合肥大學
HEFEI UNIVERSITY

Metaheuristic Optimization: State of the Art

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn · https://thomasweise.github.io

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

人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2026-10-20

Outline

1. Introduction
2. Metaheuristics
3. Discrete Optimization
4. (Multi-)Permutation Problems
5. Summary
6. Advertisement



Introduction



Introduction

- Today, we will discuss the research frontier on metaheuristic optimization.

Introduction

- Today, we will discuss the research frontier on metaheuristic optimization.
- Therefore, we will first introduce what metaheuristic optimization is and why we need.

Introduction



- Today, we will discuss the research frontier on metaheuristic optimization.
- Therefore, we will first introduce what metaheuristic optimization is and why we need.
- Then we will discuss the current SOTA on several of the many, many, many domains where metaheuristics are applied.

Introduction



- Today, we will discuss the research frontier on metaheuristic optimization.
- Therefore, we will first introduce what metaheuristic optimization is and why we need.
- Then we will discuss the current SOTA on several of the many, many, many domains where metaheuristics are applied.
- I can, of course, not discuss all algorithms or applications in detail.

Introduction



- Today, we will discuss the research frontier on metaheuristic optimization.
- Therefore, we will first introduce what metaheuristic optimization is and why we need.
- Then we will discuss the current SOTA on several of the many, many, many domains where metaheuristics are applied.
- I can, of course, not discuss all algorithms or applications in detail.
- However, I will introduce each problem, provide typical benchmarks, baselines, and SOTA algorithms.

Introduction



- Today, we will discuss the research frontier on metaheuristic optimization.
- Therefore, we will first introduce what metaheuristic optimization is and why we need.
- Then we will discuss the current SOTA on several of the many, many, many domains where metaheuristics are applied.
- I can, of course, not discuss all algorithms or applications in detail.
- However, I will introduce each problem, provide typical benchmarks, baselines, and SOTA algorithms.
- And many, many literature references.

Introduction



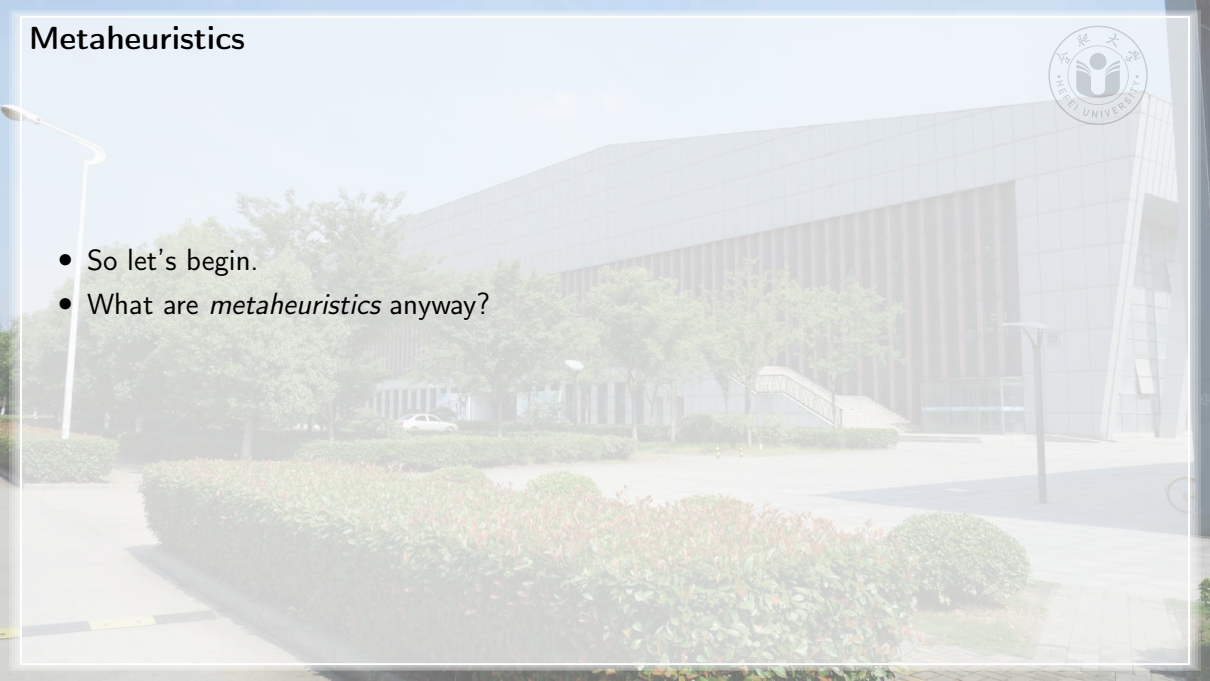
- Today, we will discuss the research frontier on metaheuristic optimization.
- Therefore, we will first introduce what metaheuristic optimization is and why we need.
- Then we will discuss the current SOTA on several of the many, many, many domains where metaheuristics are applied.
- I can, of course, not discuss all algorithms or applications in detail.
- However, I will introduce each problem, provide typical benchmarks, baselines, and SOTA algorithms.
- And many, many literature references.
- Therefore, if this lecture interests you in this field, you will have many starting points for your own research.

Metaheuristics



- So let's begin.

Metaheuristics



- So let's begin.
- What are *metaheuristics* anyway?

Metaheuristics

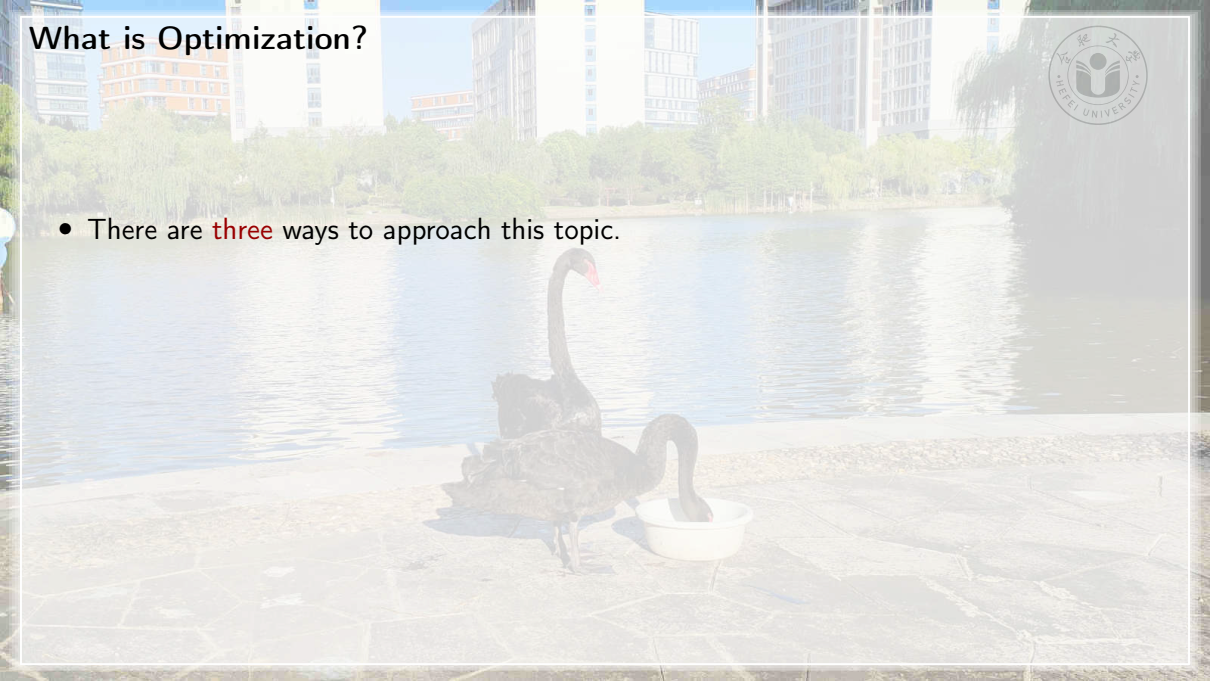
- So let's begin.
- What are *metaheuristics* anyway?
- Metaheuristics are a class of algorithms used for optimization.

Metaheuristics



- So let's begin.
- What are *metaheuristics* anyway?
- Metaheuristics are a class of algorithms used for optimization.
- So what is optimization?

What is Optimization?



- There are three ways to approach this topic.

What is Optimization?



- There are three ways to approach this topic
  1. Optimization means searching for superlatives.

What is Optimization?

- There are three ways to approach this topic
  1. Optimization means searching for superlatives.
  2. Optimization is the art of solving hard problems.

What is Optimization?

- There are three ways to approach this topic
  1. Optimization means searching for superlatives.
  2. Optimization is the art of solving hard problems.
  3. Optimization allows us to find acceptable solutions for problems where our tools from high school maths *and* traditional algorithms from computer science are no longer enough, problems where exact solutions cannot always be reached.

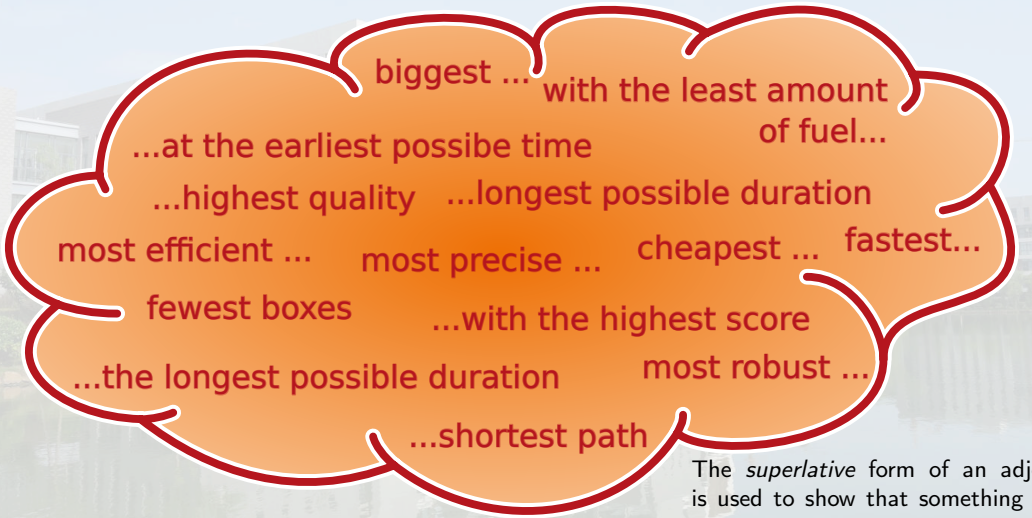
Views on Optimization, Part 1



- Let's start with the first view: Superlatives.

Optimization is the search for superlatives

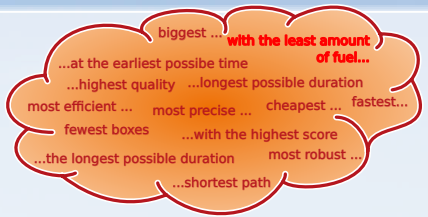
- Optimization means finding superlatives.



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

Optimization is the search for superlatives

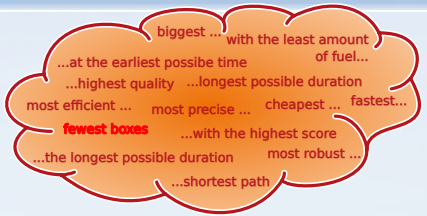
- Optimization means finding superlatives.
- Pick up and deliver packages from different places to customers using the least amount of fuel[188,233–235].

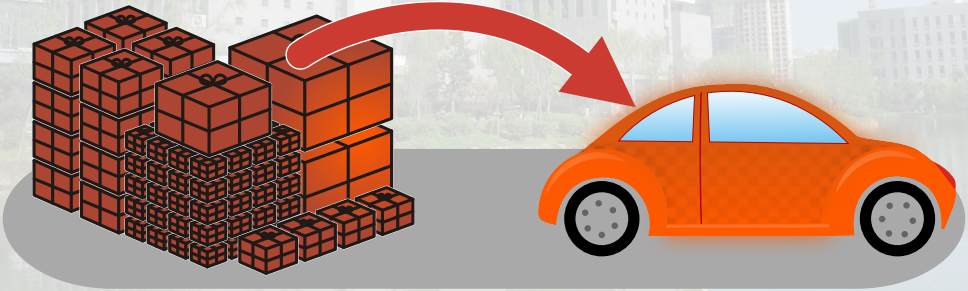




Optimization is the search for superlatives

- Optimization means finding superlatives.
- Pick up and deliver packages from different places to customers using the least amount of fuel[188,233–235].
- Pack a set of things into the fewest boxes[74,198,252,253].

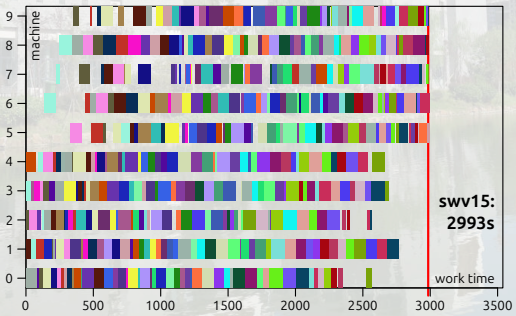




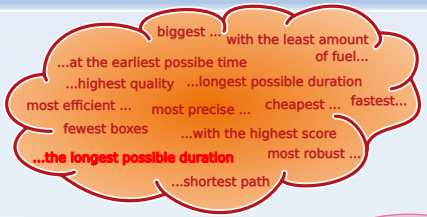
Optimization is the search for superlatives



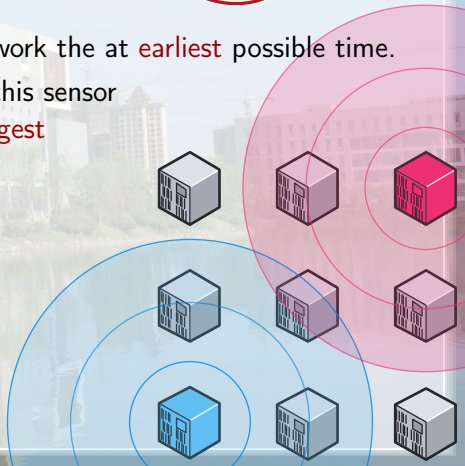
- Optimization means finding superlatives.
- Pick up and deliver packages from different places to customers using the least amount of fuel[188,233–235].
- Pack a set of things into the fewest boxes[74,198,252,253].
- Assign tasks to machines such that we can finish our work the at earliest possible time.



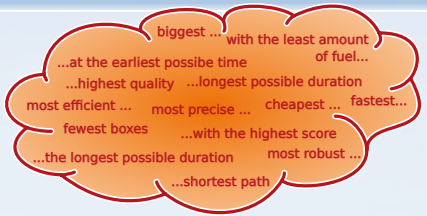
Optimization is the search for superlatives



- Optimization means finding superlatives.
- Pick up and deliver packages from different places to customers using the least amount of fuel[188,233–235].
- Pack a set of things into the fewest boxes[74,198,252,253].
- Assign tasks to machines such that we can finish our work the at earliest possible time.
- Find a strategy to manage the power of the nodes in this sensor network so that full coverage is guaranteed for the longest possible duration.



Optimization is the search for superlatives



- Optimization means finding superlatives.
- Pick up and deliver packages from different places to customers using the least amount of fuel[188,233–235].
- Pack a set of things into the fewest boxes[74,198,252,253].
- Assign tasks to machines such that we can finish our work the at earliest possible time.
- Find a strategy to manage the power of the nodes in this sensor network so that full coverage is guaranteed for the longest possible duration.
- And so on.

Views on Optimization, Part 2



- Now let us consider the second view: Solving hard problems.

Views on Optimization, Part 2



- Now let us consider the second view: Solving hard problems.
- Later, we learn how to quantify *hard*.

Views on Optimization, Part 2



- Now let us consider the second view: Solving hard problems.
- Later, we learn how to quantify *hard*.
- For now, let's first think about how to define *problem*?

What are Optimization Problems?

- There are two ways to define optimization problems.

What are Optimization Problems?

- The economic view.

Optimization Problem

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

What are Optimization Problems?

- The mathematical view.

Optimization Problem

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

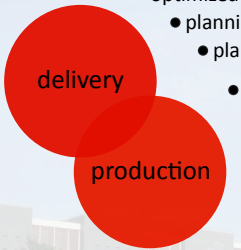
Solving an optimization problem requires finding an input element $x^\star$ within a set $\mathbb{X}$ of allowed elements for which a mathematical function $f\colon \mathbb{X} \mapsto \mathbb{R}$ takes on the smallest possible value.

Optimization in Business

- optimized logistics (business-to-customer)
  - planning and scheduling of maintenance visits
    - planning and scheduling of supply visits

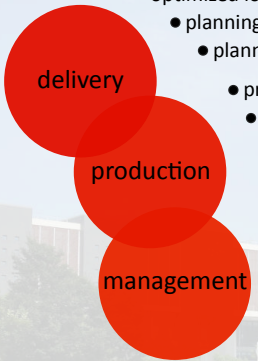


Optimization in Business



- optimized logistics (business-to-customer)
  - planning and scheduling of maintenance visits
  - planning and scheduling of supply visits
- production planning and scheduling
  - optimized assignment of jobs/orders to machines
  - optimization of production processes
  - optimization of stock-keeping
  - optimization of intra-enterprise logistics
  - optimization of supply chains
  - optimization of factory layouts and intra-factory logistics

Optimization in Business



- optimized logistics (business-to-customer)
- planning and scheduling of maintenance visits
- planning and scheduling of supply visits

- production planning and scheduling
- optimized assignment of jobs/orders to machines
- optimization of production processes
- optimization of stock-keeping
- optimization of intra-enterprise logistics
- optimization of supply chains
- optimization of factory layouts and intra-factory logistics

- scheduling of employee work
- optimal assignment of employees to tasks or customers
- optimized locations for new branch offices
  (based current or predicted future customers)

Optimization in Business



- optimized logistics (business-to-customer)
- planning and scheduling of maintenance visits
- planning and scheduling of supply visits

- production planning and scheduling
- optimized assignment of jobs/orders to machines
- optimization of production processes
- optimization of stock-keeping
- optimization of intra-enterprise logistics
- optimization of supply chains
- optimization of factory layouts and intra-factory logistics

- scheduling of employee work
- optimal assignment of employees to tasks or customers
- optimized locations for new branch offices
  (based current or predicted future customers)

- optimization of product design
- optimization of product feature configuration
- optimization of service offers
- improved tailoring of products/services to customers

Optimization in Business



- optimized logistics (business-to-customer)
- planning and scheduling of maintenance visits
- planning and scheduling of supply visits

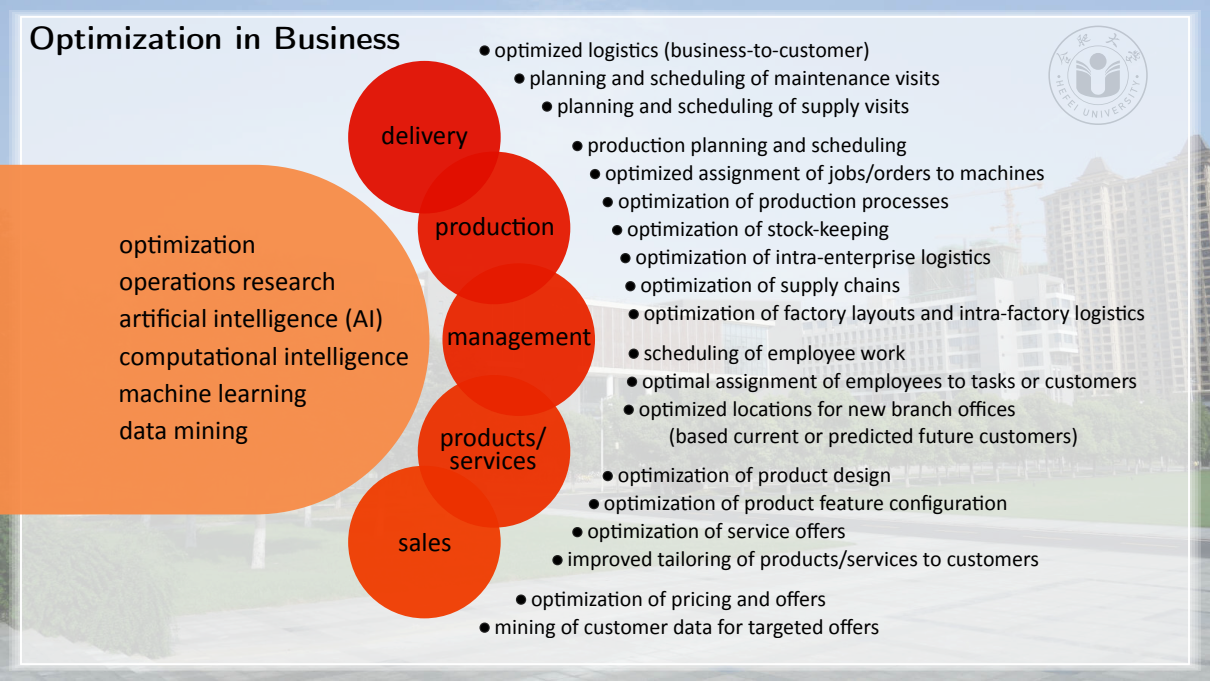
- production planning and scheduling
- optimized assignment of jobs/orders to machines
- optimization of production processes
- optimization of stock-keeping
- optimization of intra-enterprise logistics
- optimization of supply chains
- optimization of factory layouts and intra-factory logistics

- scheduling of employee work
- optimal assignment of employees to tasks or customers
- optimized locations for new branch offices
  (based current or predicted future customers)

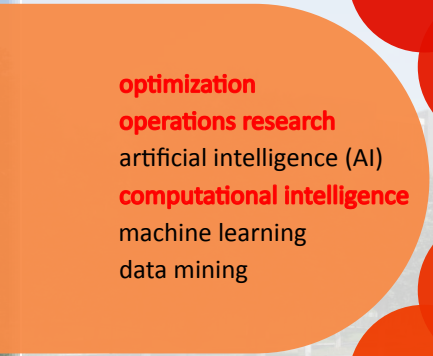
- optimization of product design
- optimization of product feature configuration
- optimization of service offers
- improved tailoring of products/services to customers

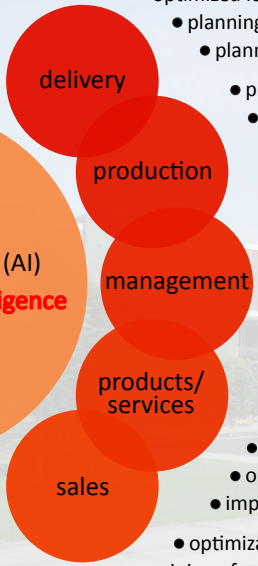
- optimization of pricing and offers
- mining of customer data for targeted offers

Optimization in Business



Optimization in Business





- optimized logistics (business-to-customer)
- planning and scheduling of maintenance visits
- planning and scheduling of supply visits

- production planning and scheduling
- optimized assignment of jobs/orders to machines
- optimization of production processes
- optimization of stock-keeping
- optimization of intra-enterprise logistics
- optimization of supply chains
- optimization of factory layouts and intra-factory logistics

- scheduling of employee work
- optimal assignment of employees to tasks or customers
- optimized locations for new branch offices
  (based current or predicted future customers)

- optimization of product design
- optimization of product feature configuration
- optimization of service offers
- improved tailoring of products/services to customers

- optimization of pricing and offers
- mining of customer data for targeted offers

Views on Optimization, Part 3



- This brings us to the third view of optimization: We need to solve problems where equations are "not enough" and for which we cannot always find the best solution.

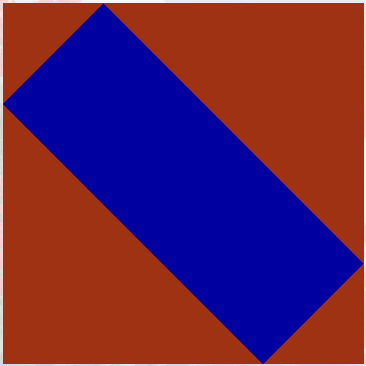
Views on Optimization, Part 3



- This brings us to the third view of optimization: We need to solve problems where equations are "not enough" and for which we cannot always find the best solution.
- Let's start with high school Maths, move to undergraduate computer science, and then approach this field of problems.

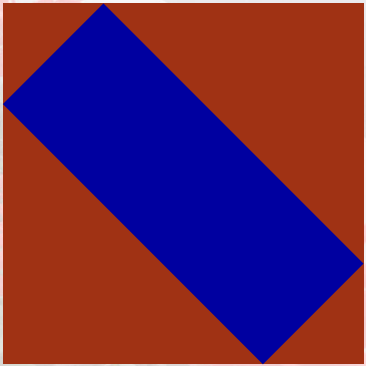
A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.



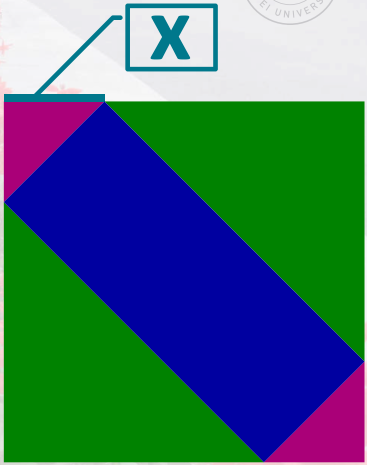
A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.
- What area has the *largest* such rectangle?



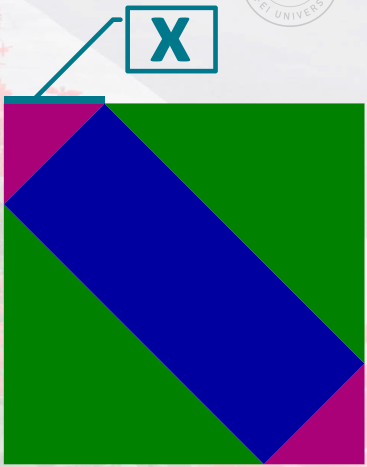
A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.
- What area has the *largest* such rectangle?
- We can solve this by realizing that $A_{\blacksquare}(x) = (13\text{cm}^2) - 2(\frac{1}{2}x^2) - 2(\frac{1}{2}(13\text{cm} - x)^2)$.



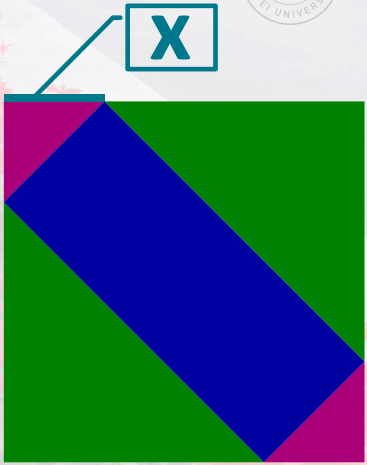
A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.
- What area has the *largest* such rectangle?
- We can solve this by realizing that $A_{\blacksquare}(x) = (13\text{cm}^2) - 2(\frac{1}{2}x^2) - 2(\frac{1}{2}(13\text{cm} - x)^2)$ and then finding the maximum by setting $0 = A'_{\blacksquare}(x) = -4x + 26\text{cm}$.



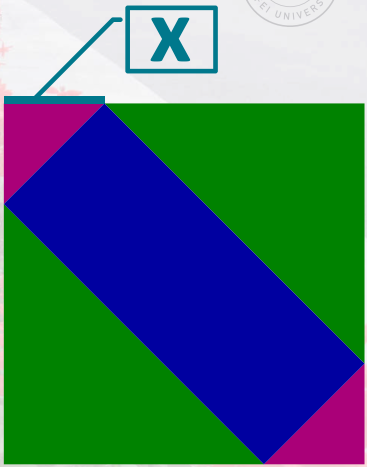
A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.
- What area has the *largest* such rectangle?
- We can solve this by realizing that $A_{\blacksquare}(x) = (13\text{cm}^2) - 2(\frac{1}{2}x^2) - 2(\frac{1}{2}(13\text{cm} - x)^2)$ and then finding the maximum by setting $0 = A'_{\blacksquare}(x) = -4x + 26\text{cm}$.
- We get $\hat{x} = 6.5\text{cm}$.



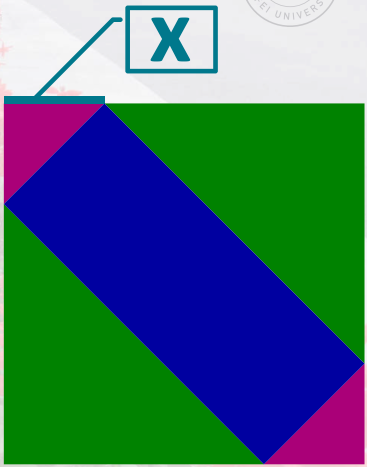
A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.
- What area has the *largest* such rectangle?
- We can solve this by realizing that $A_{\blacksquare}(x) = (13\text{cm}^2) - 2(\frac{1}{2}x^2) - 2(\frac{1}{2}(13\text{cm} - x)^2)$ and then finding the maximum by setting $0 = A'_{\blacksquare}(x) = -4x + 26\text{cm}$.
- We get $\hat{x} = 6.5\text{cm}$ and, hence, $\widehat{A_{\blacksquare}} = 84.5\text{cm}^2$.



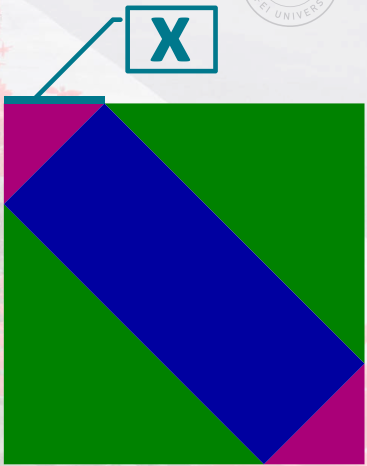
A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.
- What area has the *largest* such rectangle?
- We can solve this by realizing that $A_{\blacksquare}(x) = \left(13\text{cm}^2\right) - 2\left(\frac{1}{2}x^2\right) - 2\left(\frac{1}{2}(13\text{cm} - x)^2\right)$ and then finding the maximum by setting $0 = A'_{\blacksquare}(x) = -4x + 26\text{cm}$.
- We get $\hat{x} = 6.5\text{cm}$ and, hence, $\widehat{A_{\blacksquare}} = 84.5\text{cm}^2$.
- Maxima and minima are superlatives = optima.



A Problem that we can solve with high school Maths equations

- A rectangle should be placed inside a square with side length 13cm as sketched.
- What area has the *largest* such rectangle?
- We can solve this by realizing that $A_{\blacksquare}(x) = \left(13\text{cm}^2\right) - 2\left(\frac{1}{2}x^2\right) - 2\left(\frac{1}{2}(13\text{cm} - x)^2\right)$ and then finding the maximum by setting $0 = A'_{\blacksquare}(x) = -4x + 26\text{cm}$.
- We get $\hat{x} = 6.5\text{cm}$ and, hence, $\widehat{A_{\blacksquare}} = 84.5\text{cm}^2$.
- Maxima and minima are superlatives $=$ optima.
- Sometimes we can simply compute them and do not need fancy algorithms.



Problems that algorithms can solve efficiently



Problems that algorithms can solve efficiently

- I am at the starting point 合肥大学南二区北大门.



Problems that algorithms can solve efficiently



- I am at the starting point 合肥大学南二区北大门.
- I want to go to the goal: 食堂.

Problems that algorithms can solve efficiently



- I am at the starting point 合肥大学南二区北大门.
- I want to go to the goal: 食堂.
- How do I get there the fastest?

Problems that algorithms can solve efficiently



- I am at the starting point 合肥大学南二区北大门.
- I want to go to the goal: 食堂.
- How do I get there the fastest?
- We know the campus map.

Problems that algorithms can solve efficiently



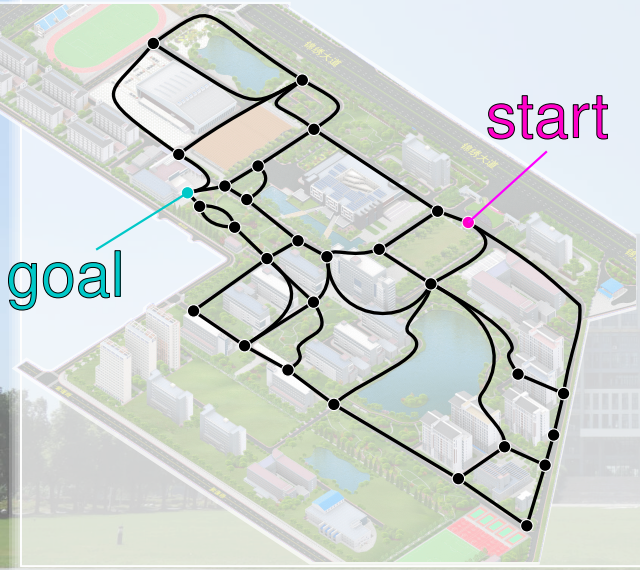
- I am at the starting point 合肥大学南二区北大门.
- I want to go to the goal: 食堂.
- How do I get there the fastest?
- We know the campus map.
- We want to compute the shortest path (before actually walking it).

Problems that algorithms can solve efficiently



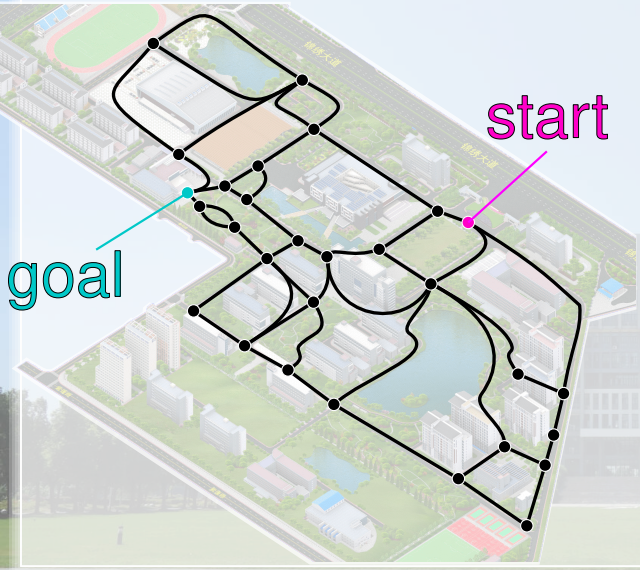
- I am at the starting point 合肥大学南二区北大门.
- I want to go to the goal: 食堂.
- How do I get there the fastest?
- We know the campus map.
- We want to compute the shortest path (before actually walking it).
- We know all the intersections where I could make turns.

Problems that algorithms can solve efficiently



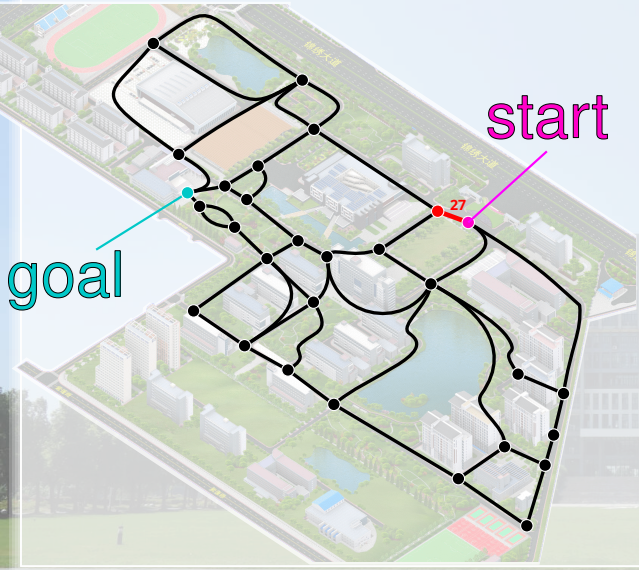
- I am at the starting point 合肥大学南二区北大门.
- I want to go to the goal: 食堂.
- How do I get there the fastest?
- We know the campus map.
- We want to compute the shortest path (before actually walking it).
- We know all the intersections where I could make turns.
- There is no mathematical equation that we can use to solve this.

Problems that algorithms can solve efficiently



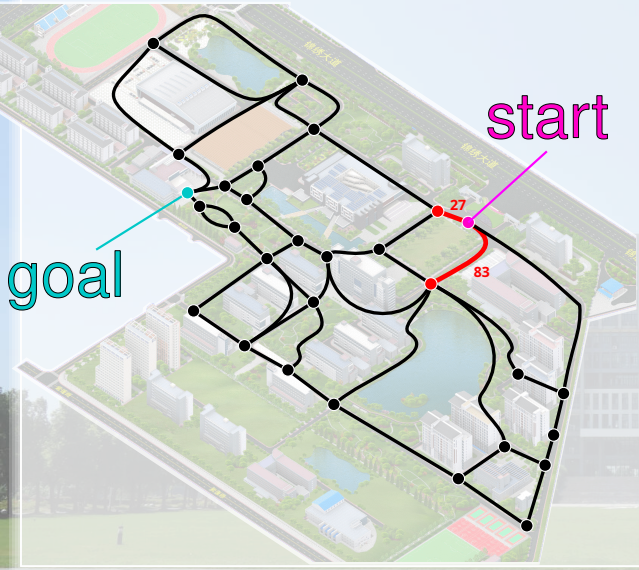
- I am at the starting point 合肥大学南二区北大门.
- I want to go to the goal: 食堂.
- How do I get there the fastest?
- We know the campus map.
- We want to compute the shortest path (before actually walking it).
- We know all the intersections where I could make turns.
- There is no mathematical equation that we can use to solve this.
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.

Problems that algorithms can solve efficiently



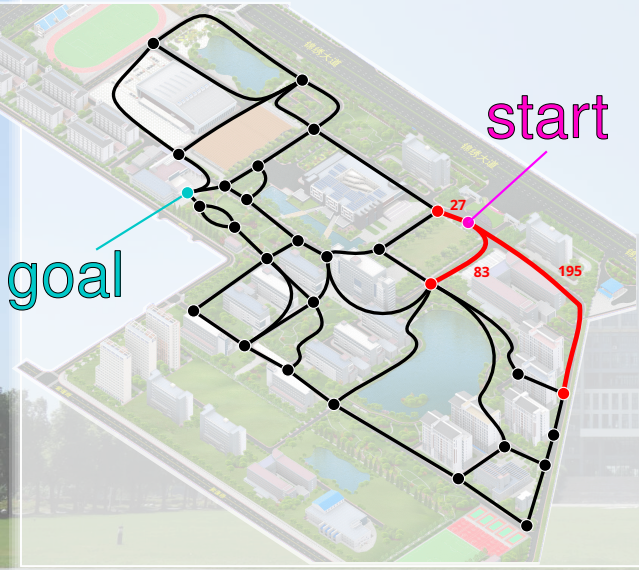
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- For example, I could walk for $27s$ to this intersection.

Problems that algorithms can solve efficiently



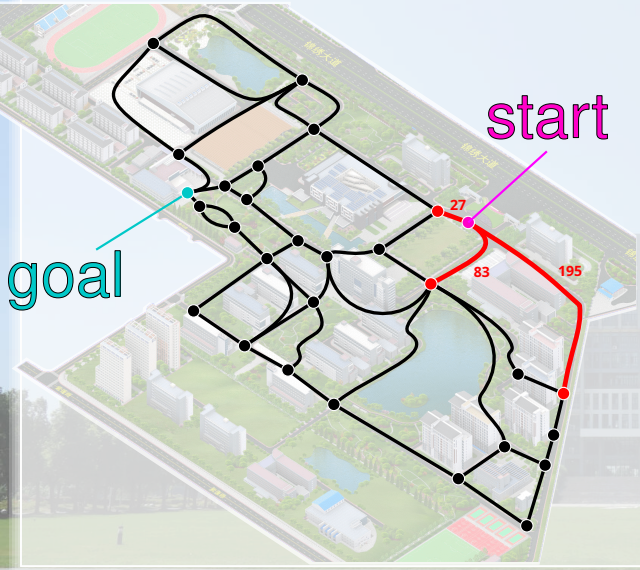
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- For example, I could walk for $27s$ to this intersection.
- Or for $83s$ to that one.

