Metaheuristic Optimization

2. The Structure of Optimization

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

website

Section Outline

- Goal 1: Learn how to solve different kinds of (optimization) problems.
- Goal 2: Learn to use optimization algorithms for that purpose.

Introduction

- Goal 1: Learn how to solve different kinds of (optimization) problems.
- Goal 2: Learn to use optimization algorithms for that purpose.
- Need to understand
  1. How to define an optimization problem formally.
  2. How an optimization algorithm works and what components it has.

- Goal 1: Learn how to solve different kinds of (optimization) problems.
- Goal 2: Learn to use optimization algorithms for that purpose.
- Need to understand
  1. How to define an optimization problem formally.
  2. How an metaheuristic optimization algorithm works and what components it has.

From the perspective of a programmer, we can say that an optimization problem has the following components:

1. a data type $\mathbb{X}$ for the possible solutions (candidate solutions)

From the perspective of a programmer, we can say that an optimization problem has the following components:

1. a data type $\mathbb{X}$ for the possible solutions (candidate solutions),
2. one (or multiple) functions $f \in \vec{f}$ which rate "how good" a candidate solution is

Components of an Optimization Problem

From the perspective of a programmer, we can say that an optimization problem has the following components:

1. a data type $\mathbb{X}$ for the possible solutions (candidate solutions),
2. one (or multiple) functions $f \in \vec{f}$ which rate "how good" a candidate solution is, and
3. a notion of what "good" actually means.

Solution Space

The first thing we need to know is what we want to find.

Solution Space

The first thing we need to know is what we want to find.
From the formal perspective, we say:

Definition (Solution Space $\mathbb{X}$)

The solution space $\mathbb{X}$ of an optimization problem is the set containing all elements x which could be solutions of the problem.

Solution Space

The first thing we need to know is what we want to find.
From the formal perspective, we say:

Definition (Solution Space $\mathbb{X}$)

The solution space $\mathbb{X}$ of an optimization problem is the set containing all elements x which could be solutions of the problem.

Definition (Candidate Solution x)

A candidate solution x of an optimization problem is an element of the solution space $\mathbb{X}$ of the problem, i.e., a potential solution of the problem.

From the programmer's perspective, we can say:

Listing: Solution space $\mathbb{X}$

```
public class MySolutionSpace extends Object {
  ...
  }

//or, instead, maybe a simple or primitive type
//or an array...
```

Solution Space

- You want to find the “best” way to go from the South Campus to Shanghai’s Bund?

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
    - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
    - a ordered list of trip-parts, each having a

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
    - a ordered list of trip-parts, each having a
    - start location and start time

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
    - a ordered list of trip-parts, each having a
    - start location and start time
    - end location and end time

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
    - a ordered list of trip-parts, each having a
    - start location and start time
    - end location and end time
    - vehicle to use

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
    - a ordered list of trip-parts, each having a
    - start location and start time
    - end location and end time
    - vehicle to use
    - $\Longrightarrow$ we can define one class for a trip part with one member variable for each of these "decision variables"

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
    - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
        - a ordered list of trip-parts, each having a
        - start location and start time
        - end location and end time
        - vehicle to use
        - $\Longrightarrow$ we can define one class for a trip part with one member variable for each of these "decision variables"
        - $\Longrightarrow$ we can define $\mathbb{X}$ use `List` of such elements

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
  - A candidate solution x is one instance of `List` (the data structure $\mathbb{X}$) which contains a sequence of these elements, with concrete settings of all variables, i.e., one possible solution of the problem

- You want to find the “best” way to go from the South Campus to Shanghai’s Bund?
  - solution space $\mathbb{X}$ is a *data structure* whose *instances* can completely describe such a trip
  - A candidate solution x is one instance of `List` (the data structure $\mathbb{X}$) which contains a sequence of these elements, with concrete settings of all variables, i.e., one possible solution of the problem
- You want to bake the perfect cookie?

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)
    - How much suggar? (real value $\in [0, 50]$g)

Solution Space

- You want to find the “best” way to go from the South Campus to Shanghai’s Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)
    - How much suggar? (real value $\in [0, 50]$g)
    - How much flour? (real value $\in [0, 250]$g)

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)
    - How much suggar? (real value $\in [0, 50]$g)
    - How much flour? (real value $\in [0, 250]$g)
    - How much honey? (real value $\in [0, 50]$g)

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)
    - How much suggar? (real value $\in [0, 50]$g)
    - How much flour? (real value $\in [0, 250]$g)
    - How much honey? (real value $\in [0, 50]$g)
    - How much chocolate? (real value $\in [0, 50]$g)

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
    - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
        - How much butter? (real value $\in [0, 50]$g)
        - How much suggar? (real value $\in [0, 50]$g)
        - How much flour? (real value $\in [0, 250]$g)
        - How much honey? (real value $\in [0, 50]$g)
        - How much chocolate? (real value $\in [0, 50]$g)
        - ...

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)
    - How much suggar? (real value $\in [0, 50]$g)
    - How much flour? (real value $\in [0, 250]$g)
    - How much honey? (real value $\in [0, 50]$g)
    - How much chocolate? (real value $\in [0, 50]$g)
    - ...
    - $\Longrightarrow$ we *could* define $\mathbb{X}$ as vector of real numbers, i.e., `double[]` in `Java`, where each element of the vector represents the amount to use of one ingredient

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)
    - How much suggar? (real value $\in [0, 50]$g)
    - How much flour? (real value $\in [0, 250]$g)
    - How much honey? (real value $\in [0, 50]$g)
    - How much chocolate? (real value $\in [0, 50]$g)
    - ...
    - $\Longrightarrow$ we *could* define $\mathbb{X}$ as vector of real numbers, i.e., `double[]` in `Java`, where each element of the vector represents the amount to use of one ingredient
  - A candidate solution x is then one concrete `double[]` with specific values for each ingredient

Solution Space

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the perfect cookie?
  - solution space $\mathbb{X}$ is a data structure whose instances can completely describe a cookie recipe!
    - How much butter? (real value $\in [0, 50]$g)
    - How much suggar? (real value $\in [0, 50]$g)
    - How much flour? (real value $\in [0, 250]$g)
    - How much honey? (real value $\in [0, 50]$g)
    - How much chocolate? (real value $\in [0, 50]$g)
    - ...
    - $\Longrightarrow$ we *could* define $\mathbb{X}$ as vector of real numbers, i.e., `double[]` in `Java`, where each element of the vector represents the amount to use of one ingredient
  - A candidate solution x is then one concrete `double[]` with specific values for each ingredient
- solution space $=$ data structure for candidate solution

Objective Function

OK. We know what we want to find. Now we need to know whether the things we find are good or not...

OK. We know what we want to find. Now we need to know whether the things we find are good or not...

Normally, there is not just good and bad, but many different degrees of solution quality.

Objective Function

OK. We know what we want to find. Now we need to know whether the things we find are good or not. . .

Normally, there is not just good and bad, but many different degrees of solution quality.

From the formal perspective, we say:

Definition (Objective Function f)

An objective function $f : \mathbb{X} \mapsto \mathbb{R}$ is a (mathematical) function which is subject to optimization

Objective Function

Now we need to know whether the things we find are good or not...
Normally, there is not just good and bad, but many different degrees of solution quality.
From the formal perspective, we say:

Definition (Objective Function f)

An objective function $f : \mathbb{X} \mapsto \mathbb{R}$ is a (mathematical) function which is subject to optimization

- Can compute a (real) solution quality value $f(x) \in \mathbb{R}$ for a given candidate solution $x \in \mathbb{X}$

Objective Function

Now we need to know whether the things we find are good or not...
Normally, there is not just good and bad, but many different degrees of solution quality.
From the formal perspective, we say:

Definition (Objective Function f)

An objective function $f : \mathbb{X} \mapsto \mathbb{R}$ is a (mathematical) function which is subject to optimization

- Can compute a (real) solution quality value $f(x) \in \mathbb{R}$ for a given candidate solution $x \in \mathbb{X}$
- Usually subject to minimization $f(x_1) < f(x_2)$ means that x_1 is better than x_2

Objective Function

From the formal perspective, we say:

Definition (Objective Function f)

An objective function $f : \mathbb{X} \mapsto \mathbb{R}$ is a (mathematical) function which is subject to optimization

- Can compute a (real) solution quality value $f(x) \in \mathbb{R}$ for a given candidate solution $x \in \mathbb{X}$
- Usually subject to minimization $f(x_1) < f(x_2)$ means that x_1 is better than x_2
- Not necessarily a function as you know it from Maths like $f(x) = x^2 + \dots$, but may be arbitrary complex, involve complicated simulations, etc.

From the programmer's perspective, we can say:

Listing: Objective Function f

```
public interface IObjectiveFunction<X> {

  public abstract double compute(final X x);
}
```

From the programmer's perspective, we can say:

Listing: Objective Function f

```
public interface IObjectiveFunction<X> {

  public abstract double compute(final X x);
}
```

- the *generic* parameter `X` stands for the solution space data structure $\mathbb{X}$

From the programmer's perspective, we can say:

Listing: Objective Function f

```
public interface IObjectiveFunction<X> {

  public abstract double compute(final X x);
}
```

- the *generic* parameter `X` stands for the solution space data structure $\mathbb{X}$
- the function `compute` implements $f(x)$ where x is an instance of `X`

From the programmer's perspective, we can say:

Listing: Objective Function f

```
public interface IObjectiveFunction<X> {

  public abstract double compute(final X x);
}
```

- the *generic* parameter `X` stands for the solution space data structure $\mathbb{X}$
- the function `compute` implements $f(x)$ where x is an instance of `X`
- as you see: $f(x)$ could be anything, could be deterministic or randomized, a simple formula, or involve running large programs like simulations

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?

Objective Function

- You want to find the “best” way to go from the South Campus to Shanghai’s Bund?
    - What does *best* mean?

Objective Function

- You want to find the “best” way to go from the South Campus to Shanghai’s Bund?
    - What does *best* mean?
        - cheapest?

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - What does *best* mean?
    - cheapest? $\Longrightarrow$ go through list of tour-parts and add up their costs, return total cost

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - What does *best* mean?
    - cheapest? $\Longrightarrow$ go through list of tour-parts and add up their costs, return total cost
    - fastest?

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - What does *best* mean?
    - cheapest? $\Longrightarrow$ go through list of tour-parts and add up their costs, return total cost
    - fastest? $\Longrightarrow$ go through list of tour-parts and add up the travel and waiting times, return total travel time

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - What does *best* mean?
    - cheapest? $\Longrightarrow$ go through list of tour-parts and add up their costs, return total cost
    - fastest? $\Longrightarrow$ go through list of tour-parts and add up the travel and waiting times, return total travel time
    - a mixture of both?

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
  - What does *best* mean?
    - cheapest? $\Longrightarrow$ go through list of tour-parts and add up their costs, return total cost
    - fastest? $\Longrightarrow$ go through list of tour-parts and add up the travel and waiting times, return total travel time
    - a mixture of both? $\Longrightarrow$ compute costs and time, return maybe *"10*(runtime in hours) + (cost in RMB)"*

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the *perfect* cookie?

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the *perfect* cookie?
  - let's say perfect $\equiv$ tastes best?

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the *perfect* cookie?
  - let's say perfect $\equiv$ tastes best? $\Longrightarrow$ for each candidate cookie receipe

Objective Function

- You want to find the “best” way to go from the South Campus to Shanghai’s Bund?
- You want to bake the *perfect* cookie?
    - let’s say perfect $\equiv$ tastes best? $\Longrightarrow$ for each candidate cookie receipe
        - print the receipe

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the *perfect* cookie?
  - let's say perfect $\equiv$ tastes best? $\Longrightarrow$ for each candidate cookie receipe
    - print the receipe
    - bake the cookie

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the *perfect* cookie?
  - let's say perfect $\equiv$ tastes best? $\Longrightarrow$ for each candidate cookie receipe
    - print the receipe
    - bake the cookie
    - eat the cookie

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the *perfect* cookie?
  - let's say perfect $\equiv$ tastes best? $\Longrightarrow$ for each candidate cookie receipe
    - print the receipe
    - bake the cookie
    - eat the cookie
    - rate its taste from 0 to 10

Objective Function

- You want to find the "best" way to go from the South Campus to Shanghai's Bund?
- You want to bake the *perfect* cookie?
  - let's say perfect $\equiv$ tastes best? $\Longrightarrow$ for each candidate cookie receipe
    - print the receipe
    - bake the cookie
    - eat the cookie
    - rate its taste from 0 to 10
    - Objective function with human interaction! Why not!

First steps when solving an optimization problem:

1. Understand the situation and all involved objects, entities, laws, constraints, etc

First steps when solving an optimization problem:

1. Understand the situation and all involved objects, entities, laws, constraints, etc
2. Define what possible solutions look like, i.e., give a data structure (programmer's point of view) or space $\mathbb{X}$ (formal point of view)

First steps when solving an optimization problem:

1. Understand the situation and all involved objects, entities, laws, constraints, etc
2. Define what possible solutions look like, i.e., give a data structure (programmer's point of view) or space $\mathbb{X}$ (formal point of view)
3. Define a function which rates how good a candidate solution is, how close it comes to what we really want as solution.

