).

[38] 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. 290).

[39] 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. 287, 289).

[40] 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. 14–17, 289).

[41] 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. 287).

References VII

[42] 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. 14–17, 249, 250, 287, 289).

[43] 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. 249, 250, 287).

[44] 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. 249, 250, 287).

[45] 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. 288).

[46] Mark Lutz. *Learning Python*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2025. ISBN: 978-1-0981-7130-8 (cit. on p. 287).

[47] 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.

[48] 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. 69–83, 288).

[49] 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. 69–83, 288).

References VIII

[50] 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. 106–111, 194–196).

[51] Siegfried Nijssen and Thomas Bäck. "An Analysis of the Behavior of Simplified Evolutionary Algorithms on Trap Functions". *IEEE Transactions on Evolutionary Computation* 7(1):11–22, Feb. 2003. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2002.806169 (cit. on pp. 150–155, 203–206).

[52] Yann Ollivier, Ludovic Arnold, Anne Auger, and Nikolaus Hansen. "Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles". *Journal of Machine Learning Research (JMLR)* 18(18):1–65, Apr. 2017. Cambridge, MA, USA: MIT Press. ISSN: 1532-4435. URL: https://jmlr.org/papers/volume18/14-467/14-467.pdf (visited on 2025-08-09) (cit. on pp. 91–96).

[53] Richard E. Overill. "Codes and Ciphers: Julius Caesar, The Enigma, and the Internet". *Journal of Logic and Computation* 12(3):543, June 2002. Oxford, Oxfordshire, England, UK: Oxford University Press. ISSN: 0955-792X. doi:10.1093/LOGCOM/12.3.543 (cit. on pp. 230–239).

[54] 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.

[55] Amit Patel. *Amit's A* Pages*. Stanford, CA, USA: Stanford University, 1997–2025. URL: https://theory.stanford.edu/~amitp/GameProgramming (visited on 2026-01-02) (cit. on pp. 4–10).

[56] 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. 288).

[57] Alexander Podlich, Thomas Weise (汤卫思), Manfred Menze, and Christian Gorldt. "Intelligente Wechselbrückensteuerung für die Logistik von Morgen". *Electronic Communications of the EASST (ECEASST)* 17(Communication in Distributed Systems), Feb. 27, 2009. The Netherlands: European Association of Software Science and Technology (EASST). ISSN: 1863-2122. doi:10.14279/tuj.eceasst.17.205. In the proceedings of *Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009* (WowKiVS 2009), 3 6, 2009, Kassel, Hessen, Germany (cit. on pp. 4–10).

References IX

[58] 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. 287).

[59] 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. 289).

[60] Ronald Linn Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Communications of the ACM (CACM)* 21(2):120–126, Feb. 1978. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/359340.359342 (cit. on pp. 230–239, 288).

[61] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Vol. 35 of Forschungsergebnisse zur Informatik. Hamburg, Germany: Verlag Dr. Kovač, 1997. ISBN: 978-3-86064-554-3 (cit. on pp. 197–199).

[62] 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. 286).

[63] 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. 288).

[64] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge, England, UK: Cambridge University Press (CUP), 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. 287).

[65] 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. 251, 252, 287).

References X

[66] 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. 200–202).

[67] 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. 289).

[68] 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), 2025. URL: https://thomasweise.github.io/databases (visited on 2025-01-05) (cit. on pp. 270, 286).

[69] 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. 69–83, 286–288).

[70] 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), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (visited on 2025-01-05) (cit. on pp. 269, 287).

[71] 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. 14–17, 286, 289).

References XI

[72] 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. 253, 254, 275).

[73] Thomas Weise (汤卫思), Li Niu (牛力), and Ke Tang (唐珂). "AOAB – Automated Optimization Algorithm Benchmarking". In: *12th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'2010)*. July 7–11, 2010, Portland, OR, USA. Ed. by Martin Pelikan and Jürgen Branke. New York, NY, USA: Association for Computing Machinery (ACM), 2010, pp. 1479–1486. ISBN: 978-1-4503-0073-5. doi:10.1145/1830761.1830763 (cit. on p. 286).

[74] Thomas Weise (汤卫思), Alexander Podlich, and Christian Gorldt. "Solving Real-World Vehicle Routing Problems with Evolutionary Algorithms". In: *Natural Intelligence for Scheduling, Planning and Packing Problems*. Ed. by Raymond Chiong and Sandeep Dhakal. Vol. 250 of Studies in Computational Intelligence (SCI). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Sept. 2009. Chap. 2, pp. 29–53. ISSN: 1860-949X. ISBN: 978-3-642-04038-2. doi:10.1007/978-3-642-04039-9_2 (cit. on pp. 4–10).