Problems that algorithms can solve efficiently



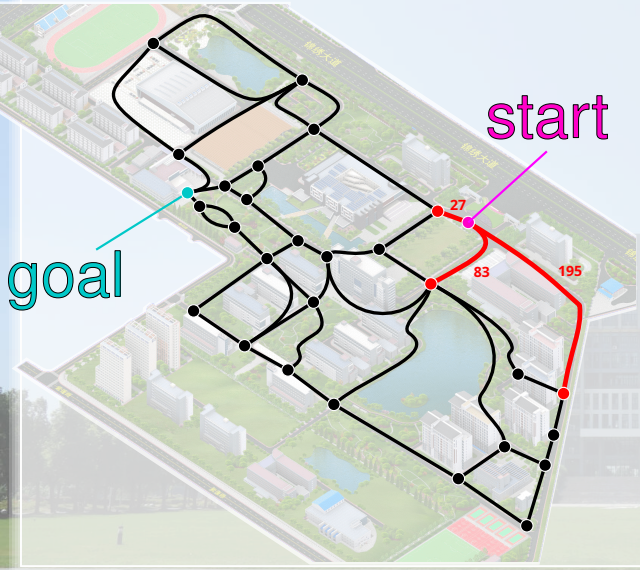
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- For example, I could walk for $27s$ to this intersection.
- Or for $83s$ to that one.
- Or for $195s$ to that one.

Problems that algorithms can solve efficiently



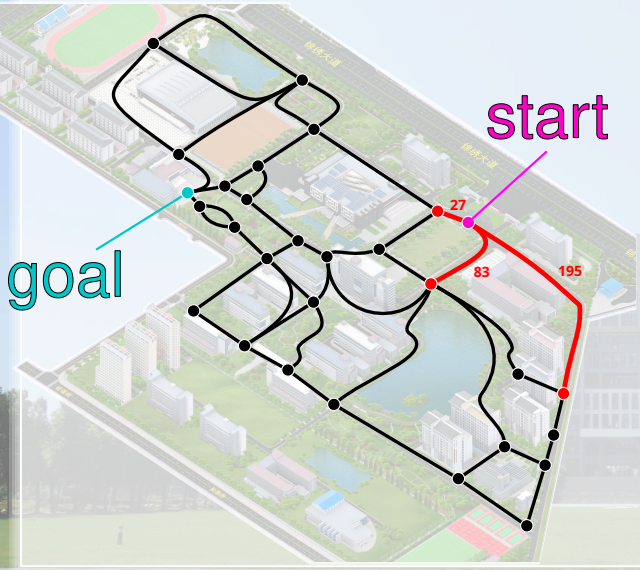
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- For example, I could walk for $27s$ to this intersection.
- Or for $83s$ to that one.
- Or for $195s$ to that one.
- We have 3 choices: (a) $27s$, (b) $83s$, (c) $195s$.

Problems that algorithms can solve efficiently



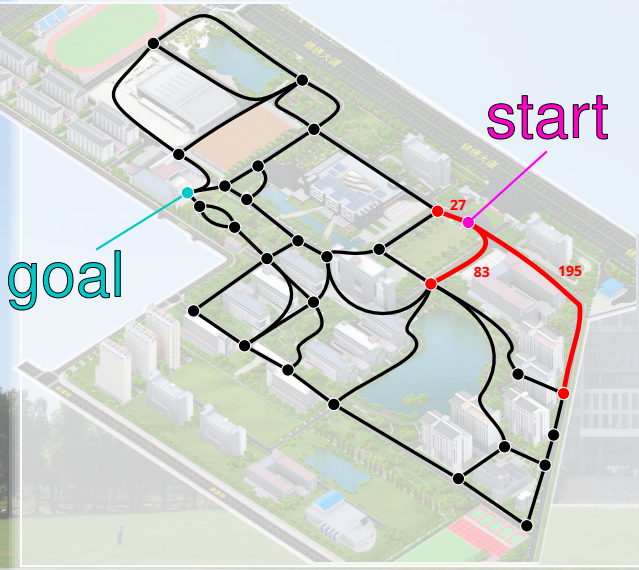
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- For example, I could walk for $27s$ to this intersection.
- Or for $83s$ to that one.
- Or for $195s$ to that one.
- We have 3 choices: (a) $27s$, (b) $83s$, (c) $195s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.

Problems that algorithms can solve efficiently



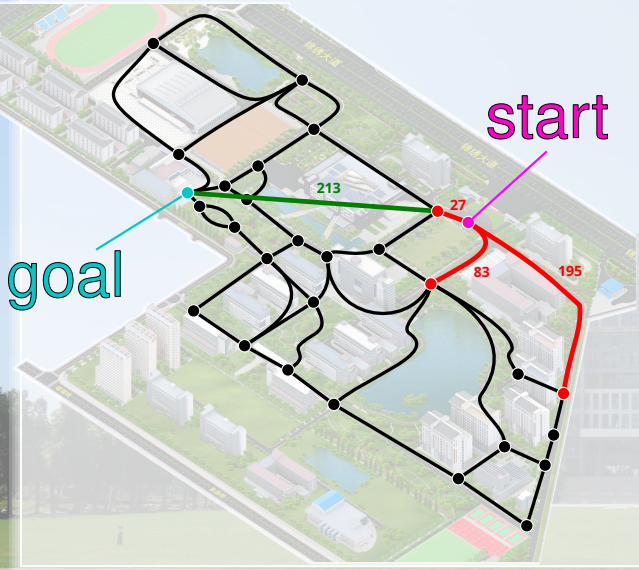
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- For example, I could walk for $27s$ to this intersection.
- Or for $83s$ to that one.
- Or for $195s$ to that one.
- We have 3 choices: (a) $27s$, (b) $83s$, (c) $195s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.

Problems that algorithms can solve efficiently



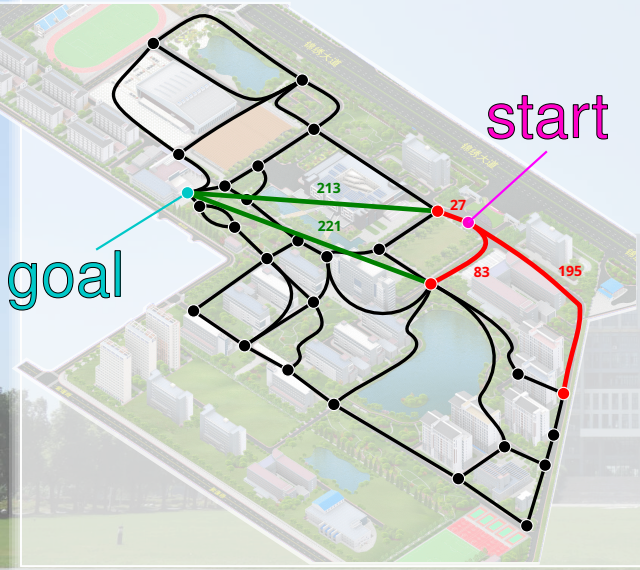
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s$, (b) $83s$, (c) $195s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.

Problems that algorithms can solve efficiently



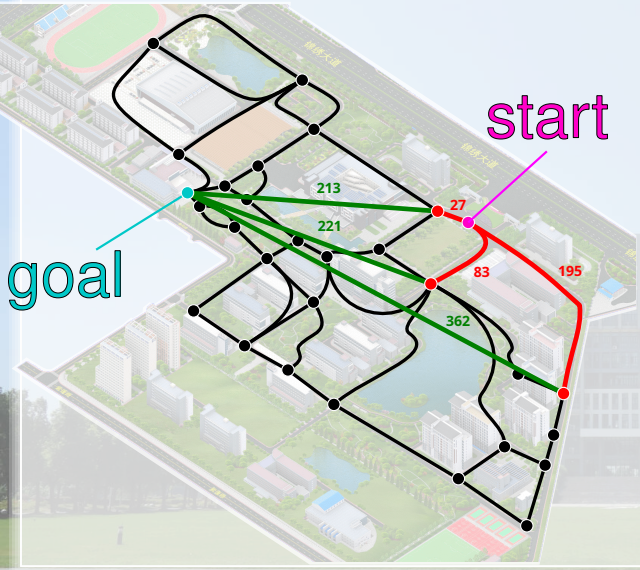
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s + 213s$, (b) $83s$, (c) $195s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.

Problems that algorithms can solve efficiently



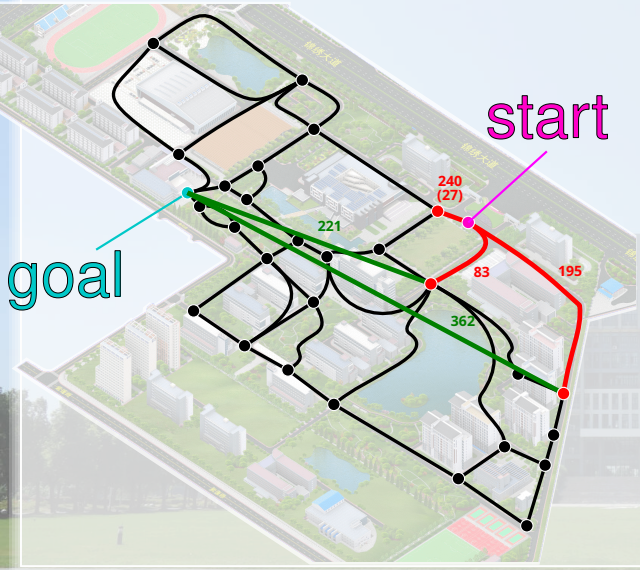
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s + 213s$, (b) $83s + 221s$, (c) $195s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.

Problems that algorithms can solve efficiently



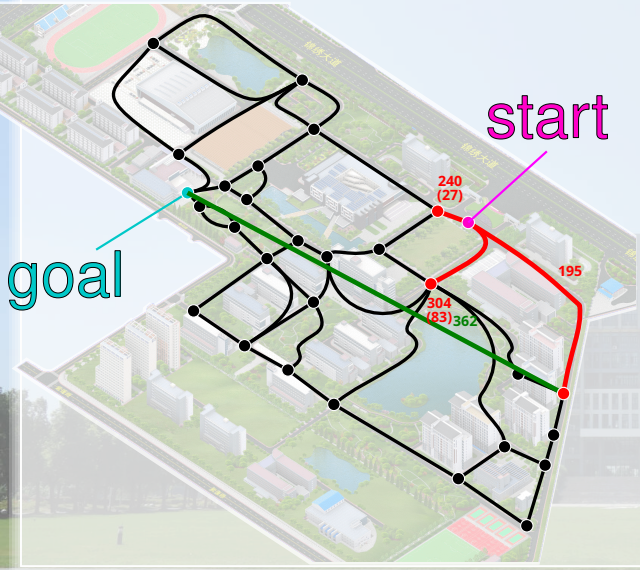
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s + 213s$, (b) $83s + 221s$, (c) $195s + 362s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.

Problems that algorithms can solve efficiently



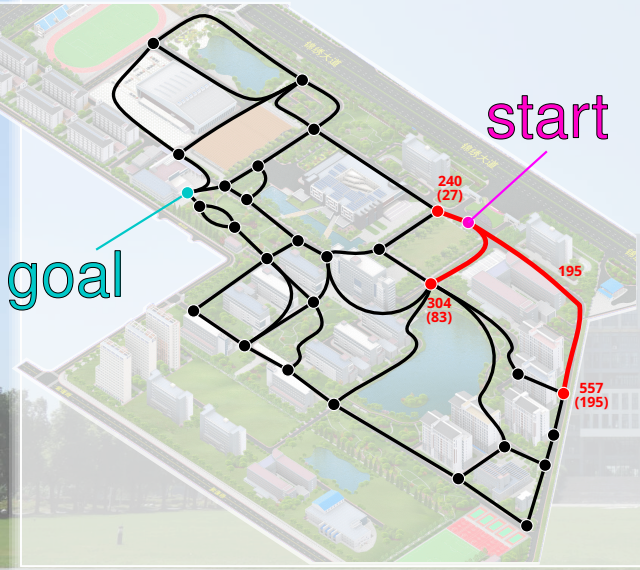
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s + 213s$, (b) $83s + 221s$, (c) $195s + 362s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.

Problems that algorithms can solve efficiently



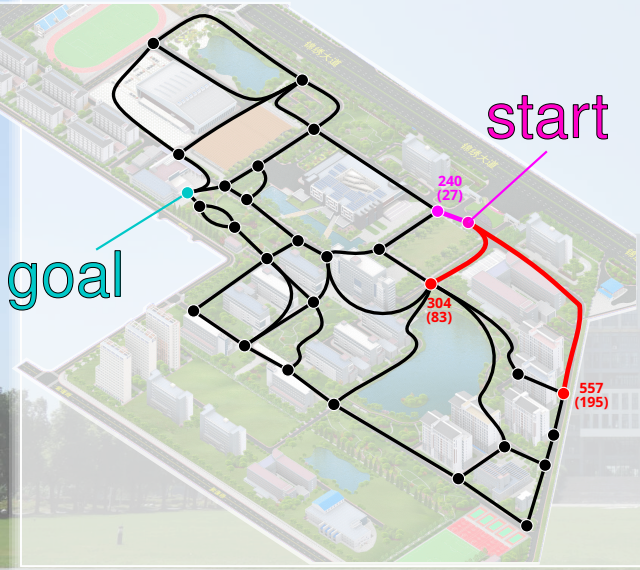
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s + 213s$, (b) $83s + 221s$, (c) $195s + 362s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.

Problems that algorithms can solve efficiently



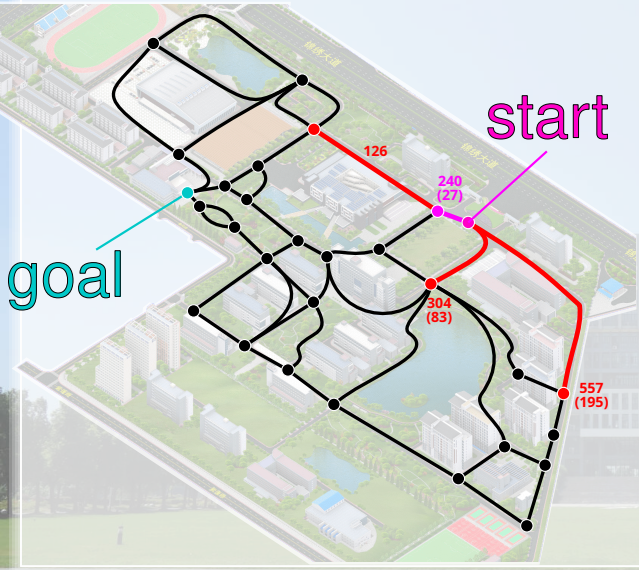
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s + 213s$, (b) $83s + 221s$, (c) $195s + 362s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.

Problems that algorithms can solve efficiently



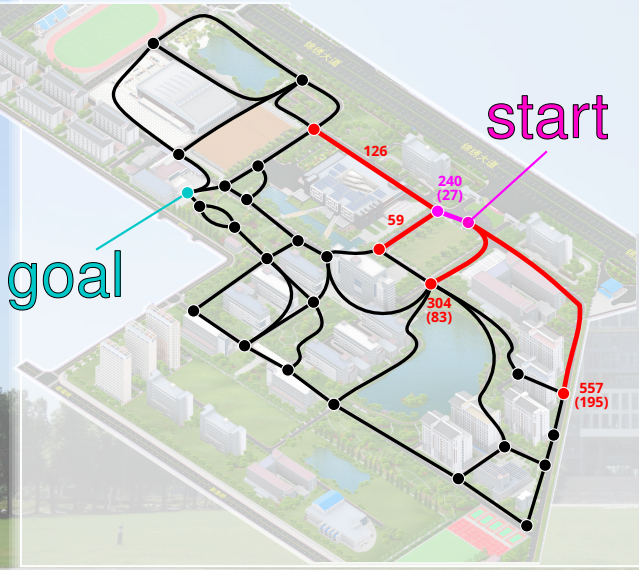
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We have 3 choices: (a) $27s + 213s$, (b) $83s + 221s$, (c) $195s + 362s$.
- For each intersection, we can compute the airline distance (as the crow flies) to the 食堂.
- The actual walking distance can never be shorter than that.
- **Adding** the two distances for each node gives us a good impression of how useful going there would be.
- (a) it is...

Problems that algorithms can solve efficiently



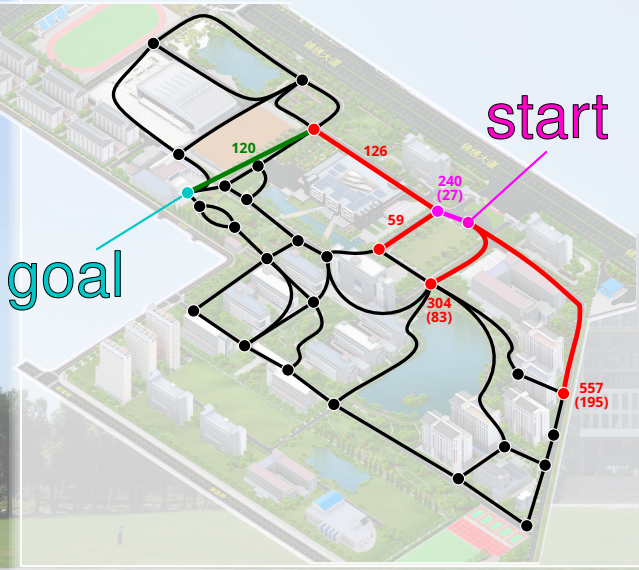
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



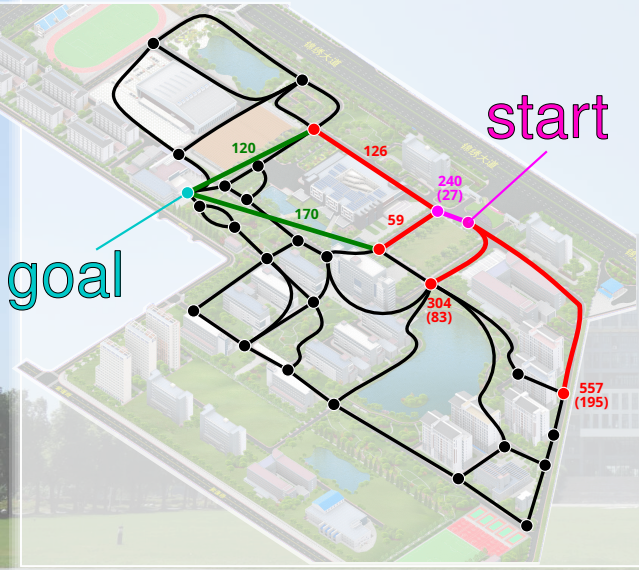
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



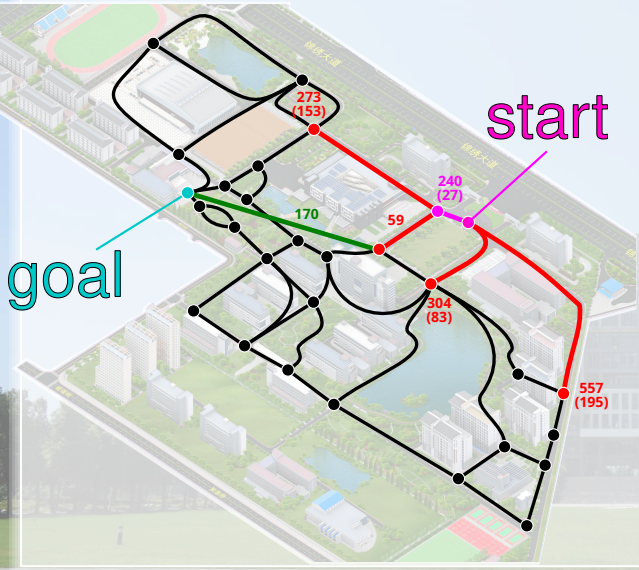
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



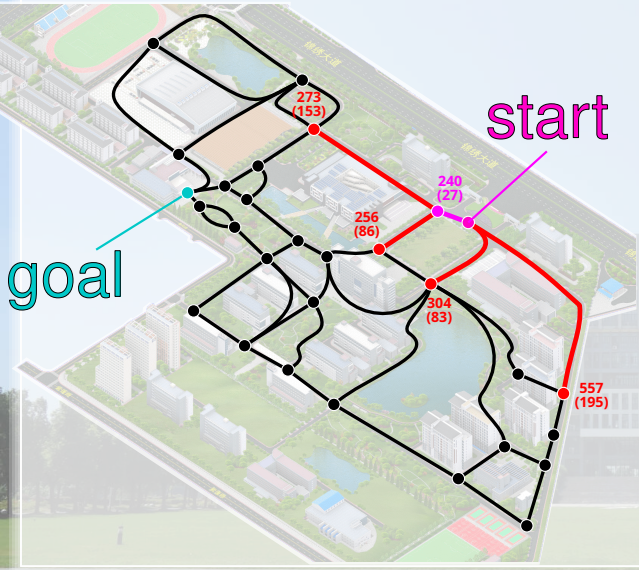
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



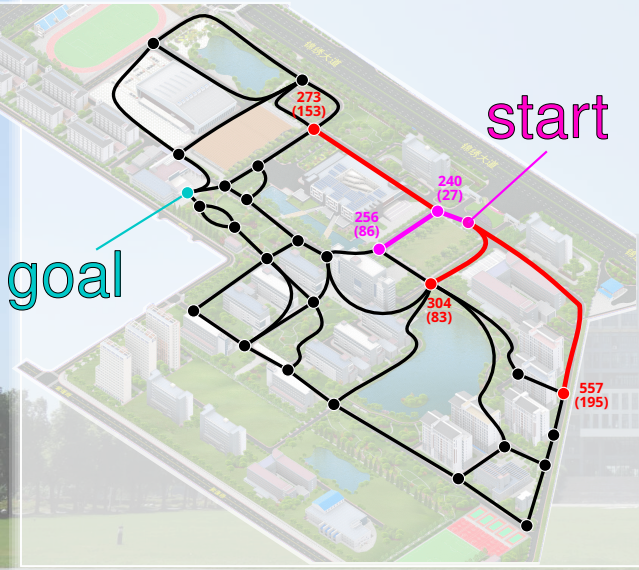
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



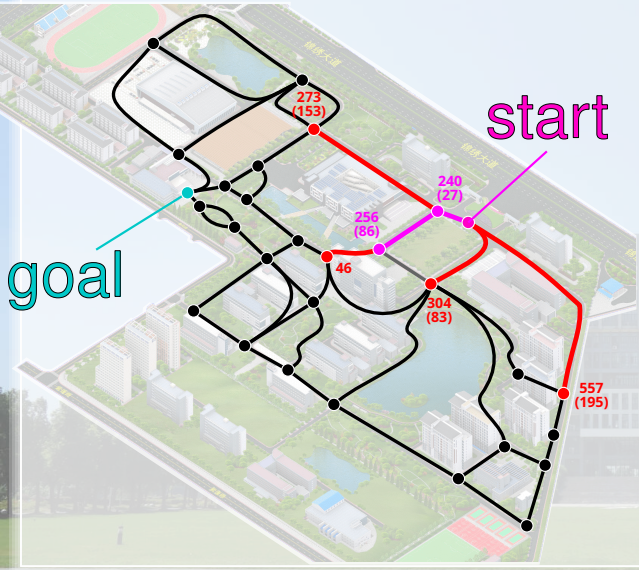
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



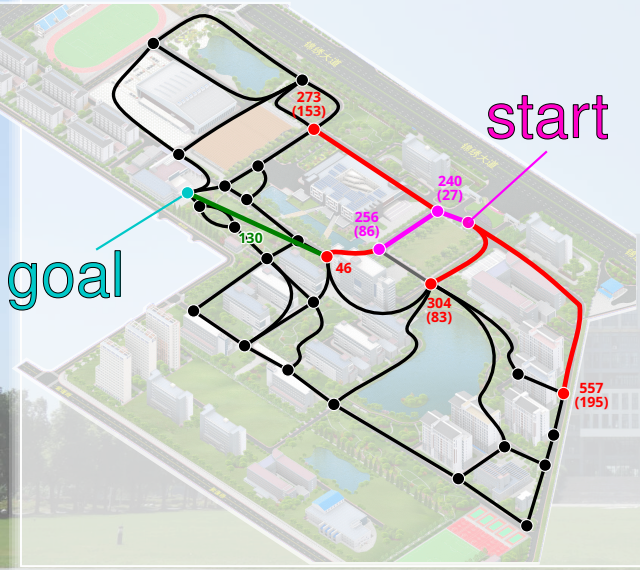
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



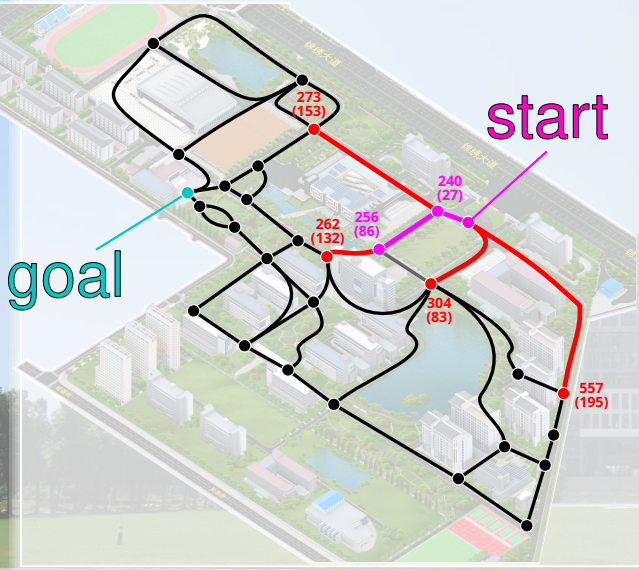
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



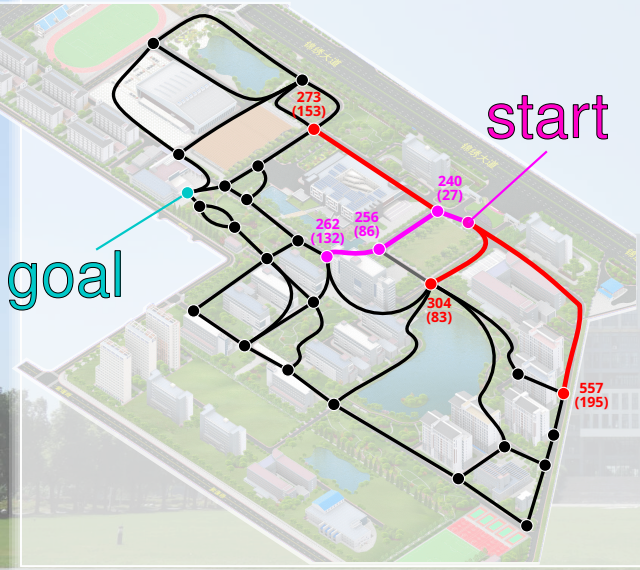
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



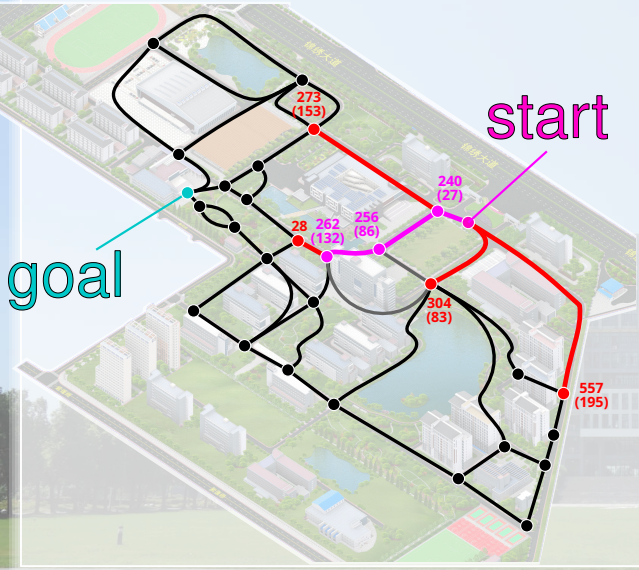
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



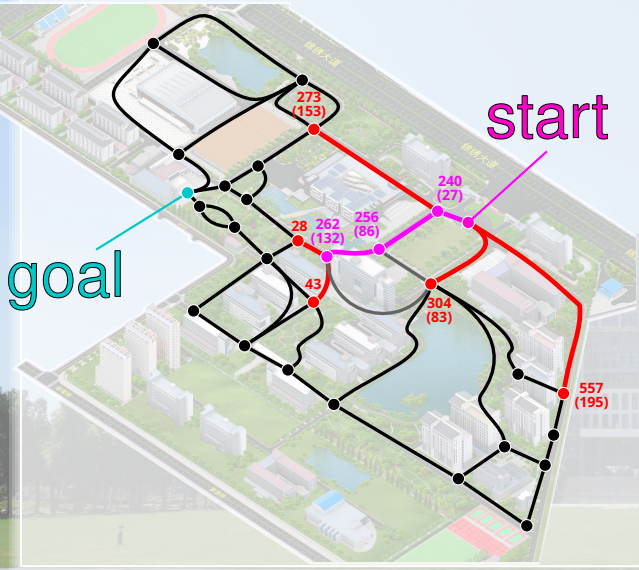
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



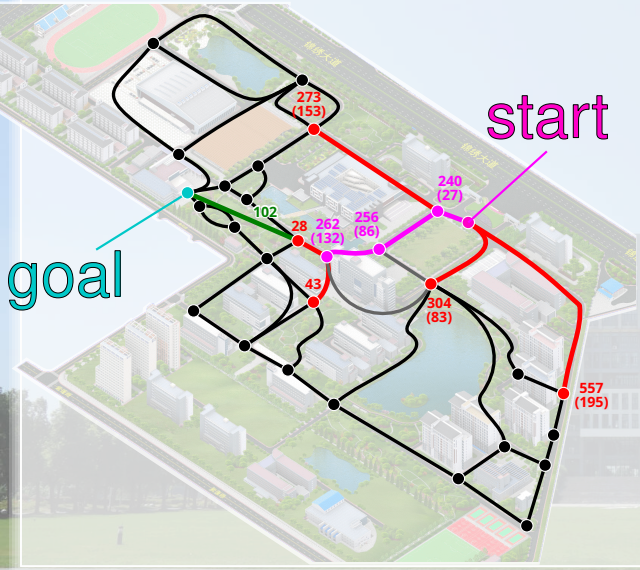
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



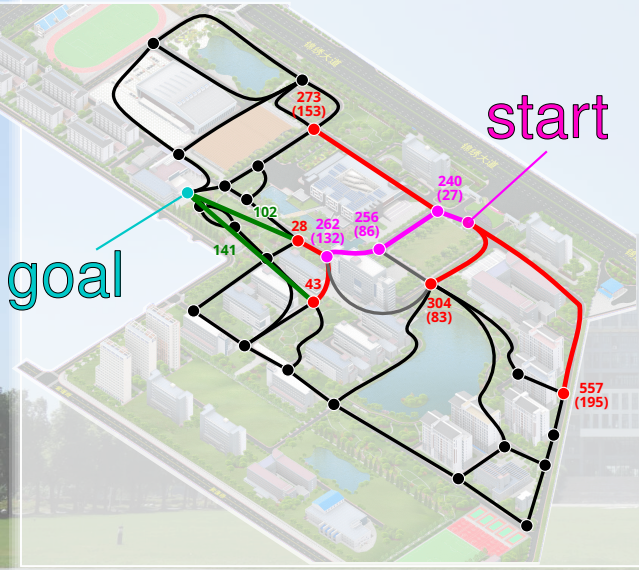
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



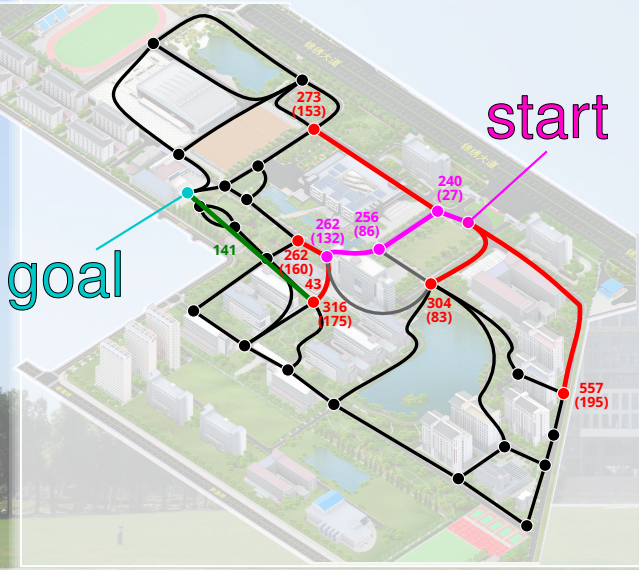
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



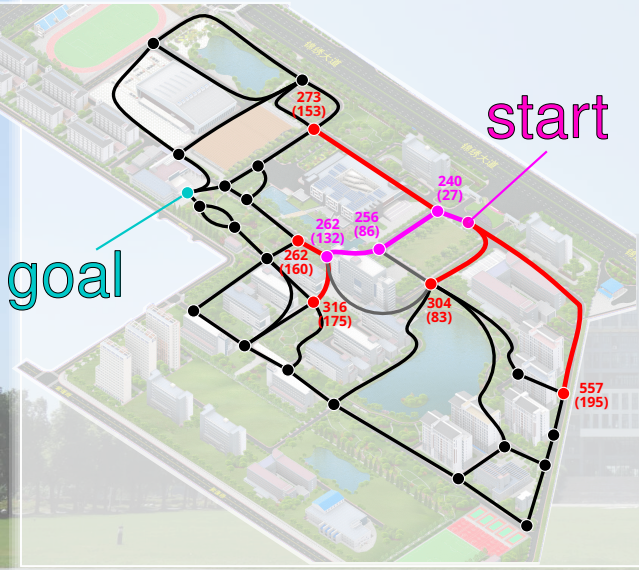
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



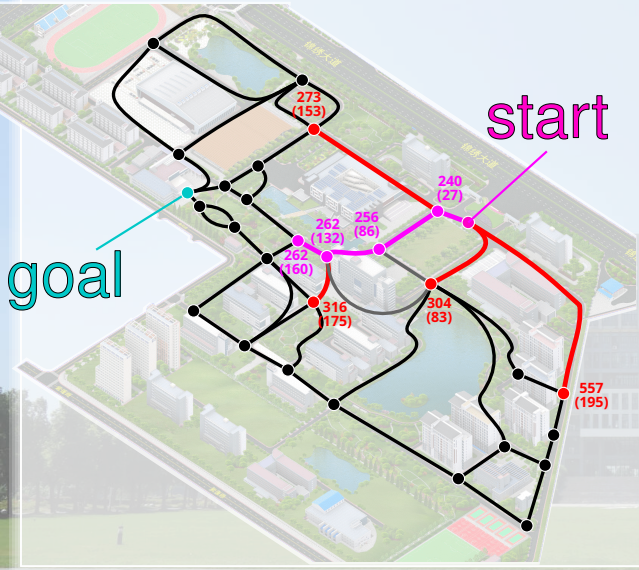
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



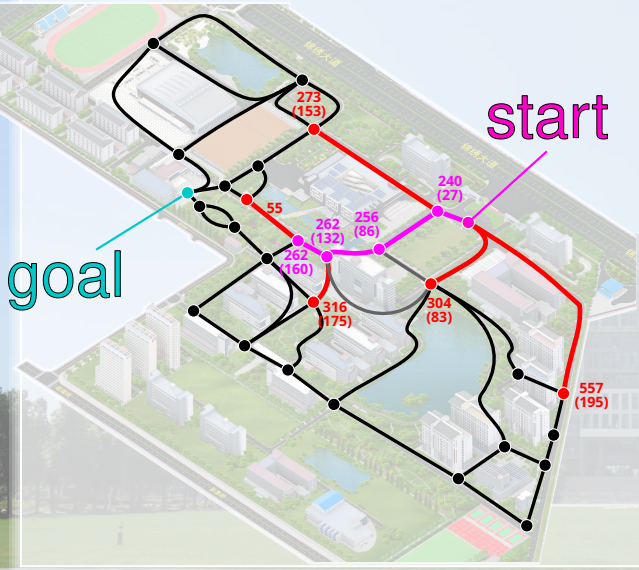
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



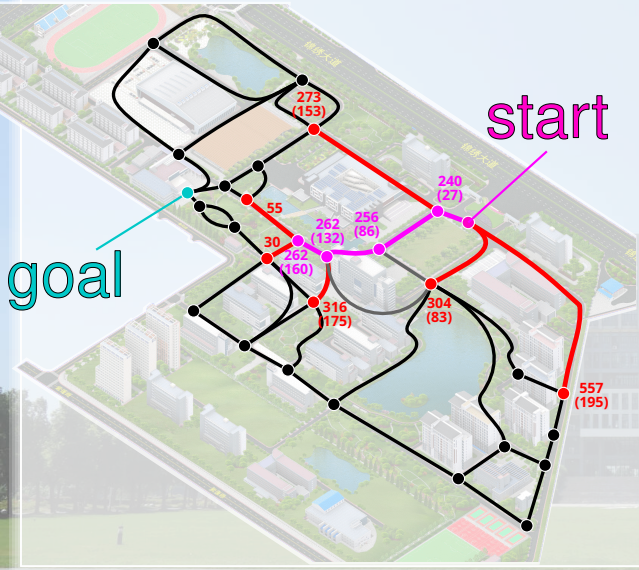
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



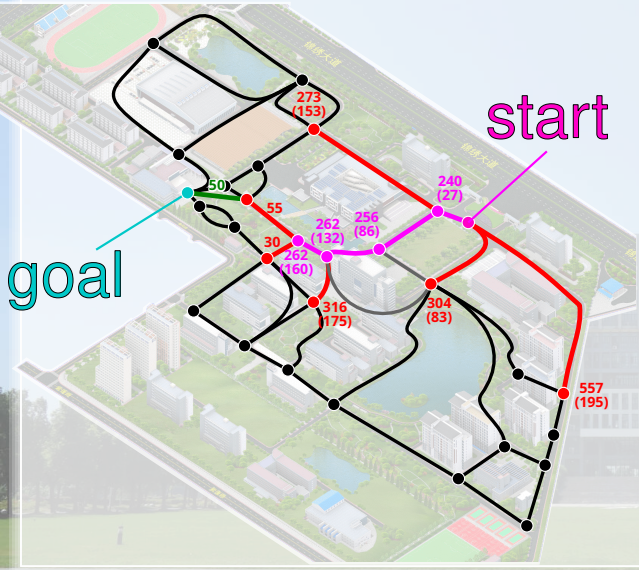
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



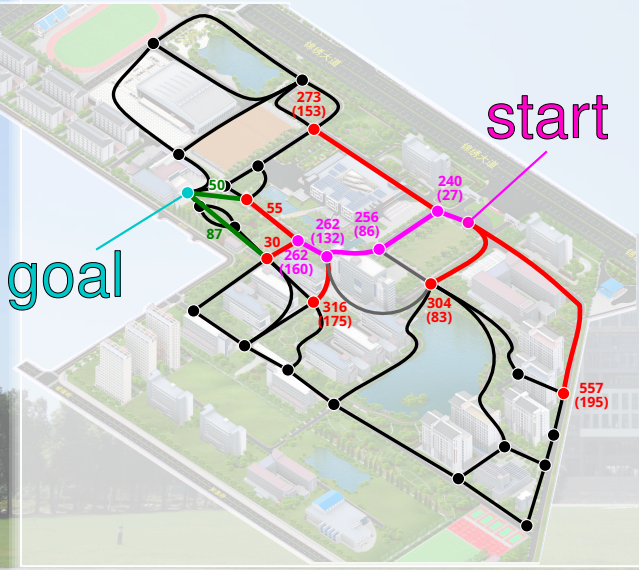
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