First steps when solving an optimization problem:

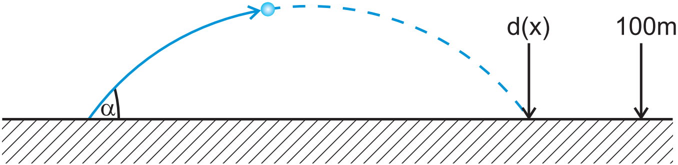
1. Understand the situation and all involved objects, entities, laws, constraints, etc
2. Define what possible solutions look like, i.e., give a data structure (programmer's point of view) or space $\mathbb{X}$ (formal point of view)
3. Define a function which rates how good a candidate solution is, how close it comes to what we really want as solution.
4. These steps are independent of how we will finally solve the problem

Solution Space and Objective

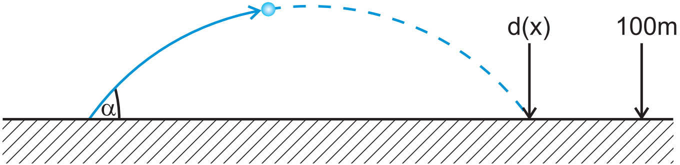
First steps when solving an optimization problem:

1. Understand the situation and all involved objects, entities, laws, constraints, etc
2. Define what possible solutions look like, i.e., give a data structure (programmer's point of view) or space $\mathbb{X}$ (formal point of view)
3. Define a function which rates how good a candidate solution is, how close it comes to what we really want as solution.
4. These steps are independent of how we will finally solve the problem
5. If you develop an optimization software for a client, it is very important to discuss these issues with the client and to formally write them down on paper! The client often does not know exactly what he/she wants AND you may misunderstand him/her...

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15^\circ$ angle) so that it lands exactly 100m away?

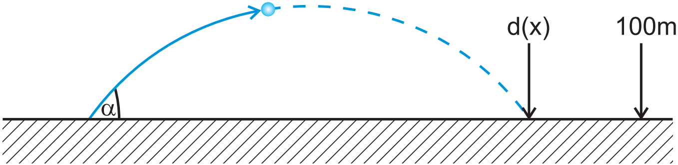


Which is the best velocity x with which I should throw a stone (in an $\alpha = 15°$ angle) so that it lands exactly 100m away?



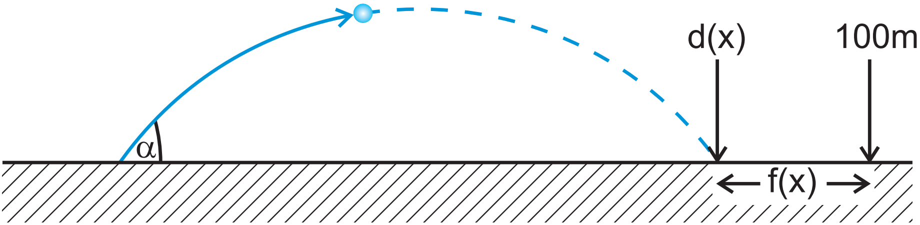
- Solution Space:
- Objective Function:

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15°$ angle) so that it lands exactly 100m away?



- Solution Space: $\mathbb{X} = \mathbb{R}^+$
- Objective Function:

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15°$ angle) so that it lands exactly 100m away?



- Solution Space: $\mathbb{X} = \mathbb{R}^+$
- Objective Function: Minimize $f(x) = |d(x) - 100m|$

  $d(x) = \frac{x^2}{g} \sin 2\alpha \approx 0.051\,\mathrm{s^2/m} * x^2$

Example: Stone's Throw

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15°$ angle) so that it lands exactly 100m away?

- Solution Space: $\mathbb{X} = \mathbb{R}^+$
- Objective Function: Minimize $f(x) = |d(x) - 100m|$
  $d(x) = \frac{x^2}{g} \sin 2\alpha \approx 0.051 \mathsf{s}^2/\mathsf{m} * x^2$

Listing: Blueprint of Objective Function for Stone's Throw Probleml

```
public final class StoneThrowObjective implements IObjectiveFunction<Number> {

  public final double compute(final Number x) {
    final double v = x.doubleValue();
    final double d = (((v * v) / 9.80665d) * Math.sin(((2.0d * 15.0d) / 180.0d) *
        Math.PI));
    return Math.abs(100d - d);
  }
}
```

Example: Stone's Throw

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15°$ angle) so that it lands exactly 100m away?

- Solution Space: $\mathbb{X} = \mathbb{R}^+$
- Objective Function: Minimize $f(x) = |d(x) - 100m|$
  $d(x) = \frac{x^2}{g} \sin 2\alpha \approx 0.051 \mathrm{s}^2/\mathrm{m} * x^2$
- Actually... not necessary!

Listing: Blueprint of Objective Function for Stone's Throw Probleml

```
public final class StoneThrowObjective implements IObjectiveFunction<Number> {

  public final double compute(final Number x) {
    final double v = x.doubleValue();
    final double d = (((v * v) / 9.80665d) * Math.sin(((2.0d * 15.0d) / 180.0d) *
        Math.PI));
    return Math.abs(100d - d);
  }
}
```

Example: Stone's Throw

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15°$ angle) so that it lands exactly 100m away?

- Solution Space: $\mathbb{X} = \mathbb{R}^+$
- Objective Function: Minimize $f(x) = |d(x) - 100m|$
  $d(x) = \frac{x^2}{g}\sin 2\alpha \approx 0.051 \mathsf{s^2/m} * x^2$
- Actually... not necessary!
- Problem can easily be solved: minimum of f known, equation is simple

Example: Stone's Throw

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15^\circ$ angle) so that it lands exactly 100m away?

- Solution Space: $\mathbb{X} = \mathbb{R}^+$
- Objective Function: Minimize $f(x) = |d(x) - 100m|$
  $d(x) = \frac{x^2}{g} \sin 2\alpha \approx 0.051 \mathrm{s^2/m} * x^2$
- Actually… not necessary!
- Problem can easily be solved: minimum of f known, equation is simple
- No optimization algorithm needed.

Which is the best velocity x with which I should throw a stone (in an $\alpha = 15^\circ$ angle) so that it lands exactly 100m away?

- Solution Space: $\mathbb{X} = \mathbb{R}^+$
- Objective Function: Minimize $f(x) = |d(x) - 100m|$

  $d(x) = \frac{x^2}{g} \sin 2\alpha \approx 0.051\mathsf{s}^2/\mathsf{m} * x^2$
- Actually... not necessary!
- Problem can easily be solved: minimum of f known, equation is simple
- No optimization algorithm needed.
- But what if the stone is an irregularly shaped object (like a chair) and we also include air drag, gravitation, wind, limit forces on the stone-throwing arm, costs for electricity of moving the joints, wear of joins, imprecision of movements, make α variable, ...?

Example: Traveling Salesman Problem

A salesman wants to visit n cities in the shortest possible time. No city should be visited twice and he wants arrive back at the origin by the end of the tour [1–3].

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

A salesman wants to visit n cities in the shortest possible time. No city should be visited twice and he wants arrive back at the origin by the end of the tour [1–3].

Definition (Traveling Salesman Problem)

The goal of the Traveling Salesman Problem (TSP) is to find a cyclic path of minimum total weight which visits all vertices of a weighted graph. [1, 2, 4, 5]

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space:

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\left\{\text{Beijing, Chengdu, Guangzhou, Hefei, Shanghai}\right\}$

Example: Traveling Salesman Problem

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\,\{\text{Beijing}, \text{Chengdu}, \text{Guangzhou}, \text{Hefei}, \text{Shanghai}\}$

  $\mathbf{\Pi}(Z) =$ set of all permutations of the elements of the given set Z

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\left\{\text{Beijing, Chengdu, Guangzhou, Hefei, Shanghai}\right\}$
  $\mathbf{\Pi}(Z) =$ set of all permutations of the elements of the given set Z
  Example: $\mathbf{\Pi}(\{123\}) = \{(1,2,3);(1,3,2);(2,1,3);(2,3,1);(3,1,2);(3,2,1)\}$

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\left\{\text{Beijing, Chengdu, Guangzhou, }\underline{\text{Hefei}}\text{, Shanghai}\right\}$

  Let us assume that the tour always starts and ends in Hefei.

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\,\{\text{Beijing}, \text{Chengdu}, \text{Guangzhou}, \text{Shanghai}\}$
  Let us assume that the tour always starts and ends in Hefei.
  Then, we can simply leave it away.

Example: Traveling Salesman Problem

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\left\{\text{Beijing}, \text{Chengdu}, \text{Guangzhou}, \text{Shanghai}\right\}$

  Let us assume that the tour always starts and ends in Hefei.

  Then, we can simply leave it away $\Rightarrow |\mathbb{X}|$ gets smaller!

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\left\{\text{Beijing}, \text{Chengdu}, \text{Guangzhou}, \text{Shanghai}\right\}$
  Let us assume that the tour always starts and ends in Hefei.
  Then, we can simply leave it away $\Rightarrow |\mathbb{X}|$ gets smaller! Good!

Example: Traveling Salesman Problem

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\left\{\text{Beijing}, \text{Chengdu}, \text{Guangzhou}, \text{Shanghai}\right\}$

  Let us assume that the tour always starts and ends in Hefei.

  Then, we can simply leave it away $\Rightarrow |\mathbb{X}|$ gets smaller! Good!
- Objective Function:

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\,\{\text{Beijing}, \text{Chengdu}, \text{Guangzhou}, \text{Shanghai}\}$
  Let us assume that the tour always starts and ends in Hefei.
  Then, we can simply leave it away $\Rightarrow |\mathbb{X}|$ gets smaller! Good!
- Objective Function: Minimize $f(x) = dist(\text{Hefei}, x[0]) + \sum_{i=0}^{2} dist(x[i], x[i+1]) + dist(x[3], \text{Hefei})$

Example: Traveling Salesman Problem

w(a,b)	Shanghai	Hefei	Guangzhou	Chengdu
Beijing	1244 km	1044 km	2174 km	1854 km
Chengdu	2095 km	1615 km	1954 km	
Guangzhou	1529 km	1257 km		
Hefei	472 km			

- Solution Space: $\mathbb{X} = \mathbf{\Pi}\left\{\text{Beijing}, \text{Chengdu}, \text{Guangzhou}, \text{Shanghai}\right\}$

  Let us assume that the tour always starts and ends in Hefei.

  Then, we can simply leave it away $\Rightarrow |\mathbb{X}|$ gets smaller! Good!

- Objective Function: Minimize $f(x) = \begin{array}{l} dist(\text{Hefei}, x[0]) + \\ \sum_{i=0}^{2} dist(x[i], x[i+1]) + \\ dist(x[3], \text{Hefei}) \end{array}$

  This formula is not so nice: we cannot simply "solve" it for a minimum $x \in \mathbb{X}$.

Listing: Solution space $\mathbb{X}$

```java
public final class ChinaTSPObjective implements IObjectiveFunction<int[]> {

  public final double compute(final int[] x) {
    double dist;

    dist = ChinaTSPObjective.distance(ChinaTSPObjective.HEFEI, x[0]);

    for (int i = 1; i < x.length; i++) {
      dist += ChinaTSPObjective.distance(x[i - 1], x[i]);
    }

    return (dist + ChinaTSPObjective.distance(x[x.length - 1], ChinaTSPObjective.HEFEI));
  }
}
```

Example: Traveling Salesman Problem

- In a TSP, we cannot directly compute the right solution

Example: Traveling Salesman Problem

- In a TSP, we cannot directly compute the right solution
- Simply test all possible solutions...

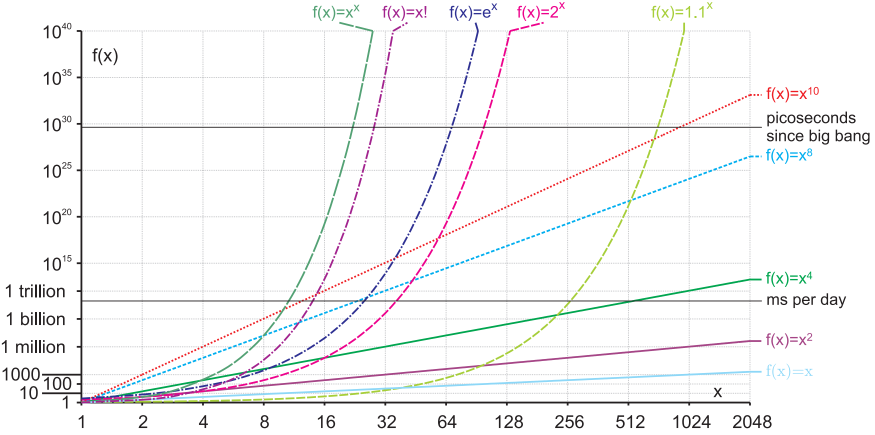
Example: Traveling Salesman Problem

- In a TSP, we cannot directly compute the right solution
- Simply test all possible solutions...

x_1	Hefei	$\rightarrow$	Beijing	$\rightarrow$	Chengdu	$\rightarrow$	Guangzhou	$\rightarrow$	Shanghai	$\rightarrow$	Hefei	7425km
x_2	Hefei	$\rightarrow$	Beijing	$\rightarrow$	Chengdu	$\rightarrow$	Shanghai	$\rightarrow$	Guangzhou	$\rightarrow$	Hefei	7566km
x_3	Hefei	$\rightarrow$	Beijing	$\rightarrow$	Guangzhou	$\rightarrow$	Chengdu	$\rightarrow$	Shanghai	$\rightarrow$	Hefei	8311km
x_4	Hefei	$\rightarrow$	Beijing	$\rightarrow$	Guangzhou	$\rightarrow$	Shanghai	$\rightarrow$	Chengdu	$\rightarrow$	Hefei	7886km
x_5	Hefei	$\rightarrow$	Beijing	$\rightarrow$	Shanghai	$\rightarrow$	Chengdu	$\rightarrow$	Guangzhou	$\rightarrow$	Hefei	7381km
x_6	Hefei	$\rightarrow$	Beijing	$\rightarrow$	Shanghai	$\rightarrow$	Guangzhou	$\rightarrow$	Chengdu	$\rightarrow$	Hefei	6815km
x_7	Hefei	$\rightarrow$	Chengdu	$\rightarrow$	Beijing	$\rightarrow$	Guangzhou	$\rightarrow$	Shanghai	$\rightarrow$	Hefei	8787km
x_8	Hefei	$\rightarrow$	Chengdu	$\rightarrow$	Beijing	$\rightarrow$	Shanghai	$\rightarrow$	Guangzhou	$\rightarrow$	Hefei	7857km
x_9	Hefei	$\rightarrow$	Chengdu	$\rightarrow$	Guangzhou	$\rightarrow$	Beijing	$\rightarrow$	Shanghai	$\rightarrow$	Hefei	8602km
x_{10}	Hefei	$\rightarrow$	Chengdu	$\rightarrow$	Shanghai	$\rightarrow$	Beijing	$\rightarrow$	Guangzhou	$\rightarrow$	Hefei	8743km
x_{11}	Hefei	$\rightarrow$	Guangzhou	$\rightarrow$	Beijing	$\rightarrow$	Chengdu	$\rightarrow$	Shanghai	$\rightarrow$	Hefei	8637km
x_{12}	Hefei	$\rightarrow$	Guangzhou	$\rightarrow$	Chengdu	$\rightarrow$	Beijing	$\rightarrow$	Shanghai	$\rightarrow$	Hefei	7566km