[75] Thomas Weise (汤卫思), Alexander Podlich, Manfred Menze, and Christian Gorldt. "Optimierte Güterverkehrsplanung mit Evolutionären Algorithmen". *Industrie Management – Zeitschrift für industrielle Geschäftsprozesse* 10(3):37–40, June 2009. Berlin, Germany: GITO mbH Verlag (cit. on pp. 4–10).

[76] Thomas Weise (汤卫思), Alexander Podlich, Kai Reinhard, Christian Gorldt, and Kurt Geihs. "Evolutionary Freight Transportation Planning". In: *Applications of Evolutionary Computing (EvoWorkshops 2009): Proceedings of EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG*. Apr. 15–17, 2009, Tübingen, Baden-Württemberg, Germany. Ed. by Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni A. Caro, Anikó Ekárt, Anna Isabel Esparcia-Alcázar, Muddassar Farooq, Andreas Fink, and Penousal Machado. Vol. 5484 of Theoretical Computer Science and General Issues (LNTCS), sub-series of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Apr. 2009, pp. 768–777. ISSN: 2512-2010. doi:10.1007/978-3-642-01129-0_87 (cit. on pp. 4–10).

[77] 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. 287).

References XII

[78] 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. 255, 256).

[79] 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. 247, 248, 255, 256).

[80] 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. 271, 286, 287).

[81] 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. 167, 221, 247, 248, 253, 254).

[82] 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. 247, 248).

[83] 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. 14–17, 289).

References XIII

[84] 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, 1989, pp. 116–123. ISBN: 978-1-55860-066-9. URL: https://www.researchgate.net/publication/2527551 (visited on 2025-08-08) (cit. on pp. 197–199, 288).

[85] 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. 257–260, 287, 289).

[86] 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. 286).

[87] Anne L. Young. *Mathematical Ciphers: From Caesar to RSA*. Vol. 25 of Mathematical World. Providence, RI, USA: American Mathematical Society (AMS), 2006. ISSN: 1055-9426. ISBN: 978-0-8218-3730-6 (cit. on pp. 230–239, 288).

[88] 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. 4–10, 26–29, 257–260, 287).

[89] 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. 4–10, 26–29, 257–260, 287).

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[9,17]. 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)[5,69].

$(\mu+\lambda)$ 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.,[62]

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*[68].

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[86].

FE *Objective function evaluations* are an implementation-independent measure of runtime for optimization algorithms[6,71,73,80]. 1 FE equals to one evaluated candidate solution during the optimization process.

Glossary II

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[12].

GP Genetic Programming[37,58,69,77] 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*[7,39] 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[11,39].

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[28], 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[13].

ML Machine Learning, see, e.g.,[64]

moptipy is the *Metaheuristic Optimization in Python* library[80]. It has been used in several different research works, including[12,42–44,65,85,88,89]. Learn more at https://thomasweise.github.io/moptipy and https://thomasweise.github.io/moptipyapps.

Python The Python programming language[29,41,46,70], i.e., what you will learn about in our book[70]. Learn more at https://python.org.

Glossary III

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[8,12,36,45]. 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[63].

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.

RSA The RSA algorithm[60,87], named after its inventors Rivest, Shamir, and Adleman, is a widely used asymmetric cryptographic technique for digital signatures, key exchange, and encryption. In the latter case, it uses a public key for encryption and a private key for decryption. It is based on the mathematical complexity of factorizing large integers.

SA Simulated Annealing is a local search that sometimes accepts a worse solution[10,31,33,56]. 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.

SATLIB is a library of benchmark instances for the Maximum Satisfiability (MaxSAT) available at https://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html[27].

SGA The Standard Genetic Algorithm[5,15,22,48,49,69] 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., [84].

Glossary IV

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[3,24,40,42,71,83]. 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[24].

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[19,85]. 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[67].

$i..j$ 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)[23].

$\mathcal{NP}$-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[23,59]. 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[11,13,39] 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[23].

$\Omega(g(x))$ If $f(x) = \Omega(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $f(x) \geq c * g(x) \geq 0 \forall x \geq x_0$[34,35]. In other words, $\Omega(g(x))$ describes a lower bound for function growth.

Glossary V

$\mathcal{O}(g(x))$ If $f(x) = \mathcal{O}(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $0 \leq f(x) \leq c * g(x) \forall x \geq x_0$[4,34,35,38]. In other words, $\mathcal{O}(g(x))$ describes an upper bound for function growth.

$\Theta(g(x))$ If $f(x) = \Theta(g(x))$, then $f(x) = \mathcal{O}(g(x))$ and $f(x) = \Omega(g(x))$[34,35]. In other words, $\Theta(g(x))$ 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 $\mathbb{Z} \subset \mathbb{R}$.