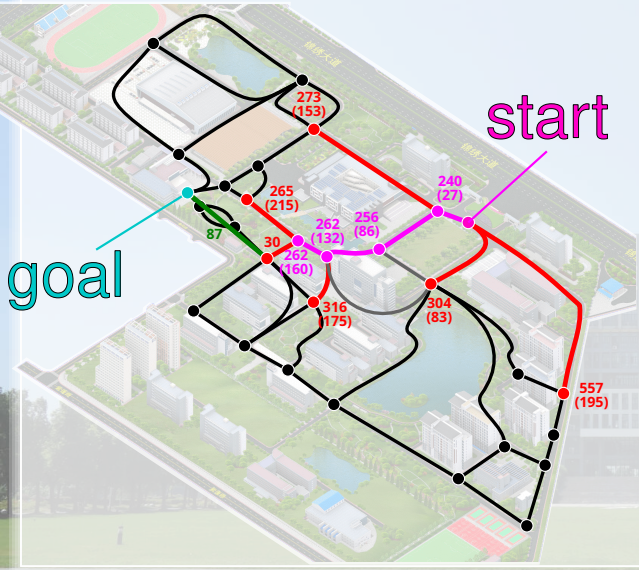
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



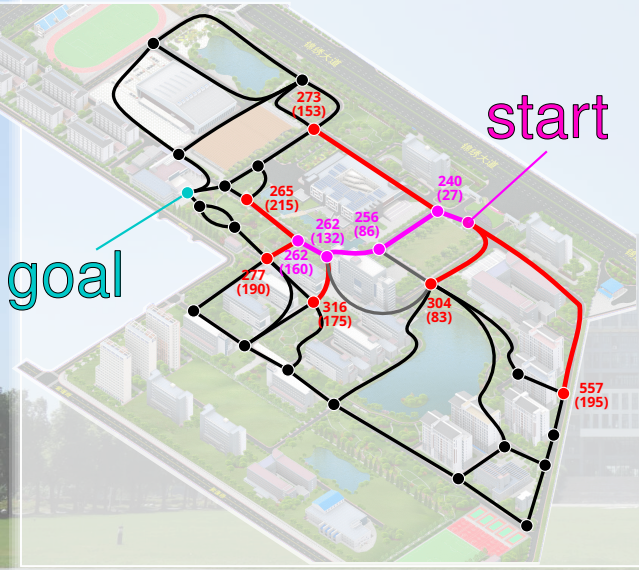
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



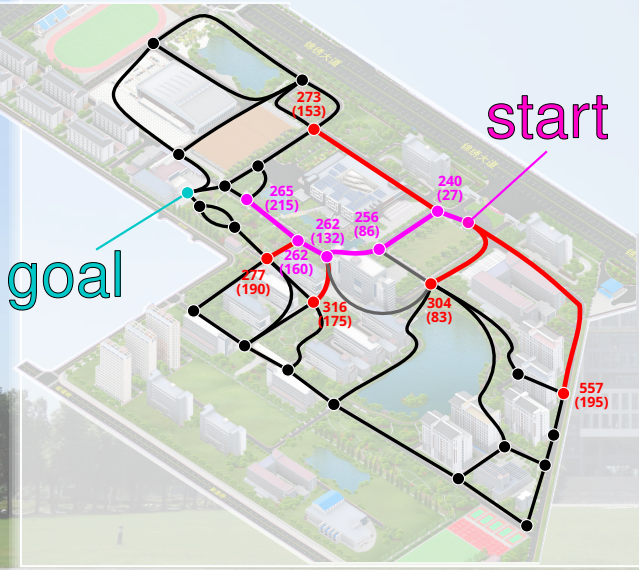
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



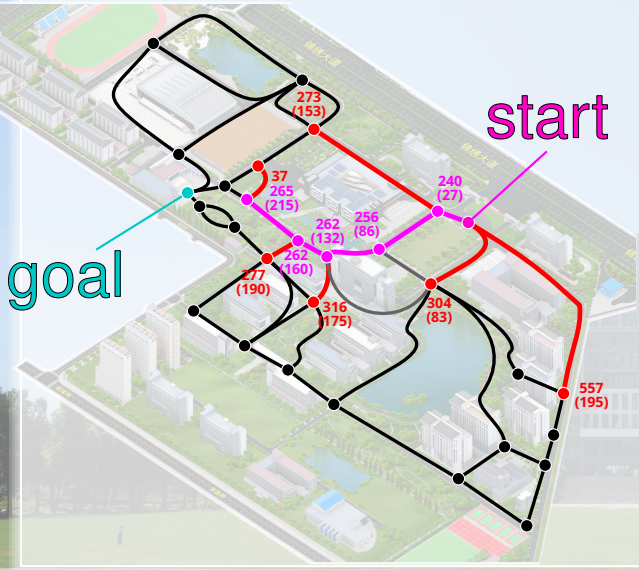
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



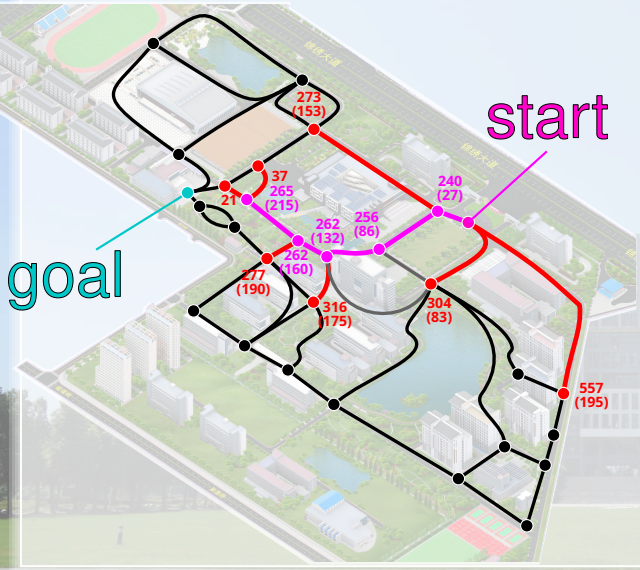
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



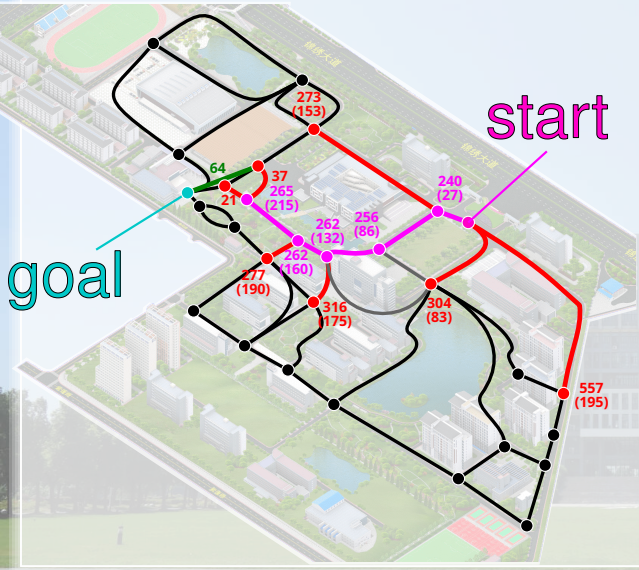
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



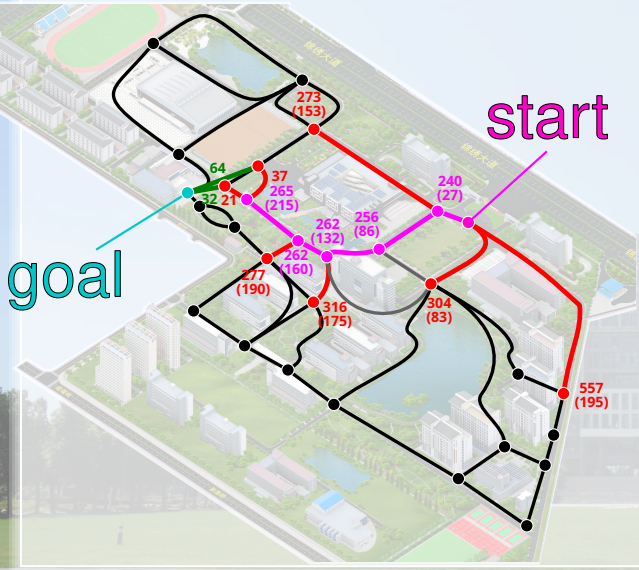
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



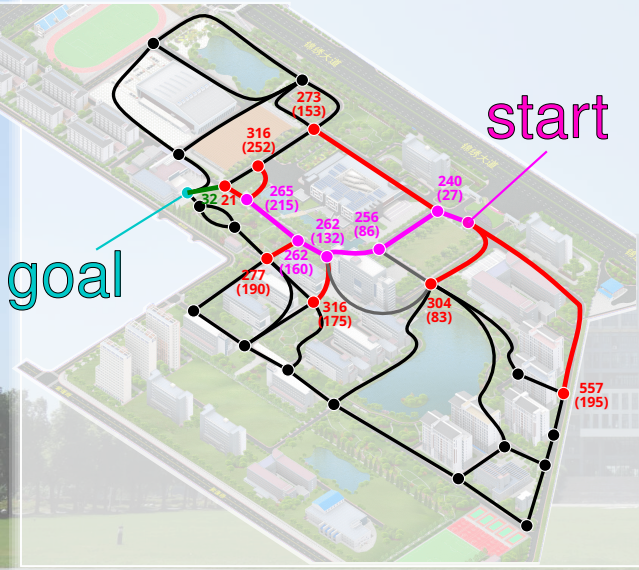
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



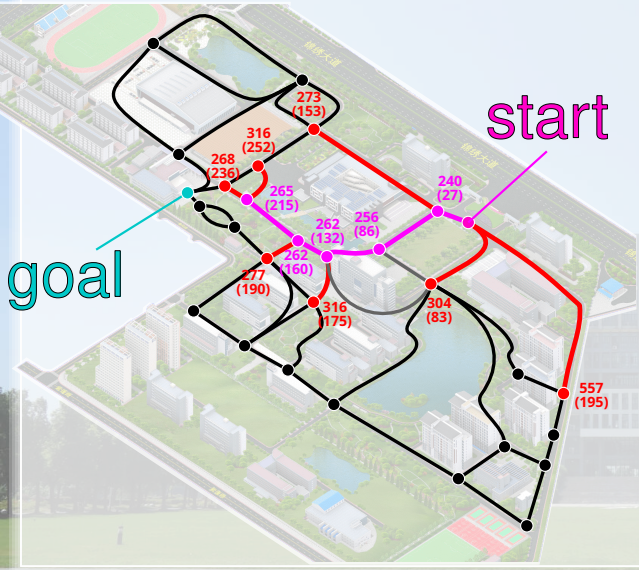
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



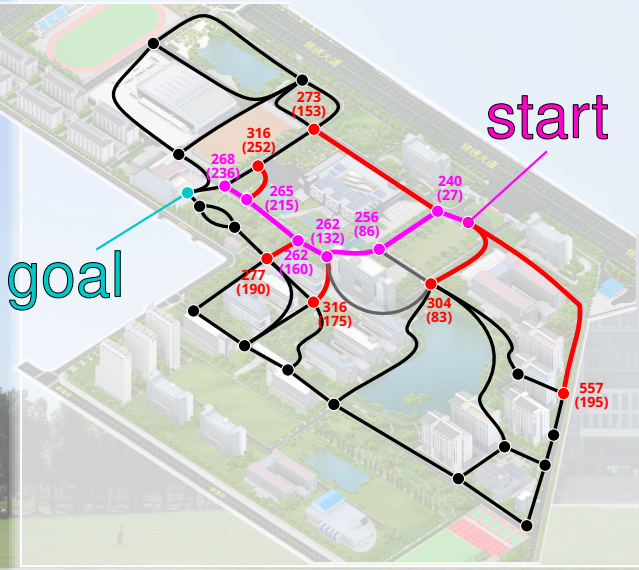
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



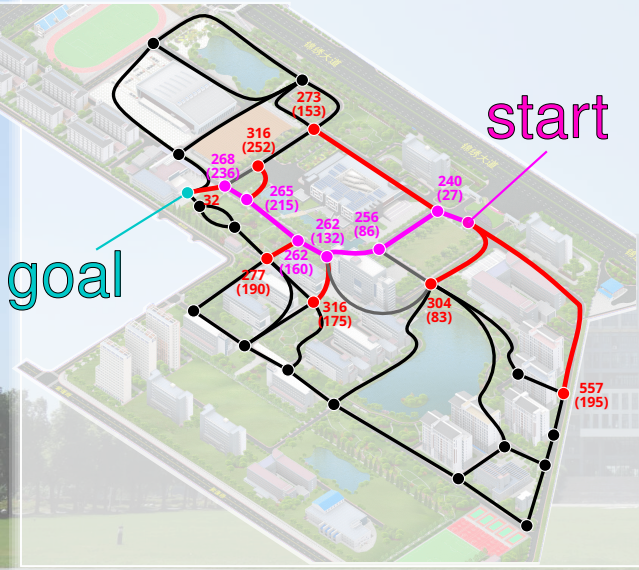
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



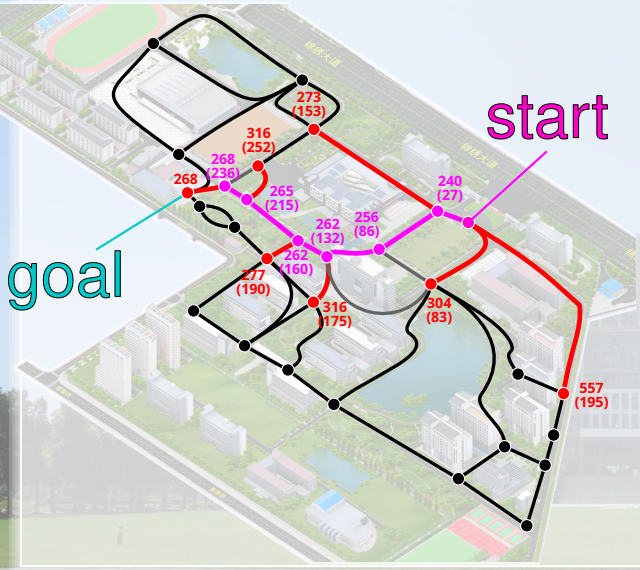
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.

Problems that algorithms can solve efficiently



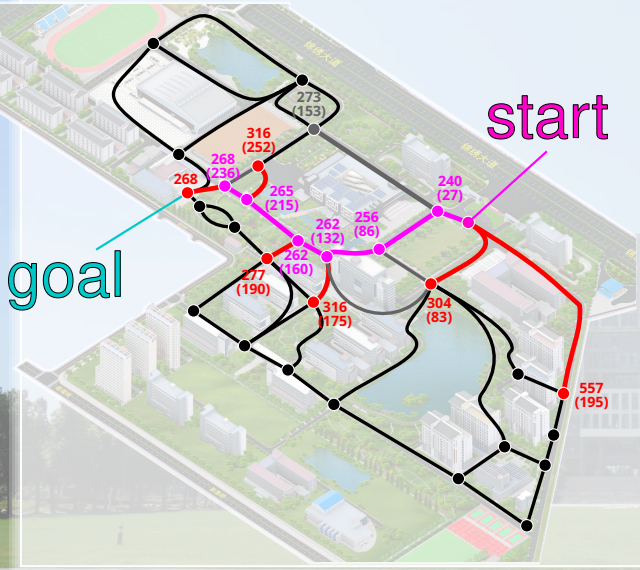
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.

Problems that algorithms can solve efficiently



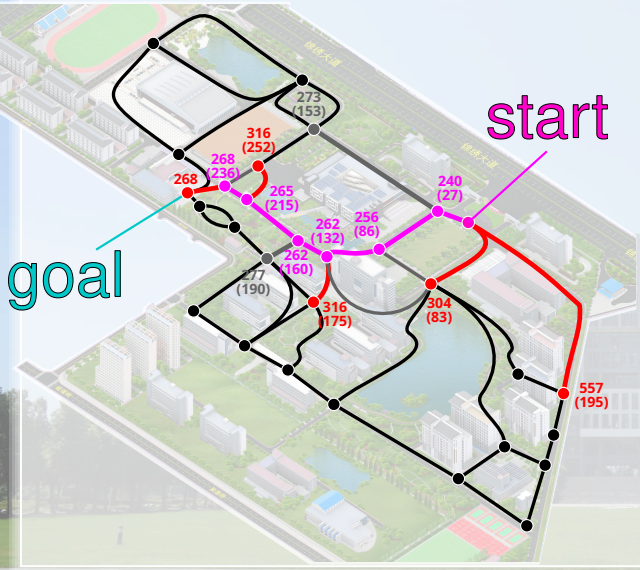
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.

Problems that algorithms can solve efficiently



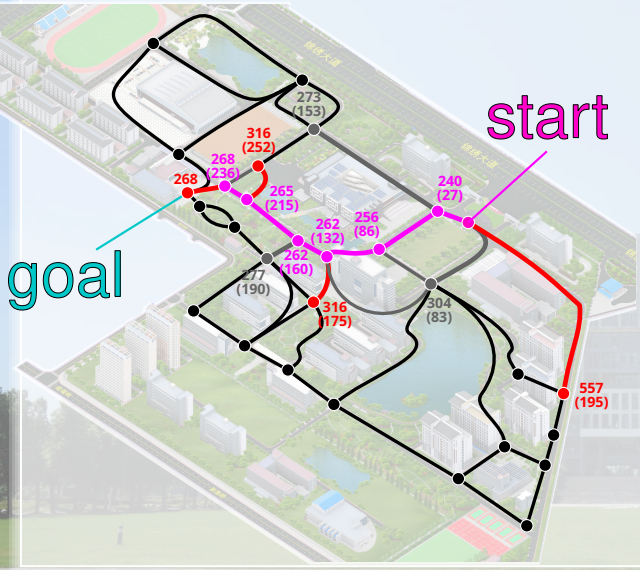
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.
- Now we need to only check the remaining paths that might be shorter, those that are at least as long can be ignored.

Problems that algorithms can solve efficiently



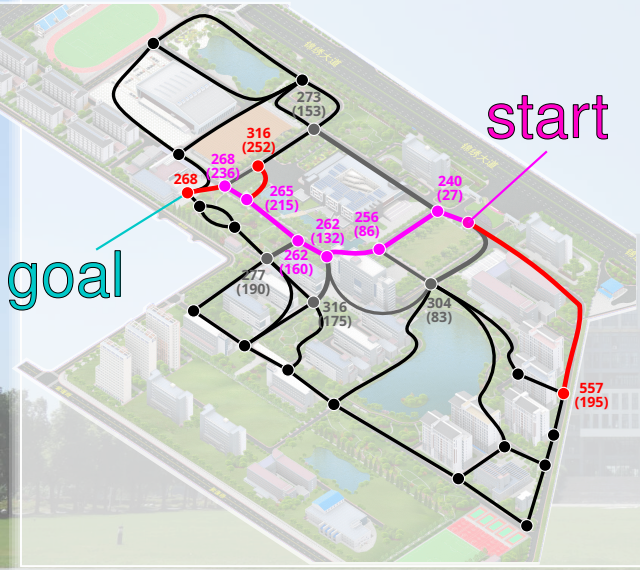
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.
- Now we need to only check the remaining paths that might be shorter, those that are at least as long can be ignored.

Problems that algorithms can solve efficiently



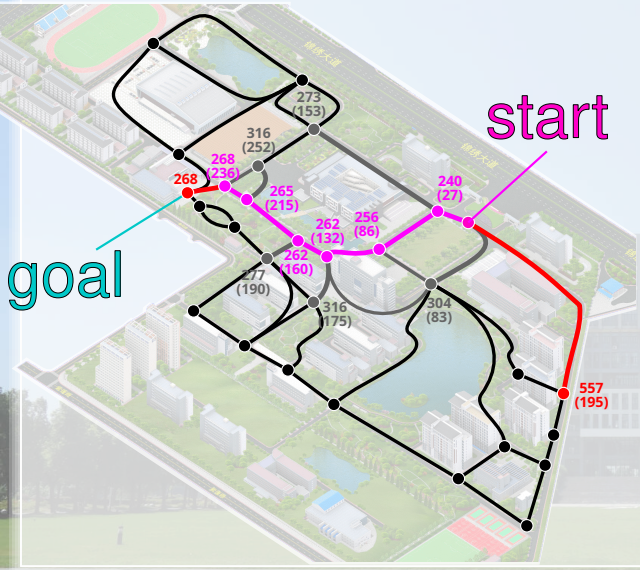
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.
- Now we need to only check the remaining paths that might be shorter, those that are at least as long can be ignored.

Problems that algorithms can solve efficiently



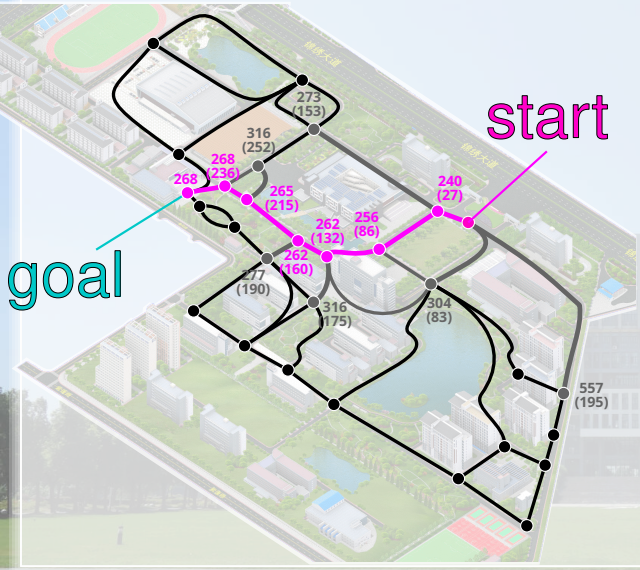
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.
- Now we need to only check the remaining paths that might be shorter, those that are at least as long can be ignored.

Problems that algorithms can solve efficiently



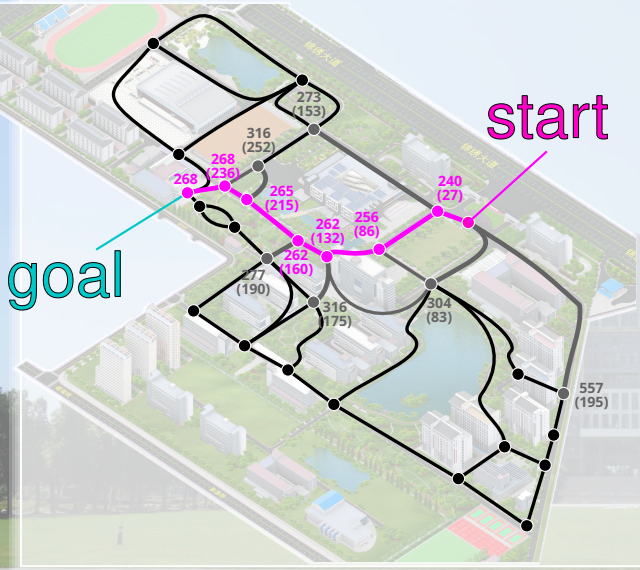
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.
- Now we need to only check the remaining paths that might be shorter, those that are at least as long can be ignored.

Problems that algorithms can solve efficiently



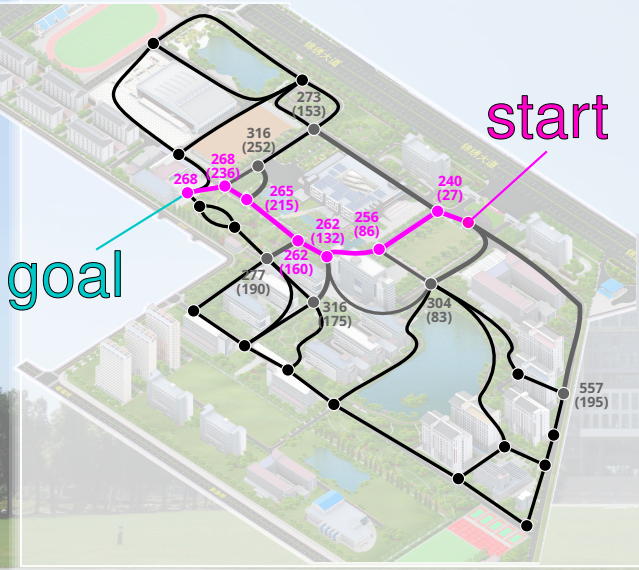
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.
- Now we need to only check the remaining paths that might be shorter, those that are at least as long can be ignored.

Problems that algorithms can solve efficiently



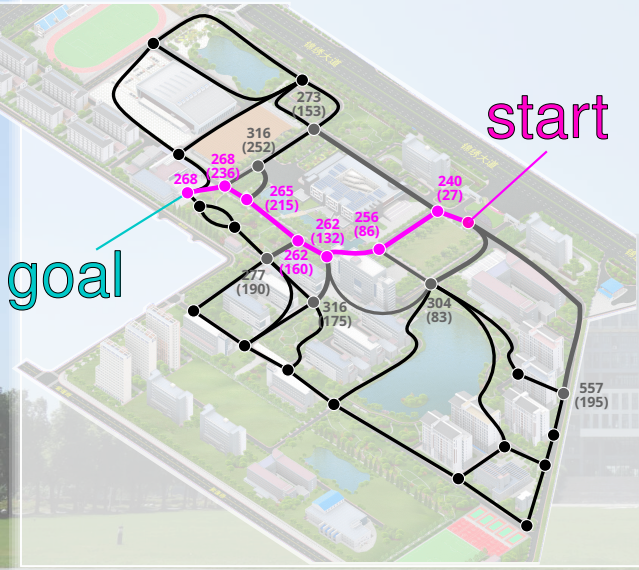
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- We continue doing this procedure until the first path to the goal is found.
- We found a first complete path from start to goal.
- It has the total length $268s$.
- Now we need to only check the remaining paths that might be shorter, those that are at least as long can be ignored.
- We confirmed the shortest possible path and it takes $268s$.

Problems that algorithms can solve efficiently



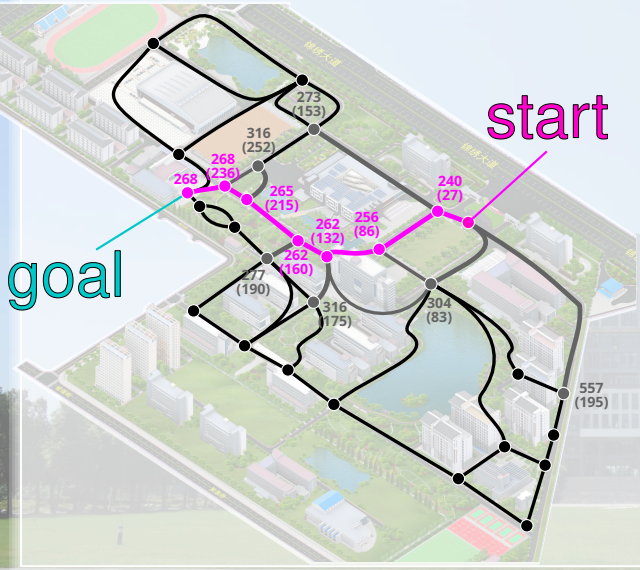
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- In the worst case, the A* Algorithm needs to "look" at all intersections (nodes) and streets (edges) once.

Problems that algorithms can solve efficiently



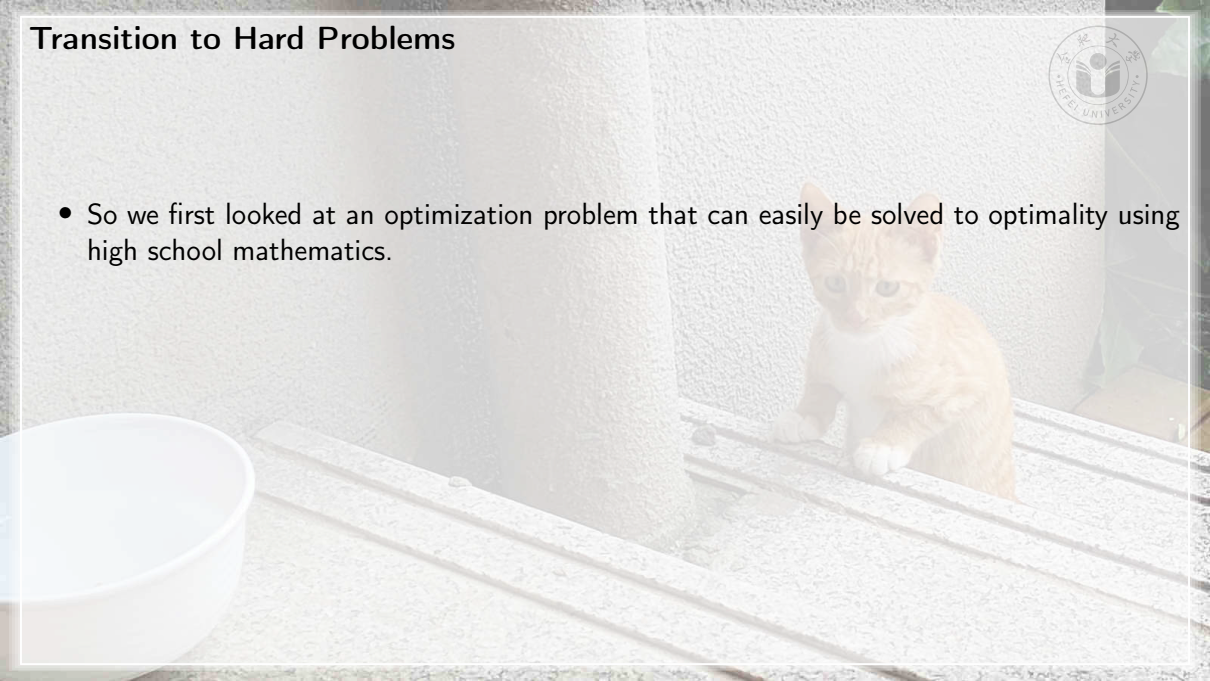
- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- In the worst case, the A* Algorithm needs to "look" at all intersections (nodes) and streets (edges) once.
- Usually, it is quite efficient to find shortest paths on maps.

Problems that algorithms can solve efficiently



- But we can use the A* Algorithm[105,183], which iteratively constructs a solution by adding a heuristic to a cost function.
- In the worst case, the A* Algorithm needs to "look" at all intersections (nodes) and streets (edges) once.
- Usually, it is quite efficient to find shortest paths on maps.
- If we look for shortest paths that do not visit any node twice, where the edge distances are not negative, where the number of choices per node is limited, and where the graph and path are not too big/long – then this algorithm is very efficient.

Transition to Hard Problems



- So we first looked at an optimization problem that can easily be solved to optimality using high school mathematics.

Transition to Hard Problems



- So we first looked at an optimization problem that can easily be solved to optimality using high school mathematics.
- Then we looked at a problem that can be solved to optimality using an algorithm that you should have learned during your Bachelor's studies.

Transition to Hard Problems



- So we first looked at an optimization problem that can easily be solved to optimality using high school mathematics.
- Then we looked at a problem that can be solved to optimality using an algorithm that you should have learned during your Bachelor's studies.
- What about *hard* problems?

Transition to Hard Problems



- So we first looked at an optimization problem that can easily be solved to optimality using high school mathematics.
- Then we looked at a problem that can be solved to optimality using an algorithm that you should have learned during your Bachelor's studies.
- What about *hard* problems?
- What about problems where traditional Maths and algorithms do not really help us much?

Transition to Hard Problems



- So we first looked at an optimization problem that can easily be solved to optimality using high school mathematics.
- Then we looked at a problem that can be solved to optimality using an algorithm that you should have learned during your Bachelor's studies.
- What about *hard* problems?
- What about problems where traditional Maths and algorithms do not really help us much?
- Here we go.

Hard Problems (1)



- There exists a group of problems that are *hard*.

Hard Problems (1)



- There exists a group of problems that are *hard*.
- Actually, most of the problems I mention at the beginning of this talk belong to this group of hard problems.

Hard Problems (1)



- There exists a group of problems that are *hard*.
- Actually, most of the problems I mention at the beginning of this talk belong to this group of hard problems.
- A hard problem cannot be solved both exactly and efficiently.

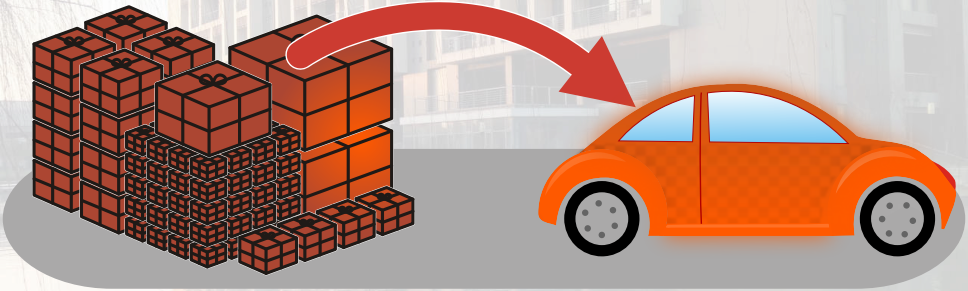
Hard Problems (1)



- There exists a group of problems that are *hard*.
- Actually, most of the problems I mention at the beginning of this talk belong to this group of hard problems.
- A hard problem cannot be solved both exactly and efficiently.
- Let's look at two quick examples.

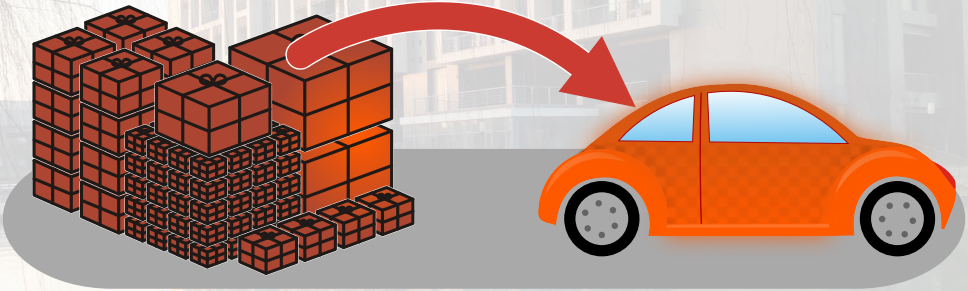
Bin Packing Problem

- My car can carry T kg of weight.



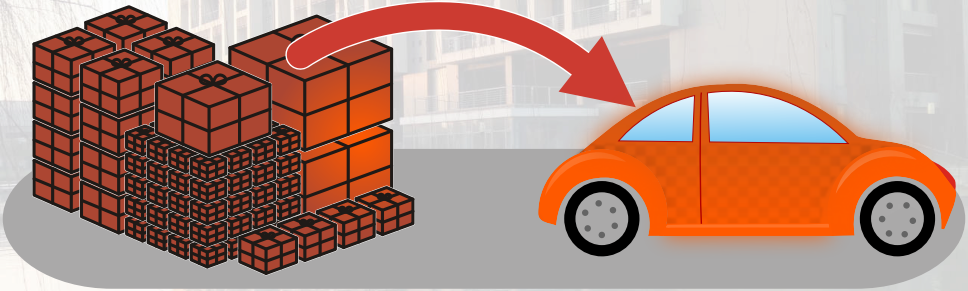
Bin Packing Problem

- My car can carry T kg of weight.
- I have n objects, each with weight w_i with i in 1 to n.



Bin Packing Problem

- My car can carry T kg of weight.
- I have n objects, each with weight w_i with i in 1 to n.
- How can I pack my car so that I can carry them from A to B with the fewest possible hauls?[252,253]



Traveling Salesperson Problem

- Find the shortest path that visits n locations and returns back to its origin.[5,138]



Hard Problems (2)

- These are just two examples from a huge family of problems called $\mathcal{NP}$-hard.

Hard Problems (2)

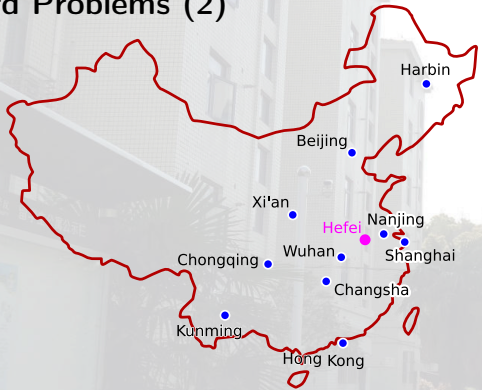
- These are just two examples from a huge family of problems called $\mathcal{NP}$-hard.
- What does that mean?

Hard Problems (2)

- These are just two examples from a huge family of problems called $\mathcal{NP}$-hard.
- What does that mean?

If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the input size n.

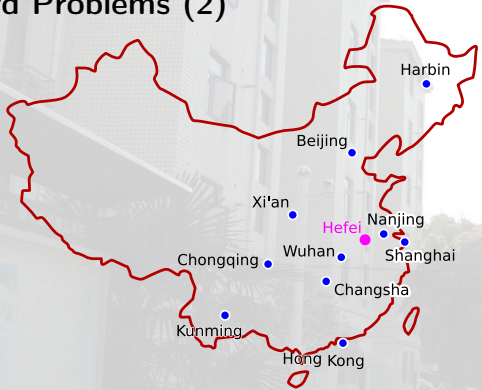
Hard Problems (2)





If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the input size n.

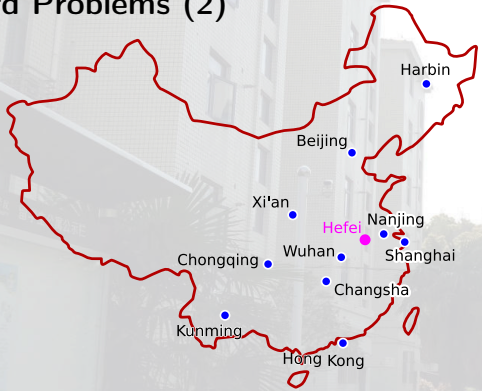
Hard Problems (2)





If an algorithm guarantees to always find the **best-possible** solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the input size n.

Hard Problems (2)



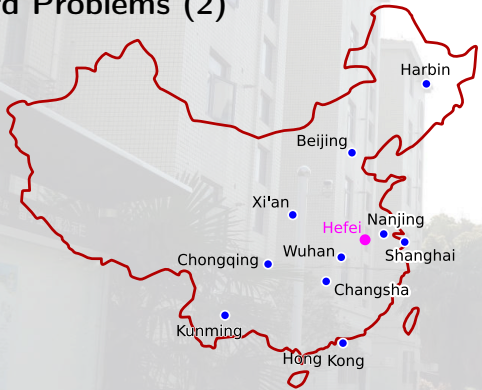
shortest round-trip tour



packing with fewest hauls

If an algorithm guarantees to always find the **best-possible** solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the input size n.

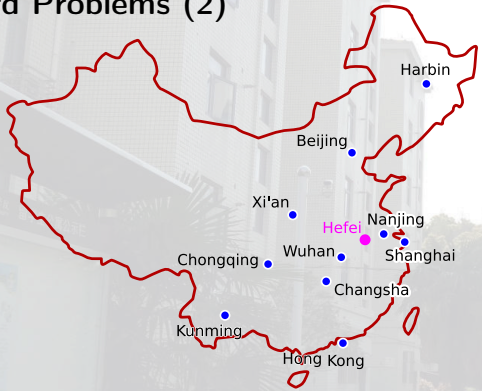
Hard Problems (2)





If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the **input size** n.

Hard Problems (2)



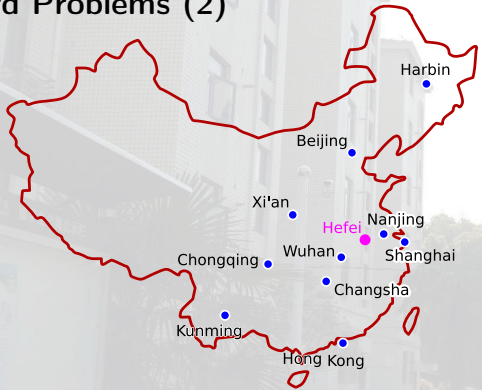
number n of cities to visit

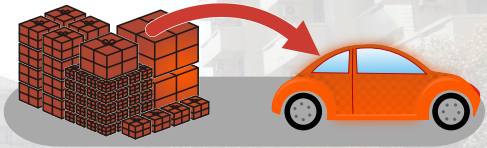


number n of items to pack

If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the **input size** n.