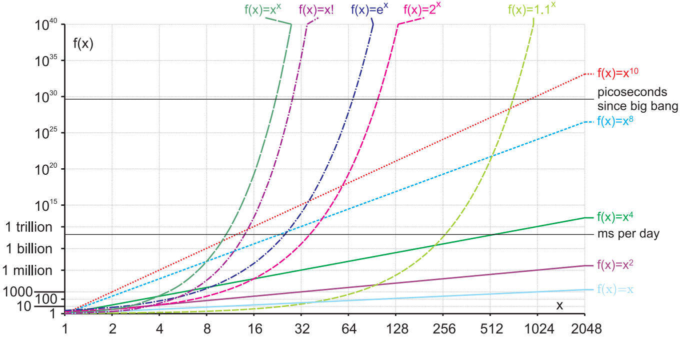
Example: Traveling Salesman Problem

- Simply test all possible solutions. . . ??
- Size of solution space: $|\mathbb{X}| = \frac{1}{2}(n-1)!$ ⇐ factorial, not exclamation mark



(Figure inspired by [6])

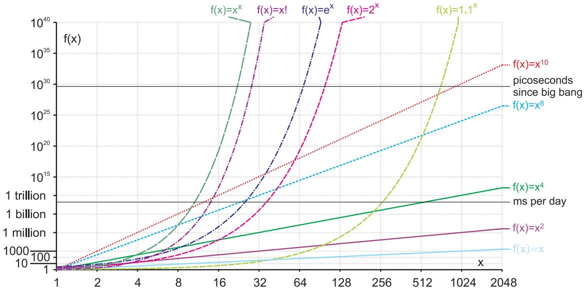
- Simply test all possible solutions. . . ??
- Size of solution space: $|\mathbb{X}| = \frac{1}{2}(n-1)!$
- Algorithm which is better than this *exhaustive enumeration* needed



(Figure inspired by [6])

Example: Traveling Salesman Problem

- Simply test all possible solutions. . . ??
- Size of solution space: $|\mathbb{X}| = \frac{1}{2}(n-1)!$
- Algorithm which is better than this *exhaustive enumeration* needed
- You will learn quite a lot of these in this lecture!



(Figure inspired by [6])

- What could be suitable solution spaces and objectives for
  1. Bin Packing [7]
  2. Circuit Layout [8, 9]
  3. Find the roots of a function $g(x)$ [10–13]
  4. Shortest Path / Routing [14–16]
  5. Find mathematical formula fitting to given data [17–19]
  6. Job Shop Scheduling [18, 20]
  7. Stock Prediction [21–24]
  8. Truss Optimization [25–27]
  9. Medical Classification [28]
  10. Airplane Wing Design [29–32]

Further Examples

- Antenna design [33–41]
- Analog Electrical Circuit Design [42–45]
- Interactive Optimization [46–51]

Section Outline

1. Introduction
2. Optimization Problem
3. What is Good?
4. Metaheuristics
5. Putting it Together
6. Summary

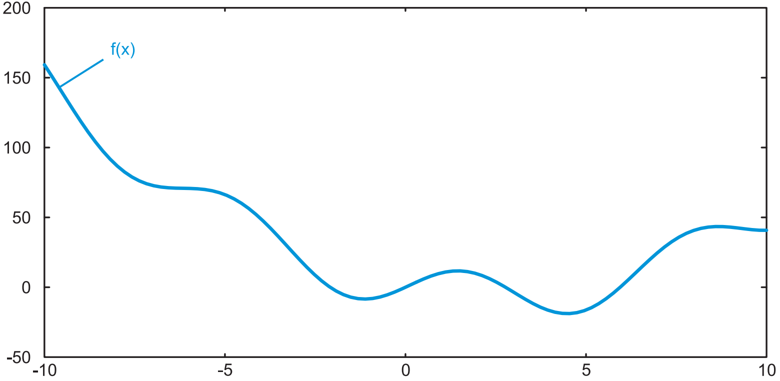
What does "good" mean?

- We want to find the good solutions for such problems.

What does "good" mean?

- We want to find the good solutions for such problems.
- But what does "good" mean?

- Assume that the objective function f is a steady, continuous, and differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ with a single real-valued parameter x.



What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Assume that the objective function f is a steady, continuous, and differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ with a single real-valued parameter x.
- In this case high school mathematics tells us what to do and what we want:

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Assume that the objective function f is a steady, continuous, and differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ with a single real-valued parameter x.
- In this case high school mathematics tells us what to do and what we want:
- We want the extrema, the minima and maxima of f

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Assume that the objective function f is a steady, continuous, and differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ with a single real-valued parameter x.
- In this case high school mathematics tells us what to do and what we want:
- We want the extrema, the minima and maxima of f
- If $\mathbb{X} \subseteq \mathbb{R}$, then for every local optimum $x^{\star}$ of f, $f'(x^{\star}) = 0$ holds.

What does “good” mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

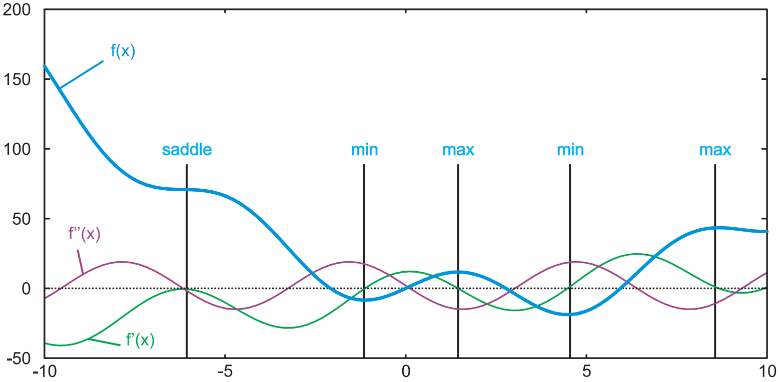
- Assume that the objective function f is a steady, continuous, and differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ with a single real-valued parameter x.
- In this case high school mathematics tells us what to do and what we want:
- We want the extrema, the minima and maxima of f
- If $\mathbb{X} \subseteq \mathbb{R}$, then for every local optimum $x^\star$ of f, $f'(x^\star) = 0$ holds.
- $(f'(x^\star) = 0) \land (f''(x^\star) > 0) \Rightarrow x^\star$ is a local minimum
- $(f'(x^\star) = 0) \land (f''(x^\star) < 0) \Rightarrow x^\star$ is a local maximum

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Assume that the objective function f is a steady, continuous, and differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ with a single real-valued parameter x.
- In this case high school mathematics tells us what to do and what we want:
- We want the extrema, the minima and maxima of f
- If $\mathbb{X} \subseteq \mathbb{R}$, then for every local optimum $x^\star$ of f, $f'(x^\star) = 0$ holds.
- $(f'(x^\star) = 0) \wedge (f''(x^\star) > 0) \Rightarrow x^\star$ is a local minimum
- $(f'(x^\star) = 0) \wedge (f''(x^\star) < 0) \Rightarrow x^\star$ is a local maximum
- sign change of f' from $-$ to $+ \Rightarrow x^\star$ is a local minimum
- sign change of f' from $+$ to $- \Rightarrow x^\star$ is a local maximum

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Assume that the objective function f is a steady, continuous, and differentiable function $f : \mathbb{R} \mapsto \mathbb{R}$ with a single real-valued parameter x.



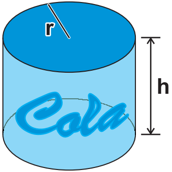
Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

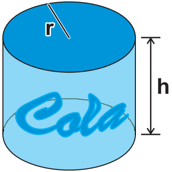
$$V(r,h) \quad = \quad \pi r^2 h \text{...... volume of cylinder} \tag{11}$$



Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint}$$



(11)

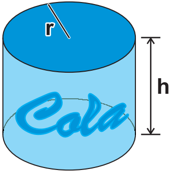
Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \quad (2)$$

$$A(r,h) = 2\pi r^2 + 2\pi r h \text{......the surface: bottom, top, and hull}$$



(11)

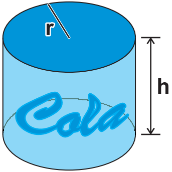
Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{ ...... volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{ ...... this volume is given: constraint} \quad (2)$$

$$A(r,h) = 2\pi r^2 + 2\pi r h \text{ ...... the surface: bottom, top, and hull} \quad (3)$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{ ...... solution space: } r \text{ and } h \text{ define a cylinder}$$



(11)

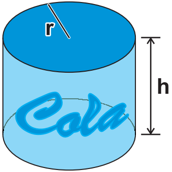
Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \quad (2)$$

$$A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \quad (3)$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2$$



(11)

Example: Cheap Cola Can

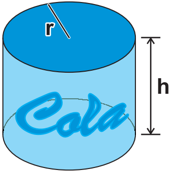
Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \tag{1}$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \tag{2}$$

$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \tag{3}$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2$$



(11)

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

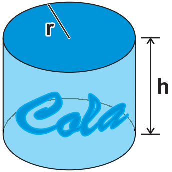
$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \quad (2)$$

$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \quad (3)$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 \quad (4)$$

$$h = \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h$$

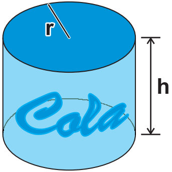


(11)

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$
\begin{aligned}
V(r,h) &= \pi r^2 h \text{......volume of cylinder} && (1)\\
V_d &= 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} && (2)\\
f(r,h) = A(r,h) &= 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} && (3)\\
\mathbb{X} &= (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 && (4)\\
h &= \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h && (5)\\
f(r) &= 2\pi r^2 + 2 * V_d r^{-1} \text{......5 in 3}
\end{aligned}
$$



(11)

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{ ...... volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{ ...... this volume is given: constraint} \quad (2)$$

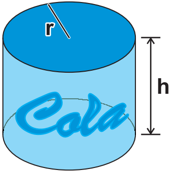
$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{ ...... objective function: material cost} \approx \text{surface} \quad (3)$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{ ...... solution space: two dimensional real vectors } \mathbb{R}^2 \quad (4)$$

$$h = \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{ ...... resolve 1 and 2 for } h \quad (5)$$

$$f(r) = 2\pi r^2 + 2 * V_d r^{-1} \text{ ...... 5 in 3} \quad (6)$$

$$f'(r) = 4\pi r + 2V_d * -r^{-2} \text{ ...... first derivative of } f \text{ for } r$$



(11)

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \quad (2)$$

$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \quad (3)$$

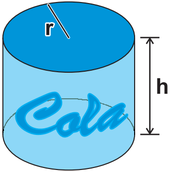
$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 \quad (4)$$

$$h = \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h \quad (5)$$

$$f(r) = 2\pi r^2 + 2 * V_d r^{-1} \text{......5 in 3} \quad (6)$$

$$f'(r) = 4\pi r + 2V_d * -r^{-2} \text{......first derivative of } f \text{ for } r \quad (7)$$

$$0 = 4\pi r - \frac{2V_d}{r^2} \text{......solve for extrema}$$



(12)

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \tag{1}$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \tag{2}$$

$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \tag{3}$$

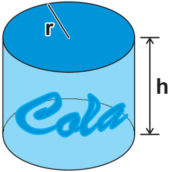
$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 \tag{4}$$

$$h = \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h \tag{5}$$

$$f(r) = 2\pi r^2 + 2 * V_d r^{-1} \text{......5 in 3} \tag{6}$$

$$f'(r) = 4\pi r + 2V_d * -r^{-2} \text{......first derivative of } f \text{ for } r \tag{7}$$

$$2V_d = 4\pi r^3 \text{......still solving...}$$

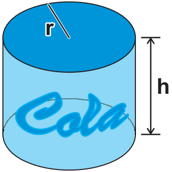


(12)

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$\begin{aligned}
V(r,h) &= \pi r^2 h \text{......volume of cylinder} && (1)\\
V_d &= 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} && (2)\\
f(r,h) = A(r,h) &= 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} && (3)\\
\mathbb{X} &= (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 && (4)\\
h &= \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h && (5)\\
f(r) &= 2\pi r^2 + 2 * V_d r^{-1} \text{......5 in 3} && (6)\\
f'(r) &= 4\pi r + 2V_d * -r^{-2} \text{......first derivative of } f \text{ for } r && (7)\\
r^\star &\approx \sqrt[3]{\frac{2 * 0.003\,55\text{m}^3}{4\pi}} \approx 0.038\text{m} \approx 3.8\text{cm} \text{......OK, } r \text{ is found}
\end{aligned}$$



(11)

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \quad (2)$$

$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \quad (3)$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 \quad (4)$$

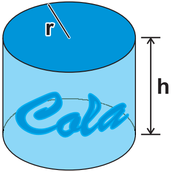
$$h = \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h \quad (5)$$

$$f(r) = 2\pi r^2 + 2 * V_d r^{-1} \text{......5 in 3} \quad (6)$$

$$f'(r) = 4\pi r + 2V_d * -r^{-2} \text{......first derivative of } f \text{ for } r \quad (7)$$

$$r^\star \approx 3.8\text{cm} \text{......OK, } r \text{ is found} \quad (8)$$

$$h^\star = \approx 7.7\text{cm} \text{......now solve for } h \text{ using 5}$$



(11)