Hard Problems (2)

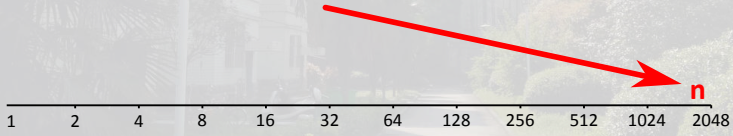




If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

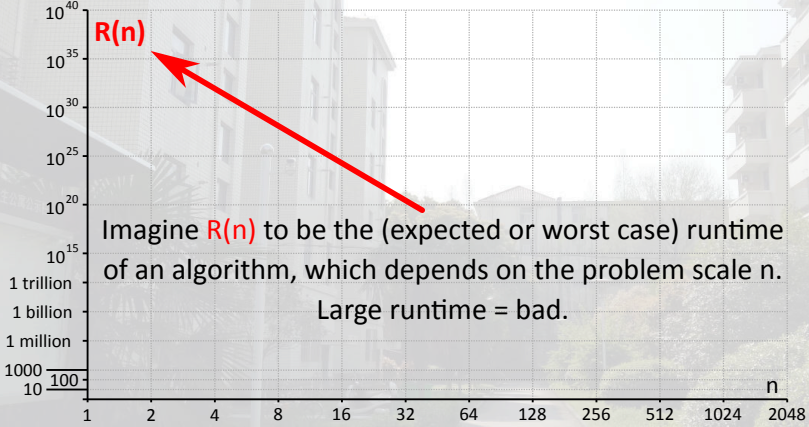
Hard Problems (2)

The problem scale n usually corresponding to the number of decision variables.



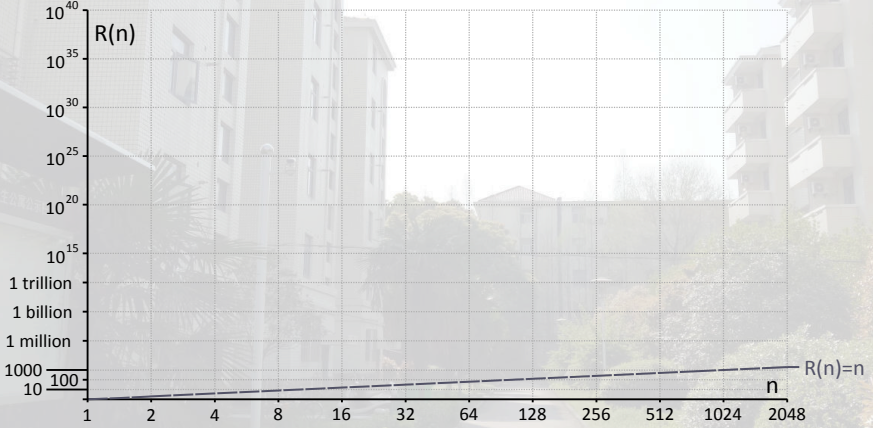
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



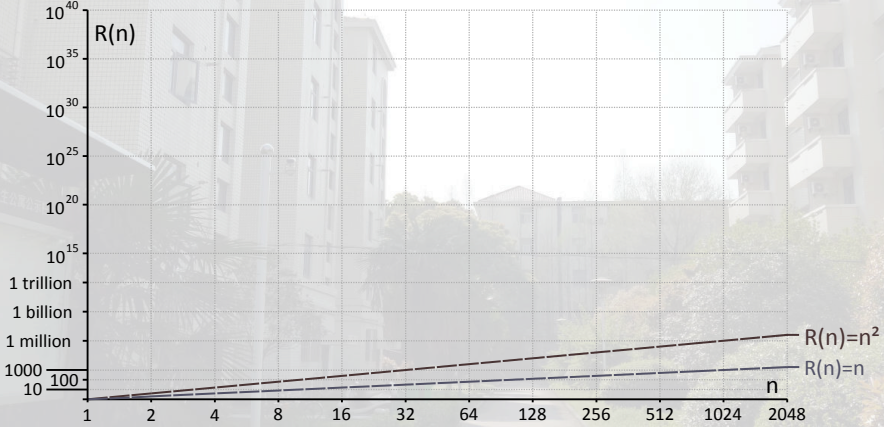
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



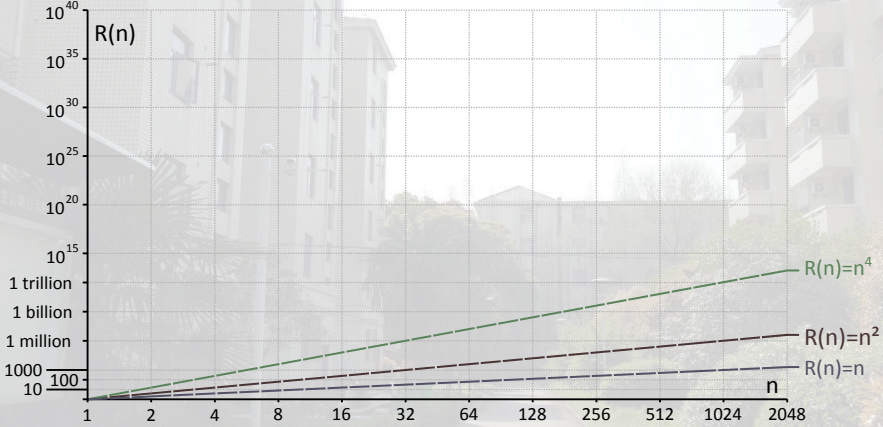
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



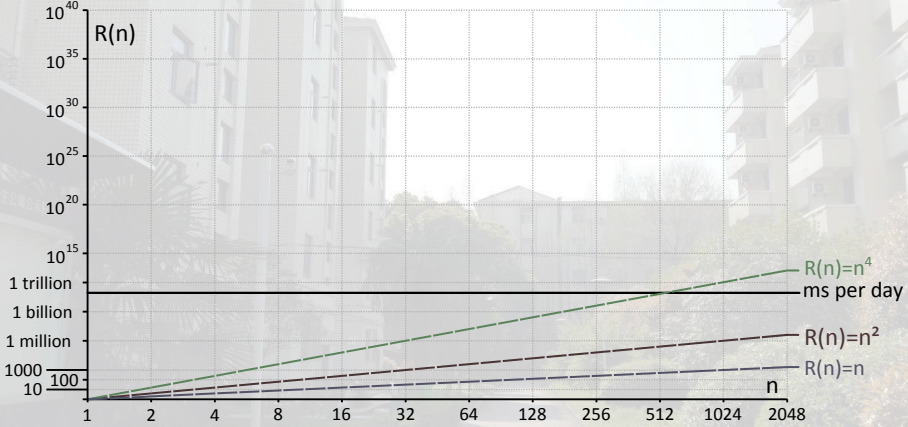
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



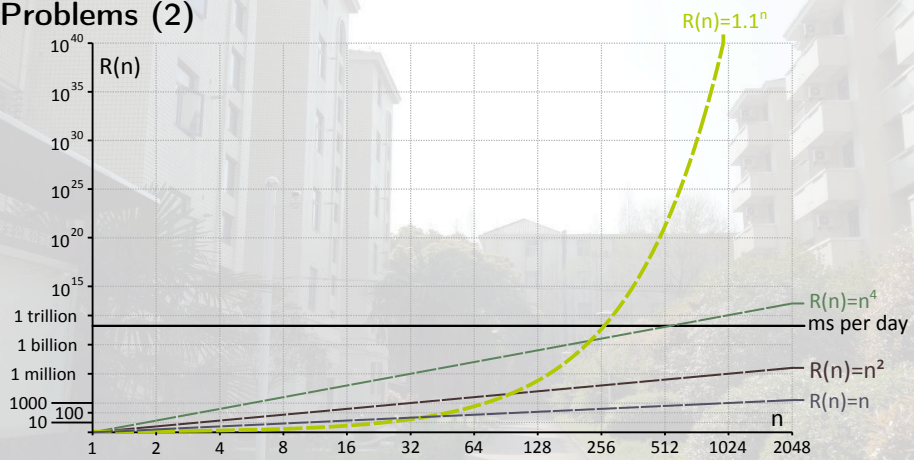
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



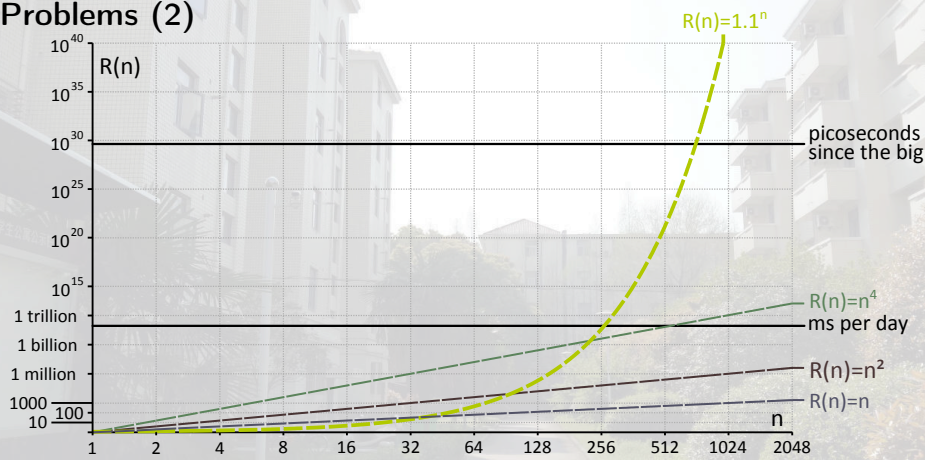
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



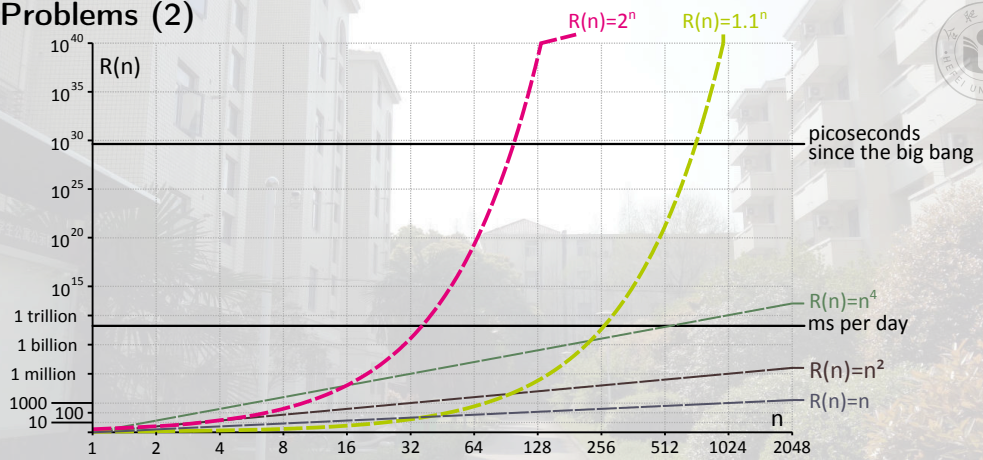
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



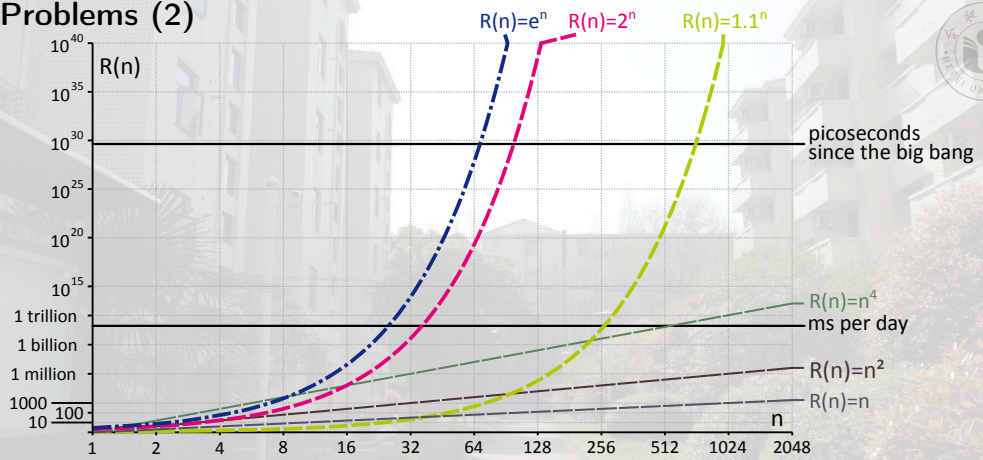
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



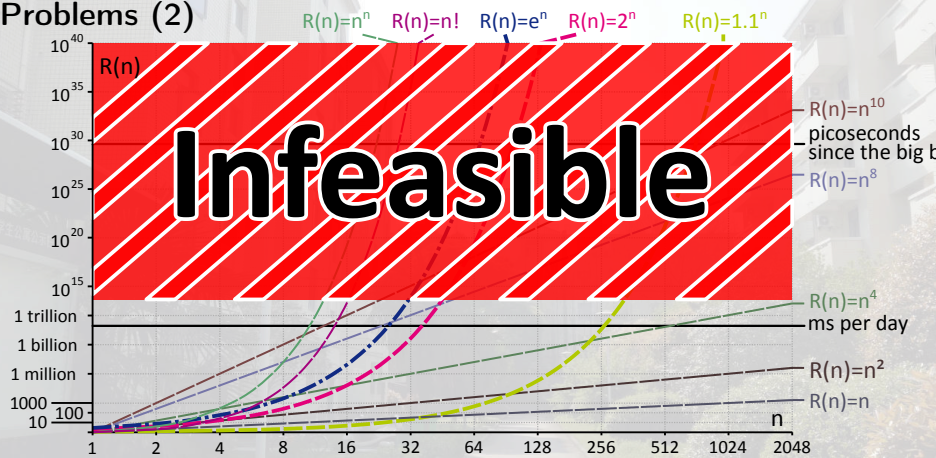
If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)



If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will **grow exponential** with the input size n.

Hard Problems (2)

- These are just two examples from a huge family of problems called $\mathcal{NP}$-hard.
- What does that mean?

If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the input size n.

Hard Problems (2)

- These are just two examples from a huge family of problems called $\mathcal{NP}$-hard.
- What does that mean?

If we want to get the optimal solution,
it will take too long.

If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its worst-case runtime will grow exponential with the input size n.

Hard Problems (2)

- These are just two examples from a huge family of problems called $\mathcal{NP}$-hard.
- What does that mean?

If we want to get the optimal solution,
it will sometimes/often/always take too long.

If an algorithm guarantees to always find the best-possible solution for an $\mathcal{NP}$-hard problem, then its **worst-case** runtime will grow exponential with the input size n.

Metaheuristics



Metaheuristics

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

Metaheuristics

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

Metaheuristics

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

Metaheuristics

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

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

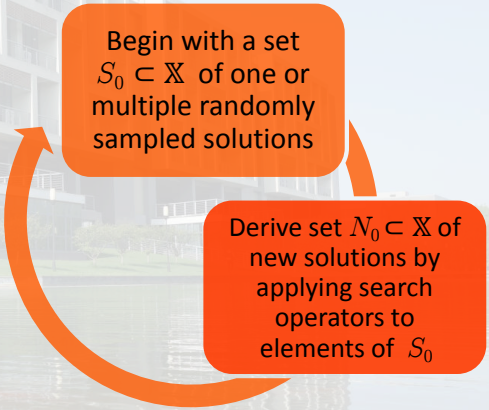
Metaheuristics

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



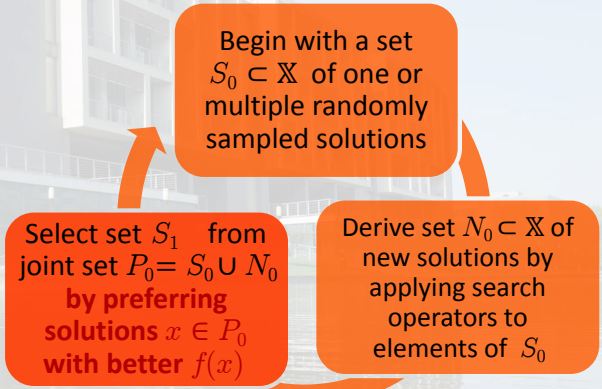
Metaheuristics

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



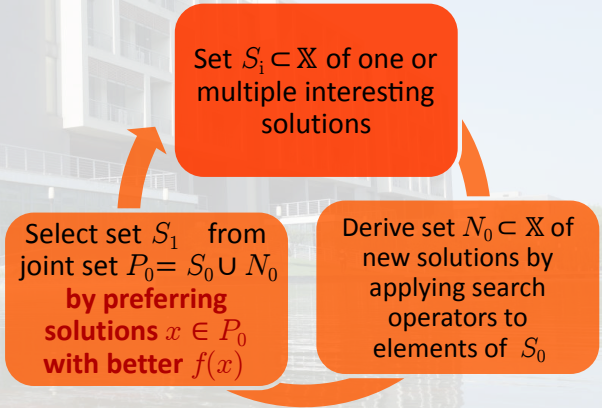
Metaheuristics

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



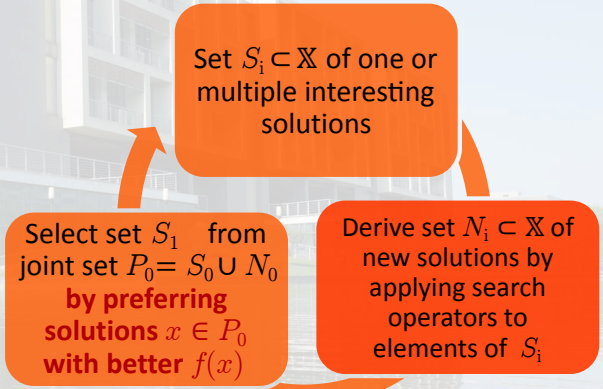
Metaheuristics

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



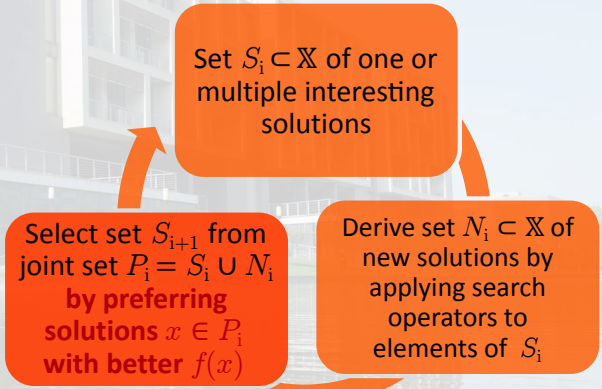
Metaheuristics

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



Metaheuristics

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



Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$. This is often used to get starting points for the search.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$. This is often used to get starting points for the search.
  - a *unary* operator receives an existing point as input and creates a slightly modified copy.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$. This is often used to get starting points for the search.
  - a *unary* operator receives an existing point as input and creates a slightly modified copy. This is often used to explore the neighborhood of good solutions.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$. This is often used to get starting points for the search.
  - a *unary* operator receives an existing point as input and creates a slightly modified copy. This is often used to explore the neighborhood of good solutions.
  - a *binary* operator receives a two existing points as input and creates a new point with elements from both of these "parents".

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$. This is often used to get starting points for the search.
  - a *unary* operator receives an existing point as input and creates a slightly modified copy. This is often used to explore the neighborhood of good solutions.
  - a *binary* operator receives a two existing points as input and creates a new point with elements from both of these "parents". This is often used to combine traits from good solutions.

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$. This is often used to get starting points for the search.
  - a *unary* operator receives an existing point as input and creates a slightly modified copy. This is often used to explore the neighborhood of good solutions.
  - a *binary* operator receives a two existing points as input and creates a new point with elements from both of these "parents". This is often used to combine traits from good solutions.
- We evaluate the quality of solutions using the objective function $f : \mathbb{X} \mapsto \mathbb{R}$, which usually is subject to minimization (smaller values are better).

Components needed for Metaheuristic Optimization

- We have a space $\mathbb{X}$ in which we search for potential solutions x.
- $\mathbb{X}$ is often very/exponentially large in the input size.
- We repeatedly sample elements x from this space $\mathbb{X}$, so we need operators to do that:
  - a *nullary* operator usually returns a randomly chosen point from $\mathbb{X}$. This is often used to get starting points for the search.
  - a *unary* operator receives an existing point as input and creates a slightly modified copy. This is often used to explore the neighborhood of good solutions.
  - a *binary* operator receives a two existing points as input and creates a new point with elements from both of these "parents". This is often used to combine traits from good solutions.
- We evaluate the quality of solutions using the objective function $f : \mathbb{X} \mapsto \mathbb{R}$, which usually is subject to minimization (smaller values are better).
- These are the usual main ingredients of metaheuristic optimization.

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS)

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
  - Advantage: Fast (and sometimes simple)

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
  - Advantage: Fast (and sometimes simple)
  - Disadvantage: May get stuck at a local optimum

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
  - Advantage: Less likely to get stuck at local optimum.

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
  - Advantage: Less likely to get stuck at local optimum.
  - Disadvantage: Slow.

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods:
  - Maintain a stochastic model M (usually a multivariate probility distribution) of what good solutions could look like, sample set N of $|N| = \lambda \geq 1$ new solutions based on M in each step, then update M.

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods:
  - Maintain a stochastic model M (usually a multivariate probility distribution) of what good solutions could look like, sample set N of $|N| = \lambda \geq 1$ new solutions based on M in each step, then update M.
  - Advantage: Sometimes really good performance.

Types of Metaheuristics



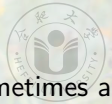
- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods:
  - Maintain a stochastic model M (usually a multivariate probility distribution) of what good solutions could look like, sample set N of $|N| = \lambda \geq 1$ new solutions based on M in each step, then update M.
  - Advantage: Sometimes really good performance.
  - Disadvantage: Complicated, sampling and model update may be slow.

Types of Metaheuristics



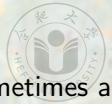
- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods:
  - Maintain a stochastic model M (usually a multivariate probility distribution) of what good solutions could look like, sample set N of $|N| = \lambda \geq 1$ new solutions based on M in each step, then update M.
- Hybrid Algorithms

Types of Metaheuristics



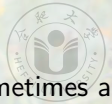
- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods:
  - Maintain a stochastic model M (usually a multivariate probility distribution) of what good solutions could look like, sample set N of $|N| = \lambda \geq 1$ new solutions based on M in each step, then update M.
- Hybrid Algorithms:
  - Usually combine model-based or population-based approach with local search (LS): Each new solution from model/population is refined with local search.

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods:
  - Maintain a stochastic model M (usually a multivariate probility distribution) of what good solutions could look like, sample set N of $|N| = \lambda \geq 1$ new solutions based on M in each step, then update M.
- Hybrid Algorithms:
  - Usually combine model-based or population-based approach with local search (LS): Each new solution from model/population is refined with local search.
  - Advantage: Fast (because LS), good global search (because of population/model)

Types of Metaheuristics



- We can build many interesting algorithms just using such components (and sometimes a bit more problem-specific information).
- stochastic local search (SLS):
  - Keep one solution x_c, sample at least one new solution x_n from its neighborhood in each step, move to better solutions as discovered.
- Population-based Methods:
  - Maintain population S of $|S| = \mu \geq 1$ solutions, sample set N of $|N| = \lambda \geq 1$ new solutions in each step by creating modified copies of the elements in S, then update S.
- Model-based Methods:
  - Maintain a stochastic model M (usually a multivariate probility distribution) of what good solutions could look like, sample set N of $|N| = \lambda \geq 1$ new solutions based on M in each step, then update M.
- Hybrid Algorithms:
  - Usually combine model-based or population-based approach with local search (LS): Each new solution from model/population is refined with local search.
  - Advantage: Fast (because LS), good global search (because of population/model)
  - Disadvantage: usually few.

Example: RLS and the $(1+1)$ EA



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

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].

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

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$
    randomly sample x_c from $\mathbb{X}$;

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$
    randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$
    randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
    while $\neg$ terminate **do**

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].

procedure $(1+1)$ EA$(f : \mathbb{X} \mapsto \mathbb{R})$
    randomly sample x_c from $\mathbb{X}$; $y_c \leftarrow f(x_c)$;
    while $\neg$ terminate **do**
        $x_n \leftarrow \texttt{move}(x_c)$;

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].

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

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].
- They accept the new solution if it is better or equally good compared to the current solution.

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

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].
- They accept the new solution if it is better or equally good compared to the current solution.

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

Example: RLS and the $(1+1)$ EA

- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].
- They accept the new solution if it is better or equally good compared to the current solution.

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

Example: RLS and the $(1+1)$ EA



- SLS with $|S_i| = |N_i| = 1$ is the simplest realization of the metaheuristic idea.
- Randomized local search (RLS) and the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) work according to this pattern[40,72].
- They accept the new solution if it is better or equally good compared to the current solution.

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- Let's start with RLS/$(1+1)$ EA and modify it to become SA.

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

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

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.

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

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

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

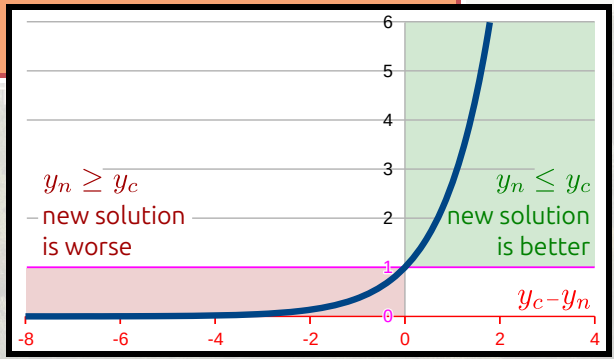
Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.

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

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

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



Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.

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

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

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.

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

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

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P is regulated by temperature schedule with parameter T_0.

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

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

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P is regulated by temperature schedule with parameter T_0.

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

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

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P is regulated by temperature schedule with parameter T_0.

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

  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while ¬ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0$;
    if $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P is regulated by temperature schedule with parameters T_0 and ε.

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

  $\tau \leftarrow 0$; ▷ τ is iteration counter
  while ¬ terminate **do**
    $x_n \leftarrow \texttt{move}(x_c)$; $y_n \leftarrow f(x_n)$;
    $\tau \leftarrow \tau + 1$;
    $T \leftarrow T_0(1-\varepsilon)^{\tau-1}$; ▷ T decreases over time
    if $\mathfrak{R}_0^1 < e^{\frac{y_c - y_n}{T}}$ **then** ▷ always true if $y_n \leq y_c$
      $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P is regulated by temperature schedule with parameters T_0 and ε.

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

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P is regulated by temperature schedule with parameters T_0 and ε.

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

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

Example: Simulated Annealing

- Simulated Annealing (SA)[126] is a local search that accepts also worsening moves, but with a probability that decreases over time and with the difference in solution quality.
- $\mathfrak{R}_0^1$ be a random number uniformly distributed in $[0, 1)$.
- High temperatures T increase the acceptance probability P of worse solutions, low T reduce it.
- P is regulated by temperature schedule with parameters T_0 and ε.

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

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

Problem Domains

- We now have at least a cursory understanding of metaheuristics.

Problem Domains

- We now have at least a cursory understanding of metaheuristics.
- So let's directly skip directly to the *research frontier* of this field.

Problem Domains

- We now have at least a cursory understanding of metaheuristics.
- So let's directly skip directly to the *research frontier* of this field.
- Let us therefore look a handful of discrete and combinatorial problems.

Problem Domains

- We now have at least a cursory understanding of metaheuristics.
- So let's directly skip directly to the *research frontier* of this field.
- Let us therefore look a handful of discrete and combinatorial problems.
- (We ignore the continuous domain $\mathbb{X} = \mathbb{R}^n$.)

Discrete Optimization





Discrete Optimization

Black-Box Discrete Optimization Benchmarks



Black-Box Discrete Optimization Benchmarks: Introduction

- Problem: Search space $\mathbb{X}$ corresponds to the set $\{0,1\}^n$ of bit strings of length n and *nothing* is known about objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$.

Black-Box Discrete Optimization Benchmarks: Introduction

- Problem: Search space $\mathbb{X}$ corresponds to the set $\{0,1\}^n$ of bit strings of length n and *nothing* is known about objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$.
- Benchmarks here often have the goal to investigate problematic aspects of discrete problems in isolation[66,250].

Black-Box Discrete Optimization Benchmarks: Introduction

- Problem: Search space $\mathbb{X}$ corresponds to the set $\{0,1\}^n$ of bit strings of length n and *nothing* is known about objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$.
- Benchmarks here often have the goal to investigate problematic aspects of discrete problems in isolation[66,250].
- The benchmarks are functions that scale with n and are usually computable in $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$.

Black-Box Discrete Optimization Benchmarks: Introduction

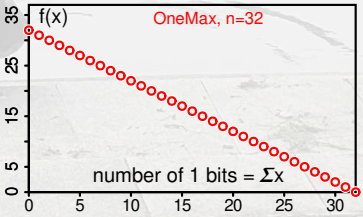
- Problem: Search space $\mathbb{X}$ corresponds to the set $\{0,1\}^n$ of bit strings of length n and *nothing* is known about objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$.
- Benchmarks here often have the goal to investigate problematic aspects of discrete problems in isolation[66,250].
- The benchmarks are functions that scale with n and are usually computable in $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$.
- Their optima are known (but the algorithms we apply to them don't know them).

Black-Box Discrete Optimization Benchmarks: Introduction

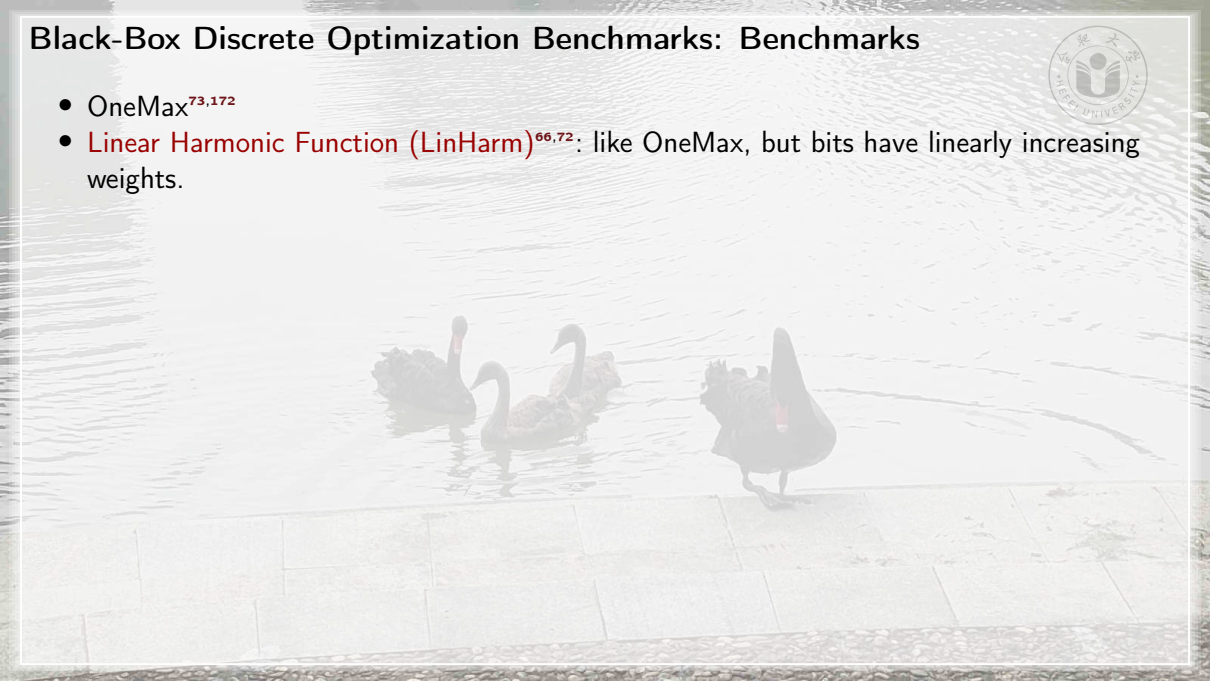
- Problem: Search space $\mathbb{X}$ corresponds to the set $\{0,1\}^n$ of bit strings of length n and *nothing* is known about objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$.
- Benchmarks here often have the goal to investigate problematic aspects of discrete problems in isolation[66,250].
- The benchmarks are functions that scale with n and are usually computable in $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$.
- Their optima are known (but the algorithms we apply to them don't know them).
- So what are such benchmarks?

Black-Box Discrete Optimization Benchmarks: Benchmarks

- OneMax[73,172]: maximize number of 1 bits, very easy, single optimum.

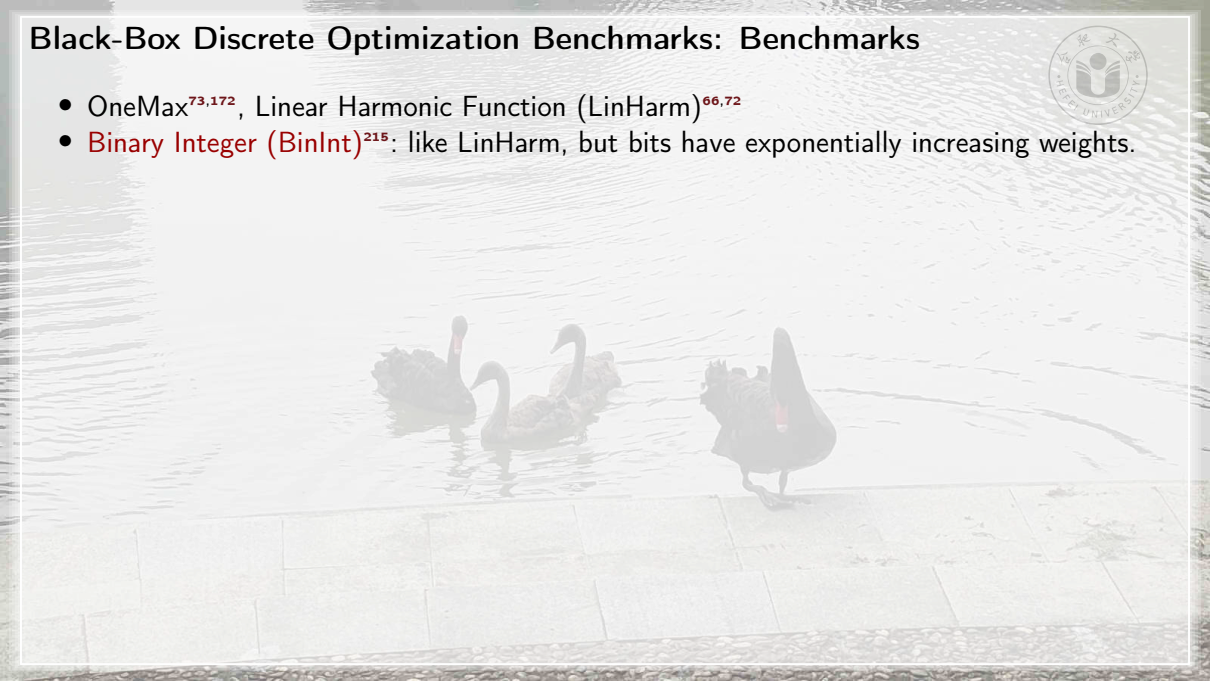


Black-Box Discrete Optimization Benchmarks: Benchmarks



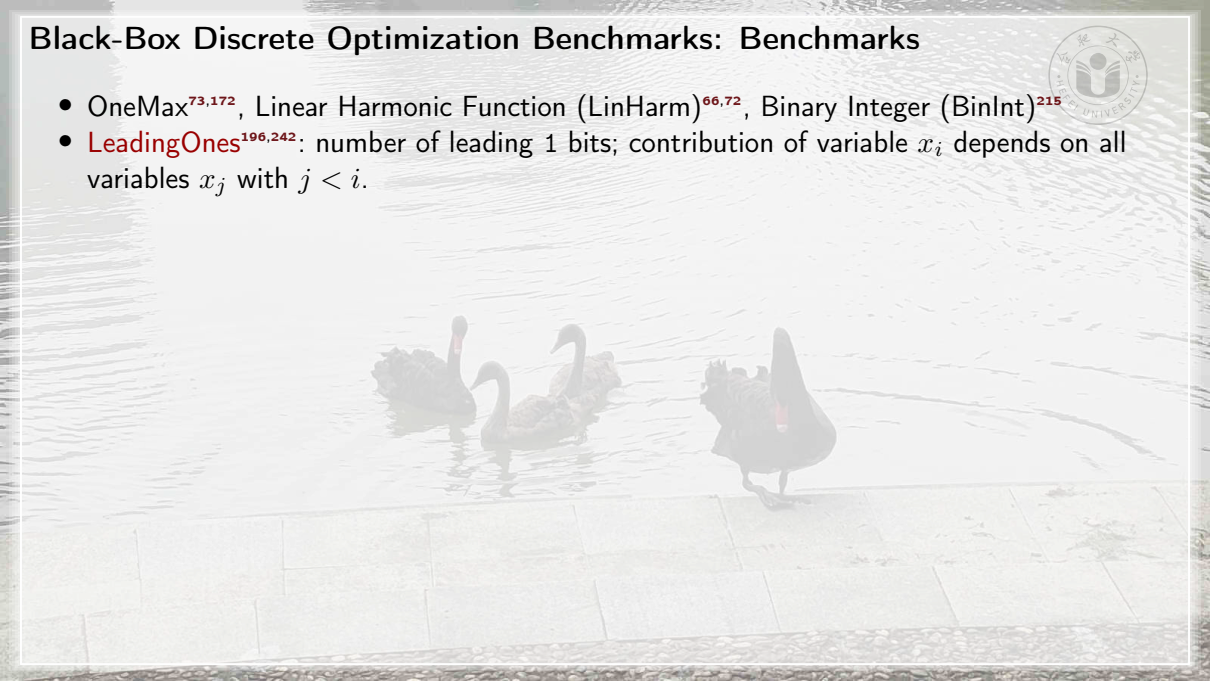
- OneMax[73,172]
- Linear Harmonic Function (LinHarm)[66,72]: like OneMax, but bits have linearly increasing weights.

Black-Box Discrete Optimization Benchmarks: Benchmarks



- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72]
- Binary Integer (BinInt)[215]: like LinHarm, but bits have exponentially increasing weights.

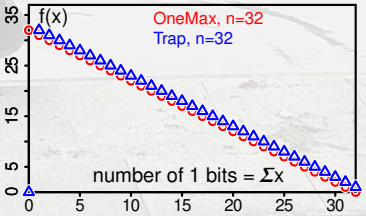
Black-Box Discrete Optimization Benchmarks: Benchmarks



- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215]
- LeadingOnes[196,242]: number of leading 1 bits; contribution of variable x_i depends on all variables x_j with $j < i$.

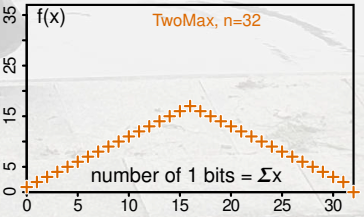
Black-Box Discrete Optimization Benchmarks: Benchmarks

- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242]
- Trap[72,176]: OneMax, but optimum and worse solution swapped – extremely hard, local search needs exponential runtime.



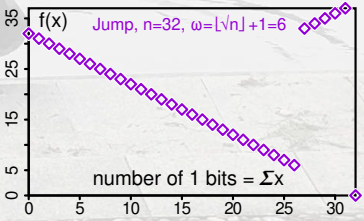
Black-Box Discrete Optimization Benchmarks: Benchmarks

- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242], Trap[72,176]
- TwoMax[87,221]: two optima, a global and a local one, of same size, local search needs exponential runtime with 50% probability.



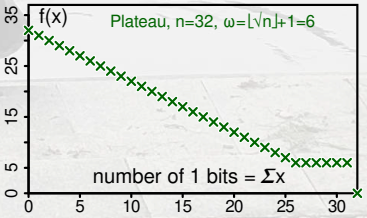
Black-Box Discrete Optimization Benchmarks: Benchmarks

- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242], Trap[72,176], TwoMax[87,221]
- Jump[72,87]: like OneMax, but last ω bit flips around optimum are deceptive = lead away from optimum.