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \quad (2)$$

$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \quad (3)$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 \quad (4)$$

$$h = \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h \quad (5)$$

$$f(r) = 2\pi r^2 + 2 * V_d r^{-1} \text{......5 in 3} \quad (6)$$

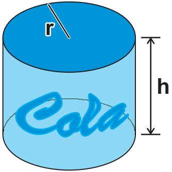
$$f'(r) = 4\pi r + 2V_d * -r^{-2} \text{......first derivative of } f \text{ for } r \quad (7)$$

$$r^\star \approx 3.8\text{cm} \text{......OK, } r \text{ is found} \quad (8)$$

$$h^\star \approx 7.7\text{cm} \text{......now solve for } h \text{ using 5} \quad (9)$$

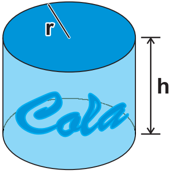
$$f''(r) = 4\pi + 4V_d r^{-3} \text{......maximum or minimum?}$$

(11)



Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.



$$
\begin{aligned}
V(r,h) &= \pi r^2 h \ldots\ldots \text{volume of cylinder} && (1)\\
V_d &= 355\text{mL} = 0.355 * 0.01\text{m}^3 \ldots\ldots \text{this volume is given: constraint} && (2)\\
f(r,h) = A(r,h) &= 2\pi r^2 + 2\pi r h \ldots\ldots \text{objective function: material cost} \approx \text{surface} && (3)\\
\mathbb{X} &= (r,h) : r,h \in \mathbb{R}^+ \ldots\ldots \text{solution space: two dimensional real vectors } \mathbb{R}^2 && (4)\\
h &= \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \ldots\ldots \text{resolve 1 and 2 for } h && (5)\\
f(r) &= 2\pi r^2 + 2 * V_d r^{-1} \ldots\ldots \text{5 in 3} && (6)\\
f'(r) &= 4\pi r + 2V_d * -r^{-2} \ldots\ldots \text{first derivative of } f \text{ for } r && (7)\\
r^\star &\approx 3.8\text{cm} \ldots\ldots \text{OK, } r \text{ is found} && (8)\\
h^\star &\approx 7.7\text{cm} \ldots\ldots \text{now solve for } h \text{ using 5} && (9)\\
f''(r) &= 4\pi + 4V_d r^{-3} \ldots\ldots \text{maximum or minimum?} && (10)\\
f''(r^\star) &> 0 \Rightarrow \text{candidate solution } x^\star = (r^\star, h^\star) \text{ is minimum} && (11)
\end{aligned}
$$

Example: Cheap Cola Can

Task: Construct a cylindrical cola can capable of holding 355mL with the minimum material costs.

$$V(r,h) = \pi r^2 h \text{......volume of cylinder} \quad (1)$$

$$V_d = 355\text{mL} = 0.355 * 0.01\text{m}^3 \text{......this volume is given: constraint} \quad (2)$$

$$f(r,h) = A(r,h) = 2\pi r^2 + 2\pi r h \text{......objective function: material cost} \approx \text{surface} \quad (3)$$

$$\mathbb{X} = (r,h) : r,h \in \mathbb{R}^+ \text{......solution space: two dimensional real vectors } \mathbb{R}^2 \quad (4)$$

$$h = \frac{V_d}{\pi r^2} = \frac{0.003\,55\text{m}^3}{\pi r^2} \text{......resolve 1 and 2 for } h \quad (5)$$

$$f(r) = 2\pi r^2 + 2 * V_d r^{-1} \text{......5 in 3} \quad (6)$$

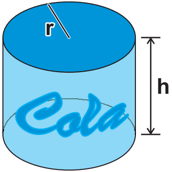
$$f'(r) = 4\pi r + 2V_d * -r^{-2} \text{......first derivative of } f \text{ for } r \quad (7)$$

$$r^\star \approx 3.8\text{cm} \text{......OK, } r \text{ is found} \quad (8)$$

$$h^\star \approx 7.7\text{cm} \text{......now solve for } h \text{ using 5} \quad (9)$$

$$f''(r) = 4\pi + 4V_d r^{-3} \text{......maximum or minimum?} \quad (10)$$

$$f''(r^\star) > 0 \Rightarrow \text{candidate solution } x^\star = (r^\star, h^\star) \text{ is minimum} \quad (11)$$



Problem solved with high school maths – no optimization algorithm needed.

What does “good” mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

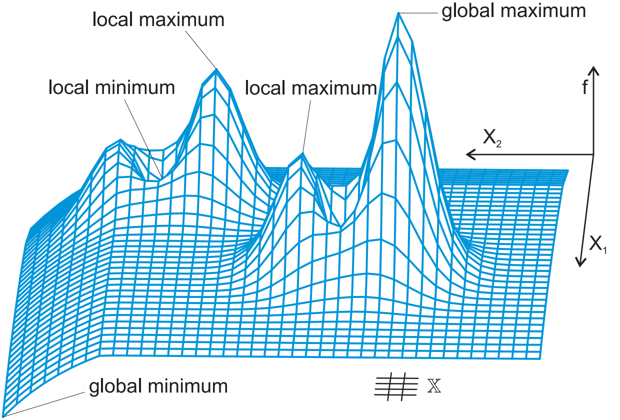
- Differentiation only possible for *differentiable* objective functions

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Differentiation only possible for *differentiable* objective functions
- Differentiation a bit more complicated for $\mathbb{X} \subseteq \mathbb{R}^n$ and large n...

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Differentiation only possible for *differentiable* objective functions
- Differentiation a bit more complicated for $\mathbb{X} \subseteq \mathbb{R}^n$ and large n...



What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Differentiation only possible for *differentiable* objective functions
- Differentiation a bit more complicated for $\mathbb{X} \subseteq \mathbb{R}^n$ and large n...
- In many cases, we have to live without the formulas from the previous slide

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Differentiation only possible for *differentiable* objective functions
- Differentiation a bit more complicated for $\mathbb{X} \subseteq \mathbb{R}^n$ and large n...
- In many cases, we have to live without the formulas from the previous slide
- Even if we can differentiate, we then need to solve the resulting equation, which is also not always analytically possible

What does “good” mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Differentiation only possible for *differentiable* objective functions
- Differentiation a bit more complicated for $\mathbb{X} \subseteq \mathbb{R}^n$ and large n...
- In many cases, we have to live without the formulas from the previous slide
- Even if we can differentiate, we then need to solve the resulting equation, which is also not always analytically possible
- Combinatorial optimization: Objective functions don't have real-valued arguments (remember the car setup and TSP problem. . .)

What does "good" mean? (for $\mathbb{X} \subseteq \mathbb{R}$)

- Differentiation only possible for *differentiable* objective functions
- Differentiation a bit more complicated for $\mathbb{X} \subseteq \mathbb{R}^n$ and large n...
- In many cases, we have to live without the formulas from the previous slide
- Even if we can differentiate, we then need to solve the resulting equation, which is also not always analytically possible
- Combinatorial optimization: Objective functions don't have real-valued arguments (remember the car setup and TSP problem...)
- Other example: Genetic Programming [17], where the solutions are tree data structures, e.g., representing mathematical formulas

What does "good" mean?

Definition (Global Minimum)

There is no element with a smaller objective value than the global minimum $\check{\check{x}}$.

What does "good" mean?

Definition (Global Minimum)

A global minimum $\check{\check{x}} \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{\check{x}}) \leq f(x) \forall x \in \mathbb{X}$.

What does “good” mean?

Definition (Global Minimum)

A global minimum $\check{\check{x}} \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{\check{x}}) \leq f(x) \forall x \in \mathbb{X}$.

Definition (Global Maximum)

There is no element with a larger objective value than the global maximum $\hat{\hat{x}}$.

What does “good” mean?

Definition (Global Minimum)

A global minimum $\check{\check{x}} \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{\check{x}}) \leq f(x) \forall x \in \mathbb{X}$.

Definition (Global Maximum)

A global maximum $\hat{\hat{x}} \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\hat{\hat{x}}) \geq f(x) \forall x \in \mathbb{X}$.

What does "good" mean?

Definition (Global Minimum)

A global minimum $\check{x} \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{x}) \leq f(x) \forall x \in \mathbb{X}$.

Definition (Global Maximum)

A global maximum $\hat{x} \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\hat{x}) \geq f(x) \forall x \in \mathbb{X}$.

Definition (Global Optimum of a Single Objective Function)

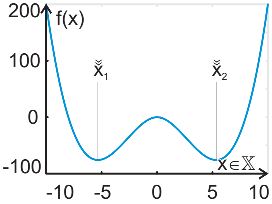
Depending on whether the objective function is subject to minimization or maximization, a global optimum is either a global minimum or a global maximum.

What does “good” mean? (Multiple Optima)

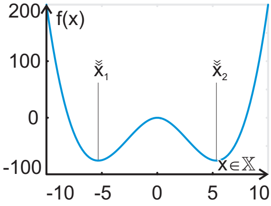
- There may be multiple global and local optima

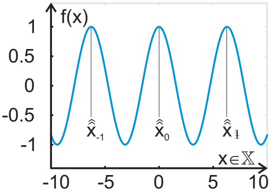
What does "good" mean? (Multiple Optima)

- There may be multiple global and local optima



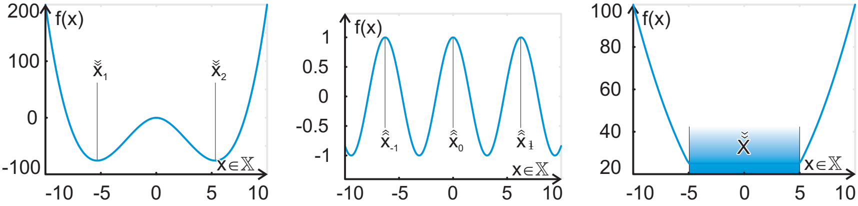
- There may be multiple global and local optima



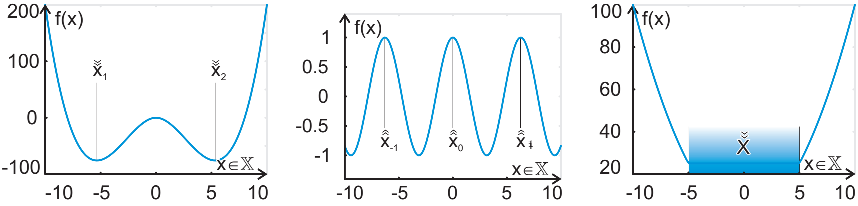


What does “good” mean? (Multiple Optima)

- There may be multiple global and local optima



- There may be multiple global and local optima



Definition (Global Optimal Set)

The optimal set $X^{\star} \subseteq \mathbb{X}$ of an optimization problem is the set that contains all its globally optimal solutions.

Examples for “good”

1. Bin Packing
2. Circuit Layout
3. Find the roots of a function $g(x)$
4. Shortest Path / Routing
5. Find mathematical formula fitting to given data
6. Job Shop Scheduling
7. Stock Prediction
8. Truss Optimization
9. Medical Classification
10. Airplane Wing Design
11. Antenna design
12. Analog Electrical Circuit Design
13. Interactive Optimization

Definition (Global Optimal Set)

The optimal set $X^{\star} \subseteq \mathbb{X}$ of an optimization problem is the set that contains all its globally optimal solutions.

Optimization Result

Definition (Global Optimal Set)

The optimal set $X^{\star} \subseteq \mathbb{X}$ of an optimization problem is the set that contains all its globally optimal solutions.

- Often, we cannot get the global optimal set...

Definition (Global Optimal Set)

The optimal set $X^{\star} \subseteq \mathbb{X}$ of an optimization problem is the set that contains all its globally optimal solutions.

- Often, we cannot get the global optimal set...
- ...but only an approximation $\tilde{X}$ of it.

Optimization Result

Definition (Global Optimal Set)

The optimal set $X^{\star} \subseteq \mathbb{X}$ of an optimization problem is the set that contains all its globally optimal solutions.

- Often, we cannot get the global optimal set...
- ...but only an approximation $\tilde{X}$ of it.

Definition (Optimization Result $\tilde{X}$)

The set $\tilde{X} \subseteq \mathbb{X}$ contains output elements $\tilde{x} \in \mathbb{X}$ of an optimization process.

Definition (Global Optimal Set)

The optimal set $X^{\star} \subseteq \mathbb{X}$ of an optimization problem is the set that contains all its globally optimal solutions.

- Often, we cannot get the global optimal set…
- …but only an approximation $\tilde{X}$ of it.

Definition (Optimization Result $\tilde{X}$)

The set $\tilde{X} \subseteq \mathbb{X}$ contains output elements $\tilde{x} \in \mathbb{X}$ of an optimization process.

- usually we only return one single solution $\tilde{x}$, i.e., $\tilde{X} \equiv \{\tilde{x}\}$

Components of a Optimization Problem

Now we have discussed the basic components of an optimization problem from a more mathematical point of view.

1. the solution space $\mathbb{X}$,
2. the objective function(s) $f : \mathbb{X} \mapsto \mathbb{R}$, and
3. the concept of "good" (minimize? maximize? multi-objective?).

- What is the situation?

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.
     - Instead, we often only have the objective function f which provides a quality value for each candidate solution x.

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.
     - Instead, we often only have the objective function f which provides a quality value for each candidate solution x.
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)?

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.
     - Instead, we often only have the objective function f which provides a quality value for each candidate solution x.
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)?
     - If so, we will do that and are finished. We don't need an optimization algorithm.

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.
     - Instead, we often only have the objective function f which provides a quality value for each candidate solution x.
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)?
     - If so, we will do that and are finished. We don't need an optimization algorithm.
     - Often, we cannot.

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.
     - Instead, we often only have the objective function f which provides a quality value for each candidate solution x.
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)?
     - If so, we will do that and are finished. We don't need an optimization algorithm.
     - Often, we cannot.
     - Often, we can calculate f and have a rough idea of it, but cannot directly solve it.

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.
     - Instead, we often only have the objective function f which provides a quality value for each candidate solution x.
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)?
     - If so, we will do that and are finished. We don't need an optimization algorithm.
     - Often, we cannot.
     - Often, we can calculate f and have a rough idea of it, but cannot directly solve it.
  4. Can we simply test all candidate solution $x \in \mathbb{X}$?