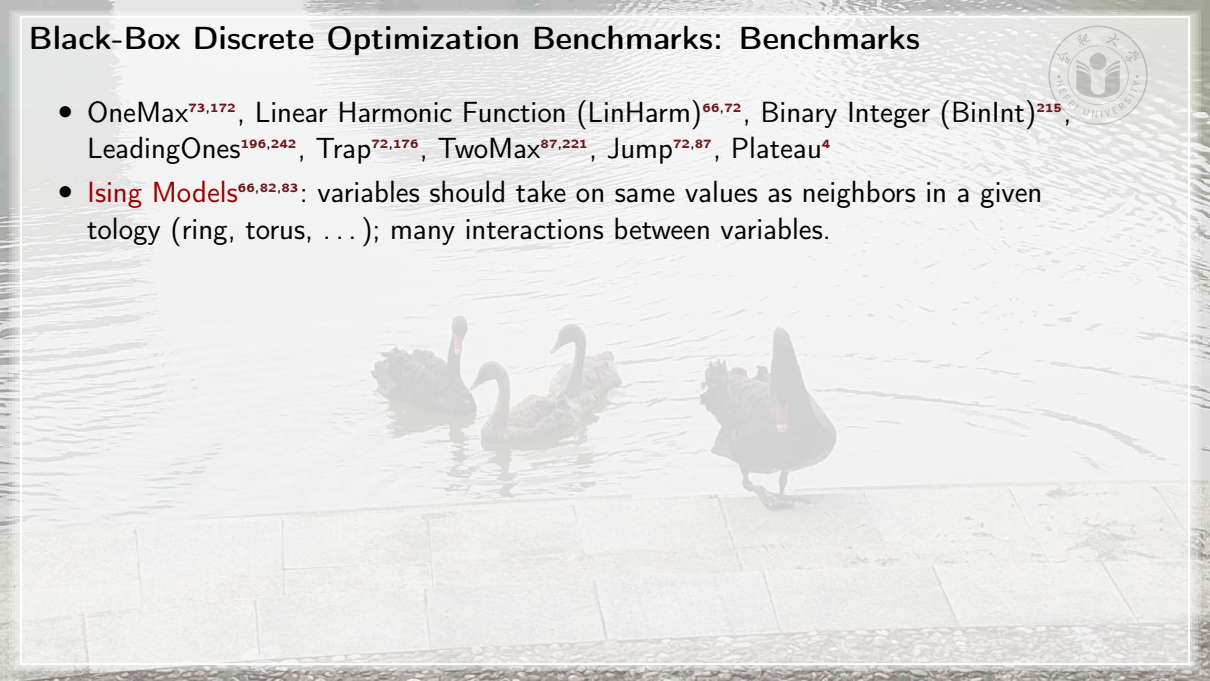


Black-Box Discrete Optimization Benchmarks: Benchmarks

- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242], Trap[72,176], TwoMax[87,221], Jump[72,87]
- Plateau[4]: like Jump, but last ω bit fips around optimum are neutral = have same objective value.



Black-Box Discrete Optimization Benchmarks: Benchmarks

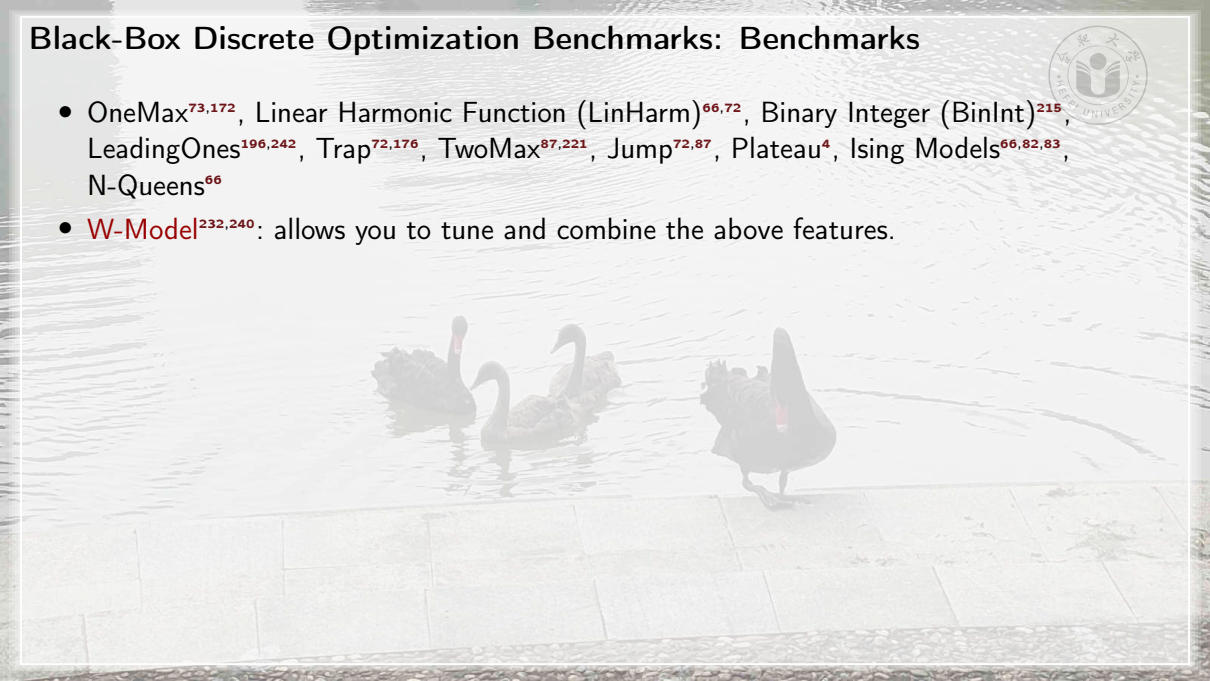


- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242], Trap[72,176], TwoMax[87,221], Jump[72,87], Plateau[4]
- Ising Models[66,82,83]: variables should take on same values as neighbors in a given tology (ring, torus, ...); many interactions between variables.

Black-Box Discrete Optimization Benchmarks: Benchmarks

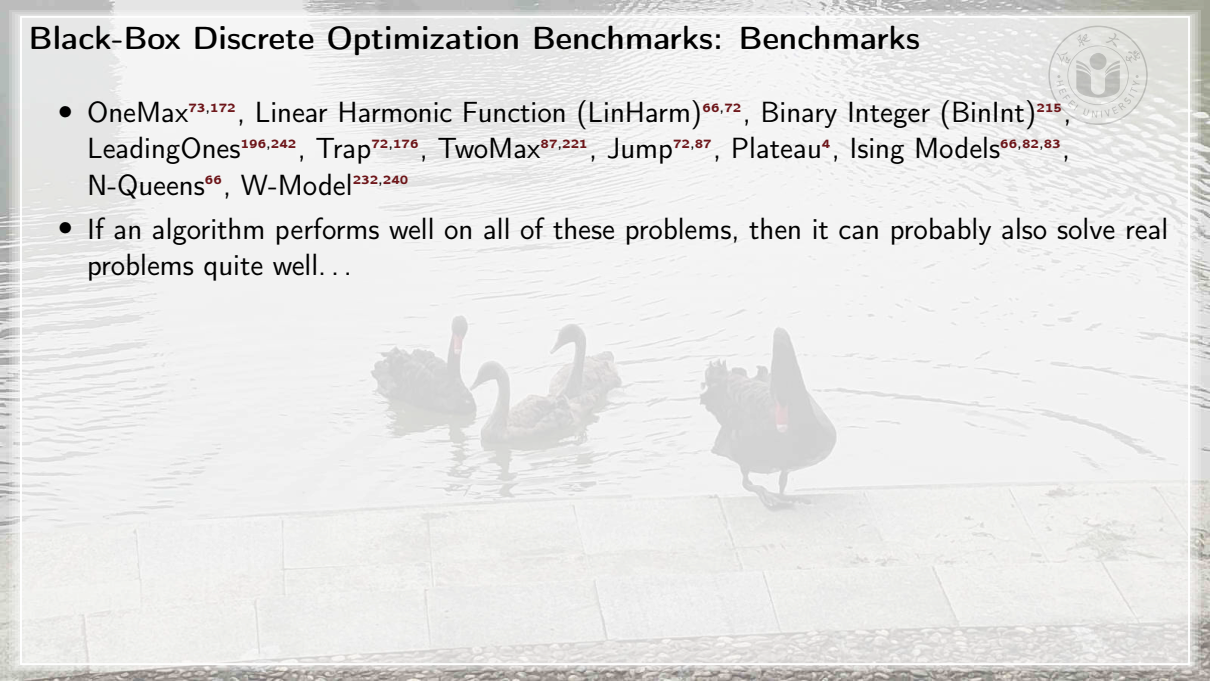
- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242], Trap[72,176], TwoMax[87,221], Jump[72,87], Plateau[4], Ising Models[66,82,83]
- N-Queens[66]: place N queens on a chessboard so that they cannot beat each other, many interactions between variables.

Black-Box Discrete Optimization Benchmarks: Benchmarks



- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242], Trap[72,176], TwoMax[87,221], Jump[72,87], Plateau[4], Ising Models[66,82,83], N-Queens[66]
- W-Model[232,240]: allows you to tune and combine the above features.

Black-Box Discrete Optimization Benchmarks: Benchmarks



- OneMax[73,172], Linear Harmonic Function (LinHarm)[66,72], Binary Integer (BinInt)[215], LeadingOnes[196,242], Trap[72,176], TwoMax[87,221], Jump[72,87], Plateau[4], Ising Models[66,82,83], N-Queens[66], W-Model[232,240]
- If an algorithm performs well on all of these problems, then it can probably also solve real problems quite well...

Black-Box Discrete Optimization Benchmarks: Baselines



- So what are the baseline algorithms for black-box discrete optimization, i.e., the algorithms that you absolutely must beat and against which any new idea should be compared?

Black-Box Discrete Optimization Benchmarks: Baselines



- So what are the baseline algorithms for black-box discrete optimization, i.e., the algorithms that you absolutely must beat and against which any new idea should be compared?
  - randomized local search (RLS): unary operator flips 1 bit

Black-Box Discrete Optimization Benchmarks: Baselines



- So what are the baseline algorithms for black-box discrete optimization, i.e., the algorithms that you absolutely must beat and against which any new idea should be compared?
  - randomized local search (RLS): unary operator flips 1 bit
  - $(1 + 1)$ EA: unary operator flips each bit with probability $1/n$

Black-Box Discrete Optimization Benchmarks: Baselines



- So what are the baseline algorithms for black-box discrete optimization, i.e., the algorithms that you absolutely must beat and against which any new idea should be compared?
  - randomized local search (RLS): unary operator flips 1 bit
  - $(1+1)$ EA: unary operator flips each bit with probability $1/n$
  - Standard Genetic Algorithm (SGA)[9,95,226]: traditional population-based GA, now outdated

Black-Box Discrete Optimization Benchmarks: Baselines



- So what are the baseline algorithms for black-box discrete optimization, i.e., the algorithms that you absolutely must beat and against which any new idea should be compared?
  - randomized local search (RLS): unary operator flips 1 bit
  - $(1+1)$ EA: unary operator flips each bit with probability $1/n$
  - Standard Genetic Algorithm (SGA)[9,95,226]: traditional population-based GA, now outdated
  - Tabu Search (TS): like RLS, but forbit flipping of recently flipped bits to avoid cycles

Black-Box Discrete Optimization Benchmarks: SOTA



- The SOTA algorithms are the most interesting methods currently put forward by research.

Black-Box Discrete Optimization Benchmarks: SOTA



- The SOTA algorithms are the most interesting methods currently put forward by research.
  - The Self-Adjusting $(1 + (\lambda, \lambda))$ GA by Carvalho Pinto and Doerr [40, 61]: linear runtime on OneMax.

Black-Box Discrete Optimization Benchmarks: SOTA



- The SOTA algorithms are the most interesting methods currently put forward by research.
  - The Self-Adjusting $(1 + (\lambda, \lambda))$ GA by Carvalho Pinto and Doerr [40, 61]: linear runtime on OneMax.
  - The Greedy $(2 + 1)$ GA by Sudholt [39, 40, 208, 209]: An algorithm using a binary operator and proven to be twice as fast as any unary-operator only black-box method on OneMax.

Black-Box Discrete Optimization Benchmarks: SOTA

- The SOTA algorithms are the most interesting methods currently put forward by research.
  - The Self-Adjusting $(1+(\lambda,\lambda))$ GA by Carvalho Pinto and Doerr [40, 61]: linear runtime on OneMax.
  - The Greedy $(2+1)$ GA by Sudholt [39, 40, 208, 209]: An algorithm using a binary operator and proven to be twice as fast as any unary-operator only black-box method on OneMax.
  - $(1+1)$ FEA by Weise (汤卫思), Wu (吴志泽), Li (李新路), and Chen (陈岩) [238, 239, 250], i.e., the $(1+1)$ EA with Frequency Fitness Assignment (FFA): the *only* algorithm achieving polynomial runtime on Trap, TwoMax, and Jump. Invented here.

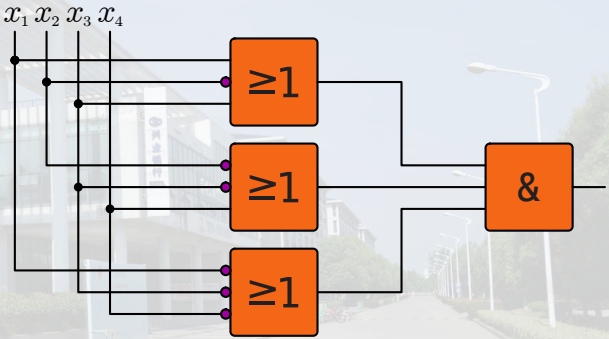


Discrete Optimization

Maximum Satisfiability Problem

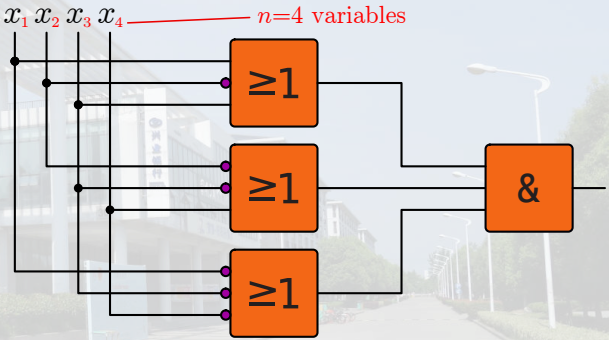
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



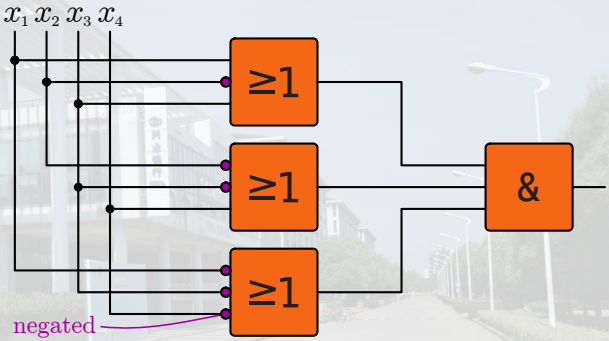
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



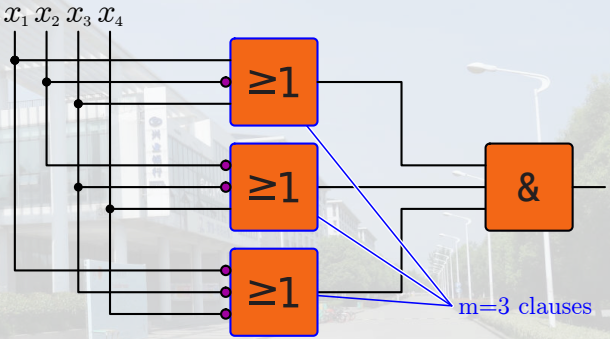
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



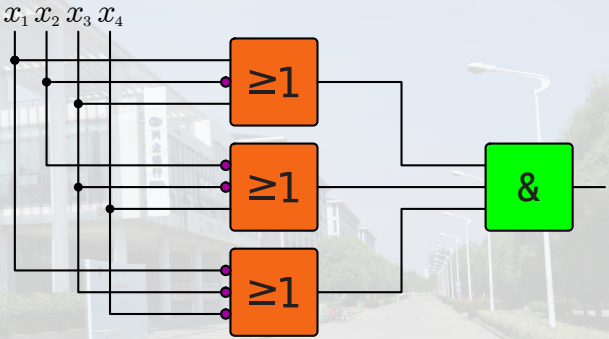
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



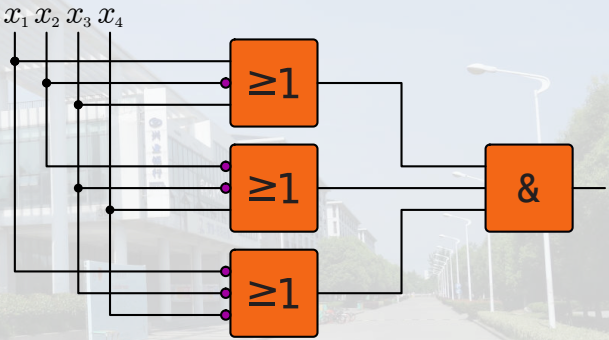
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.



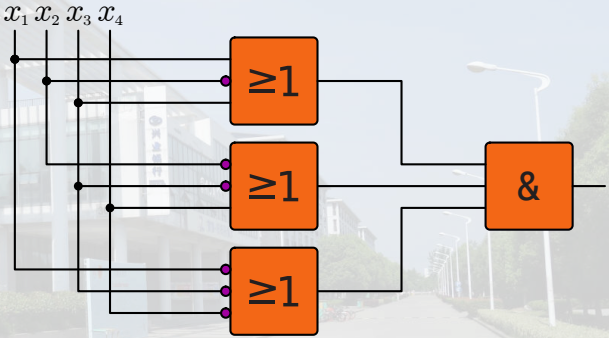
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.
- $\mathbb{X}$ is the set of all possible bit strings of length n.



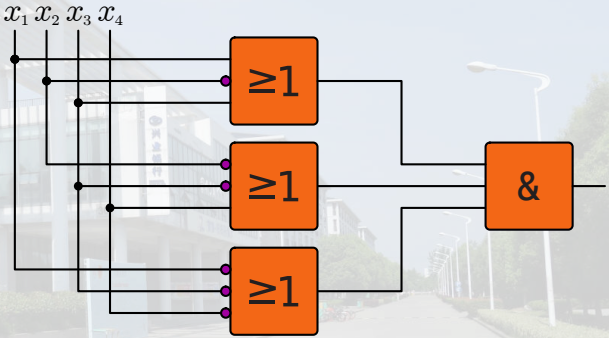
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.
- $\mathbb{X}$ is the set of all possible bit strings of length n.
- The objective function $f : \mathbb{X} \mapsto \mathbb{N}_0$ is the number of unsatisfied OR-clauses.



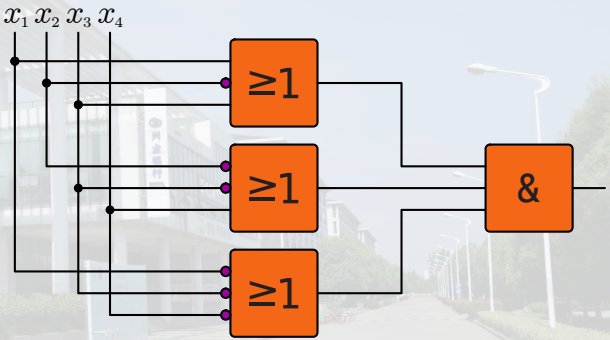
Maximum Satisfiability Problem: Introduction

- The goal of the Maximum Satisfiability (MaxSAT)[48,109] problem is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.
- $\mathbb{X}$ is the set of all possible bit strings of length n.
- The objective function $f : \mathbb{X} \mapsto \mathbb{N}_0$ is the number of unsatisfied OR-clauses.
- If the problem can be solved, then the optimum $x^\star \in \mathbb{X}$ has $f(x^\star) = 0$, i.e., all clauses satisfied, i.e., $F(x^\star) =$ True.



Maximum Satisfiability Problem: Introduction

- The goal of MaxSAT is to find a setting of n variables that makes a Boolean formula F become True. The variables appear directly or negated in m OR-clauses, whose results flow into one AND-clause.
- $\mathbb{X}$ is the set of all possible bit strings of length n.
- The objective function $f : \mathbb{X} \mapsto \mathbb{N}_0$ is the number of unsatisfied OR-clauses.
- If the problem can be solved, then the optimum $x^\star \in \mathbb{X}$ has $f(x^\star) = 0$, i.e., all clauses satisfied, i.e., $F(x^\star) = \text{True}$.
- MaxSAT is $\mathcal{NP}$-complete[48].



Maximum Satisfiability Problem: Benchmarks

- At so-called phase transition region with $\frac{m}{n} \approx 4.26$, the average instance hardness for SLS algorithms is maximal[63,64,111], so instances from this region are most interesting.

Maximum Satisfiability Problem: Benchmarks

- At so-called phase transition region with $\frac{m}{n} \approx 4.26$, the average instance hardness for SLS algorithms is maximal[63,64,111], so instances from this region are most interesting.
- SATLIB[110,111] provides satisfiable benchmark instances from the phase transition region (i.e., the hardest type of instances) for different scales $n \in 20 \cup \{25i \; \forall i \in 2..10\}$, which is enough to test black-box algorithms.

Maximum Satisfiability Problem: Benchmarks



- At so-called phase transition region with $\frac{m}{n} \approx 4.26$, the average instance hardness for SLS algorithms is maximal[63,64,111], so instances from this region are most interesting.
- SATLIB[110,111] provides satisfiable benchmark instances from the phase transition region (i.e., the hardest type of instances) for different scales $n \in 20 \cup \{25i \; \forall i \in 2..10\}$, which is enough to test black-box algorithms.
- The MaxSAT Evaluation (MSE) offers a much more comprehensive and up-to-date set of benchmark instances[117] as well as solvers and results[222].

Maximum Satisfiability: Solver Types



- We can distinguish *complete* and *incomplete* MaxSAT solvers[96].

Maximum Satisfiability: Solver Types



- We can distinguish *complete* and *incomplete* MaxSAT solvers[96].
  - Complete Solvers: guaranteed to output a provably optimal solution to any instance, i.e., a solution that solves as many clauses as possible, given enough runtime and memory.

Maximum Satisfiability: Solver Types



- We can distinguish *complete* and *incomplete* MaxSAT solvers[96].
  - Complete Solvers: guaranteed to output a provably optimal solution to any instance, i.e., a solution that solves as many clauses as possible, given enough runtime and memory.
  - Incomplete Solvers: provide good solutions quickly, but may not guarantee optimality.

Maximum Satisfiability: Solver Types



- We can distinguish *complete* and *incomplete* MaxSAT solvers[96].
  - Complete Solvers: guaranteed to output a provably optimal solution to any instance, i.e., a solution that solves as many clauses as possible, given enough runtime and memory.
  - Incomplete Solvers: provide good solutions quickly, but may not guarantee optimality.
- Metaheuristics tend to be incomplete solvers and these are therefore more interesting for us.

Maximum Satisfiability Problem: Baselines

- Traditional incomplete solvers that are the very bottom line against we should compare new idea at least are...

Maximum Satisfiability Problem: Baselines

- Traditional incomplete solvers that are the very bottom line against we should compare new idea at least are:
  - As black-box baselines, we can use the algorithms already listed for discrete benchmarks.

Maximum Satisfiability Problem: Baselines

- Traditional incomplete solvers that are the very bottom line against we should compare new idea at least are:
  - As black-box baselines, we can use the algorithms already listed for discrete benchmarks.
  - A good problem-specific baseline is WalkSAT by Selman, Kautz, and Cohen [199].

Maximum Satisfiability Problem: SOTA



Maximum Satisfiability Problem: SOTA



Maximum Satisfiability Problem: SOTA



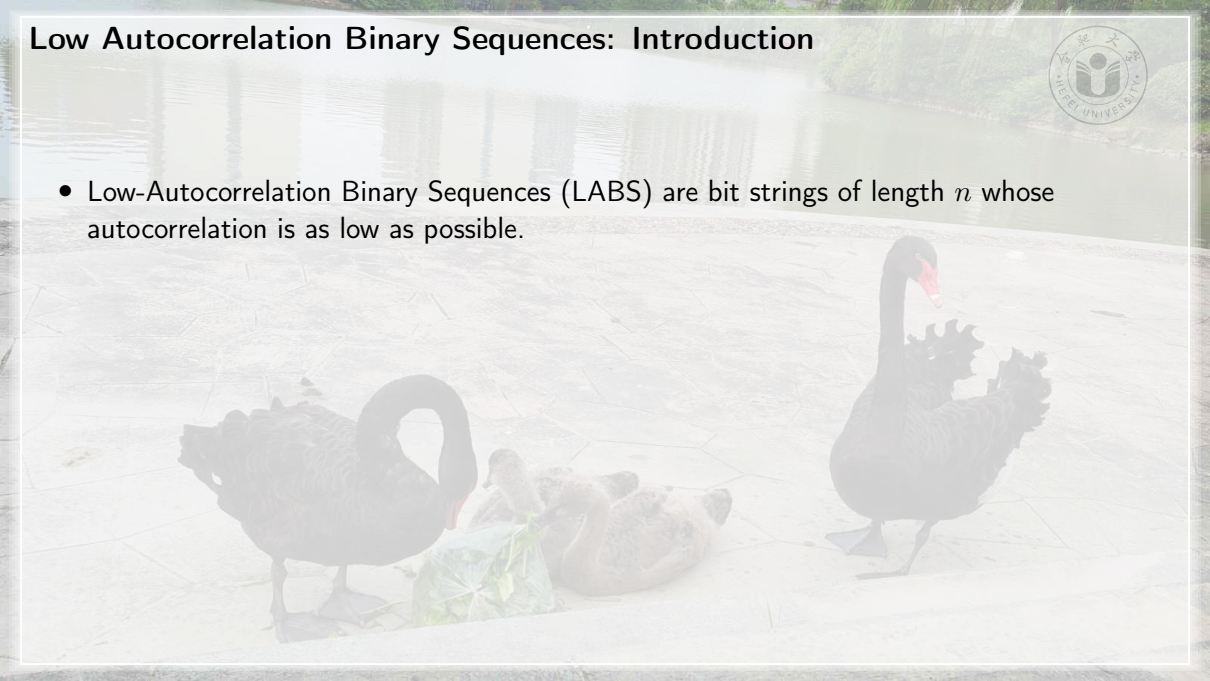
- The SOTA for heuristic incomplete solvers comprises SLS algorithms, often based on ideas dating back to WalkSAT, such as
  - Adaptive Memory-based Local Search (AMLS) by Lü and Hao [155] or
  - NuWLS by Chu, Cai, and Luo [46].



Discrete Optimization

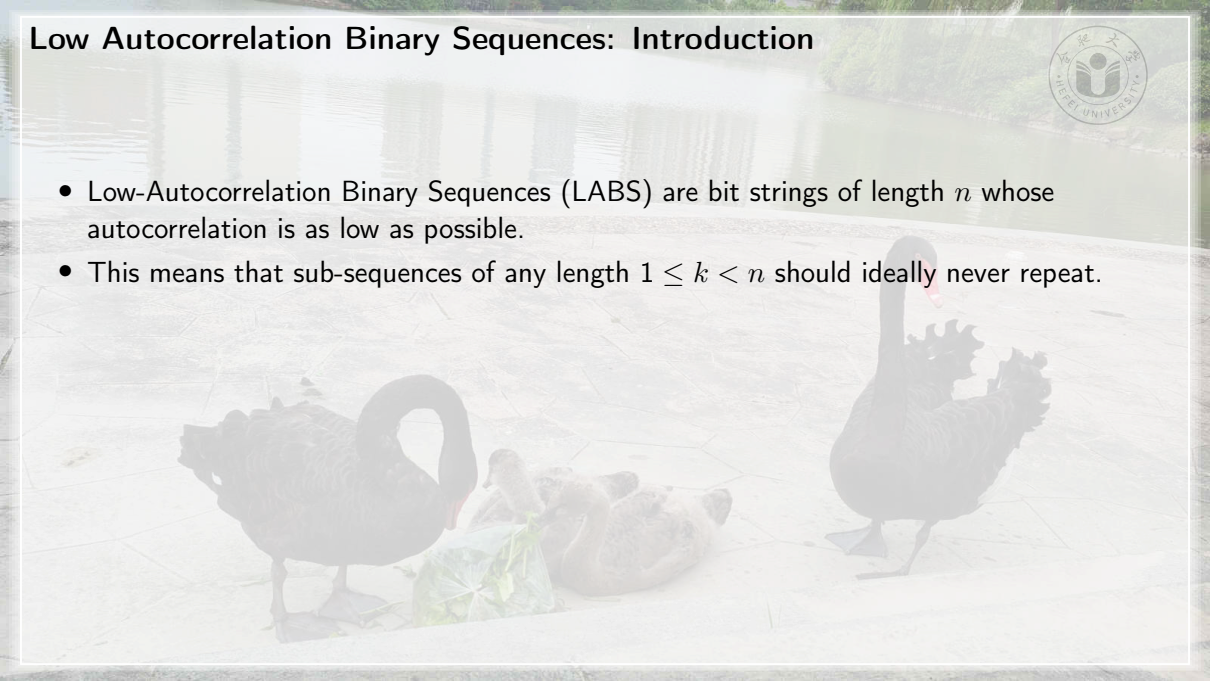
Low Autocorrelation Binary Sequences

Low Autocorrelation Binary Sequences: Introduction



- Low-Autocorrelation Binary Sequences (LABS) are bit strings of length n whose autocorrelation is as low as possible.

Low Autocorrelation Binary Sequences: Introduction



- Low-Autocorrelation Binary Sequences (LABS) are bit strings of length n whose autocorrelation is as low as possible.
- This means that sub-sequences of any length $1 \leq k < n$ should ideally never repeat.

Low Autocorrelation Binary Sequences: Introduction



- Low-Autocorrelation Binary Sequences (LABS) are bit strings of length n whose autocorrelation is as low as possible.
- This means that sub-sequences of any length $1 \leq k < n$ should ideally never repeat.
- The objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$ represents the energy of a bit string x and can be defined as $f(x) = \sum_{k=1}^{n-1} \left(\sum_{i=0}^{n-1-k} s_i \cdot s_{i+k} \right)^2$ with $s_j = 2x_j - 1$.

Low Autocorrelation Binary Sequences: Introduction



- Low-Autocorrelation Binary Sequences (LABS) are bit strings of length n whose autocorrelation is as low as possible.
- This means that sub-sequences of any length $1 \leq k < n$ should ideally never repeat.
- The objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$ represents the energy of a bit string x and can be defined as $f(x) = \sum_{k=1}^{n-1} \left(\sum_{i=0}^{n-1-k} s_i \cdot s_{i+k} \right)^2$ with $s_j = 2x_j - 1$.
- Such sequences are needed for communication engineering[33,171], cryptography[171], as well as modulation pulses in radar and sonar ranging[179].

Low Autocorrelation Binary Sequences: Introduction



- Low-Autocorrelation Binary Sequences (LABS) are bit strings of length n whose autocorrelation is as low as possible.
- This means that sub-sequences of any length $1 \leq k < n$ should ideally never repeat.
- The objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$ represents the energy of a bit string x and can be defined as $f(x) = \sum_{k=1}^{n-1} \left(\sum_{i=0}^{n-1-k} s_i \cdot s_{i+k} \right)^2$ with $s_j = 2x_j - 1$.
- Such sequences are needed for communication engineering[33,171], cryptography[171], as well as modulation pulses in radar and sonar ranging[179].
- The LABS is $\mathcal{NP}$-hard[219].

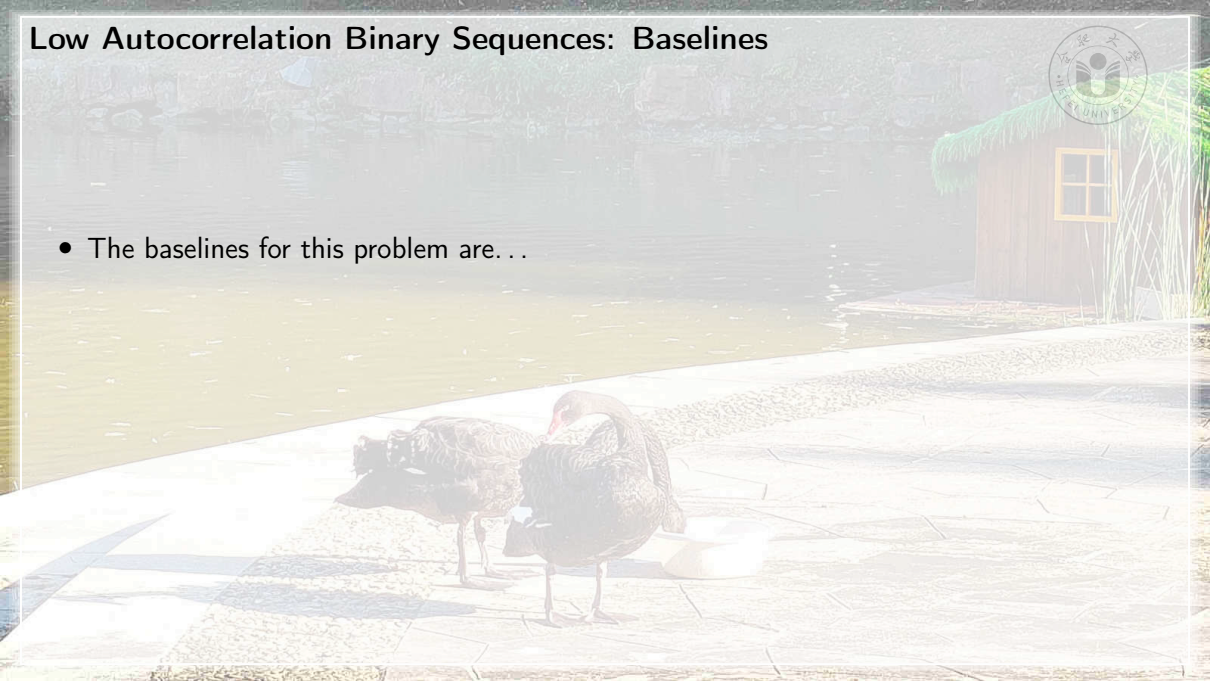
Low Autocorrelation Binary Sequences: Benchmarks

- The problem itself is also the benchmark.

Low Autocorrelation Binary Sequences: Benchmarks

- The problem itself is also the benchmark.
- So we only got the objective function f for different values of n...

Low Autocorrelation Binary Sequences: Baselines



- The baselines for this problem are...

Low Autocorrelation Binary Sequences: Baselines

- The baselines for this problem are:
  - The algorithms listed under discrete optimization benchmarks.

Low Autocorrelation Binary Sequences: Baselines



- The baselines for this problem are:
  - The algorithms listed under discrete optimization benchmarks.
  - For small scales n the GA by Militzer, Zamparelli, and Beule [164] or the exhaustive enumeration by Mertens [161].

Low Autocorrelation Binary Sequences: Baselines



- The baselines for this problem are:
  - The algorithms listed under discrete optimization benchmarks.
  - For small scales n the GA by Militzer, Zamparelli, and Beule [164] or the exhaustive enumeration by Mertens [161].
  - See also the (outdated) online repository `https://github.com/borkob/git_labs` by Bošković, Brglez, and Brest [32].

Low Autocorrelation Binary Sequences: SOTA



- The SOTA, i.e., the best current algorithms for LABS are...

Low Autocorrelation Binary Sequences: SOTA

- The SOTA, i.e., the best current algorithms for LABS are:
  - the algorithm by Dimitrov [58]

Low Autocorrelation Binary Sequences: SOTA



- The SOTA, i.e., the best current algorithms for LABS are:
  - the algorithm by Dimitrov [58],
  - the Memetic Algorithm (MA) by Mow, Du, and Wu [171]

Low Autocorrelation Binary Sequences: SOTA

- The SOTA, i.e., the best current algorithms for LABS are:
  - the algorithm by Dimitrov [58],
  - the Memetic Algorithm (MA) by Mow, Du, and Wu [171],
  - the xLastovka-family of methods by Bošković and Brest [31, 34]

Low Autocorrelation Binary Sequences: SOTA

- The SOTA, i.e., the best current algorithms for LABS are:
  - the algorithm by Dimitrov [58],
  - the Memetic Algorithm (MA) by Mow, Du, and Wu [171],
  - the xLastovka-family of methods by Bošković and Brest [31, 34], and
  - the SLS by Dimitrov, Baitcheva, and Nikolov [59] (= best-performing method).

(Multi-)Permutation Problems



(Multi-)Permutation Problems

Black-Box Permutation Problems



Black-Box Permutation Problems: Introduction

- The search space $\mathbb{X}$ corresponds to the set of all permutations of the first n natural numbers.

Black-Box Permutation Problems: Introduction

- The search space $\mathbb{X}$ corresponds to the set of all permutations of the first n natural numbers.
- The algorithms know nothing about the problem nature, they only can sample permutations x and compute their objective values $f(x)$.

Black-Box Permutation Problems: Introduction



- The search space $\mathbb{X}$ corresponds to the set of all permutations of the first n natural numbers.
- The algorithms know nothing about the problem nature, they only can sample permutations x and compute their objective values $f(x)$.
- When faced with a new problem where solutions can be mapped to permutations, it makes sense to try the baseline algorithms from this field.

Black-Box Permutation Problems: Baselines



- Some baselines for any permutation-based problem include...

Black-Box Permutation Problems: Baselines



- Some baselines for any permutation-based problem include:
  - randomized local search (RLS) with a unary operator that swaps two elements

Black-Box Permutation Problems: Baselines



- Some baselines for any permutation-based problem include:
  - randomized local search (RLS) with a unary operator that swaps two elements,
  - RLS with a unary operator that swaps a binomially distributed number of elements (similar to the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) for permutations)

Black-Box Permutation Problems: Baselines



- Some baselines for any permutation-based problem include:
  - randomized local search (RLS) with a unary operator that swaps two elements,
  - RLS with a unary operator that swaps a binomially distributed number of elements (similar to the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) for permutations),
  - Simulated Annealing (SA) with the same operators and suitable exponential temperature schedule setups

Black-Box Permutation Problems: Baselines



- Some baselines for any permutation-based problem include:
  - randomized local search (RLS) with a unary operator that swaps two elements,
  - RLS with a unary operator that swaps a binomially distributed number of elements (similar to the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) for permutations),
  - Simulated Annealing (SA) with the same operators and suitable exponential temperature schedule setups,
  - Tabu Search (TS) with the same operator that forbids moving values to recently-used indicess

Black-Box Permutation Problems: Baselines



- Some baselines for any permutation-based problem include:
  - randomized local search (RLS) with a unary operator that swaps two elements,
  - RLS with a unary operator that swaps a binomially distributed number of elements (similar to the $(1+1)$ evolutionary algorithm ($(1+1)$ EA) for permutations),
  - Simulated Annealing (SA) with the same operators and suitable exponential temperature schedule setups,
  - Tabu Search (TS) with the same operator that forbids moving values to recently-used indicess, and a
  - $(\mu+\lambda)$ evolutionary algorithm ($(\mu+\lambda)$ EA) with these operators and maybe Alternating Position Crossover (AP)[134,135].

(Multi-)Permutation Problems

Traveling Salesperson Problem



Traveling Salesperson Problem: Introduction

- The goal of a TSP is to find the shortest round-trip tour through n cities.

Traveling Salesperson Problem: Introduction



- The goal of a TSP is to find the shortest round-trip tour through n cities.
- Solutions are often encoded as permutations x, where each city gets a number and the order of numbers in the permutation corresponds to the order of cities in the tour.

Traveling Salesperson Problem: Introduction



- The goal of a TSP is to find the shortest round-trip tour through n cities.
- Solutions are often encoded as permutations x, where each city gets a number and the order of numbers in the permutation corresponds to the order of cities in the tour.
- The distances between the n cities are given in a $n \times n$ distance matrix A, where $A[a, b]$ is the distance from city a to city b.

Traveling Salesperson Problem: Introduction



- The goal of a TSP is to find the shortest round-trip tour through n cities.
- Solutions are often encoded as permutations x, where each city gets a number and the order of numbers in the permutation corresponds to the order of cities in the tour.
- The distances between the n cities are given in a $n \times n$ distance matrix A, where $A[a, b]$ is the distance from city a to city b.
- Hence, the objective function subject to minimization is $f(x) = A[x_n, x_1] + \sum_{i=1}^{n-1} A[x_i, x_{i+1}]$.

Traveling Salesperson Problem: Introduction



- The goal of a TSP is to find the shortest round-trip tour through n cities.
- Solutions are often encoded as permutations x, where each city gets a number and the order of numbers in the permutation corresponds to the order of cities in the tour.
- The distances between the n cities are given in a $n \times n$ distance matrix A, where $A[a, b]$ is the distance from city a to city b.
- Hence, the objective function subject to minimization is $f(x) = A[x_n, x_1] + \sum_{i=1}^{n-1} A[x_i, x_{i+1}]$.
- The TSP has many applications[5], including in transportation, logistics, (circuit) plotting and drilling problems[102], telescope aiming, and genome sequencing.

Traveling Salesperson Problem: Introduction



- The goal of a TSP is to find the shortest round-trip tour through n cities.
- Solutions are often encoded as permutations x, where each city gets a number and the order of numbers in the permutation corresponds to the order of cities in the tour.
- The distances between the n cities are given in a $n \times n$ distance matrix A, where $A[a, b]$ is the distance from city a to city b.
- Hence, the objective function subject to minimization is $f(x) = A[x_n, x_1] + \sum_{i=1}^{n-1} A[x_i, x_{i+1}]$.
- The TSP has many applications[5], including in transportation, logistics, (circuit) plotting and drilling problems[102], telescope aiming, and genome sequencing.
- The TSP is $\mathcal{NP}$-hard[89,103,132].

Traveling Salesperson Problem: Benchmarks



- The commone benchmark instances for the TSP can be obtained from...

Traveling Salesperson Problem: Benchmarks



- The commone benchmark instances for the TSP can be obtained from:
  - TSPLib with 110 symmetric instances with scales $n \in 14..85\,900$ and 19 asymmetric instances with scales $n \in 17..443$[194,195].

Traveling Salesperson Problem: Benchmarks



- The commone benchmark instances for the TSP can be obtained from:
  - TSPLib with 110 symmetric instances with scales $n \in 14..85\,900$ and 19 asymmetric instances with scales $n \in 17..443$[194,195].
  - The instances of the 8th DIMACS challenge[122–124], which have up to 10 000 000 cities.

Traveling Salesperson Problem: Benchmarks



- The commone benchmark instances for the TSP can be obtained from:
  - TSPLib with 110 symmetric instances with scales $n \in 14..85\,900$ and 19 asymmetric instances with scales $n \in 17..443$[194,195].
  - The instances of the 8th DIMACS challenge[122–124], which have up to 10 000 000 cities.
  - The *World TSP* repository[51] with many real-world geographical instances, including one with all 1 904 711 registered populated places of the Earth at some point in time in the 2000s.

Traveling Salesperson Problem: Benchmarks



- The commone benchmark instances for the TSP can be obtained from:
  - TSPLib with 110 symmetric instances with scales $n \in 14..85\,900$ and 19 asymmetric instances with scales $n \in 17..443$[194,195].
  - The instances of the 8th DIMACS challenge[122–124], which have up to 10 000 000 cities.
  - The *World TSP* repository[51] with many real-world geographical instances, including one with all 1 904 711 registered populated places of the Earth at some point in time in the 2000s.
  - The TSP Suite, which offers an integrated Java environment for benchmarking algorithms on the TSPLib instances[230].

Traveling Salesperson Problem: Baselines



- Besides permutation baselines, the baseline for the TSP is the Lin-Kernighan (LK) algorithm[146], which exchanges variable number of edges in a tour in each step.

Traveling Salesperson Problem: SOTA



- The SOTA on the TSP comprises...

Traveling Salesperson Problem: SOTA



- The SOTA on the TSP comprises:
  - The Lin-Kernighan Helsgaun (LKH) algorithm building on top of the LK algorithm[106,107].

Traveling Salesperson Problem: SOTA



- The SOTA on the TSP comprises:
  - The Lin-Kernighan Helsgaun (LKH) algorithm building on top of the LK algorithm[106,107].
  - An evolutionary algorithm (EA) with edge assembly crossover (EAX) as binary operator by Nagata and Kobayashi [173–175].

Traveling Salesperson Problem: SOTA



- The SOTA on the TSP comprises:
  - The Lin-Kernighan Helsgaun (LKH) algorithm building on top of the LK algorithm[106,107].
  - An evolutionary algorithm (EA) with edge assembly crossover (EAX) as binary operator by Nagata and Kobayashi [173–175].
  - An Memetic Algorithm (MA) with Generalized Partition Crossover (GPX) as binary operator by Whitley, Hains, and Howe [243].

Traveling Salesperson Problem: SOTA



- The SOTA on the TSP comprises:
  - The Lin-Kernighan Helsgaun (LKH) algorithm building on top of the LK algorithm[106,107].
  - An evolutionary algorithm (EA) with edge assembly crossover (EAX) as binary operator by Nagata and Kobayashi [173–175].
  - An Memetic Algorithm (MA) with Generalized Partition Crossover (GPX) as binary operator by Whitley, Hains, and Howe [243].
  - Especially: LKH run in an MA-fashion with GPX as binary operator as proposed by Tinós, Helsgaun, and Whitley [218].

Traveling Salesperson Problem: SOTA



- The SOTA on the TSP comprises:
  - The Lin-Kernighan Helsgaun (LKH) algorithm building on top of the LK algorithm[106,107].
  - An evolutionary algorithm (EA) with edge assembly crossover (EAX) as binary operator by Nagata and Kobayashi [173–175].
  - An Memetic Algorithm (MA) with Generalized Partition Crossover (GPX) as binary operator by Whitley, Hains, and Howe [243].
  - Especially: LKH run in an MA-fashion with GPX as binary operator as proposed by Tinós, Helsgaun, and Whitley [218].
  - Concorde, an exact solver for the TSP by Applegate, Bixby, Chvátal, and Cook [5, 49, 50], which even solved the largest instance of TSPLib (with 85 900 cities) to optimality! (This algorithm uses LK-style algorithms methods like LKH to find its starting point.)

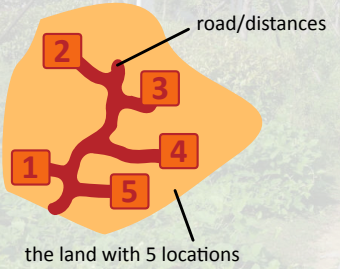
(Multi-)Permutation Problems

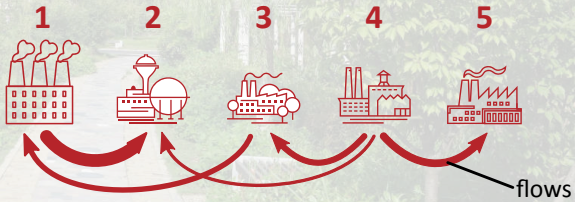
Quadratic Assignment Problem



Quadratic Assignment Problem: Introduction

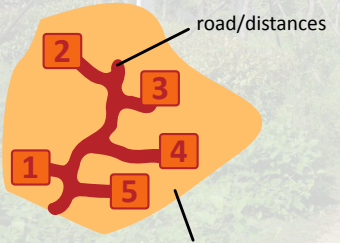
- In a Quadratic Assignment Problem (QAP), n facilities need to be assigned to n locations[36,43,130,151].



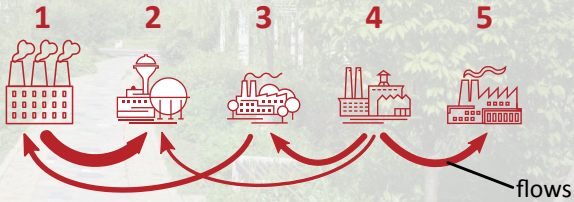


Quadratic Assignment Problem: Introduction

- In a Quadratic Assignment Problem (QAP), n facilities need to be assigned to n locations[36,43,130,151].
- Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed.



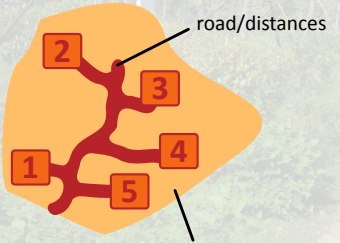
the land with 5 locations



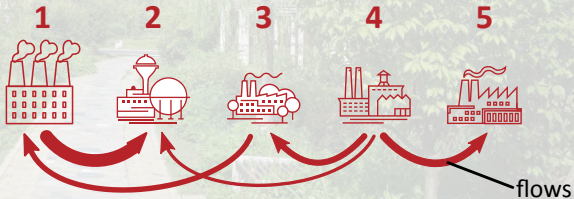
the 5 workshops which need to be assigned to the 5 locations and the different material flows between them

Quadratic Assignment Problem: Introduction

- In a Quadratic Assignment Problem (QAP), n facilities need to be assigned to n locations[36,43,130,151].
- Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed.
- A is the distance matrix and $A[a, b]$ is the distance from location a to location b.



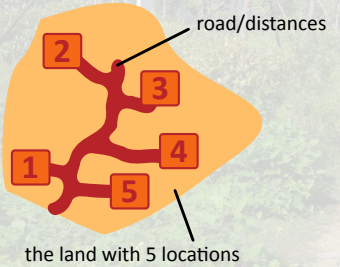
the land with 5 locations



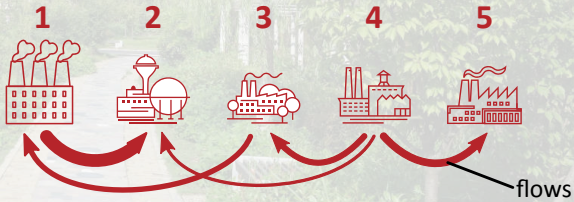
the 5 workshops which need to be assigned to the 5 locations and the different material flows between them

Quadratic Assignment Problem: Introduction

- In a Quadratic Assignment Problem (QAP), n facilities need to be assigned to n locations[36,43,130,151].
- Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed.
- A is the distance matrix and $A[a, b]$ is the distance from location a to location b.
- B is the flow matrix and $B[i, j]$ is the amount of material flowing (e.g., daily) from facility i to facility j.



the land with 5 locations



the 5 workshops which need to be assigned to the 5 locations and the different material flows between them

Quadratic Assignment Problem: Introduction



- In a Quadratic Assignment Problem (QAP), n facilities need to be assigned to n locations[36,43,130,151].
- Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed.
- A is the distance matrix and $A[a, b]$ is the distance from location a to location b.
- B is the flow matrix and $B[i, j]$ is the amount of material flowing (e.g., daily) from facility i to facility j.
- We want to minimize the objective function $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A[x_i, x_j] * B[i, j]$, i.e., we want to minimize the sum of the products of distance and flows.

Quadratic Assignment Problem: Introduction



- In a Quadratic Assignment Problem (QAP), n facilities need to be assigned to n locations[36,43,130,151].
- Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed.
- A is the distance matrix and $A[a, b]$ is the distance from location a to location b.
- B is the flow matrix and $B[i, j]$ is the amount of material flowing (e.g., daily) from facility i to facility j.
- We want to minimize the objective function $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A[x_i, x_j] * B[i, j]$, i.e., we want to minimize the sum of the products of distance and flows.
- The QAP has applications such as building layout[52,76,131], keyboard layout[38], circuit design[77], wiring[204], and scheduling[203].

Quadratic Assignment Problem: Introduction



- In a Quadratic Assignment Problem (QAP), n facilities need to be assigned to n locations[36,43,130,151].
- Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed.
- A is the distance matrix and $A[a, b]$ is the distance from location a to location b.
- B is the flow matrix and $B[i, j]$ is the amount of material flowing (e.g., daily) from facility i to facility j.
- We want to minimize the objective function $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A[x_i, x_j] * B[i, j]$, i.e., we want to minimize the sum of the products of distance and flows.
- The QAP has applications such as building layout[52,76,131], keyboard layout[38], circuit design[77], wiring[204], and scheduling[203].
- The QAP is $\mathcal{NP}$-complete[71,197] and even among this problem class, is considered as particularly hard for metaheuristics[20,36,151,207].

Quadratic Assignment Problem: Benchmarks



- The typical benchmark for the QAP are the 136 instances of QAPLIB[35,37], 28 of which have not yet been solved to optimality[3] (at least by 2023).

Quadratic Assignment Problem: Baselines



- Algorithms that have performed well on the QAP and therefore are baselines include...

Quadratic Assignment Problem: Baselines



- Algorithms that have performed well on the QAP and therefore are baselines include:
  - the robust Tabu Search (TS) algorithms by Taillard [211, 212] and Misevičius [165]

Quadratic Assignment Problem: Baselines



- Algorithms that have performed well on the QAP and therefore are baselines include:
  - the robust Tabu Search (TS) algorithms by Taillard [211, 212] and Misevičius [165],
  - Simulated Annealing (SA) by Thonemann and Bölte [217], Misevičius [166]

Quadratic Assignment Problem: Baselines



- Algorithms that have performed well on the QAP and therefore are baselines include:
  - the robust Tabu Search (TS) algorithms by Taillard [211, 212] and Misevičius [165],
  - Simulated Annealing (SA) by Thonemann and Bölte [217], Misevičius [166],
  - the Reactive Tabu Search (RTS) by Battiti and Tecchiolli [13, 14]

Quadratic Assignment Problem: Baselines



- Algorithms that have performed well on the QAP and therefore are baselines include:
  - the robust Tabu Search (TS) algorithms by Taillard [211, 212] and Misevičius [165],
  - Simulated Annealing (SA) by Thonemann and Bölte [217], Misevičius [166],
  - the Reactive Tabu Search (RTS) by Battiti and Tecchiolli [13, 14], and
  - the iterated local search (ILS) by Stützle [207].

Quadratic Assignment Problem: SOTA

- Beham, Affenzeller, and Pitzer [20] provide a nice experimental overview on the SOTA of the QAP.

Quadratic Assignment Problem: SOTA



- Beham, Affenzeller, and Pitzer [20] provide a nice experimental overview on the SOTA of the QAP.
- Apart from this, we can identify the following methods:
  - the Memetic Algorithms (MAs) by Fleurent and Ferland [85] and Taillard and Gambardella [213], which use the TS from [212] as local search,

Quadratic Assignment Problem: SOTA

- Beham, Affenzeller, and Pitzer [20] provide a nice experimental overview on the SOTA of the QAP.
- Apart from this, we can identify the following methods:
  - the Memetic Algorithms (MAs) by Fleurent and Ferland [85] and Taillard and Gambardella [213], which use the TS from [212] as local search,
  - the MA by Merz and Freisleben [162], and

Quadratic Assignment Problem: SOTA

- Beham, Affenzeller, and Pitzer [20] provide a nice experimental overview on the SOTA of the QAP.
- Apart from this, we can identify the following methods:
  - the Memetic Algorithms (MAs) by Fleurent and Ferland [85] and Taillard and Gambardella [213], which use the TS from [212] as local search,
  - the MA by Merz and Freisleben [162], and
  - the Iterated TS by Misevičius [168].

(Multi-)Permutation Problems

Job Shop Scheduling Problem

Job Shop Scheduling Problem: Introduction



- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.

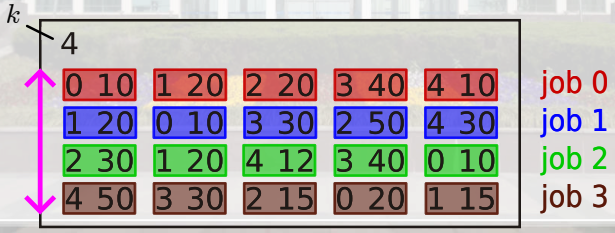
Job Shop Scheduling Problem: Introduction



- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.

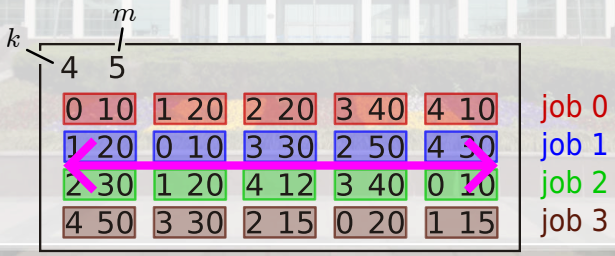
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.



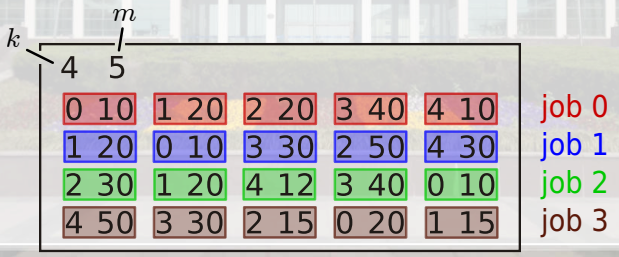
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.



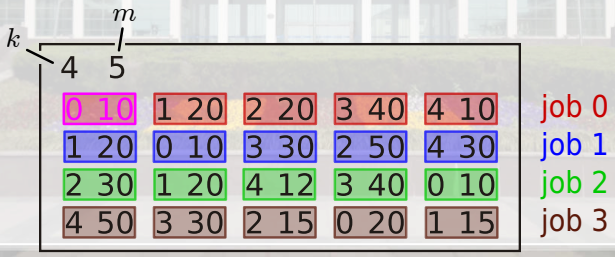
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.



Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0.



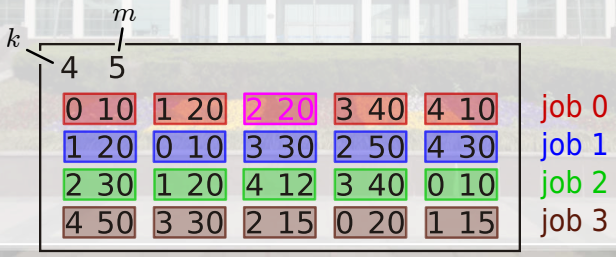
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0, then it goes for 20s to machine 1.



Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0, then it goes for 20s to machine 1, then for 20s to machine 2.



Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0, then it goes for 20s to machine 1, then for 20s to machine 2, then for 40s to machine 3.



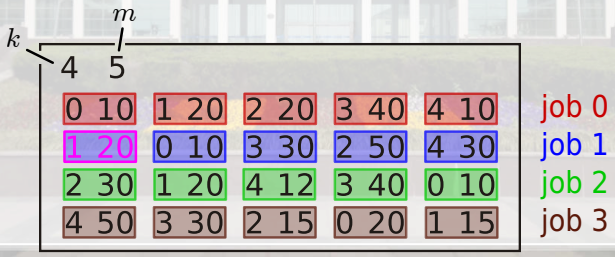
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0, then it goes for 20s to machine 1, then for 20s to machine 2, then for 40s to machine 3, and finally for 10s to machine 4.



Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0, then it goes for 20s to machine 1, then for 20s to machine 2, then for 40s to machine 3, and finally for 10s to machine 4.
- Job 1 first goes for 20s to machine 1



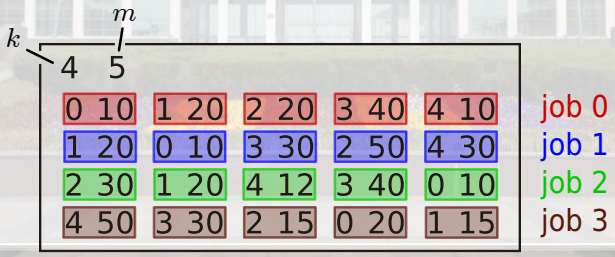
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0, then it goes for 20s to machine 1, then for 20s to machine 2, then for 40s to machine 3, and finally for 10s to machine 4.
- Job 1 first goes for 20s to machine 1, then it goes for 10s to machine 0.



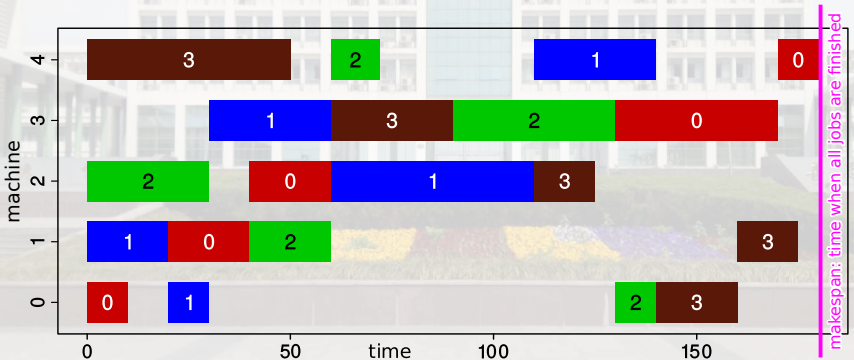
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- A JSSP instance is defined by k jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- Job 0 first goes for 10s to machine 0, then it goes for 20s to machine 1, then for 20s to machine 2, then for 40s to machine 3, and finally for 10s to machine 4.
- Job 1 first goes for 20s to machine 1, then it goes for 10s to machine 0. . . . . and so on.



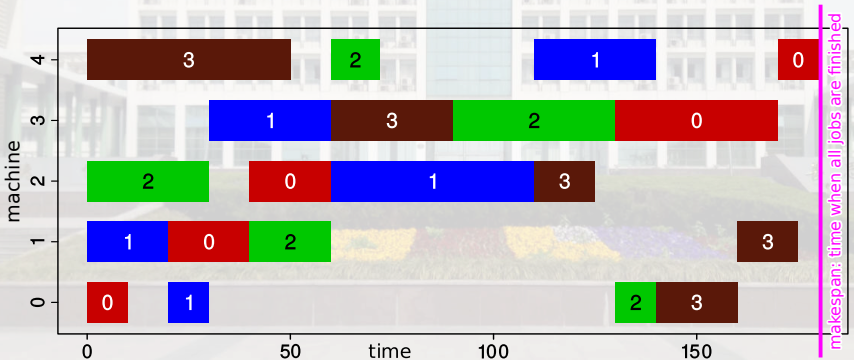
Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- The goal is to find the assignment of jobs to machines with the smallest possible makespan, i.e., the schedule (Gantt chart[127,244]) that finishes the fastest.



Job Shop Scheduling Problem: Introduction

- The Job Shop Scheduling Problem (JSSP) [28,45,101,137] is one of the most prominent and well-studied scheduling tasks.
- The goal is to find the assignment of jobs to machines with the smallest possible makespan, i.e., the schedule (Gantt chart[127,244]) that finishes the fastest.
- The JSSP is a $\mathcal{NP}$-hard[42,136,137].



Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.
- However, it is possible to map permutations of operations to Gantt charts.

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.
- However, it is possible to map permutations of operations to Gantt charts.
- And we know how to search in a space of permutations...

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.
- However, it is possible to map permutations of operations to Gantt charts.
- And we know how to search in a space of permutations...
- Idea by Gen, Tsujimura, and Kubota [90], Bierwirth [26, 27], and Shi, Iima, and Sannomiya [200]:

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.
- However, it is possible to map permutations of operations to Gantt charts.
- And we know how to search in a space of permutations...
- Idea by Gen, Tsujimura, and Kubota [90], Bierwirth [26, 27], and Shi, Iima, and Sannomiya [200]:
  - The search space $\mathbb{X}$ corresponds of permutations with repetitions x, where each of the k job IDs appears m times, i.e., the problem scale is $n = k * m$.

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.
- However, it is possible to map permutations of operations to Gantt charts.
- And we know how to search in a space of permutations...
- Idea by Gen, Tsujimura, and Kubota [90], Bierwirth [26, 27], and Shi, Iima, and Sannomiya [200]:
  - The search space $\mathbb{X}$ corresponds of permutations with repetitions x, where each of the k job IDs appears m times, i.e., the problem scale is $n = k * m$.
  - The solution space $\mathbb{Y}$ is the set of correct Gantt charts y.

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.
- However, it is possible to map permutations of operations to Gantt charts.
- And we know how to search in a space of permutations...
- Idea by Gen, Tsujimura, and Kubota [90], Bierwirth [26, 27], and Shi, Iima, and Sannomiya [200]:
  - The search space $\mathbb{X}$ corresponds of permutations with repetitions x, where each of the k job IDs appears m times, i.e., the problem scale is $n = k * m$.
  - The solution space $\mathbb{Y}$ is the set of correct Gantt charts y.
  - A decoding function $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ is designed that constructs a Gantt chart $y = \gamma(x)$ from such a permutation x.

Job Shop Scheduling Problem: Possible Metaheuristic Approach



- The vast majority of the SOTA algorithms for the JSSP use graph-based representations for solutions[28,177].
- However, schedules / Gantt charts / graphs are complex data structures.
- Defining search operators for them is a bit more complicated than working on permutations or bit strings.
- However, it is possible to map permutations of operations to Gantt charts.
- And we know how to search in a space of permutations...
- Idea by Gen, Tsujimura, and Kubota [90], Bierwirth [26, 27], and Shi, Iima, and Sannomiya [200]:
  - The search space $\mathbb{X}$ corresponds of permutations with repetitions x, where each of the k job IDs appears m times, i.e., the problem scale is $n = k * m$.
  - The solution space $\mathbb{Y}$ is the set of correct Gantt charts y.
  - A decoding function $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ is designed that constructs a Gantt chart $y = \gamma(x)$ from such a permutation x.
  - Then we can use the algorithms and operators we already know...

Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)

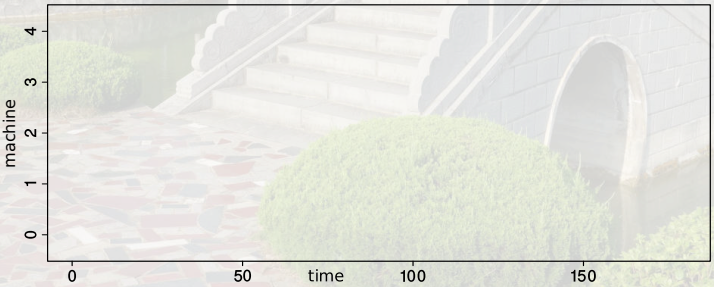
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



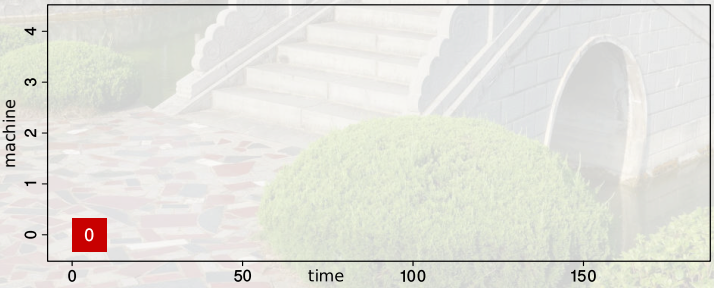
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

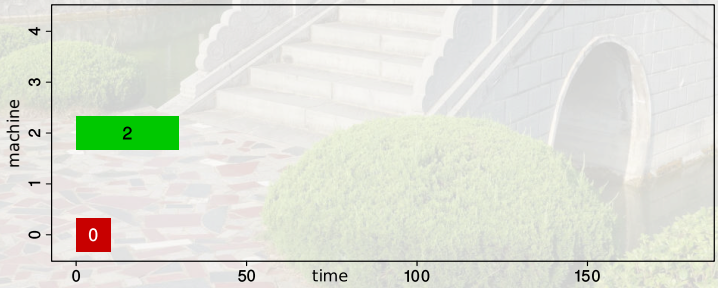
- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

k / m

4	5	
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

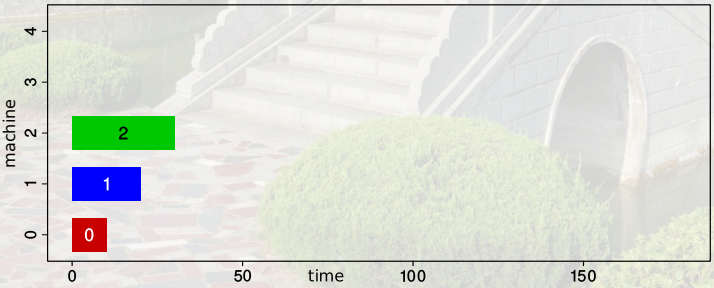
- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

k \ m	4	5	
0	0 10	1 20	job 0
1	1 20	0 10	job 1
2	2 30	1 20	job 2
3	4 50	3 30	job 3

2 20	3 40	4 10			
3 30	2 50	4 30			
4 12	3 40	0 10			
2 15	0 20	1 15			

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



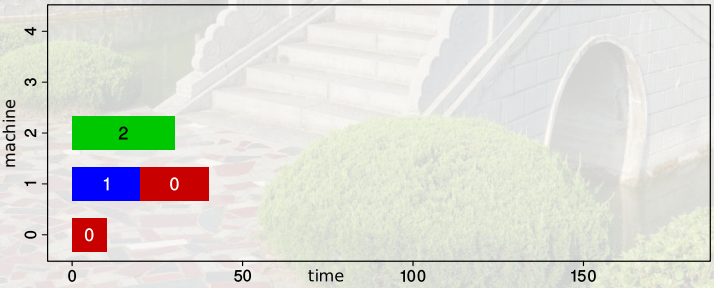
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



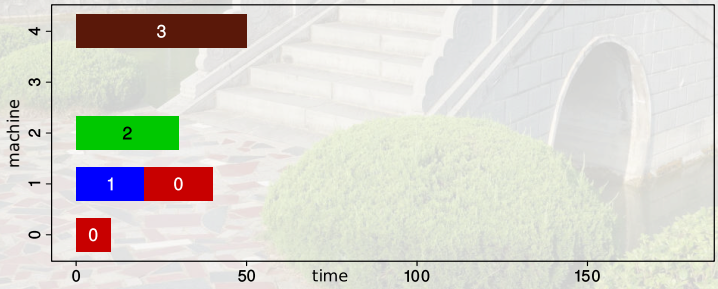
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



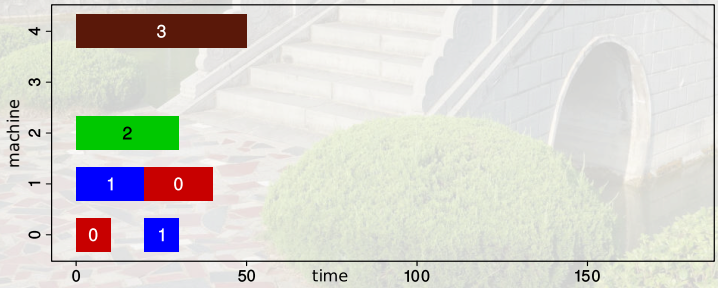
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

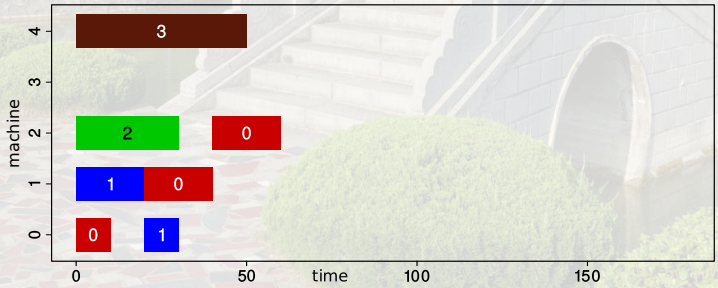
- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

k \ m	4	5	
0	0 10	1 20	job 0
1	1 20	0 10	job 1
2	2 30	1 20	job 2
3	4 50	3 30	job 3

2 20	3 40	4 10	job 0
3 30	2 50	4 30	job 1
4 12	3 40	0 10	job 2
2 15	0 20	1 15	job 3

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



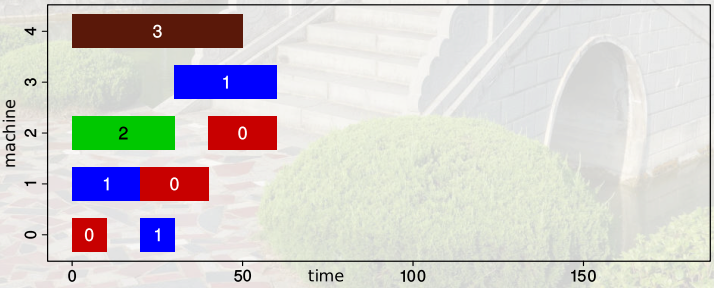
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

k \ m: 4 5

0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



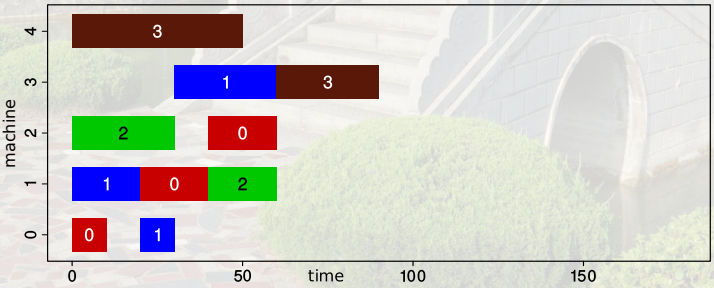
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

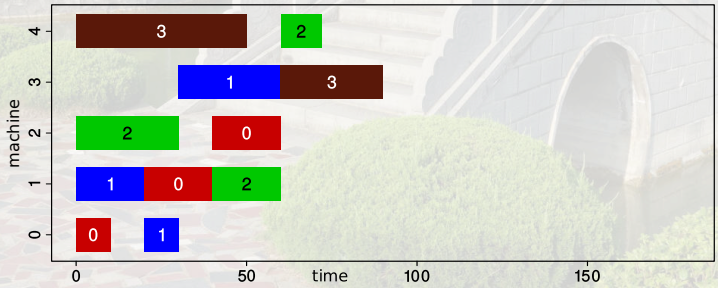
k / m

4	5	
0 10	1 20	job 0
1 20	0 10	job 1
2 30	1 20	job 2
4 50	3 30	job 3

2 20	3 40	4 10	
3 30	2 50	4 30	
4 12	3 40	0 10	
2 15	0 20	1 15	

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

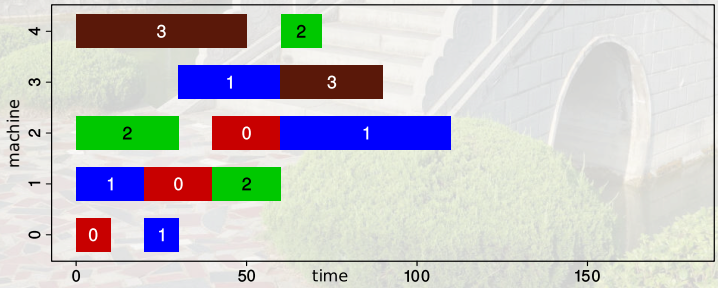
k \ m

4	5	
0 10	1 20	job 0
1 20	0 10	job 1
2 30	1 20	job 2
4 50	3 30	job 3

2 20	3 40	4 10
3 30	2 50	4 30
4 12	3 40	0 10
2 15	0 20	1 15

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



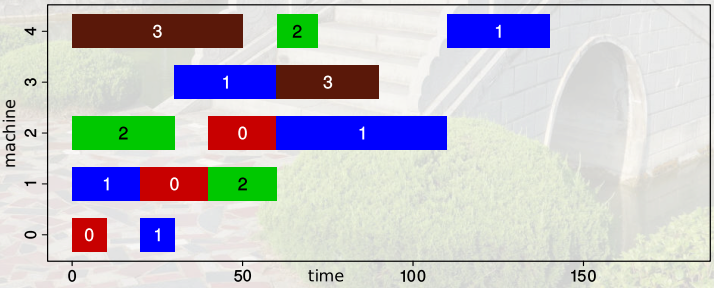
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



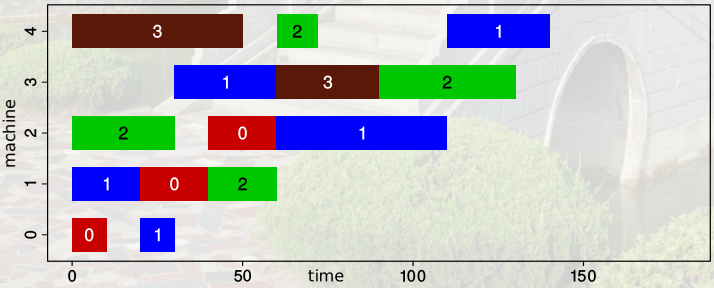
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



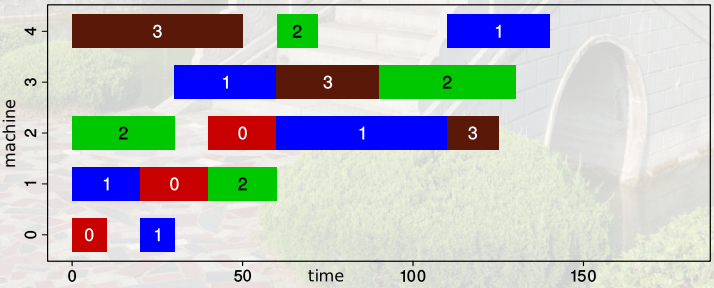
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

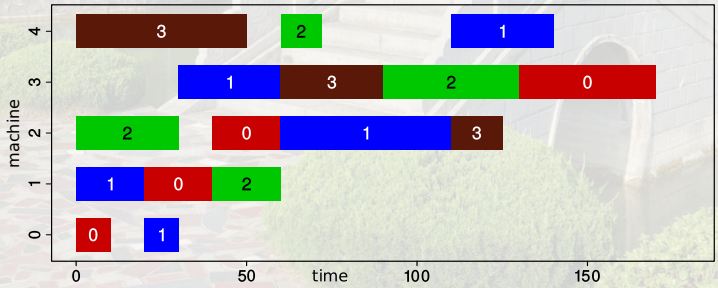
- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

k / m

4	5	
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

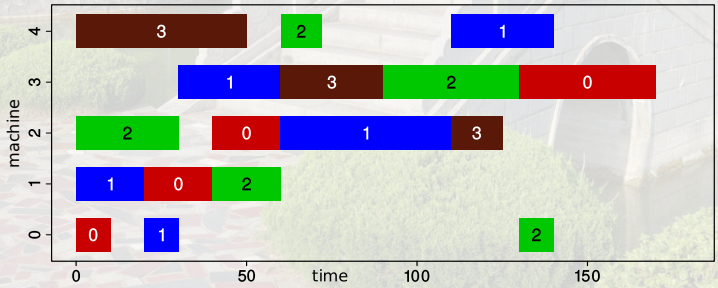
- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

k \ m

4	5	
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Possible Metaheuristic Approach

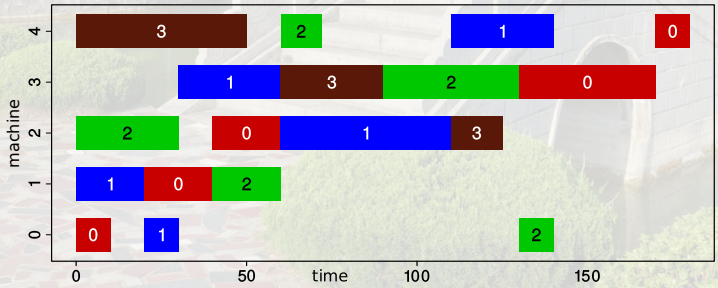
- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.

k \ m

4	5	
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

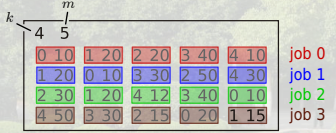
example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



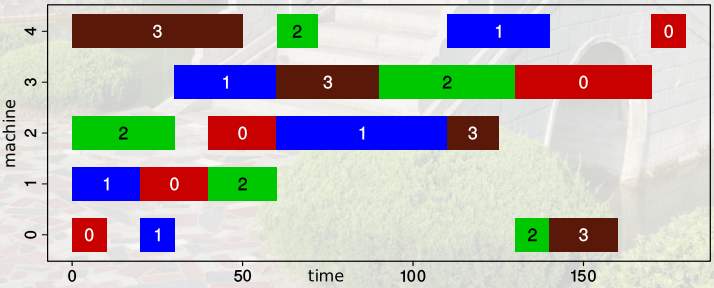
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



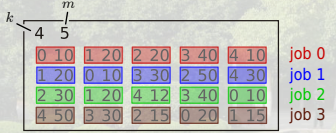
example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



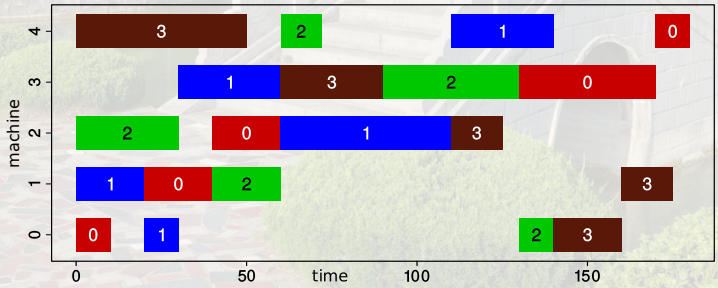
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



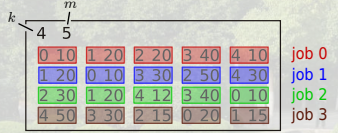
example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



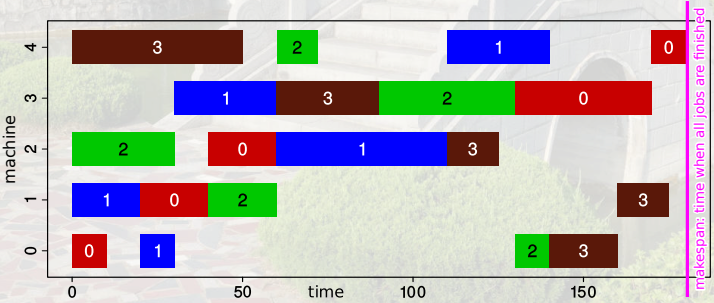
Job Shop Scheduling Problem: Possible Metaheuristic Approach

- The decoder γ processes the permutations $x \in \mathbb{X}$ from beginning to end and places the operations belonging to the listed jobs at their corresponding machines at the earliest possible time in a Gantt chart $y \in \mathbb{Y}$.



example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)



Job Shop Scheduling Problem: Benchmark

- The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[57], *ft*[84], *la*[139], *orb*[6], *swv*[205], *ta*[210], and *yn*[247].