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have?
     - No. If so, we are already finished.
     - Instead, we often only have the objective function f which provides a quality value for each candidate solution x.
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)?
     - If so, we will do that and are finished. We don't need an optimization algorithm.
     - Often, we cannot.
     - Often, we can calculate f and have a rough idea of it, but cannot directly solve it.
  4. Can we simply test all candidate solution $x \in \mathbb{X}$?
     - No. There are too many... (remember the TSP)

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO
- So what can we do?

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO
- So what can we do?
  - We know the data structure for elements of $\mathbb{X}$.

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO
- So what can we do?
  - We know the data structure for elements of $\mathbb{X}$.
  - So we can randomly create instances x!

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO
- So what can we do?
  - We know the data structure for elements of $\mathbb{X}$.
  - So we can randomly create instances x!
  - And we can modify some existing (previously created) instances x!

Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO
- So what can we do?
  - We know the data structure for elements of $\mathbb{X}$.
  - So we can randomly create instances x!
  - And we can modify some existing (previously created) instances x!
  - And we can maybe even combine existing instances x_1 and x_2!

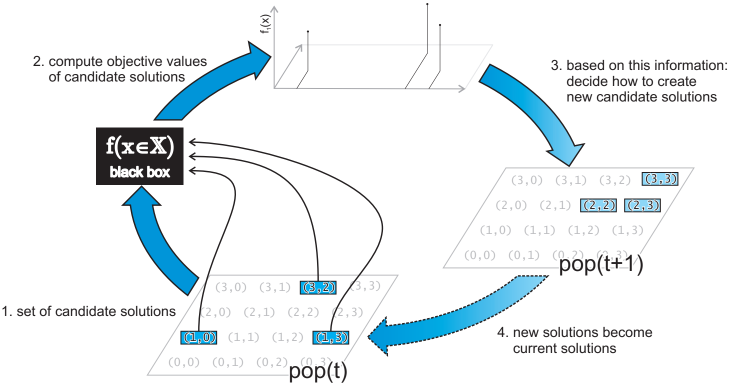
Situation & Idea

- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO
- So what can we do?
  - We know the data structure for elements of $\mathbb{X}$.
  - So we can randomly create instances x!
  - And we can modify some existing (previously created) instances x!
  - And we can maybe even combine existing instances x_1 and x_2!
  - If we do this well or can learn how to do this best, we can win!

Situation & Idea

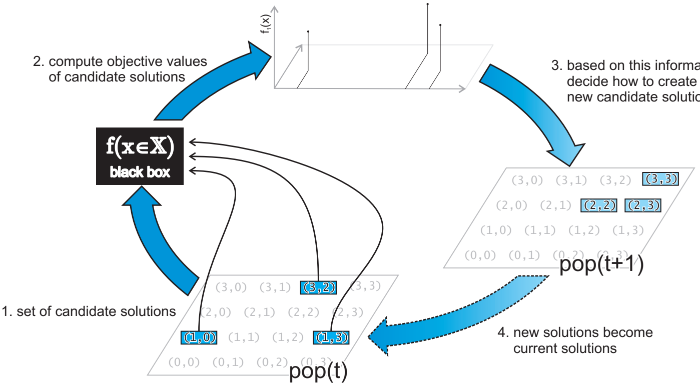
- What is the situation?
  1. We have a potentially extremely large set $\mathbb{X}$ of solutions.
  2. Do we know exactly what features good (or the best) solutions have? NO
  3. Can we "directly" solve f for the optima (e.g., by differentiating it)? NO
  4. Can we simply test all candidate solution $x \in \mathbb{X}$? NO
- So what can we do?
  - We know the data structure for elements of $\mathbb{X}$.
  - So we can randomly create instances x!
  - And we can modify some existing (previously created) instances x!
  - And we can maybe even combine existing instances x_1 and x_2!
  - If we do this well or can learn how to do this best, we can win!
  - This is the idea behind all *metaheuristics*

- Start with one (or multiple) initially generated candidate solutions (we call this set of solutions "population" pop)

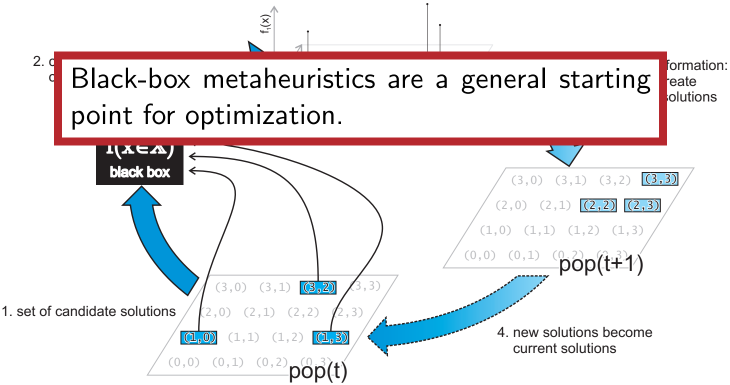


How a Metaheuristic Works

- Start with one (or multiple) initially generated candidate solutions (we call this set of solutions "population" pop)
- Iteratively refine the solution(s) in a loop (e.g., by making small random changes)



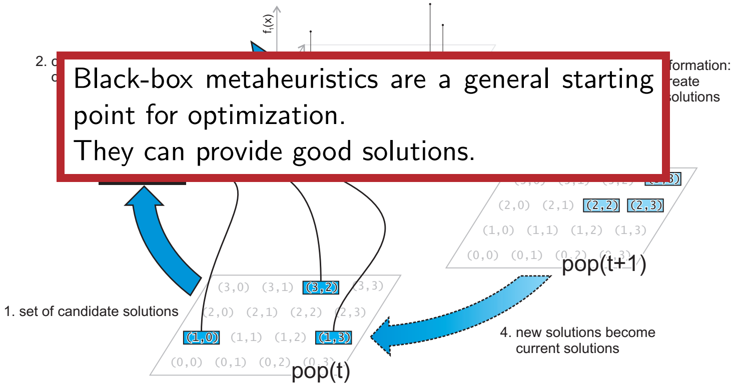
- Start with one (or multiple) initially generated candidate solutions (we call this set of solutions "population" pop)
- Iteratively refine the solution(s) in a loop (e.g., by making small random changes)



Black-box metaheuristics are a general starting point for optimization.

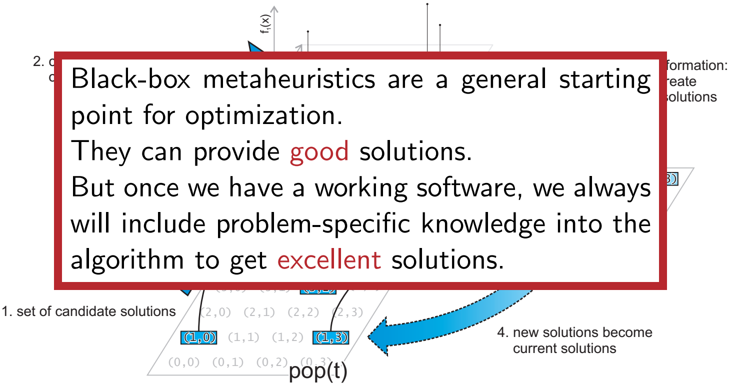
How a Metaheuristic Works

- Start with one (or multiple) initially generated candidate solutions (we call this set of solutions "population" pop)
- Iteratively refine the solution(s) in a loop (e.g., by making small random changes)



Black-box metaheuristics are a general starting point for optimization.
They can provide good solutions.

- Start with one (or multiple) initially generated candidate solutions (we call this set of solutions "population" pop)
- Iteratively refine the solution(s) in a loop (e.g., by making small random changes)



Black-box metaheuristics are a general starting point for optimization.
They can provide good solutions.
But once we have a working software, we always will include problem-specific knowledge into the algorithm to get excellent solutions.

Search Space

- OK, we have a data structure $\mathbb{X}$ for the candidate solutions and an objective function $f : \mathbb{X} \mapsto \mathbb{R}$ telling us how good they are

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively

- We now want to create instances of $\mathbb{X}$ and then “improve” them iteratively
- This means that we need

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- This means that we need:
    - a method for creating an instance of $\mathbb{X}$

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- This means that we need:
  - a method for creating an instance of $\mathbb{X}$
  - a method for changing (and hopefully improving) an instance of $\mathbb{X}$

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- This means that we need:
    - a method for creating an instance of $\mathbb{X}$
    - a method for changing (and hopefully improving) an instance of $\mathbb{X}$

… a lot of code that we need to write for *each* optimization problem…

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- This means that we need ... a lot of code that we need to write for *each* optimization problem... and we did not even talk about how the metaheuristic algorithm itself works

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- This means that we need ... a lot of code that we need to write for *each* optimization problem...
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
  Example: Finding the roots of a real function $g(x)$ (use real vector)

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
- What if $\mathbb{X}$ is not any well-known data structure?

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
- What if $\mathbb{X}$ is not any well-known data structure?
- Try to see if there is a well-known data structure $\mathbb{G}$ that can be translated to $\mathbb{X}$

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
- What if $\mathbb{X}$ is not any well-known data structure?
- Try to see if there is a well-known data structure $\mathbb{G}$ that can be translated to $\mathbb{X}$
  Example: a bit string can be translated to a text describing which features a BMW has

Search Space

- We now want to create instances of $\mathbb{X}$ and then “improve” them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
- What if $\mathbb{X}$ is not any well-known data structure?
- Try to see if there is a well-known data structure $\mathbb{G}$ that can be translated to $\mathbb{X}$
  Example: a cookie receipe internally can be represented as vector of real numbers, just translate it to text the grandma can read

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
- What if $\mathbb{X}$ is not any well-known data structure?
- Try to see if there is a well-known data structure $\mathbb{G}$ that can be translated to $\mathbb{X}$
  Example: the shape of an airplane wing can be represented as vector of real numbers, just translate it to a textual description of the wing

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
- What if $\mathbb{X}$ is not any well-known data structure?
- Try to see if there is a well-known data structure $\mathbb{G}$ that can be translated to $\mathbb{X}$
- Besides the solution space $\mathbb{X}$ we can use a search space $\mathbb{G}$

Search Space

- We now want to create instances of $\mathbb{X}$ and then "improve" them iteratively
- Idea: Sometimes, we can use well-known data structures for $\mathbb{X}$, e.g., $\mathbb{R}^n \equiv$ `double[]`
- If we know good operators for $\mathbb{R}^n$, we can re-use them for all problems that have $\mathbb{X} \equiv \mathbb{R}^n$!
- What if $\mathbb{X}$ is not any well-known data structure?
- Try to see if there is a well-known data structure $\mathbb{G}$ that can be translated to $\mathbb{X}$
- Besides the solution space $\mathbb{X}$ we can use a search space $\mathbb{G}$ an *internal* data structure for representing the possible solutions from $\mathbb{X}$

Search Space

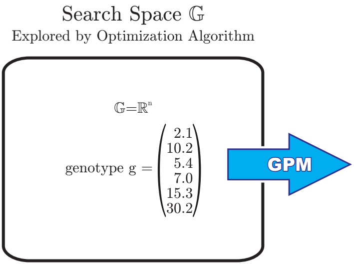
Search Space $\mathbb{G}$

Explored by Optimization Algorithm

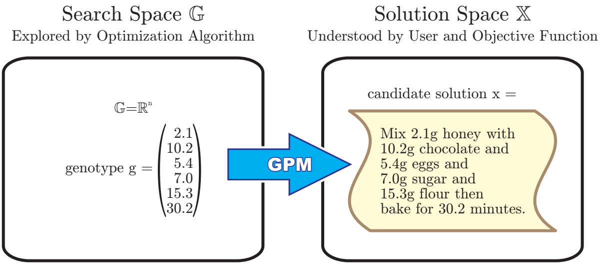
$\mathbb{G}=\mathbb{R}^n$

$$\text{genotype } g = \begin{pmatrix} 2.1 \\ 10.2 \\ 5.4 \\ 7.0 \\ 15.3 \\ 30.2 \end{pmatrix}$$

As a metaphor based on biological genetics, the search space is often called *genome*, points in the search space are called *genotypes*, the solution space (solution space) is called *phenome*, its elements are called *phenotypes*, and the translation between phenotypes and genotypes is called *genotype-phenotype-mapping*.

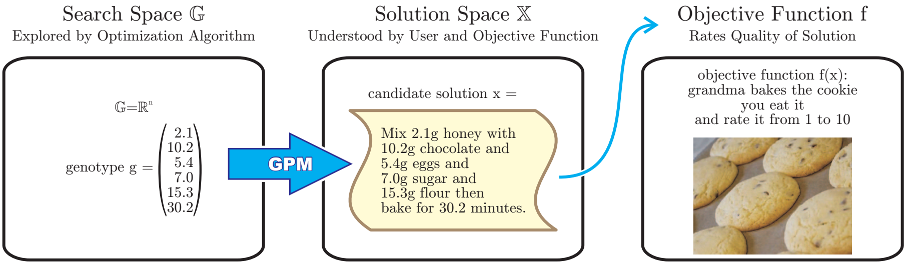


As a metaphor based on biological genetics, the search space is often called *genome*, points in the search space are called *genotypes*, the solution space (solution space) is called *phenome*, its elements are called *phenotypes*, and the translation between phenotypes and genotypes is called *genotype-phenotype-mapping*.



As a metaphor based on biological genetics, the search space is often called *genome*, points in the search space are called *genotypes*, the solution space (solution space) is called *phenome*, its elements are called *phenotypes*, and the translation between phenotypes and genotypes is called *genotype-phenotype-mapping*.

Search Space



As a metaphor based on biological genetics, the search space is often called *genome*, points in the search space are called *genotypes*, the solution space (solution space) is called *phenome*, its elements are called *phenotypes*, and the translation between phenotypes and genotypes is called *genotype-phenotype-mapping*.