Job Shop Scheduling Problem: Benchmark



- The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[57], *ft*[84], *la*[139], *orb*[6], *swv*[205], *ta*[210], and *yn*[247].
- Their data can be found (sometimes partially) in several repositories in the Internet, such as
  - the OR-Library managed by Beasley [18, 19]
    https://people.brunel.ac.uk/~mastjjb/jeb/info.html

Job Shop Scheduling Problem: Benchmark



- The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[57], *ft*[84], *la*[139], *orb*[6], *swv*[205], *ta*[210], and *yn*[247].
- Their data can be found (sometimes partially) in several repositories in the Internet, such as
  - the OR-Library managed by Beasley [18, 19] https://people.brunel.ac.uk/~mastjjb/jeb/info.html,
  - Shylo's page http://optimizizer.com/jobshop.php

Job Shop Scheduling Problem: Benchmark



- The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[57], *ft*[84], *la*[139], *orb*[6], *swv*[205], *ta*[210], and *yn*[247].
- Their data can be found (sometimes partially) in several repositories in the Internet, such as
  - the OR-Library managed by Beasley [18, 19] https://people.brunel.ac.uk/~mastjjb/jeb/info.html,
  - Shylo's page http://optimizizer.com/jobshop.php, or
  - my own repository [227] at https://github.com/thomasWeise/jsspInstancesAndResults, where I collect all the above problem instances and many results from existing works.

Job Shop Scheduling Problem: Benchmark

- The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[57], *ft*[84], *la*[139], *orb*[6], *swv*[205], *ta*[210], and *yn*[247].

```
+++++++++++++++++++++++++++++
Lawrence 15x10 instance (Table 7, instance 4); also called (setb4) or (B4)
15 10
7  8 9 75 0 72 6 74 4 30 8 43 2 38 5 98 1 26 3 19
6 19 8 73 3 43 0 23 1 85 4 39 5 13 9 26 2 67 7  9
1 50 3 93 5 80 4  7 0 55 2 61 6 57 8 72 9 42 7 46
1 68 7 43 4 99 6 60 5 68 0 91 8 11 3 96 9 11 2 72
7 84 2 34 8 40 5  7 1 70 6 74 3 12 0 43 9 69 4 30
8 60 0 49 4 59 5 72 9 63 1 69 7 99 6 45 3 27 2  9
6 71 2 91 8 65 1 90 9 98 4  8 7 50 0 75 5 37 3 17
8 62 7 90 5 98 3 31 2 91 4 38 9 72 1  9 0 72 6 49
4 35 0 39 9 74 5 25 7 47 3 52 2 63 8 21 6 35 1 80
9 58 0  5 3 50 8 52 1 88 6 20 2 68 5 24 4 53 7 57
7 99 3 91 4 33 5 19 2 18 6 38 0 24 9 35 1 49 8  9
0 68 3 60 2 77 7 10 8 60 5 15 9 72 1 18 6 90 4 18
9 79 1 60 3 56 6 91 2 40 8 86 7 72 0 80 5 89 4 51
4 10 2 92 5 23 6 46 8 40 7 72 3  6 1 23 0 95 9 34
```

15 jobs

10 machines

Job Shop Scheduling Problem: Benchmark

- The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[57], *ft*[84], *la*[139], *orb*[6], *swv*[205], *ta*[210], and *yn*[247].

15 jobs

10 machines

```
+++++++++++++++++++++++++++
Lawrence 15x10 instance (Table 7, instance 4); also called (setb4) or (84)
15 10
 7  8  9 75  0 72  6 74  4 30  8 43  2 38  5 98  1 26  3 19
 6 19  8 73  3 43  0 23  1 85  4 39  5 13  9 26  2 67  7  9
 1 50  3 93  5 80  4  7  0 55  2 61  6 57  8 72  9 42  7 46
 1 68  7 43  4 99  6 60  5 68  0 91  8 11  3 96  9 11  2 72
 7 84  2 34  8 40  5  7  1 70  6 74  3 12  0 43  9 69  4 30
 8 60  0 49  4 59  5 72  9 63  1 49  7 99  6 45  3 27  2  9
 6 71  2 91  8 65  1 90  9 98  4  8  7 50  0 75  5 37  3 17
 8 62  7 90  5 98  3 31  2 91  4 38  9 72  1  9  0 72  6 49
 4 35  0 39  9 74  5 25  7 47  3 52  2 63  8 21  6 35  1 80
 9 58  0  5  3 50  8 52  1 88  6 20  2 68  5 24  4 53  7 57
 7 99  3 91  4 33  5 19  2 18  6 38  0 24  9 35  1 49  8  9
 0 68  3 60  2 77  7 10  8 60  5 15  9 72  1 18  6 90  4 18
 9 79  1 60  3 56  6 91  2 40  8 86  7 72  0 80  5 89  4 51
 4 10  2 92  5 23  6 46  8 40  7 72  3  6  1 23  9 95  9 34
 2 24  5 29  9 49  8 55  0 47  6 77  3 77  7  8  1 28  4 48
```

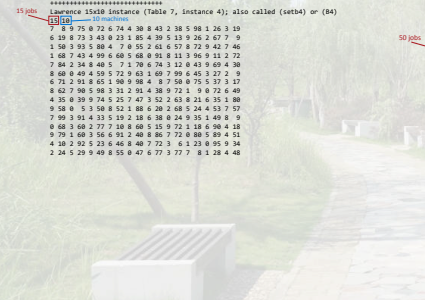
50 jobs

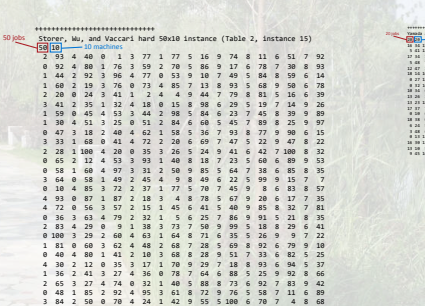
10 machines

```
+++++++++++++++++++++++++++++
Storer, Wu, and Vaccari hard 50x10 instance (Table 2, instance 15)
50 10
 2  93  4  40  0   1  3  77  1  77  5  16  9  74  8  11  6  51  7  92
 0  92  4  80  1  76  3  59  2  70  5  86  9  17  6  78  7  30  8  93
 1  44  2  92  3  96  4  77  0  53  9  10  7  49  5  84  8  59  6  14
 1  60  2  19  3  76  0  73  4  85  7  13  8  93  5  68  9  50  6  78
 2  20  0  24  3  41  1   2  4   4  9  44  7  79  8  81  5  16  6  39
 3  41  2  35  1  32  4  18  0  15  8  98  6  29  5  19  7  14  9  26
 1  59  0  45  4  53  3  44  2  98  5  84  6  23  7  45  8  39  9  89
 1  30  4  51  3  25  0  51  2  84  6  60  5  45  7  89  8  25  9  97
 0  47  3  18  2  40  4  62  1  58  5  36  7  93  8  77  9  90  6  15
 3  33  1  68  0  41  4  72  2  20  6  69  7  47  5  22  9  47  8  22
 2  28  1 100  4  20  0  35  3  26  5  24  9  41  6  42  7 100  8  32
 0  65  2  12  4  53  3  93  1  40  8  18  7  23  5  60  6  89  9  53
 0  58  1  60  4  97  3  31  2  50  9  85  5  64  7  38  6  85  8  35
 3  64  0  58  1  49  2  45  4   9  8  49  6  22  5  99  9  15  7   7
 0  10  4  85  3  72  2  37  1  77  5  70  7  45  9   8  6  83  8  57
 4  93  0  87  1  87  2  18  3   4  8  78  5  67  9  20  6  17  7  35
 4  72  0  56  3  57  2  15  1  45  6  41  5  40  9  85  8  32  7  81
 0  36  3  63  4  79  2  32  1   5  6  25  7  86  9  91  5  21  8  35
 2  83  4  29  0   9  1  38  3  73  7  50  9  99  5  18  8  29  6  41
 0 100  3  29  2  60  4  63  1  64  8  71  6  35  5  26  9   9  7  22
 1  81  0  60  3  62  4  48  2  68  7  28  5  69  8  92  6  79  9  10
 0  40  4  80  1  41  2  10  3  68  8  28  9  51  7  33  6  82  5  25
```

Job Shop Scheduling Problem: Benchmark

- The 8 most common sets of JSSP benchmark instances are *abz*[1], *dmu*[57], *ft*[84], *la*[139], *orb*[6], *swv*[205], *ta*[210], and *yn*[247].





Job Shop Scheduling Problem: Baselines



- See the large study by Weise (汤卫思) [227] at https://github.com/thomasWeise/jsspInstancesAndResults for old/classical methods that are baselines on the JSSP.

Job Shop Scheduling Problem: Baselines



- See the large study by Weise (汤卫思) [227] at https://github.com/thomasWeise/jsspInstancesAndResults for old/classical methods that are baselines on the JSSP.
- Good classical algorithms to outperform are the Tabu Search (TS) algorithms by Nowicki and Smutnicki [177] and Pezzella and Merelli [185] or the local search (LS) by Balas and Vazacopoulos [11].

Job Shop Scheduling Problem: SOTA



- The SOTA on the JSSP includes...

Job Shop Scheduling Problem: SOTA

- The SOTA on the JSSP includes:
  - the Memetic Algorithms (MAs) by Henning [108]

Job Shop Scheduling Problem: SOTA



- The SOTA on the JSSP includes:
  - the Memetic Algorithms (MAs) by Henning [108],
  - the Tabu Search (TS) algorithms by Shylo and Shams [202], Peng, Lü, and Cheng [184], and Zhang, Shao, Rao, and Qiu [251]

Job Shop Scheduling Problem: SOTA



- The SOTA on the JSSP includes:
  - the Memetic Algorithms (MAs) by Henning [108],
  - the Tabu Search (TS) algorithms by Shylo and Shams [202], Peng, Lü, and Cheng [184], and Zhang, Shao, Rao, and Qiu [251],
  - the SA-related method by Pardalos, Shylo, and Vazacopoulos [181]

Job Shop Scheduling Problem: SOTA

- The SOTA on the JSSP includes:
  - the Memetic Algorithms (MAs) by Henning [108],
  - the Tabu Search (TS) algorithms by Shylo and Shams [202], Peng, Lü, and Cheng [184], and Zhang, Shao, Rao, and Qiu [251],
  - the SA-related method by Pardalos, Shylo, and Vazacopoulos [181],
  - the Biased Random Key Genetic Algorithm (BRKGA), here realized as MA combining continuous optimization, the permutation-based encoding, and TS by Gonçalves and Resende [98]

Job Shop Scheduling Problem: SOTA



- The SOTA on the JSSP includes:
  - the Memetic Algorithms (MAs) by Henning [108],
  - the Tabu Search (TS) algorithms by Shylo and Shams [202], Peng, Lü, and Cheng [184], and Zhang, Shao, Rao, and Qiu [251],
  - the SA-related method by Pardalos, Shylo, and Vazacopoulos [181],
  - the Biased Random Key Genetic Algorithm (BRKGA), here realized as MA combining continuous optimization, the permutation-based encoding, and TS by Gonçalves and Resende [98], and
  - the algorithm by Vilím, Laborie, and Shaw [224].



(Multi-)Permutation Problems

2D Bin Packing

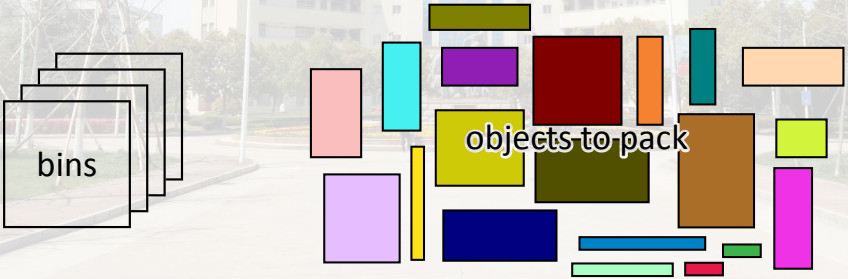
2D Bin Packing Problem: Introduction

- In a two-dimensional Bin Packing Problem (2BP), we have a set of n rectangles[114,148,150].



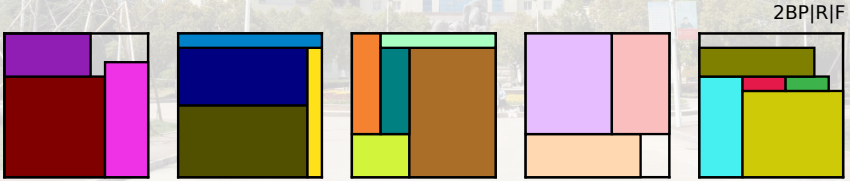
2D Bin Packing Problem: Introduction

- In a two-dimensional Bin Packing Problem (2BP), we have a set of n rectangles and we want to pack them into rectangular bins without overlap such that their edges are parallel to the bin edges, with the goal to use as few bins as possible[114,148,150].



2D Bin Packing Problem: Introduction

- In a two-dimensional Bin Packing Problem (2BP), we have a set of n rectangles and we want to pack them into rectangular bins without overlap such that their edges are parallel to the bin edges, with the goal to use as few bins as possible[114,148,150].



2D Bin Packing Problem: Introduction

- In a two-dimensional Bin Packing Problem (2BP), we have a set of n rectangles and we want to pack them into rectangular bins without overlap such that their edges are parallel to the bin edges[114,148,150].
- Typical flavors of this problem include the two-dimensional Bin Packing Problem with free cuts and rotation (2BP|R|F) and the oriented two-dimensional Bin Packing Problem with free cuts (2BP|O|F).

2D Bin Packing Problem: Introduction

- In a two-dimensional Bin Packing Problem (2BP), we have a set of n rectangles and we want to pack them into rectangular bins without overlap such that their edges are parallel to the bin edges[114,148,150].
- Typical flavors of this problem include the two-dimensional Bin Packing Problem with free cuts and rotation (2BP|R|F) and the oriented two-dimensional Bin Packing Problem with free cuts (2BP|O|F).
- The 2BP is strongly $\mathcal{NP}$-hard[150].

2D Bin Packing Problem: Introduction

- In a two-dimensional Bin Packing Problem (2BP), we have a set of n rectangles and we want to pack them into rectangular bins without overlap such that their edges are parallel to the bin edges[114,148,150].
- Typical flavors of this problem include the two-dimensional Bin Packing Problem with free cuts and rotation (2BP|R|F) and the oriented two-dimensional Bin Packing Problem with free cuts (2BP|O|F).
- The 2BP is strongly $\mathcal{NP}$-hard[150].
- The problem is closely related to the 2D-CSP, where the goal is to cut rectangles from larger rectangles...

2D Bin Packing Problem: Introduction

- In a two-dimensional Bin Packing Problem (2BP), we have a set of n rectangles and we want to pack them into rectangular bins without overlap such that their edges are parallel to the bin edges[114,148,150].
- Typical flavors of this problem include the two-dimensional Bin Packing Problem with free cuts and rotation (2BP|R|F) and the oriented two-dimensional Bin Packing Problem with free cuts (2BP|O|F).
- The 2BP is strongly $\mathcal{NP}$-hard[150].
- The problem is closely related to the 2D-CSP, where the goal is to cut rectangles from larger rectangles...
- It has many practical applications, e.g., in cutting (wood, glas)[150], packing (transportation, warehouse)[150], newspaper page layout[206], as well as in scheduling and build formation in additive manufacturing[142,187].

2D Bin Packing Problem: Possible Metaheuristic Approach

- Packings are complicated data structures: For each rectangle, we need to store the bin index and coordinate where it is stored.

2D Bin Packing Problem: Possible Metaheuristic Approach

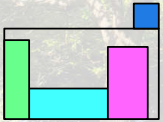
- Packings are complicated data structures: For each rectangle, we need to store the bin index and coordinate where it is stored.
- Designing search operators for these data structures is not easy, as we must not create overlaps when moving rectangles around.

2D Bin Packing Problem: Possible Metaheuristic Approach

- Packings are complicated data structures: For each rectangle, we need to store the bin index and coordinate where it is stored.
- Designing search operators for these data structures is not easy, as we must not create overlaps when moving rectangles around.
- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.

2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- The permutations are processed from beginning to end. The rectangle represented by the current number is placed on the top-right corner of the bin.



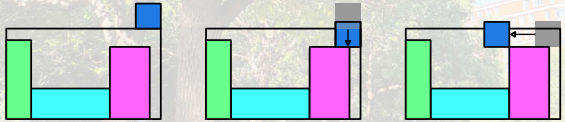
2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- As soon as the current rectangle can be moved down, it is moved down.



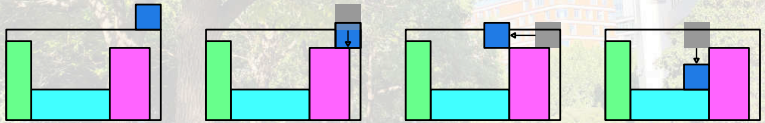
2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- Once it cannot be moved down anymore, it is moved to the left (until it could be moved down again).



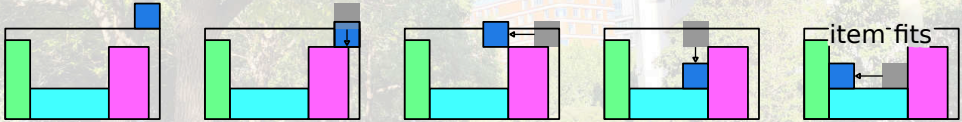
2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- The rectangle is moved down-left (with down prioritized) until it cannot be moved anymore.



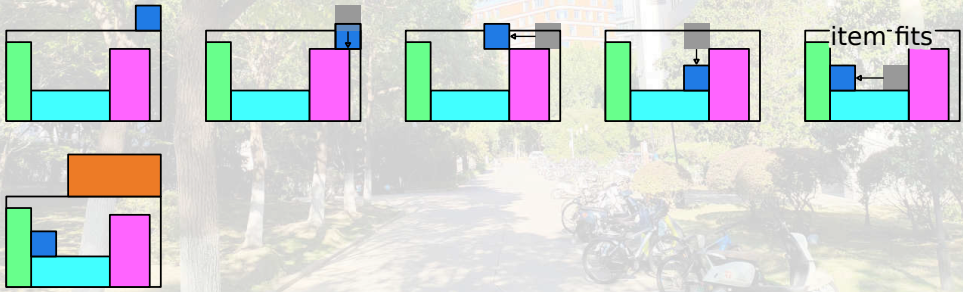
2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- Once it cannot be moved anymore, we check if it is completely inside the bin.



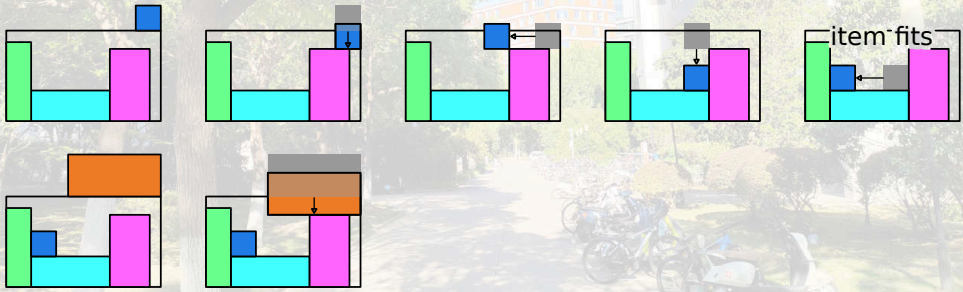
2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- Then we insert the next rectangle in the same way.



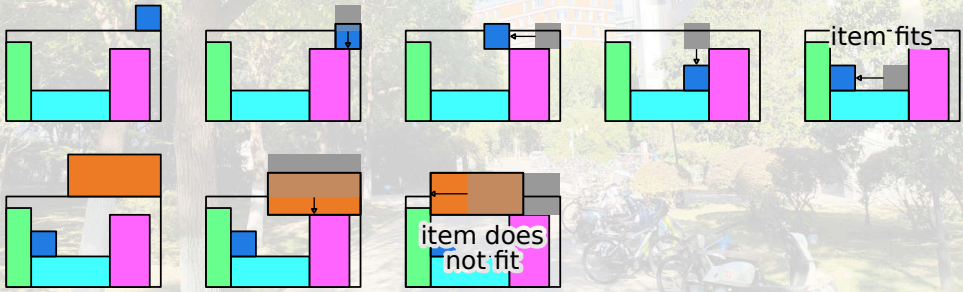
2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- Then we insert the next rectangle in the same way.



2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- Then we insert the next rectangle in the same way.



2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- If the rectangle does not fit into the bin, then we close the bin...



2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- ... and put the rectangle at the bottom-left corner of the next bin and then use this bin for all future insertions.



2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- If objects are allowed to be rotated by 90°, i.e., if we are working on 2BP|R|F, we can allow the elements in the permutation to be signed, where "$-i$" identifies rectangle i, but rotated by 90°.

2D Bin Packing Problem: Possible Metaheuristic Approach

- We can follow the Improved Bottom Left (IBL)-idea by Liu (刘德全) and Teng (滕弘飞) [147] as adopted in[252,253] that uses a permutation of the first n natural numbers to represent the order in which rectangles should be packed.
- If objects are allowed to be rotated by 90°, i.e., if we are working on 2BP|R|F, we can allow the elements in the permutation to be signed, where "$-i$" identifies rectangle i, but rotated by 90°.
- Either way: We can now use the same permutation search spaces and the operators and algorithms we already have developed for the TSP, QAP, and JSSP (maybe with that slight modification) to search for two-dimensional packings!

2D Bin Packing Problem: Benchmarks

- The standard benchmark set in this field is 2DPackLib[115,116], which can be found at https://site.unibo.it/operations-research/en/research/2dpacklib.

2D Bin Packing Problem: Benchmarks

- The standard benchmark set in this field is 2DPackLib[115,116], which can be found at https://site.unibo.it/operations-research/en/research/2dpacklib.
  - 10 instances of type *beng*[21] with 20 and 200 items

2D Bin Packing Problem: Benchmarks

- The standard benchmark set in this field is 2DPackLib[115,116], which can be found at https://site.unibo.it/operations-research/en/research/2dpacklib.
  - 10 instances of type *beng*[21] with 20 and 200 items,
  - 10 *class* instance sets[22,160] with 5 group of 10 instances each, with up to 100 items

2D Bin Packing Problem: Benchmarks

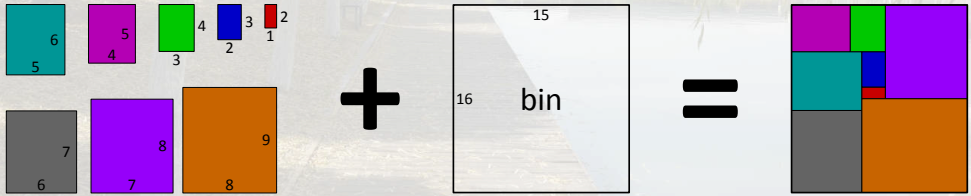
- The standard benchmark set in this field is 2DPackLib[115,116], which can be found at https://site.unibo.it/operations-research/en/research/2dpacklib.
  - 10 instances of type *beng*[21] with 20 and 200 items,
  - 10 *class* instance sets[22,160] with 5 group of 10 instances each, with up to 100 items, and
  - the *A* instance set[157] offers 43 instances with 13 to 809 items.

2D Bin Packing Problem: Benchmarks

- The standard benchmark set in this field is 2DPackLib[115,116], which can be found at https://site.unibo.it/operations-research/en/research/2dpacklib.
  - 10 instances of type *beng*[21] with 20 and 200 items,
  - 10 *class* instance sets[22,160] with 5 group of 10 instances each, with up to 100 items, and
  - the *A* instance set[157] offers 43 instances with 13 to 809 items.
- The almost-squares-in-almost-squares (Asqas) problem[220] asks us to put objects of sizes 1×2, 2×3, $\ldots$, $n \times (n+1)$ into a similarly almost-square bin without waste.

2D Bin Packing Problem: Benchmarks

- The standard benchmark set in this field is 2DPackLib[115,116], which can be found at https://site.unibo.it/operations-research/en/research/2dpacklib.
  - 10 instances of type *beng*[21] with 20 and 200 items,
  - 10 *class* instance sets[22,160] with 5 group of 10 instances each, with up to 100 items, and
  - the *A* instance set[157] offers 43 instances with 13 to 809 items.
- The almost-squares-in-almost-squares (Asqas) problem[220] asks us to put objects of sizes 1×2, 2×3, $\ldots$, $n \times (n+1)$ into a similarly almost-square bin without waste. It has only 4 interesting instances, with bin sizes 5×4 for $n = 3$, 15×16 for $n = 8$, 55×56 for $n = 20$, and 119×120 for $n = 34$.



2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F.

2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F:
  - the BRKGA by Gonçalves and Resende [97]

2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F:
  - the BRKGA by Gonçalves and Resende [97],
  - the hyper-heuristic by Beyaz, Dökeroğlu, and Cosar [25]

2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F:
  - the BRKGA by Gonçalves and Resende [97],
  - the hyper-heuristic by Beyaz, Dökeroğlu, and Cosar [25],
  - the multi-crossover EA by Lee [141]

2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F:
  - the BRKGA by Gonçalves and Resende [97],
  - the hyper-heuristic by Beyaz, Dökeroğlu, and Cosar [25],
  - the multi-crossover EA by Lee [141],
  - the EA by Blum and Schmid [30]

2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F:
  - the BRKGA by Gonçalves and Resende [97],
  - the hyper-heuristic by Beyaz, Dökeroğlu, and Cosar [25],
  - the multi-crossover EA by Lee [141],
  - the EA by Blum and Schmid [30], and
  - the hybrid GRASP/VND algorithm by Parreño, Alvarez-Valdés, Oliveira, and Tamarit [182].

2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F:
  - the BRKGA by Gonçalves and Resende [97],
  - the hyper-heuristic by Beyaz, Dökeroğlu, and Cosar [25],
  - the multi-crossover EA by Lee [141],
  - the EA by Blum and Schmid [30], and
  - the hybrid GRASP/VND algorithm by Parreño, Alvarez-Valdés, Oliveira, and Tamarit [182].
  - For small instances, the exact method by Cid-Garcia and Rios-Solis [47].

2D Bin Packing Problem: SOTA

- Let's look at the SOTA on the 2BP|R|F and the 2BP|O|F:
  - the BRKGA by Gonçalves and Resende [97],
  - the hyper-heuristic by Beyaz, Dökeroğlu, and Cosar [25],
  - the multi-crossover EA by Lee [141],
  - the EA by Blum and Schmid [30], and
  - the hybrid GRASP/VND algorithm by Parreño, Alvarez-Valdés, Oliveira, and Tamarit [182].
  - For small instances, the exact method by Cid-Garcia and Rios-Solis [47].
  - Actually, the simple RLS by Zhao (赵睿), Wu (吴志泽), van den Berg, Thürer, Liang (梁天宇), Tan (檀明), and Weise (汤卫思) [252, 253] outperforms these methods on the 2BP|O|F and is quite competitive on the 2BP|R|F.

Summary

Summary

- Today, I gave you a brief introduction into optimization.

Summary



- Today, I gave you a brief introduction into optimization.
- You learned roughly what it is and saw some basic metaheuristic algorithms.

Summary



- Today, I gave you a brief introduction into optimization.
- You learned roughly what it is and saw some basic metaheuristic algorithms.
- Then we looked at different examples for optimization problems.

Summary



- Today, I gave you a brief introduction into optimization.
- You learned roughly what it is and saw some basic metaheuristic algorithms.
- Then we looked at different examples for optimization problems.
- What did we learn?

Algorithm Families



- Among the SOTA, we can find many SLS methods.

Algorithm Families



- Among the SOTA, we can find many SLS methods.
- Tabu Search (TS) is quite successful, Simulated Annealing (SA) is less often among the SOTA.

Algorithm Families



- Among the SOTA, we can find many SLS methods.
- Tabu Search (TS) is quite successful, Simulated Annealing (SA) is less often among the SOTA.
- Pure population-based methods like EAs are rarely SOTA.

Algorithm Families



- Among the SOTA, we can find many SLS methods.
- Tabu Search (TS) is quite successful, Simulated Annealing (SA) is less often among the SOTA.
- Pure population-based methods like EAs are rarely SOTA.
- However, if hybridized with LS, i.e., in the form of MAs, they rock.

Encodings

- Surprisingly many problems where the solutions are complex data structures can still be solved using simple metaheuristics.

Encodings



- Surprisingly many problems where the solutions are complex data structures can still be solved using simple metaheuristics.
- Here, we often use a simple search space $\mathbb{X}$ and map it to the complex solution space $\mathbb{Y}$ using a decoding function $\gamma : \mathbb{X} \mapsto \mathbb{Y}$.

Encodings



- Surprisingly many problems where the solutions are complex data structures can still be solved using simple metaheuristics.
- Here, we often use a simple search space $\mathbb{X}$ and map it to the complex solution space $\mathbb{Y}$ using a decoding function $\gamma : \mathbb{X} \mapsto \mathbb{Y}$.
- Typical examples were the JSSP and the 2BP, where permutations are translated to Gantt charts and packings, respectively.

Generalized Problems



- Many of the problems introduced here were presented in a "pure" form, but in the real world, they are often more complicated.

Generalized Problems

- Many of the problems introduced here were presented in a "pure" form, but in the real world, they are often more complicated.
- In a real-world JSSP, there may be transportation times for material between machines, which may even be job-specific.

Generalized Problems



- Many of the problems introduced here were presented in a "pure" form, but in the real world, they are often more complicated.
- In a real-world JSSP, there may be transportation times for material between machines, which may even be job-specific.
- In a real-world MaxSAT, the clauses may have different weights, i.e., some may be more important than others.

Generalized Problems



- Many of the problems introduced here were presented in a "pure" form, but in the real world, they are often more complicated.
- In a real-world JSSP, there may be transportation times for material between machines, which may even be job-specific.
- In a real-world MaxSAT, the clauses may have different weights, i.e., some may be more important than others.
- And so on.

Generalized Problems



- Many of the problems introduced here were presented in a "pure" form, but in the real world, they are often more complicated.
- In a real-world JSSP, there may be transportation times for material between machines, which may even be job-specific.
- In a real-world MaxSAT, the clauses may have different weights, i.e., some may be more important than others.
- And so on.
- Such special problems can often still be tackled with metaheuristic, which often are the best methods for them.

Generalized Problems



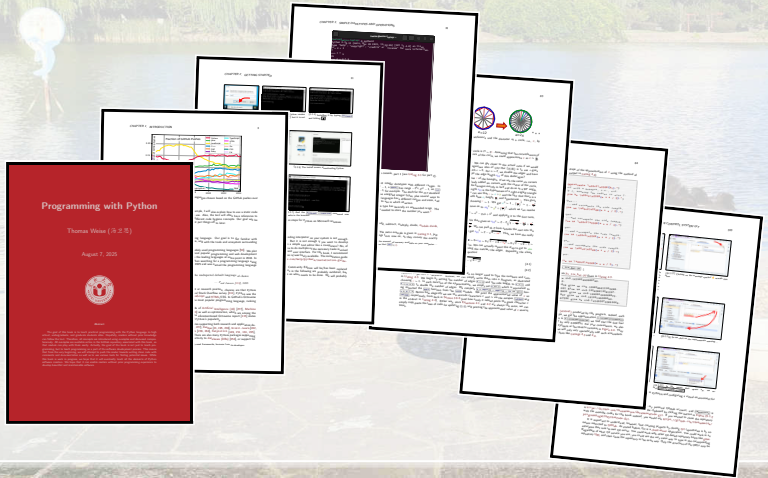
- Many of the problems introduced here were presented in a "pure" form, but in the real world, they are often more complicated.
- In a real-world JSSP, there may be transportation times for material between machines, which may even be job-specific.
- In a real-world MaxSAT, the clauses may have different weights, i.e., some may be more important than others.
- And so on.
- Such special problems can often still be tackled with metaheuristic, which often are the best methods for them.
- If you understand these concepts, then you can tackle basically *any* problem occuring in management, logistics, warehousing, planning, scheduling, packing, or even engineering.

Advertisement



Programming with Python

We have a freely available course book on *Programming with Python* at `https://thomasweise.github.io/programmingWithPython`, with focus on practical software development using the Python ecosystem of tools[229].



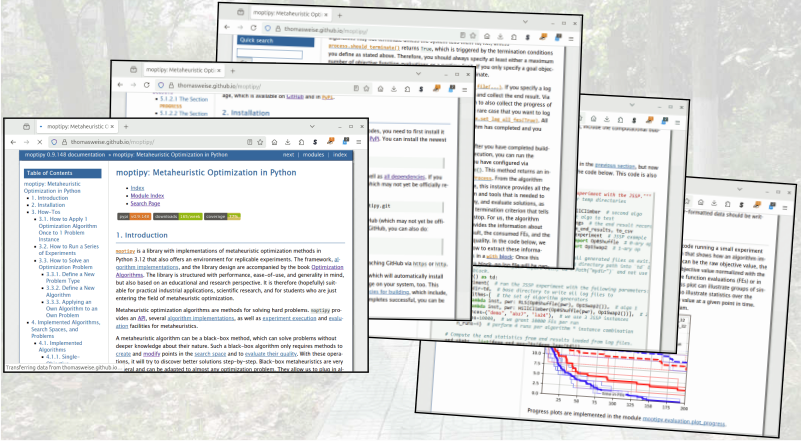
Databases

We have a freely available course book on *Databases* at `https://thomasweise.github.io/databases`, with actual practical examples using a real database management system (DBMS)[225].



Metaheuristic Optimization in Python: moptipy

We offer moptipy[237], a mature open source Python package for metaheuristic optimization, which implements several algorithms, can run self-documenting experiments in parallel and in a distributed fashion, and offers statistical evaluation tools.



谢谢您们！

Thank you!

Vielen Dank!



References I

[1] Joseph Adams, Egon Balas, and Daniel Zawack. "The Shifting Bottleneck Procedure for Job Shop Scheduling". *Management Science* 34(3):391–401, Mar. 1988. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.34.3.391 (cit. on pp. 386–392, 504).

[2] Hernán E. Aguirre and Keiki Takadama, eds. *20th Genetic and Evolutionary Computation Conference (GECCO'2018)*. July 15–19, 2018, Kyōto, Kyōto-fu, Japan. New York, NY, USA: Association for Computing Machinery (ACM), 2018. ISBN: 978-1-4503-5618-3. doi:10.1145/3205455.

[3] Miguel F. Anjos. *The QAPLIB Challenge*. Edinburgh, Scotland, UK: Miguel F. Anjos @ University of Edinburgh, Nov. 19, 2023. URL: https://www.miguelanjos.com/projects/2999 (visited on 2026-09-28). The document lists the 28 instances of QAPLIB [35,37] that had not yet been solved to optimality by the time of its writing. (Cit. on pp. 325, 508).

[4] Denis Antipov and Benjamin Doerr. "Precise Runtime Analysis for Plateaus". In: *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Vol. 2. Sept. 8–12, 2018, Coimbra, Portugal. Ed. by Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and L. Darrell Whitley. Vol. 11102 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018, pp. 117–128. ISSN: 0302-9743. ISBN: 978-3-319-99258-7. doi:10.1007/978-3-319-99259-4_10 (cit. on pp. 226–237, 507).

[5] David Lee Applegate, Robert E. Bixby, Vašek Chvátal, and William John Cook. *The Traveling Salesman Problem: A Computational Study*. 2nd ed. Vol. 17 of Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 978-0-691-12993-8. This book also discusses the Concorde TSP solver. (Cit. on pp. 128, 299–304, 311–316, 500, 510).

[6] David Lee Applegate and William John Cook. "A Computational Study of the Job-Shop Scheduling Problem". *ORSA Journal on Computing* 3(2):149–156, May 1991. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.3.2.149. The JSSP instances used were generated in Bonn in 1986. (Cit. on pp. 386–392, 504).

[7] Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and L. Darrell Whitley, eds. *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Vol. 2. Sept. 8–12, 2018, Coimbra, Portugal. Vol. 11102 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018. ISSN: 0302-9743. ISBN: 978-3-319-99258-7. doi:10.1007/978-3-319-99259-4.

References II

[8] Paul Gustav Heinrich Bachmann. *Die Analytische Zahlentheorie / Dargestellt von Paul Bachmann*. Vol. Zweiter Theil of Zahlentheorie: Versuch einer Gesamtdarstellung dieser Wissenschaft in ihren Haupttheilen. Leipzig, Sachsen, Germany: B. G. Teubner, 1894. ISBN: 978-1-4181-6963-3. URL: http://gallica.bnf.fr/ark:/12148/bpt6k994750 (visited on 2023-12-13) (cit. on p. 514).

[9] Thomas Bäck, David B. Fogel, and Zbigniew "Zbyszek" Michalewicz, eds. *Handbook of Evolutionary Computation*. Bristol, England, UK: IOP Publishing Ltd and Oxford, Oxfordshire, England, UK: Oxford University Press, 1997. ISBN: 978-0-7503-0392-7 (cit. on pp. 238–242, 501, 509).

[10] Egon Balas and Alkis Vazacopoulos. *Guided Local Search with Shifting Bottleneck for Job Shop Scheduling*. Management Science Research Report MSSR–609. Pittsburgh, PA, USA: Graduate School of Industrial Administration (GSIA), Carnegie-Mellon University, Oct. 1994. Revised 11 1995, see also [11]. (Cit. on p. 462).

[11] Egon Balas and Alkis Vazacopoulos. "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling". *Management Science* 44(2):262–275, Feb. 1998. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.44.2.262. See also [10]. (Cit. on pp. 393, 394, 462).

[12] Roberto Battiti and Giampietro Tecchiolli. "Local Search with Memory: Benchmarking RTS". *OR Spectrum: Quantitative Approaches in Management. Formerly Operations-Research-Spektrum* 17:67–86, June 1995. London, England, UK: Springer Nature Limited. ISSN: 0171-6468. doi:10.1007/BF01719249 (cit. on p. 508).

[13] Roberto Battiti and Giampietro Tecchiolli. "Simulated Annealing and Tabu Search in the Long Run: A Comparison on QAP Tasks". *Computers & Mathematics with Applications* 28(6):1–8, Sept. 1994. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0898-1221. doi:10.1016/0898-1221(94)00147-2 (cit. on pp. 326–330, 508, 510).

[14] Roberto Battiti and Giampietro Tecchiolli. "The Reactive Tabu Search". 6(2):126–140, May 1994. doi:10.1287/ijoc.6.2.126. Applications of RTS to the 0-1 Knapsack and QAP. (Cit. on pp. 326–330, 508, 510).