Search Space

From the programmer's perspective, we can say:

Listing: Search space $\mathbb{G}$

```
public class MySearchSpace extends Object {
  ...
  }

//or, instead, maybe a simple or primitive type
//or an array...
```

Definition (Genotype-Phenotype Mapping)

The genotype-phenotype mapping (GPM) $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$ is a left-total binary relation which maps the elements of the search space $\mathbb{G}$ to elements in the solution space $\mathbb{X}$.

Definition (Genotype-Phenotype Mapping)

The genotype-phenotype mapping (GPM) $\mathrm{gpm}: \mathbb{G} \mapsto \mathbb{X}$ is a left-total binary relation which maps the elements of the search space $\mathbb{G}$ to elements in the solution space $\mathbb{X}$.

- if $\mathbb{G} = \mathbb{X}$, the genotype-phenotype mapping is (usually) the identity mapping

Genotype-Phenotype Mapping

Definition (Genotype-Phenotype Mapping)

The genotype-phenotype mapping (GPM) $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$ is a left-total binary relation which maps the elements of the search space $\mathbb{G}$ to elements in the solution space $\mathbb{X}$.

- if $\mathbb{G} = \mathbb{X}$, the genotype-phenotype mapping is (usually) the identity mapping
- this is often the case, but not always [25, 52]

From the programmer's perspective, we can say:

Listing: Mapping from search- to solution space: $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$

```
public interface IGPM<G, X> {
  public abstract X gpm(final G genotype);
}
```

Search Space + GPM

Search Space $\mathbb{G}$

Explored by Optimization Algorithm

$\mathbb{G}=\mathbb{R}^n$

$$\text{genotype } g = \begin{pmatrix} 2.1 \\ 10.2 \\ 5.4 \\ 7.0 \\ 15.3 \\ 30.2 \end{pmatrix}$$

GPM

Solution Space $\mathbb{X}$

Understood by User and Objective Function

candidate solution x =

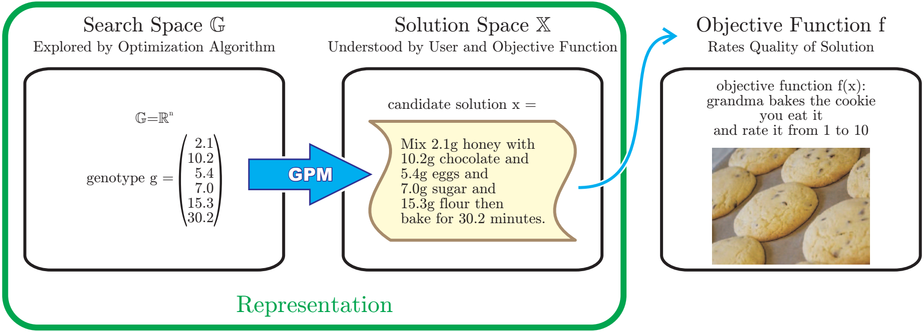
Mix 2.1g honey with
10.2g chocolate and
5.4g eggs and
7.0g sugar and
15.3g flour then
bake for 30.2 minutes.

Objective Function f

Rates Quality of Solution

objective function f(x):
grandma bakes the cookie
you eat it
and rate it from 1 to 10

Search Space + GPM



- $\mathbb{G}$, $\mathbb{X}$, and gpm together are called Representation

Search Space + GPM

Search Space $\mathbb{G}$
Explored by Optimization Algorithm

$\mathbb{G}=\mathbb{R}^n$

$$\text{genotype } g = \begin{pmatrix} 2.1 \\ 10.2 \\ 5.4 \\ 7.0 \\ 15.3 \\ 30.2 \end{pmatrix}$$

GPM

Solution Space $\mathbb{X}$
Understood by User and Objective Function

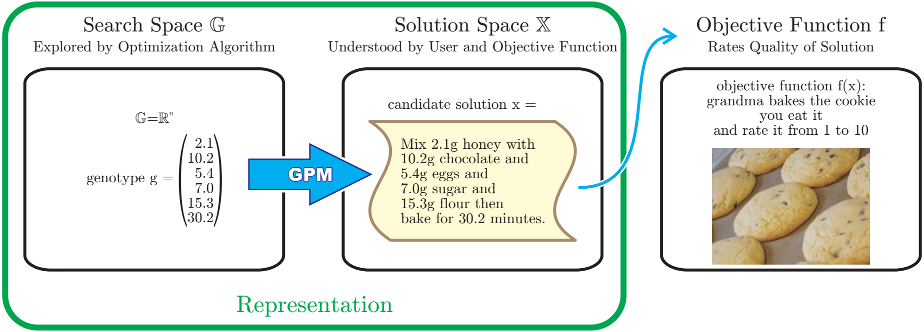
candidate solution x =

Mix 2.1g honey with
10.2g chocolate and
5.4g eggs and
7.0g sugar and
15.3g flour then
bake for 30.2 minutes.

Objective Function f
Rates Quality of Solution

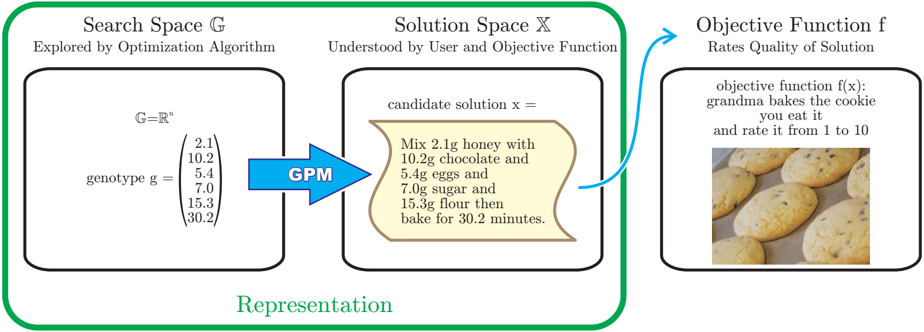
objective function f(x):
grandma bakes the cookie
you eat it
and rate it from 1 to 10

Representation



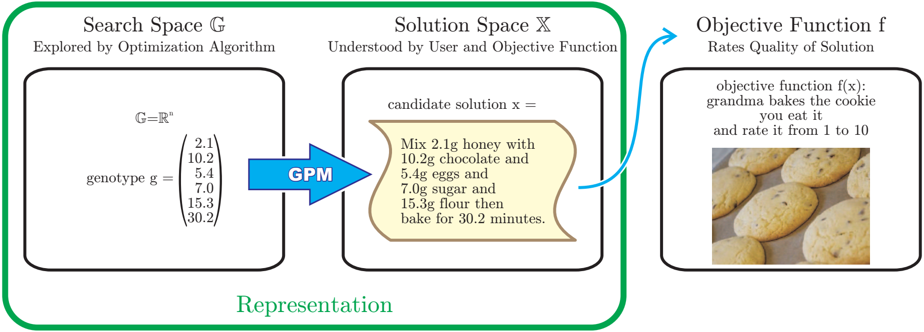
- The choice of the representation has tremendous impact on the results!

Search Space + GPM



- The choice of the representation has tremendous impact on the results, e.g.,
  - It determines which solutions can be found.

Search Space + GPM



- The choice of the representation has tremendous impact on the results, e.g.,
  - It determines which solutions can be found.
  - It determines the number of potential solutions.

Search Operations

- So, we have $\mathbb{X}$, f, $\mathbb{G}$, and $\mathrm{gpm}\ldots$ what else do we need?

- So, we have $\mathbb{X}$, f, $\mathbb{G}$, and $\mathrm{gpm}\ldots$ what else do we need?
  - An operation which creates instances of data structure $\mathbb{G}$,

Search Operations

- So, we have $\mathbb{X}$, f, $\mathbb{G}$, and $\mathrm{gpm}\ldots$ what else do we need?
  - An operation which creates instances of data structure $\mathbb{G}$,
  - An operators which can create a modified copy of an element of $\mathbb{G}$

- So, we have $\mathbb{X}$, f, $\mathbb{G}$, and $\mathrm{gpm}\ldots$ what else do we need?
  - An operation which creates instances of data structure $\mathbb{G}$,
  - An operators which can create a modified copy of an element of $\mathbb{G}$

Definition (Search Operation)

A search operation receives 0 or more elements from the search space $\mathbb{G}$ as parameter and returns a new genotype.

Search Operations

From the programmer's perspective, we can say:

Listing: Nullary search operation $\text{searchOp} : \emptyset \mapsto \mathbb{G}$

```
public interface INullarySearchOperation<G> {
  public abstract G create(final Random r);
}
```

From the programmer's perspective, we can say:

Listing: Nullary search operation $\mathrm{searchOp} : \emptyset \mapsto \mathbb{G}$

```
public interface INullarySearchOperation<G> {
  public abstract G create(final Random r);
}
```

- *Null*ary $\Longrightarrow$ 0 arguments from $\mathbb{G}$ (except from the random number generator)

From the programmer's perspective, we can say:

Listing: Nullary search operation $\mathrm{searchOp} : \emptyset \mapsto \mathbb{G}$

```
public interface INullarySearchOperation<G> {
  public abstract G create(final Random r);
}
```

- *Null*ary $\Longrightarrow$ 0 arguments from $\mathbb{G}$ (except from the random number generator)
- `G` is a generic data structure to be replaced by the search space $\mathbb{G}$

From the programmer's perspective, we can say:

Listing: Nullary search operation $\mathrm{searchOp} : \emptyset \mapsto \mathbb{G}$

```java
public interface INullarySearchOperation<G> {
  public abstract G create(final Random r);
}
```

- *Null*ary $\Longrightarrow$ 0 arguments from $\mathbb{G}$ (except from the random number generator)
- `G` is a generic data structure to be replaced by the search space $\mathbb{G}$
- `create` returns one instance of `G`.

Search Operations

From the programmer's perspective, we can say:

Listing: Nullary search operation $\mathrm{searchOp} : \emptyset \mapsto \mathbb{G}$

```
public interface INullarySearchOperation<G> {
  public abstract G create(final Random r);
}
```

- *Null*ary $\Longrightarrow$ 0 arguments from $\mathbb{G}$ (except from the random number generator)
- `G` is a generic data structure to be replaced by the search space $\mathbb{G}$
- `create` returns one instance of `G`.
- this could be a random instance or an instance constructed using some particular algorithm

From the programmer's perspective, we can say:

Listing: Unary search operation $\mathrm{searchOp} : \mathbb{G} \mapsto \mathbb{G}$

```
public interface IUnarySearchOperation<G> {
  public abstract G mutate(final G parent, //
      final Random r);
}
```

From the programmer's perspective, we can say:

Listing: Unary search operation $\text{searchOp} : \mathbb{G} \mapsto \mathbb{G}$

```
public interface IUnarySearchOperation<G> {
  public abstract G mutate(final G parent, //
      final Random r);
}
```

- *Un*ary $\Longrightarrow$ 1 argument from $\mathbb{G}$ (plus a random number generator)

From the programmer's perspective, we can say:

Listing: Unary search operation $\mathrm{searchOp} : \mathbb{G} \mapsto \mathbb{G}$

```
public interface IUnarySearchOperation<G> {
  public abstract G mutate(final G parent, //
      final Random r);
}
```

- *Un*ary $\Longrightarrow$ 1 argument from $\mathbb{G}$ (plus a random number generator)
- `G` is a generic data structure to be replaced by the search space $\mathbb{G}$

From the programmer's perspective, we can say:

Listing: Unary search operation $\mathrm{searchOp} : \mathbb{G} \mapsto \mathbb{G}$

```
public interface IUnarySearchOperation<G> {
  public abstract G mutate(final G parent, //
      final Random r);
}
```

- *Un*ary $\Longrightarrow$ 1 argument from $\mathbb{G}$ (plus a random number generator)
- `G` is a generic data structure to be replaced by the search space $\mathbb{G}$
- `mutate` receives one instance of `G` as parameter.

From the programmer's perspective, we can say:

Listing: Unary search operation $\mathrm{searchOp} : \mathbb{G} \mapsto \mathbb{G}$

```java
public interface IUnarySearchOperation<G> {
  public abstract G mutate(final G parent, //
      final Random r);
}
```

- *Unary* $\Longrightarrow 1$ argument from $\mathbb{G}$ (plus a random number generator)
- `G` is a generic data structure to be replaced by the search space $\mathbb{G}$
- `mutate` receives one instance of `G` as parameter.
- it then returns a modified copy of that instance.

From the programmer's perspective, we can say:

Listing: Unary search operation $\mathrm{searchOp} : \mathbb{G} \mapsto \mathbb{G}$

```
public interface IUnarySearchOperation<G> {
  public abstract G mutate(final G parent, //
      final Random r);
}
```

- *Unary* $\Longrightarrow 1$ argument from $\mathbb{G}$ (plus a random number generator)
- `G` is a generic data structure to be replaced by the search space $\mathbb{G}$
- `mutate` receives one instance of `G` as parameter.
- it then returns a modified copy of that instance.
- the modification is usually small and random

Termination Criterion

- So, we have $\mathbb{X}$, f, $\mathbb{G}$, and $\mathrm{gpm}\ldots$ what else do we need?
  - An operation which creates instances of data structure $\mathbb{G}$,
  - Operators which modify or combines such data structures

Termination Criterion

- So, we have $\mathbb{X}$, f, $\mathbb{G}$, and $\mathrm{gpm}\ldots$ what else do we need?
  - An operation which creates instances of data structure $\mathbb{G}$,
  - Operators which modify or combines such data structures
  - A method that tells us when the algorithm should stop.

Termination Criterion

- So, we have $\mathbb{X}$, f, $\mathbb{G}$, and $\mathrm{gpm}\ldots$ what else do we need?
  - An operation which creates instances of data structure $\mathbb{G}$,
  - Operators which modify or combines such data structures
  - A method that tells us when the algorithm should stop.

Definition (Termination Criterion)

When the termination criterion function becomes `true`, the optimization process will stop and return its results.

Termination Criterion

Definition (Termination Criterion)

When the termination criterion function becomes `true`, the optimization process will stop and return its results.

- Termination criterion may utilize all information gathered by the optimization algorithm so far

Termination Criterion

Definition (Termination Criterion)

When the termination criterion function becomes `true`, the optimization process will stop and return its results.

- Termination criterion may utilize all information gathered by the optimization algorithm so far
- Many different criteria possible [53–56]

Definition (Termination Criterion)

When the termination criterion function becomes `true`, the optimization process will stop and return its results.

- Termination criterion may utilize all information gathered by the optimization algorithm so far
- Many different criteria possible [53–56]:
  1. maximum computation time

Definition (Termination Criterion)

When the termination criterion function becomes `true`, the optimization process will stop and return its results.

- Termination criterion may utilize all information gathered by the optimization algorithm so far
- Many different criteria possible [53–56]:
  1. maximum computation time
  2. maximum number of objective function evaluations

Definition (Termination Criterion)

When the termination criterion function becomes `true`, the optimization process will stop and return its results.

- Termination criterion may utilize all information gathered by the optimization algorithm so far
- Many different criteria possible [53–56]:
  1. maximum computation time
  2. maximum number of objective function evaluations
  3. stop when no further improvement can be detected

Definition (Termination Criterion)

When the termination criterion function becomes `true`, the optimization process will stop and return its results.

- Termination criterion may utilize all information gathered by the optimization algorithm so far
- Many different criteria possible [53–56]:
  1. maximum computation time
  2. maximum number of objective function evaluations
  3. stop when no further improvement can be detected
  4. stop when a sufficiently good solution has been detected
  5. ...

Listing: Termination Criterion $\text{shouldTerminate} : \mapsto \{\texttt{true}, \texttt{false}\}$

```
public interface ITerminationCriterion {
  public abstract boolean shouldTerminate();
}
```

Listing: Termination Criterion $\text{shouldTerminate} :\mapsto \{\texttt{true}, \texttt{false}\}$

```
public interface ITerminationCriterion {
  public abstract boolean shouldTerminate();
}
```

- Directly after every time the optimization algorithm has created a new solution x and computed $f(x)$, it must call `shouldTerminate()`

Listing: Termination Criterion $\text{shouldTerminate} :\mapsto \{\texttt{true}, \texttt{false}\}$

```
public interface ITerminationCriterion {
  public abstract boolean shouldTerminate();
}
```

- Directly after every time the optimization algorithm has created a new solution x and computed $f(x)$, it must call `shouldTerminate()`
- If `shouldTerminate()` returns `true`, the algorithm must immediately stop

Listing: Termination Criterion shouldTerminate $:\mapsto \{\texttt{true}, \texttt{false}\}$

```
public interface ITerminationCriterion {
  public abstract boolean shouldTerminate();
}
```

- Directly after every time the optimization algorithm has created a new solution x and computed $f(x)$, it must call `shouldTerminate()`
- If `shouldTerminate()` returns `true`, the algorithm must immediately stop and return the best solution candidate it has seen so far
- One could implement `ITerminationCriterion` and `IObjectiveFunction` in the same object to stop once a goal solution quality was reached.

Example Termination Criterion

Listing: A criterion stopping after a given amount of steps.

```
public class MaxSteps implements ITerminationCriterion {
  /** the number of remaining steps */
  private int m_remaining;


  public MaxSteps(final int steps) {
    super();
    this.m_remaining = steps;
  }


  public boolean shouldTerminate() {
    return ((--this.m_remaining) < 0);
  }
}
```

Section Outline

Putting it Together

- An optimization problem is defined by

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$
  - a nullary search operation to create new points in $\mathbb{G}$

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$
  - a nullary search operation to create new points in $\mathbb{G}$
  - a unary search operation to modify existing points in $\mathbb{G}$

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$
  - a nullary search operation to create new points in $\mathbb{G}$
  - a unary search operation to modify existing points in $\mathbb{G}$
  - a mapping gpm that translates the internal representation $\mathbb{G}$ to candidate solutions in $\mathbb{X}$

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$
  - a nullary search operation to create new points in $\mathbb{G}$
  - a unary search operation to modify existing points in $\mathbb{G}$
  - a mapping gpm that translates the internal representation $\mathbb{G}$ to candidate solutions in $\mathbb{X}$
  - a termination criterion to know when to stop

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$
  - a nullary search operation to create new points in $\mathbb{G}$
  - a unary search operation to modify existing points in $\mathbb{G}$
  - a mapping gpm that translates the internal representation $\mathbb{G}$ to candidate solutions in $\mathbb{X}$
  - a termination criterion to know when to stop
- It will give us

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$
  - a nullary search operation to create new points in $\mathbb{G}$
  - a unary search operation to modify existing points in $\mathbb{G}$
  - a mapping gpm that translates the internal representation $\mathbb{G}$ to candidate solutions in $\mathbb{X}$
  - a termination criterion to know when to stop
- It will give us:
  - a set of solutions $\tilde{X} \subseteq \mathbb{X}$.

Putting it Together

- An optimization problem is defined by:
  - a solution space $\mathbb{X}$
  - (at least) one objective function f
  - a notion of good (let us assume: minimization)
- An optimization algorithm furthermore needs:
  - a search space $\mathbb{G}$ usually the same as $\mathbb{X}$
  - a nullary search operation to create new points in $\mathbb{G}$
  - a unary search operation to modify existing points in $\mathbb{G}$
  - a mapping gpm that translates the internal representation $\mathbb{G}$ to candidate solutions in $\mathbb{X}$
  - a termination criterion to know when to stop
- It will give us:
  - a usually one of solution $\tilde{x} \in \mathbb{X}$.

Putting it Together

Then a metaheuristic, black-box optimization looks like:

Definition (Individual)

An individual is a record where we can store all information that belongs to a solution, such as the genotype $g \in \mathbb{G}$, the corresponding phenotype $x \in \mathbb{X}$, and the objective value that we get when computing $f(x)$.

Section Outline

Summary

- Most metaheuristic optimization algorithms consist of common types of modules

Summary

- Most metaheuristic optimization algorithms consist of common types of modules
- Most often, specific sets and transformations are involved

Summary

- Most metaheuristic optimization algorithms consist of common types of modules
- Most often, specific sets and transformations are involved
- Search space $\mathbb{G}$ with genotypes g

Summary

- Most metaheuristic optimization algorithms consist of common types of modules
- Most often, specific sets and transformations are involved
- Search space $\mathbb{G}$ with genotypes g
- A set of search operations that can create, modify, or combine the elements from $\mathbb{G}$

- Most metaheuristic optimization algorithms consist of common types of modules
- Most often, specific sets and transformations are involved
- Search space $\mathbb{G}$ with genotypes g
- A set of search operations that can create, modify, or combine the elements from $\mathbb{G}$
- Solution space $\mathbb{X}$ with phenotypes x

Summary

- Most metaheuristic optimization algorithms consist of common types of modules
- Most often, specific sets and transformations are involved
- Search space $\mathbb{G}$ with genotypes g
- A set of search operations that can create, modify, or combine the elements from $\mathbb{G}$
- Solution space $\mathbb{X}$ with phenotypes x
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$

Summary

- Most metaheuristic optimization algorithms consist of common types of modules
- Most often, specific sets and transformations are involved
- Search space $\mathbb{G}$ with genotypes g
- A set of search operations that can create, modify, or combine the elements from $\mathbb{G}$
- Solution space $\mathbb{X}$ with phenotypes x
- Genotype-phenotype mapping $\mathrm{gpm} : \mathbb{G} \mapsto \mathbb{X}$
- Objective Functions $f : \mathbb{X} \mapsto \mathbb{R}$

谢谢
Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. David Lee Applegate, Robert E. Bixby, Vašek Chvátal, and William John Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, February 2007. ISBN 0-691-12993-2 and 978-0-691-12993-8. URL http://books.google.de/books?id=nmF4rVNJMVsC.
2. Eugene Leighton (Gene) 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, UK: Wiley Interscience, September 1985. ISBN 0-471-90413-9 and 978-0-471-90413-7. URL http://books.google.de/books?id=BXBGAAAAYAAJ.
3. Gregory Z. Gutin and Abraham P. Punnen, editors. *The Traveling Salesman Problem and its Variations*, volume 12 of *Combinatorial Optimization*. Norwell, MA, USA: Kluwer Academic Publishers, 2002. ISBN 0-306-48213-4, 1-4020-0664-0, and 978-1-4020-0664-7. doi: 10.1007/b101971. URL http://books.google.de/books?id=TRYkPg_Xf20C.
4. Bernhard Friedrich Voigt. *Der Handlungsreisende – wie er sein soll und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur*. Ilmenau, Germany: Voigt, 1832. Excerpt: "... Durch geeignete Auswahl und Planung der Tour kann man oft so viel Zeit sparen, daß wir einige Vorschläge zu machen haben. ... Der wichtigste Aspekt ist, so viele Orte wie möglich zu erreichen, ohne einen Ort zweimal zu besuchen. ...".
5. Federico Greco, editor. *Traveling Salesman Problem*. Vienna, Austria: IN-TECH Education and Publishing, September 2008. ISBN 978-953-7619-10-7. URL http://intechweb.org/downloadfinal.php?is=978-953-7619-10-7&type=B.
6. Ashish Sabharwal. Combinatorial problems i: Finding solutions. In Silvio Franz, Matteo Marsili, and Haijun Zhou, editors, *2nd Asian-Pacific School on Statistical Physics and Interdisciplinary Applications*, Beijing, China, March 3–14, 2008. Triest, Italy: Abdus Salam International Centre for Theoretical Physics (ICTP), Beijing, China: Chinese Center of Advanced Science and Technology (CCAST), and Beijing, China: Chinese Academy of Sciences, Kavli Institute of Theoretical Physics China (KITPC). URL http://www.cs.cornell.edu/~sabhar/tutorials/kitpc08-combinatorial-problems-I.ppt.
7. Michael R. Garey and David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. New York, NY, USA: W. H. Freeman and Company, 1979. ISBN 0-7167-1044-7, 0-7167-1045-5, 978-0-7167-1044-8, and 978-0-7167-1045-5. URL http://books.google.de/books?id=mdBxHAAACAAJ.
8. Scott Kirkpatrick, Charles Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science Magazine*, 220(4598):671–680, May 13, 1983. doi: 10.1126/science.220.4598.671. URL http://fezzik.ucd.ie/msc/cscs/ga/kirkpatrick83optimization.pdf.

Bibliography II

9. Tatiana Kalganova and Julian Francis Miller. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In Adrian Stoica, Jason D. Lohn, and Didier Keymeulen, editors, *Evolvable Hardware – Proceedings of 1st NASA/DoD Workshop on Evolvable Hardware (EH'99)*, pages 54–63, Pasadena, CA, USA: Jet Propulsion Laboratory, California Institute of Technology (Caltech), June 19–21, 1999. Washington, DC, USA: IEEE Computer Society. doi: 10.1109/EH.1999.785435. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.948.
10. Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 3(2):82–102, July 1999. doi: 10.1109/4235.771163. URL http://www.u-aizu.ac.jp/~yliu/publication/tec22r2_online.ps.gz.
11. Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In Richard K. Belew and Lashon Bernard Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, pages 2–9, San Diego, CA, USA: University of California (UCSD), July 13–16, 1991. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://130.203.133.121:8080/viewdoc/summary?doi=10.1.1.42.3375.
12. Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Natural Computing Series. Basel, Switzerland: Birkhäuser Verlag, 2005. ISBN 3-540-20950-6, 3-540-31306-0, 978-3-540-20950-8, and 978-3-540-31306-9. URL http://books.google.de/books?id=S67vX-KqVqUC.
13. Zbigniew Michalewicz. Genetic algorithms, numerical optimization, and constraints. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95)*, pages 151–158., Pittsburgh, PA, USA: University of Pittsburgh, July 15–19, 1995. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://www.cs.adelaide.edu.au/~zbyszek/Papers/p16.pdf.
14. Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. URL http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf.
15. Robert W Floyd. Algorithm 97 (shortest path). *Communications of the ACM (CACM)*, 5(6):345, June 1, 1962. doi: 10.1145/367766.368168.
16. Stephen Warshall. A theorem on boolean matrices. *Journal of the Association for Computing Machinery (JACM)*, 9(1): 11–12, January 1962. doi: 10.1145/321105.321107.
17. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford Books. Cambridge, MA, USA: MIT Press, December 1992. ISBN 0-262-11170-5 and 978-0-262-11170-6. URL http://books.google.de/books?id=Bhtxo60BV0EC. 1992 first edition, 1993 second edition.

18. Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz, editors. *Variants of Evolutionary Algorithms for Real-World Applications*. Berlin/Heidelberg: Springer-Verlag, 2011. ISBN 978-3-642-23423-1 and 978-3-642-23424-8. doi: 10.1007/978-3-642-23424-8. URL http://books.google.de/books?id=B2ONePP40MEC.
19. Douglas A. Augusto and Helio Joseé Correa Barbosa. Symbolic regression via genetic programming. In Felipe M. G. França and Carlos H. C. Ribeiro, editors, *Proceedings of the VI Brazilian Symposium on Neural Networks (SBRN'00)*, pages 173–178, Rio de Janeiro, RJ, Brazil, November 22–25, 2000. Washington, DC, USA: IEEE Computer Society. doi: 10.1109/SBRN.2000.889734.
20. Federico Della Croce, Roberto Tadei, and Giuseppe Volta. A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15–24, January 1995. doi: 10.1016/0305-0548(93)E0015-L.
21. Edward P. K. Tsang, Jin Li, Sheri Marina Markose, Hakan Er, Abdellah Salhi, and Guilia Iori. Eddie in financial decision making. *Journal of Management and Economics*, 4(4), November 2000. URL http://www.bracil.net/finance/papers/Tsang-Eddie-JMgtEcon2000.pdf.
22. Edward P. K. Tsang, Paul Yung, and Jin Li. Eddie-automation – a decision support tool for financial forecasting. *Decision Support Systems*, 37(4):559–565, September 2004. doi: 10.1016/S0167-9236(03)00087-3. URL http://www.bracil.net/finance/papers/TsYuLi-Eddie-Dss2004.pdf.
23. Pu Wang, Edward P. K. Tsang, Thomas Weise, Ke Tang, and Xin Yao. Using gp to evolve decision rules for classification in financial data sets. In Fuchun Sun, Yingxu Wang, Jianhua Lu, Bo Zhang, Witold Kinsner, and Lotfi A. Zadeh, editors, *Proceedings of the 9th IEEE International Conference on Cognitive Informatics (ICCI'10)*, pages 722–727, Beijing, China: Tsinghua University, July 7–9, 2010. Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/COGINF.2010.5599820.
24. Pu Wang, Thomas Weise, and Raymond Chiong. Novel evolutionary algorithms for supervised classification problems: An experimental study. *Evolutionary Intelligence*, 4(1):3–16, January 12, 2011. doi: 10.1007/s12065-010-0047-7.
25. Alexandre Devert, Thomas Weise, and Ke Tang. A study on scalable representations for evolutionary optimization of ground structures. *Evolutionary Computation*, 20(3):453–472, Fall 2012. doi: 10.1162/EVCO_a_00054. URL http://www.marmakoide.org/download/publications/devweita-ecj-preprint.pdf.
26. Wolfgang Achtziger and Mathias Stolpe. Truss topology optimization with discrete design variables – guaranteed global optimality and benchmark examples. *Structural and Multidisciplinary Optimization*, 34(1):1–20, July 2007. doi: 10.1007/s00158-006-0074-2.

27. Fabiano Luis de Sousa and Walter Kenkiti Takahashi. Discrete optimal design of trusses by generalized extremal optimization. In José Herskovits, Sandro Mazorche, and Alfredo Canelas, editors, *Proceedings of the 6th World Congresses of Structural and Multidisciplinary Optimization (WCSMO6)*, Rio de Janeiro, RJ, Brazil, May 30–June 3, 2005. Rio de Janeiro, RJ, Brazil: COPPE Publication. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.76.8740.
28. Faten Kharbat, Larry Bull, and Mohammed Odeh. Mining breast cancer data with xcs. In Dirk Thierens, Hans-Georg Beyer, Josh C. Bongard, Jürgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev P. Kumar, Julian Francis Miller, Jason H. Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stützle, Richard A. Watson, and Ingo Wegener, editors, *Proceedings of 9th Genetic and Evolutionary Computation Conference (GECCO'07)*, pages 2066–2073, London, UK: University College London (UCL), July 7–11, 2007. New York, NY, USA: ACM Press. doi: 10.1145/1276958.1277362. URL http://www.cs.york.ac.uk/rts/docs/GECCO_2007/docs/p2066.pdf.
29. Shigeru Obayashi. Multidisciplinary design optimization of aircraft wing planform based on evolutionary algorithms. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC'98)*, volume 4, pages 3148–3153, La Jolla, CA, USA, October 11–14, 1998. Los Alamitos, CA, USA: IEEE Computer Society Press. doi: 10.1109/ICSMC.1998.726486. URL http://www.lania.mx/~ccoello/obayashi98a.pdf.gz.
30. Akira Oyama. *Wing Design Using Evolutionary Algorithm*. PhD thesis, Tokyo, Japan: Tokyo University, Department of Aeronautics and Space Engineering, March 2000. URL http://flab.eng.isas.ac.jp/member/oyama/index2e.html.
31. Miroslav Červenka and Vojtěch Křesálek. Aerodynamic wing optimisation using soma evolutionary algorithm. In Natalio Krasnogor, María Belén Melián-Batista, José Andrés Moreno Pérez, J. Marcos Moreno-Vega, and David Alejandro Pelta, editors, *Proceedings of the 3rd International Workshop Nature Inspired Cooperative Strategies for Optimization (NICSO'08)*, volume 236/2009 of *Studies in Computational Intelligence*, pages 127–138, Puerto de la Cruz, Tenerife, Spain, November 12–14, 2008. Berlin/Heidelberg: Springer-Verlag. doi: 10.1007/978-3-642-03211-0_11.
32. Miroslav Červenka and Ivan Zelinka. Application of evolutionary algorithm on aerodynamic wing optimisation. In *Proceedings of the 2nd European Computing Conference (ECC'08)*, Malta, September 11–13, 2008. URL http://www.wseas.us/e-library/conferences/2008/malta/ecc/ecc53.pdf.

Bibliography V

33. Jason D. Lohn, Gregory S. Hornby, and Derek Linden. An evolved antenna for deployment on nasa's space technology 5 mission. In Una-May O'Reilly, Gwoing Tina Yu, Rick L. Riolo, and Bill Worzel, editors, *Genetic Programming Theory and Practice II, Proceedings of the Second Workshop on Genetic Programming (GPTP'04)*, volume 8 of *Genetic Programming Series*, pages 301–315, Ann Arbor, MI, USA: University of Michigan, Center for the Study of Complex Systems (CSCS), May 13–15, 2004. Boston, MA, USA: Kluwer Publishers. doi: 10.1007/b101112. URL http://ic.arc.nasa.gov/people/hornby/papers/lohn_gptp04.ps.gz.
34. Jason D. Lohn, William F. Kraus, and Derek Linden. Evolutionary optimization of a quadrifilar helical antenna. In *IEEE AP-S International Symposium and USNC/URSI National Radio Science Meeting*, San Antonio, TX, USA, June 16–21, 2002. Piscataway, NJ, USA: IEEE Computer Society. URL http://ti.arc.nasa.gov/people/jlohn/Papers/aps2002.pdf.
35. Jason D. Lohn, Derek Linden, Gregory S. Hornby, William F. Kraus, and Adán Rodríguez-Arroyo. Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In Jason D. Lohn, editor, *The 2003 NASA/DoD Conference on Evolvable Hardware (EH'03)*, pages 155–163, Chicago, IL, USA, June 9–11, 2003. Washington, DC, USA: IEEE Computer Society. doi: 10.1109/EH.2003.1217660.
36. Andrew Lewis, Gerhard Weis, Marcus Randall, Amir Galehdar, and David Thiel. Optimising efficiency and gain of small meander line rfid antennas using ant colony system. In *10th IEEE Congress on Evolutionary Computation (CEC'09)*, pages 1486–1492, Trondheim, Norway, May 18–21, 2009. Piscataway, NJ, USA: IEEE Computer Society. doi: 10.1109/CEC.2009.4983118.
37. Hosung Choo, Adrian Hutani, Luiz Cezar Trintinalia, and Hao Ling. Shape optimisation of broadband microstrip antennas using genetic algorithm. *Electronics Letters*, 36(25):2057–2058, December 7, 2000. doi: 10.1049/el:20001452.
38. Neela Chattoraj and Jibendu Sekhar Roy. Application of genetic algorithm to the optimization of microstrip antennas with and without superstrate. *Mikrotalasna Revija (Microwave Review)*, 2(6), November 2006. URL http://www.mwr.medianis.net/pdf/Vol12No2-06-NChattoraj.pdf.
39. Matthias John and Max J. Ammann. Design of a wide-band printed antenna using a genetic algorithm on an array of overlapping sub-patches. In Duixian Liu and Brian Gaucher, editors, *2006 IEEE International Workshop on Antenna Technology: Small Antennas and Novel Metamaterials (iWAT'06)*, pages 92–95, White Plains, NY, USA: Crowne Plaza Hotel, 2006. Piscataway, NJ, USA: IEEE Computer Society. URL http://www.ctvr.ie/docs/RF%20Pubs/01608983.pdf.
40. Matthias John and Max J. Ammann. Optimisation of a wide-band printed monopole antenna using a genetic algorithm. In *Loughborough Antennas & Propagation Conference (PAPC'06)*, pages 237–240. Loughborough, Leicestershire, UK: Loughborough University, April 11–12, 2006. URL http://www.ctvr.ie/docs/RF%20Pubs/LAPC_2006_MJ.pdf.

Bibliography VI

41. Tian-Li Yu, Scott Santarelli, and David Edward Goldberg. Military antenna design using a simple genetic algorithm and hboa. In Martin Pelikan, Kumara Sastry, and Erick Cantú-Paz, editors, *Scalable Optimization via Probabilistic Modeling – From Algorithms to Applications*, volume 33 of *Studies in Computational Intelligence*, chapter 12, pages 275–289. Berlin/Heidelberg: Springer-Verlag, 2006. doi: 10.1007/978-3-540-34954-9.
42. John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. Automatic design of analog electrical circuits using genetic programming. In Hugh Cartwright, editor, *Intelligent Data Analysis in Science*, volume 4 of *Oxford Chemistry Masters*, chapter 8, pages 172–200. New York, NY, USA: Oxford University Press, Inc., June 2000. URL `http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/koza_2000_idas.html`.
43. John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. The design of analog circuits by means of genetic programming. In Peter John Bentley, editor, *Evolutionary Design by Computers*, chapter 16, pages 365–385. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., May 1999. URL `http://www.genetic-programming.com/jkpdf/edc1999.pdf`.
44. Jason D. Lohn and Silvano P. Colombano. Automated analog circuit synthesis using a linear representation. In Moshe Sipper, Daniel Mange, and Andrés Pérez-Uribe, editors, *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES'98)*, volume 1478/1998 of *Lecture Notes in Computer Science (LNCS)*, pages 125–133, Lausanne, Switzerland, September 23–25, 1999. Berlin, Germany: Springer-Verlag GmbH. URL `http://ti.arc.nasa.gov/people/jlohn/bio.html`.
45. Lyudmila Zinchenko, Matthias Radecker, and Fabio Bisogno. Application of the univariate marginal distribution algorithm to mixed analogue - digital circuit design and optimisation. In Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni A. Di Caro, Rolf Drechsler, Muddassar Farooq, Andreas Fink, Evelyne Lutton, Penousal Machado, Stefan Minner, Michael O'Neill, Juan Romero, Franz Rothlauf, Giovanni Squillero, Hideyuki Takagi, A. Şima Uyar, and Shengxiang Yang, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2007: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog (EvoWorkshops'07)*, volume 4448/2007 of *Lecture Notes in Computer Science (LNCS)*, pages 431–438, València, Spain, April 11–13, 2007. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/978-3-540-71805-5_48.
46. Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capacities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, September 2001. doi: 10.1109/5.949485. URL `http://www.design.kyushu-u.ac.jp/~takagi/TAKAGI/IECsurvey.html`.

Bibliography VII

47. Jeanine Graf and Wolfgang Banzhaf. Interactive evolution of images. In John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming (EP'95)*, Bradford Books, pages 53–65, San Diego, CA, USA, March 1–2, 1995. Cambridge, MA, USA: MIT Press. URL `http://citeseer.ist.psu.edu/110968.html`.
48. Brad Johanson and Riccardo Poli. Gp-music: An interactive genetic programming system for music generation with automated fitness raters. Technical Report CSRP-98-13, Birmingham, UK: University of Birmingham, School of Computer Science, 1998. URL `http://graphics.stanford.edu/~bjohanso/gp-music/tech-report/`.
49. Colin G. Johnson and Riccardo Poli. Gp-music: An interactive genetic programming system for music generation with automated fitness raters. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David Edward Goldberg, Hitoshi Iba, and Rick L. Riolo, editors, *Proceedings of the Third Annual Genetic Programming Conference (GP'98)*, pages 181–186, Madison, WI, USA: University of Wisconsin, July 22–25, 1998. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL `http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/johanson_1998_GP-Music.html`.
50. Joshua R. Smith. Designing biomorphs with an interactive genetic algorithm. In Richard K. Belew and Lashon Bernard Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, pages 535–538, San Diego, CA, USA: University of California (UCSD), July 13–16, 1991. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL `http://web.media.mit.edu/~jrs/biomorphs.pdf`.
51. Lothar Thiele, Kaisa Miettinen, Pekka J. Korhonen, and Julian Molina. A preference-based interactive evolutionary algorithm for multiobjective optimization. HSE Working Paper W-412, Helsinki, Finland: Helsinki School of Economics (HSE, Helsingin kauppakorkeakoulu), January 2007. URL `http://hsepubl.lib.hse.fi/pdf/wp/w412.pdf`.
52. Thomas Weise, Ke Tang, and Alexandre Devert. A developmental solution to (dynamic) capacitated arc routing problems using genetic programming. In Terence Soule and Jason H. Moore, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'12)*, pages 831–838, Philadelphia, PA, USA: Doubletree by Hilton Hotel Philadelphia Center City, July 7–11, 2012. New York, NY, USA: Association for Computing Machinery (ACM). doi: 10.1145/2330163.2330278.
53. Felix Streichert. Evolutionäre algorithmen: Implementation und anwendungen im asset-management-bereich (evolutionary algorithms and their application to asset management). Master's thesis, Stuttgart, Germany: Universität Stuttgart, Institut A für Mechanik, August 2001. URL `http://www-ra.informatik.uni-tuebingen.de/mitarb/streiche`.

Bibliography VIII

54. Parag C. Pendharkar and Gary J. Koehler. A general steady state distribution based stopping criteria for finite length genetic algorithms. *European Journal of Operational Research (EJOR)*, 176(3):1436–1451, February 2007. doi: 10.1016/j.ejor.2005.10.050.
55. Karin Zielinski and Rainer Laur. Stopping criteria for constrained optimization with particle swarms. In Bogdan Filipič and Jurij Šilc, editors, *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA'06)*, Informacijska Družba (Information Society), pages 45–54, Ljubljana, Slovenia: Jožef Stefan International Postgraduate School, October 9–10, 2006. Ljubljana, Slovenia: Jožef Stefan Institute. URL http://www.item.uni-bremen.de/staff/zilli/zielinski06stopping_PSO.pdf.
56. Karin Zielinski and Rainer Laur. Stopping criteria for a constrained single-objective particle swarm optimization algorithm. *Informatica*, 31(1):51–59, 2007. URL http://www.item.uni-bremen.de/staff/zilli/zielinski07informatica.pdf.