[15] Roberto Battiti and Giampietro Tecchiolli. "Training Neural Nets with the Reactive Tabu Search". *IEEE Transactions on Neural Networks* 6(5):1185–1200, Sept. 1995. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1045-9227. doi:10.1109/72.410361. See also "Training Neural Nets with the Reactive Tabu Search, Technical Report UTM 421, Trento, Alto Adige/Südtirol, Italy: University of Trento, 11 1993" (cit. on p. 508).

References III

[16] James C. Bean. "Genetic Algorithms and Random Keys for Sequencing and Optimization". *ORSA Journal on Computing* 6(2):107–220, Spr. 1994. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.6.2.154. Applies the Random Keys encoding to scheduling and the QAP. See [17]. (Cit. on pp. 463, 500, 508).

[17] James C. Bean. *Genetics and Random Keys for Sequencing and Optimization*. Tech. rep. 92-43. Ann Arbor, MI, USA: University of Michigan, Department of Industrial & Operations Engineering, June 30, 1992. URL: https://www.math.wm.edu/~rrkinc/homework/RdmKeyGA.pdf (visited on 2026-10-01). See [16]. (Cit. on pp. 463, 500, 508).

[18] John Edward Beasley. "OR-Library: Distributing Test Problems by Electronic Mail". *The Journal of the Operational Research Society (JORS)* 41:1069–1072, Nov. 1990. London, England, UK: Taylor and Francis Ltd. and London, England, UK: Springer Nature Limited. ISSN: 0160-5682. doi:10.1057/jors.1990.166. See the website [19] (cit. on pp. 386–389, 463, 504, 507).

[19] John Edward Beasley. *Welcome to OR-Library*. Uxbridge, Middlesex, England, UK: Brunel University of London, June 1990–Feb. 2018. URL: https://people.brunel.ac.uk/~mastjjb/jeb/info.html (visited on 2026-09-30). See the article [18]. Contains instances of the JSSP, 1D- and 2D-Packing, Knapsack, periodic TSPs, amongst others. (Cit. on pp. 386–389, 463, 504, 507).

[20] Andreas Beham, Michael Affenzeller, and Erik Pitzer. "Metaheuristic Algorithms for the Quadratic Assignment Problem: Performance and Comparison". In: *Innovative Technologies in Management and Science*. Ed. by Ryszard Klempous and Jan Nikodem. Vol. 10 of Topics in Intelligent Engineering and Informatics (TIEI). Cham, Switzerland: Springer, 2015, pp. 171–190. ISSN: 2193-9411. ISBN: 978-3-319-12651-7. doi:10.1007/978-3-319-12652-4_10. Algorithms like SLS/ILS, SA, TS, Variable Neighborhood Search (VNS), EAs, MAs are compared and, with the exception of SA and EA, all perform best on some instance groups of QAPLIB. (Cit. on pp. 318–324, 331–334, 507).

[21] Bengt-Erik Bengtsson. "Packing Rectangular Pieces— A Heuristic Approach". *The Computer Journal* 25(3):353–357, Aug. 1982. Oxford, Oxfordshire, England, UK: Oxford University Press. ISSN: 0010-4620. doi:10.1093/comjnl/25.3.353 (cit. on pp. 424–429, 498, 499).

[22] Judith Osterhoudt Berkey and Pearl Y. Wang. "Two-Dimensional Finite Bin-Packing Algorithms". *The Journal of the Operational Research Society (JORS)* 38(5):423–429, May 1987. London, England, UK: Taylor and Francis Ltd. and London, England, UK: Springer Nature Limited. ISSN: 0160-5682. doi:10.1057/jors.1987.70 (cit. on pp. 424–429, 499).

References IV

[23] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) and Newsgroups: alt.hypertext, Aug. 6, 1991. URL: https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt (visited on 2025-02-05) (cit. on p. 512).

[24] Tim Berners-Lee, Larry Masinter, and Mark P. McCahill. *Uniform Resource Locators (URL)*. Request for Comments (RFC) 1738. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Dec. 1994. URL: https://www.ietf.org/rfc/rfc1738.txt (visited on 2025-02-05) (cit. on p. 511).

[25] Muhammed Beyaz, Tansel Dökeroğlu, and Ahmet Cosar. "Robust Hyper-Heuristic Algorithms for the Offline Oriented/Non-Oriented 2D Bin Packing Problems". *Applied Soft Computing (ASOC)* 36:236–245, Nov. 2015. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 1568-4946. doi:10.1016/j.asoc.2015.06.063 (cit. on pp. 430–437, 498).

[26] Christian Bierwirth. "A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms". *OR Spectrum: Quantitative Approaches in Management. Formerly Operations-Research-Spektrum* 17(2–3):87–92, June 1995. London, England, UK: Springer Nature Limited. ISSN: 0171-6468. doi:10.1007/BF01719250. URL: https://www.researchgate.net/publication/240263036 (visited on 2026-09-30). Proposes a permutations-with-repetitions based encoding for the JSSP, see also [27]. (Cit. on pp. 351–360, 464, 474, 504, 513).

[27] Christian Bierwirth, Dirk C. Mattfeld, and Herbert Kopfer. "On Permutation Representations for Scheduling Problems". In: *4th International Conference on Parallel Problem Solving from Nature (PPSN IV)*. Sept. 22–24, 1996, Berlin, Germany. Ed. by Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel. Vol. 1141 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1996, pp. 310–318. ISSN: 0302-9743. ISBN: 978-3-540-61723-5. doi:10.1007/3-540-61723-X_995. URL: https://www.researchgate.net/publication/2753293 (visited on 2026-09-30). Proposes a permutations-with-repetitions based encoding for the JSSP, see also [26]. (Cit. on pp. 351–360, 464, 474, 504, 513).

[28] Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. "The Job Shop Scheduling Problem: Conventional and New Solution Techniques". *European Journal of Operational Research* 93(1):1–33, Aug. 1996. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(95)00362-2 (cit. on pp. 336–360, 504).

[29] Joshua Bloch. *Effective Java*. Reading, MA, USA: Addison-Wesley Professional, May 2008. ISBN: 978-0-321-35668-0 (cit. on p. 503).

References V

[30] Christian Blum and Verena Schmid. "Solving the 2D Bin Packing Problem by Means of a Hybrid Evolutionary Algorithm". In: *International Conference on Computational Science (ICCS'2013)*. June 5–7, 2013, Barcelona, Spain. Ed. by Vassil Alexandrov, Michael Lees, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot. Vol. 18 of Procedia Computer Science. Amsterdam, The Netherlands: Elsevier B.V., 2013, pp. 899–908. ISSN: 1877-0509. doi:10.1016/j.procs.2013.05.255. An EA for the 2BP|O|F using the heuristic by Wong and Lee [245]. (Cit. on pp. 430–437, 498).

[31] Borko Bošković and Janez Brest. "Computational Search of Long Skew-Symmetric Binary Sequences with High Merit Factors". *MENDEL – Soft Computing Journal* 28(2):17–24, Dec. 2022. Brno, Czech Republic: Brno University of Technology (Vysoké učení technické v Brně). ISSN: 1803-3814. doi:10.13164/mendel.2022.2.017 (cit. on pp. 282–286, 504).

[32] Borko Bošković, Franc Brglez, and Janez Brest. *A GitHub Archive for Solvers and Solutions of the* labs *Problem*. San Francisco, CA, USA: GitHub Inc, Jan. 2016. URL: https://github.com/borkob/git_labs (visited on 2026-09-20) (cit. on pp. 278–281).

[33] Borko Bošković, Franc Brglez, and Janez Brest. "Low-Autocorrelation Binary Sequences: On Improved Merit Factors and Runtime Predictions to Achieve Them". *Applied Soft Computing (ASOC)* 56:262–285, July 2017. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 1568-4946. doi:10.1016/J.ASOC.2017.02.024 (cit. on pp. 271–275, 504).

[34] Janez Brest and Borko Bošković. "A Heuristic Algorithm for a Low Autocorrelation Binary Sequence Problem with Odd Length and High Merit Factor". *IEEE Access* 6:4127–4134, Jan. 2018. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 2169-3536. doi:10.1109/ACCESS.2018.2789916 (cit. on pp. 282–286, 504).

[35] Rainer E. Burkard, Eranda Çela, Stefan E. Karisch, Franz Rendl, Miguel F. Anjos, and Peter M. Hahn. *QAPLIB – A Quadratic Assignment Problem Library – Problem Instances and Solutions*. Edinburgh, Scotland, UK: University of Edinburgh, Apr. 20, 2012–Mar. 31, 2022. doi:10.7488/ds/3428. This is the 2012 status of the QAPLIB dataset from [37] (cit. on pp. 325, 461, 466, 508).

[36] Rainer E. Burkard, Eranda Çela, Panos Miltiades Pardalos, and Leonidas S. Pitsoulis. "The Quadratic Assignment Problem". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1713–1809. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_27 (cit. on pp. 318–324, 507).

References VI

[37] Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. "QAPLIB – A Quadratic Assignment Problem Library". *Journal of Global Optimization* 10(4):391–403, June 1997. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1023/A:1008293323270. This article introduces QAPLIB, a set of benchmark problems for the QAP, now hosted at [35]. (Cit. on pp. 325, 461, 465, 508).

[38] Rainer E. Burkard and Josef Offermann. "Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme". *Zeitschrift für Operations Research (ZOR). now Mathematical Methods of Operations Research (MMOR)* 21:B121–B132, Aug. 1977. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag; now London, England, UK: Springer Nature Limited. ISSN: 0340-9422. doi:10.1007/BF01918175. Keyboard layout modeled as QAPs. (Cit. on pp. 318–324, 507).

[39] Eduardo Carvalho Pinto and Carola Doerr. "A Simple Proof for the Usefulness of Crossover in Black-Box Optimization". In: *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Vol. 2. Sept. 8–12, 2018, Coimbra, Portugal. Ed. by Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and L. Darrell Whitley. Vol. 11102 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018, pp. 29–41. ISSN: 0302-9743. ISBN: 978-3-319-99258-7. doi:10.1007/978-3-319-99259-4_3 (cit. on pp. 243–246, 502).

[40] Eduardo Carvalho Pinto and Carola Doerr. *Towards a More Practice-Aware Runtime Analysis of Evolutionary Algorithms*. arXiv.org: Computing Research Repository (CoRR) abs/1812.00493. Ithaca, NY, USA: Cornell Universiy Library, Dec. 3, 2018. doi:10.48550/arXiv.1812.00493. URL: https://arxiv.org/abs/1812.00493 (visited on 2025-08-08). arXiv:1812.00493v1 [cs.NE] 3 Dec 2018 (cit. on pp. 191–201, 243–246, 497, 502, 509).

[41] Vladimír Černý. "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm". *Journal of Optimization Theory and Applications* 45(1):41–51, Jan. 1985. New York, NY, USA: Springer Science+Business Media, LLC. ISSN: 0022-3239. doi:10.1007/BF00940812 (cit. on p. 508).

[42] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. "A Review of Machine Scheduling: Complexity, Algorithms and Approximability". In: *Handbook of Combinatorial Optimization*. Ed. by Panos Miltiades Pardalos, Ding-Zhu Du, and Ronald Lewis Graham. 1st ed. Boston, MA, USA: Springer, 1998, pp. 1493–1641. ISBN: 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_25. See also pages 21–169 in volume 3/3 by Norwell, MA, USA: Kluwer Academic Publishers. Discusses, amongst others, that the JSSP is $\mathcal{NP}$-hard. (Cit. on pp. 336–350, 504, 514).

References VII

[43] Jiayang Chen (陈嘉阳), Zhize Wu (吴志泽), Sarah Louise Thomson, and Thomas Weise (汤卫思). "Frequency Fitness Assignment: Optimization Without Bias for Good Solution Outperforms Randomized Local Search on the Quadratic Assignment Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 27–37. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888600003837 (cit. on pp. 318–324, 501, 506, 507).

[44] T. C. Edwin Cheng, Bo Peng, and Zhipeng Lü. "A Hybrid Evolutionary Algorithm to Solve the Job Shop Scheduling Problem". *Annals of Operations Research* 242(2):223–237, July 2016. London, England, UK: Springer Nature Limited. ISSN: 0254-5330. doi:10.1007/s10479-013-1332-5. A MA utlizing TS (cit. on p. 505).

[45] Philippe Chrétienne, Edward G. Coffman, Jan Karel Lenstra, and Zhen Liu, eds. *Scheduling Theory and Its Applications*. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., Sept. 1995. ISBN: 978-0-471-94059-3 (cit. on pp. 336–350, 504).

[46] Yi Chu, Shaowei Cai, and Chuan Luo. "NuWLS: Improving Local Search for (Weighted) Partial MaxSAT by New Weighting Techniques". In: *37th AAAI Conference on Artificial Intelligence (AAAI'2023)*. Feb. 7–14, 2023, Washington, D.C., USA. Ed. by Brian Williams, Yiling Chen, and Jennifer Nevi. Washington, D.C., USA: AAAI Press, 2023, pp. 3915–3923. ISSN: 2159-5399. ISBN: 978-1-57735-880-0. doi:10.1609/AAAI.V37I4.25505. Introduces NuWLS. (Cit. on pp. 267–269, 506).

[47] Nestor M. Cid-Garcia and Yasmin A. Rios-Solis. "Positions and Covering: A Two-Stage Methodology to Obtain Optimal Solutions for the 2D-Bin Packing Problem". *PLoS ONE* 15(4):e0229358, Apr. 2020. San Francisco, CA, USA: Public Library of Science (PLOS). ISSN: 1932-6203. doi:10.1371/journal.pone.0229358. An exact method for the 2BP|R|F and the 2BP|O|F. (Cit. on pp. 430–437, 498).

[48] Stephen Arthur Cook. "The Complexity of Theorem-Proving Procedures". In: *Third Annual ACM Symposium on Theory of Computing (STOC'1971)*. May 3–5, 1971, Shaker Heights, OH, USA. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. New York, NY, USA: Association for Computing Machinery (ACM), 1971, pp. 151–158. ISBN: 978-1-4503-7464-4. doi:10.1145/800157.805047 (cit. on pp. 248–256, 505, 514).

[49] William John Cook. *Concorde TSP Solver*. Waterloo, ON, Canada: University of Waterloo, 1997. URL: https://math.uwaterloo.ca/tsp/concorde (visited on 2026-09-26) (cit. on pp. 311–316, 500).

References VIII

[50] William John Cook. *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton, NJ, USA: Princeton University Press, 2012–Nov. 9, 2014. ISBN: 978-0-691-16352-9 (cit. on pp. 311–316, 500, 505).

[51] William John Cook. *World TSP*. Waterloo, ON, Canada: University of Waterloo, 2013–Oct. 25, 2025. URL: http://www.math.uwaterloo.ca/tsp/world (visited on 2026-01-03) (cit. on pp. 305–309, 505, 510).

[52] Cemre Çubukçuoğlu, Pirouz Nourian, M. Fatih Tasgetiren, I. Sevil Sariyildiz, and Shervin Azadi. "Hospital Layout Design Renovation as a Quadratic Assignment Problem with Geodesic Distances". *Journal of Building Engineering* 44:102952, Dec. 2021. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 2352-7102. doi:10.1016/j.jobe.2021.102952 (cit. on pp. 318–324, 507).

[53] Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*. Vol. 4 of Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, 2006. ISBN: 978-0-262-04194-2. URL: https://www.researchgate.net/publication/220740669 (visited on 2025-08-08) (cit. on pp. 501, 509).

[54] *Definition of Operations Research*. University of Western Ontario, London, ON, Canada: International Federation of Operational Research Societies (IFORS), 2020. URL: https://www.ifors.org/what-is-or (visited on 2026-01-01) (cit. on p. 507).

[55] Paul Deitel, Harvey Deitel, and Abbey Deitel. *Internet & World Wide Web: How to Program*. 5th ed. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (cit. on p. 512).

[56] Maxence Delorme, Manuel Iori, and Silvano Martello. "Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms". *European Journal of Operational Research* 255(1):1–20, Nov. 2016. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2016.04.030 (cit. on p. 500).

[57] Ebru Demirkol, Sanjay V. Mehta, and Reha Uzsoy. "Benchmarks for Shop Scheduling Problems". *European Journal of Operational Research* 109(1):137–141, Aug. 1998. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(97)00019-2 (cit. on pp. 386–392, 504).

[58] Miroslav Dimitrov. *New Classes of Binary Sequences with High Merit Factor*. arXiv.org: Computing Research Repository (CoRR). Ithaca, NY, USA: Cornell Universiy Library, June 24, 2022. doi:10.48550/arXiv.2206.12070. URL: https://arxiv.org/pdf/2206.12070 (visited on 2026-09-21). arXiv:2206.12070v1 [cs.IT] 24 Jun 2022 (cit. on pp. 282–286, 504).

References IX

[59] Miroslav Dimitrov, Tsonka Baitcheva, and Nikolay Nikolov. "Efficient Generation of Low Autocorrelation Binary Sequences". *IEEE Signal Processing Letters* 27:341–345, 2020. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1558-2361. doi:10.1109/LSP.2020.2972127 (cit. on pp. 282–286, 504).

[60] Slobodan Dmitrović. *Modern C for Absolute Beginners: A Friendly Introduction to the C Programming Language*. New York, NY, USA: Apress Media, LLC, Mar. 2024. ISBN: 979-8-8688-0224-9 (cit. on p. 500).

[61] Benjamin Doerr and Carola Doerr. "Optimal Static and Self-Adjusting Parameter Choices for the $(1 + (\lambda, \lambda))$ Genetic Algorithm". *Algorithmica* 80(5):1658–1709, May 2018. London, England, UK: Springer Nature Limited. ISSN: 0178-4617. doi:10.1007/s00453-017-0354-9 (cit. on pp. 243–246, 509).

[62] Benjamin Doerr, Carola Doerr, and Franziska Ebel. "From Black-Box Complexity to Designing New Genetic Algorithms". *Theoretical Computer Science* 567:87–104, Feb. 2015. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/J.TCS.2014.11.028 (cit. on p. 506).

[63] Benjamin Doerr, Frank Neumann, and Andrew M. Sutton. "Improved Runtime Bounds for the (1+1) EA on Random 3-CNF Formulas Based on Fitness-Distance Correlation". In: *17th Genetic and Evolutionary Computation Conference (GECCO'2015)*. July 11–15, 2015, Madrid, Spain. Ed. by Sara Silva and Anna Isabel Esparcia-Alcázar. New York, NY, USA: Association for Computing Machinery (ACM), 2015, pp. 1415–1422. ISBN: 978-1-4503-3472-3. doi:10.1145/2739480.2754659 (cit. on pp. 257–259).

[64] Benjamin Doerr, Frank Neumann, and Andrew M. Sutton. "Time Complexity Analysis of Evolutionary Algorithms on Random Satisfiable k-CNF Formulas". *Algorithmica* 78(2):561–586, 2017. London, England, UK: Springer Nature Limited. ISSN: 0178-4617. doi:10.1007/s00453-016-0190-3 (cit. on pp. 257–259).

[65] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, and Ofer M. ShirThomas Bäck. "Benchmarking Discrete Optimization Heuristics with IOHprofiler". In: *21st Genetic and Evolutionary Computation Conference Companion (GECCO'2019)*. July 13–17, 2019, Prague, Czech Republic. Ed. by Manuel López-Ibáñez, Anne Auger, and Thomas Stützle. New York, NY, USA: Association for Computing Machinery (ACM), 2019, pp. 1798–1806. ISBN: 978-1-4503-6748-6. doi:10.1145/3319619.3326810. See also [66,67] (cit. on pp. 469, 470).

[66] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, and Ofer M. ShirThomas Bäck. "Benchmarking Discrete Optimization Heuristics with IOHprofiler". *Applied Soft Computing (ASOC)* 88:106027, Mar. 2020. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 1568-4946. doi:10.1016/j.asoc.2019.106027. See also [65,67] (cit. on pp. 221–237, 469, 470, 497, 503, 506).

References X

[67] Carola Doerr, Furong Ye, Naama Horesh, Hao Wang, and Ofer M. ShirThomas Bäck. "Benchmarking Discrete Optimization Heuristics with IOHprofiler". (abs/1912.09237), Jan. 31, 2021. doi:10.48550/arXiv.1912.09237. URL: https://arxiv.org/abs/1912.09237 (visited on 2024-06-27). arXiv:1912.09237v2 [cs.NE] 31 Jan 2021, see also [65,66] (cit. on p. 469).

[68] Marco Dorigo, Mauro Birattari, and Thomas Stützle. "Ant Colony Optimization". *IEEE Computational Intelligence Magazine (CIM)* 1(4):28–39, Nov. 2006. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1556-603X. doi:10.1109/MCI.2006.329691. URL: https://www.researchgate.net/publication/308953674 (visited on 2025-07-25) (cit. on p. 499).

[69] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. "Ant System: Optimization by a Colony of Cooperating Agents". *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics* 26(1):29–41, Feb. 1996. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1083-4419. doi:10.1109/3477.484436. URL: https://www.researchgate.net/publication/5589170 (visited on 2025-07-26) (cit. on p. 499).

[70] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004. ISBN: 978-0-262-04219-2 (cit. on p. 499).

[71] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Éric D. Taillard. *Metaheuristics for Hard Optimization – Methods and Case Studies*. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2006. ISBN: 978-3-540-23022-9. doi:10.1007/3-540-30966-7. Discusses SA, TS, EAs, and Ant Colony Optimization (ACO). (Cit. on pp. 318–324, 507).

[72] Stefan Droste, Thomas Jansen, and Ingo Wegener. "On the Analysis of the $(1+1)$ Evolutionary Algorithm". *Theoretical Computer Science* 276(1-2):51–81, Apr. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/S0304-3975(01)00182-7 (cit. on pp. 191–201, 226–237, 497, 504, 505, 509).

[73] Stefan Droste, Thomas Jansen, and Ingo Wegener. "Upper and Lower Bounds for Randomized Search Heuristics in Black-Box Optimization". *Theoretical Computer Science* 39(4):525–544, July 2006. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1007/S00224-004-1177-Z (cit. on pp. 226–237, 506).

[74] Harald Dyckhoff and Ute Finke. *Cutting and Packing in Production and Distribution: A Typology and Bibliography*. Contributions to Management Science. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag, 1992. ISSN: 1431-1941. ISBN: 978-3-642-63487-1. doi:10.1007/978-3-642-58165-6 (cit. on pp. 20–25, 500).

References XI

[75] Kelly Easton, George L. Nemhauser, and Michael A. Trick. "The Traveling Tournament Problem Description and Benchmarks". In: *7th International Conference on Principles and Practice of Constraint Programming (CP'01)*. Nov. 26–Dec. 1, 2001, Paphos, Cyprus. Ed. by Toby Walsh. Vol. 2239 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2001, pp. 580–584. ISSN: 0302-9743. ISBN: 978-3-540-42863-3. doi:10.1007/3-540-45578-7_43 (cit. on p. 511).

[76] Alwalid N. Elshafei. "Hospital Layout as a Quadratic Assignment Problem". *The Journal of the Operational Research Society (JORS)* 28(1):167–179, Apr. 1977. London, England, UK: Taylor and Francis Ltd. and London, England, UK: Springer Nature Limited. ISSN: 0160-5682. doi:10.1057/jors.1977.29 (cit. on pp. 318–324, 507).

[77] Bernhard Eschermann and Hans-Joachim Wunderlich. "Optimized Synthesis of Self-Testable Finite State Machines". In: *20th International Symposium Fault-Tolerant Computing (FTCS'1990): Digest of Papers*. June 26–28, 1990, Newcastle Upon Tyne, England, UK. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 1990. ISBN: 978-0-8186-2051-5. doi:10.1109/FTCS.1990.89393 (cit. on pp. 318–324, 507).

[78] Thomas A. Feo and Mauricio G. C. Resende. "Greedy Randomized Adaptive Search Procedures". *Journal of Global Optimization* 6(2):109–133, Mar. 1995. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1007/BF01096763. Introduces Greedy Randomized Adaptive Search Procedures (GRASPs) and applies them to the set covering problem, among others, and lists many problem domains. (Cit. on p. 502).

[79] Luca Ferrari and Enrico Pirozzi. *Learn PostgreSQL*. 2nd ed. Birmingham, England, UK: Packt Publishing Ltd, Oct. 2023. ISBN: 978-1-83763-564-1 (cit. on p. 507).

[80] Paola Festa and Mauricio G. C. Resende. "An Annotated Bibliography of GRASP—Part I: Algorithms". *International Transactions on Operational Research (ITOR)* 16(1):1–24, Jan. 2009. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0969-6016. doi:10.1111/j.1475-3995.2009.00663.x. See also [81] (cit. on pp. 471, 502).

[81] Paola Festa and Mauricio G. C. Resende. "An Annotated Bibliography of GRASP—Part II: Applications". *International Transactions on Operational Research (ITOR)* 16(2):131–172, Mar. 2009. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0969-6016. doi:10.1111/j.1475-3995.2009.00664.x. See also [80] (cit. on pp. 471, 502).

References XII

[82] Simon Fischer. "A Polynomial Upper Bound for a Mutation-Based Algorithm on the Two-Dimensional Ising Model". In: *6th Genetic and Evolutionary Computation Conference (GECCO'2004), Part I*. June 26–30, 2004, Seattle, WA, USA. Ed. by Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell. Vol. 3102 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2004, pp. 1100–1112. ISSN: 0302-9743. ISBN: 978-3-540-22344-3. doi:10.1007/978-3-540-24854-5_108 (cit. on pp. 226–237, 503).

[83] Simon Fischer and Ingo Wegener. "The One-Dimensional Ising Model: Mutation versus Recombination". *Theoretical Computer Science* 344(2-3):208–225, Nov. 2005. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0304-3975. doi:10.1016/J.TCS.2005.04.002 (cit. on pp. 226–237, 503).

[84] Henry Fisher and Gerald L. Thompson. "Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules". In: *Industrial Scheduling*. Ed. by John F. Muth and Gerald L. Thompson. Englewood Cliffs, NJ, USA: Prentice-Hall, 1963. Chap. 3.2, pp. 225–251 (cit. on pp. 386–392, 504).

[85] Charles Fleurent and Jacques A. Ferland. "Genetic Hybrids for the Quadratic Assignment Problem". In: *Quadratic Assignment and Related Problems, Proceedings of a DIMACS Workshop*. May 20–21, 1993, New Brunswick, NJ, USA: Rutgers University. Ed. by Panos Miltiades Pardalos and Henry Wolkowicz. Providence, RI, USA: American Mathematical Society (AMS), 1994, pp. 173–187. ISBN: 978-0-8218-6607-8. A MA using the TS from [212] as local search performs well. (Cit. on pp. 331–334, 505, 507).

[86] Tobias Friedrich, Pietro Simone Oliveto, Dirk Sudholt, and Carsten Witt. "Analysis of Diversity-Preserving Mechanisms for Global Exploration". *Evolutionary Computation* 17(4):455–476, Win. 2009. Cambridge, MA, USA: MIT Press. ISSN: 1063-6560. doi:10.1162/evco.2009.17.4.17401 (cit. on p. 511).

[87] Tobias Friedrich, Francesco Quinzan, and Markus Wagner. "Escaping Large Deceptive Basins of Attraction with Heavy-Tailed Mutation Operators". In: *20th Genetic and Evolutionary Computation Conference (GECCO'2018)*. July 15–19, 2018, Kyōto, Kyōto-fu, Japan. Ed. by Hernán E. Aguirre and Keiki Takadama. New York, NY, USA: Association for Computing Machinery (ACM), 2018, pp. 293–300. ISBN: 978-1-4503-5618-3. doi:10.1145/3205455.3205515 (cit. on pp. 226–237, 504, 511).

References XIII

[88] José E. Gallardo, Carlos Cotta, and Antonio J. Fernández. "A Memetic Algorithm for the Low Autocorrelation Binary Sequence Problem". In: *9th Genetic and Evolutionary Computation Conference (GECCO'2007)*. July 7–11, 2007, London, England, UK. Ed. by Hod Lipson. New York, NY, USA: Association for Computing Machinery (ACM), 2007, pp. 1226–1233. ISBN: 978-1-59593-697-4. doi:10.1145/1276958.1277195 (cit. on p. 504).

[89] Michael R. Garey and David Stifler Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman and Company, 1979. ISBN: 978-0-7167-1045-5 (cit. on pp. 299–304, 510).

[90] Mitsuo Gen, Yasuhiro Tsujimura, and Erika Kubota. "Solving Job-Shop Scheduling Problems by Genetic Algorithm". In: *Humans, Information and Technology: Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics (SMC'1994)*. Vol. 2. Oct. 2–5, 1994, San Antonio, TX, USA. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 1994, pp. 1577–1582. ISBN: 978-0-7803-2129-8. doi:10.1109/ICSMC.1994.400072. Proposes a permutations-with-repetitions based encoding for the JSSP. (Cit. on pp. 351–360, 474, 504, 513).

[91] Fred Glover. "Tabu Search – Part I". *ORSA Journal on Computing* 1(3):190–206, Sum. 1989. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.1.3.190 (cit. on p. 510).

[92] Fred Glover. "Tabu Search – Part II". *ORSA Journal on Computing* 2(1):4–32, Win. 1990. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.2.1.4 (cit. on p. 510).

[93] Fred Glover and Gary A. Kochenberger, eds. *Handbook of Metaheuristics*. Vol. 57 of International Series in Operations Research & Management Science (ISOR). Cham, Switzerland: Springer, 2003. ISSN: 0884-8289. ISBN: 978-0-306-48056-0. doi:10.1007/b101874 (cit. on p. 506).

[94] Fred Glover, Éric D. Taillard, and Dominique de Werra. "A User's Guide to Tabu Search". *Annals of Operations Research* 41(1):3–28, Mar. 1993. London, England, UK: Springer Nature Limited. ISSN: 0254-5330. doi:10.1007/BF02078647. Special issue on TS. (Cit. on p. 510).

[95] David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1989. ISBN: 978-0-201-15767-3 (cit. on pp. 238–242, 501, 509).

References XIV

[96] Carla P. Gomes, Henry A. Kautz, Ashish Sabharwal, and Bart Selman. "Satisfiability Solvers". In: *Handbook of Knowledge Representation*. Ed. by Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter. Vol. 3 of Foundations of Artificial Intelligence. Amsterdam, The Netherlands: Elsevier B.V., 2008. Chap. 2, pp. 89–134. ISSN: 1574-6526. URL: https://www.cs.cornell.edu/gomes/papers/SATSolvers-KR-Handbook.pdf (visited on 2026-09-20) (cit. on pp. 260–263).

[97] José Fernando Gonçalves and Mauricio G. C. Resende. "A Biased Random Key Genetic Algorithm for 2BP and 3D Bin Packing Problems". *International Journal of Production Economics (IJPE)* 145(2):500–510, Oct. 2013. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0925-5273. doi:10.1016/j.ijpe.2013.04.019. Applies the BRKGA to the 2BP|R|F and 2BP|O|F as well as 3D packing. Continuous vectors are translated to permutations and orientation data which is then the input of a placement heuristic. (Cit. on pp. 430–437, 498, 500).

[98] José Fernando Gonçalves and Mauricio G. C. Resende. "An Extended Akers Graphical Method with a Biased Random-Key Genetic Algorithm for Job-Shop Scheduling". *International Transactions on Operational Research (ITOR)* 21(2):215–246, Mar. 2014. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0969-6016. doi:10.1111/itor.12044. URL: http://mauricio.resende.info/doc/brkga-jss2011.pdf (visited on 2026-09-30). The BRKGA basically is a MA using a continuous encoding of permutations that are decoded to schedules using the method from [26,27,90,200] and then refined via a TS. (Cit. on pp. 395–400, 500, 504).

[99] José Fernando Gonçalves and Mauricio G. C. Resende. "Biased Random-Key Genetic Algorithms for Combinatorial Optimization". *Journal of Heuristics* 17:487–525, Oct. 2010. London, England, UK: Springer Nature Limited. ISSN: 1381-1231. doi:10.1007/s10732-010-9143-1 (cit. on p. 500).

[100] Michael T. Goodrich. *A Gentle Introduction to NP-Completeness*. Irvine, CA, USA: University of California, Irvine, Apr. 2022. URL: https://ics.uci.edu/~goodrich/teach/cs165/notes/NPComplete.pdf (visited on 2025-08-01) (cit. on p. 514).

[101] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and Alexander Hendrik George Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey". Annals of Discrete Mathematics (5):287–326. Ed. by Peter Ladislaw Hammer, Ellis Lane Johnson, and Bernhard H. Korte, 1979. ISSN: 0167-5060. doi:10.1016/S0167-5060(08)70356-X. An early review on JSSPs and FSSPs. (Cit. on pp. 336–350, 501, 504).

References XV

[102] Martin Grötschel, Michael Jünger, and Gerhard Reinelt. "Optimal Control of Plotting and Drilling Machines: A Case Study". *Zeitschrift für Operations Research (ZOR). now Mathematical Methods of Operations Research (MMOR)* 35(1):61–84, Jan. 1991. Heidelberg, Baden-Württemberg, Germany: Physica-Verlag; now London, England, UK: Springer Nature Limited. ISSN: 0340-9422. doi:10.1007/BF01415960. This paper tackles circuit board plotting and drilling problems by modelling (parts of) them as symmetric Traveling Salesperson Problems (STSPs). (Cit. on pp. 299–304, 510).

[103] Gregory Z. Gutin and Abraham P. Punnen, eds. *The Traveling Salesman Problem and its Variations*. Vol. 12 of Combinatorial Optimization (COOP). New York, NY, USA: Springer New York, May 2002. ISSN: 1388-3011. doi:10.1007/b101971 (cit. on pp. 299–304, 510).

[104] Pierre Hansen, Nenad Mladenović, and José Andrés Moreno Pérez. "Variable Neighbourhood Search: Methods and Applications. Invited Survey". *4OR* 6(4):319–360, Dec. 2008. ISSN: 1619-4500. doi:10.1007/s10288-008-0089-1. URL: https://www.researchgate.net/publication/225155945 (visited on 2026-10-02) (cit. on p. 512).

[105] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107, July 31, 1968. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 0536-1567. doi:10.1109/TSSC.1968.300136 (cit. on pp. 50–115, 499).

[106] Keld Helsgaun. "General k-opt Submoves for the Lin–Kernighan TSP Heuristic". *Mathematical Programming Computation* 1(2-3):119–163, July 2009. London, England, UK: Springer Nature Limited. ISSN: 1867-2949. doi:10.1007/s12532-009-0004-6. URL: https://www2.seas.gwu.edu/~simhaweb/champalg/tsp/papers/Helsgaun2.pdf (visited on 2026-09-26) (cit. on pp. 311–316, 505).

[107] Keld Helsgaun. "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic". 126(1):106–130, Oct. 2000. doi:10.1016/S0377-2217(99)00284-2. URL: http://akira.ruc.dk/~keld/research/LKH/LKH-2.0/DOC/LKH_REPORT.pdf (visited on 2026-09-26) (cit. on pp. 311–316, 505).

[108] André Henning. "Praktische Job-Shop Scheduling-Probleme". PhD thesis. Jena, Thüringen, Germany: Friedrich-Schiller-Universität Jena, Aug. 2002. URL: http://www.db-thueringen.de/servlets/DocumentServlet?id=873 (visited on 2026-09-30). Alternate url: https://nbn-resolving.org/urn:nbn:de:gbv:27-20060809-115700-4. Applies MA-like algorithms to the JSSP. (Cit. on pp. 395–400, 504, 505).

References XVI

[109] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Applications*. The Morgan Kaufmann Series in Artificial Intelligence. Amsterdam, The Netherlands: Elsevier B.V., 2005. ISBN: 978-1-55860-872-6. doi:10.1016/B978-1-55860-872-6.X5016-1 (cit. on pp. 248–255, 505, 509).

[110] Holger H. Hoos and Thomas Stützle. *SATLIB – The Satisfiability Library*. 1.4.4. Vancouver, BC, Canada: The University of British Columbia (UBC), May 11, 2001. URL: https://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html (visited on 2026-09-20). See paper [111] (cit. on pp. 257–259, 476, 509).

[111] Holger H. Hoos and Thomas Stützle. "SATLIB: An Online Resource for Research on SAT". In: *SAT2000 – Highlights of Satisfiability Research in the Year 2000*. Ed. by Ian Philip Gent, Toby Walsh, and Hans van Maaren. Vol. 63 of Frontiers in Artificial Intelligence and Applications. Amsterdam, The Netherlands: IOS Press BV, Jan. 2000, pp. 283–292. ISSN: 0922-6389. ISBN: 978-1-58603-061-2. See the website [110] (cit. on pp. 257–259, 476, 509).

[112] John Hunt. *A Beginners Guide to Python 3 Programming*. 2nd ed. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISSN: 1863-7310. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (cit. on p. 507).

[113] *Information Technology – Universal Coded Character Set (UCS)*. International Standard ISO/IEC 10646:2020. Geneva, Switzerland: International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), Dec. 2020 (cit. on p. 496).

[114] Manuel Iori, Vinícius Loti de Lima, Silvano Martello, Flávio K. Miyazaw, and Michele Monaci. "Exact Solution Techniques for Two-Dimensional Cutting and Packing". *European Journal of Operational Research* 289(2):399–415, Mar. 2021. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2020.06.050. See also arXiv:2004.12619v1 [math.OC] 27 Apr 2020 at doi:10.1016/j.ejor.2020.06.050. (Cit. on pp. 402–408, 498).

[115] Manuel Iori, Vinícius Loti de Lima, Silvano Martello, and Michele Monaci. *2DPackLib*. Bologna, Emilia-Romagna, Italy: Università di Bologna, 2021–2026. URL: https://site.unibo.it/operations-research/en/research/2dpacklib (visited on 2026-10-01). The accompanying paper for this benchmark set is [116]. (Cit. on pp. 424–429, 476, 499).

[116] Manuel Iori, Vinícius Loti de Lima, Silvano Martello, and Michele Monaci. "2DPackLib: A Two-Dimensional Cutting and Packing Library". *Optimization Letters* 16(2):471–480, Mar. 2022. London, England, UK: Springer Nature Limited. ISSN: 1862-4472. doi:10.1007/s11590-021-01808-y. URL: https://www.researchgate.net/publication/355191467 (visited on 2026-10-01). This benchmark set is available at https://site.unibo.it/operations-research/en/research/2dpacklib [115], (cit. on pp. 424–429, 476, 499).

References XVII

[117] Christoph Jabs, Dieter Vandesande, Jeremias Berg, Johannes Fichte, George Katsirelos, Ruben Martins, André Schidler, and Dominik Schreiber. *MaxSAT Evaluation: Complete Benchmark Collection*. 2026th ed. Geneva, Switzerland: zenodo (hosted by European Organization for Nuclear Research (CERN)), July 9, 2026. doi:10.5281/zenodo.21283148. All versions of this dataset can be reached directly via doi:10.5281/zenodo.21283147. See also https://github.com/maxsat-evaluations/data and https://maxsat-evaluations.github.io. (Cit. on pp. 257–259, 506).

[118] Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. "Monte Carlo Techniques in Code Optimization". In: *15th Annual Workshop on Microprogramming (MICRO 15)*. Oct. 5–7, 1982. Ed. by Joseph Allen Fisher, William J. Tracz, and Bill Hopkins. Palo Alto, CA, USA: Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE) and New York, NY, USA: Association for Computing Machinery (ACM), Oct. 1982, pp. 143–148. doi:10.5555/800036.800944. See [119] (cit. on p. 477).

[119] Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. "Monte Carlo Techniques in Code Optimization". *ACM SIGMICRO Newsletter* 13(4):143–148, Dec. 1982. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 1050-916X. doi:10.1145/1014194.800944. See [118] (cit. on pp. 477, 508).

[120] Stefan Jakobs. "On Genetic Algorithms for the Packing of Polygons". *European Journal of Operational Research* 88(1):165–181, Jan. 1996. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(94)00166-9 (cit. on p. 502).

[121] Wassim Jaziri, ed. *Tabu Search*. London, England, UK: IntechOpen Limited, Aug. 1, 2008. ISBN: 978-3-902613-34-9. doi:10.5772/65. URL: https://www.intechopen.com/books/6114 (visited on 2026-09-30) (cit. on p. 510).

[122] David Stifler Johnson, Gregory Z. Gutin, Lyle Andrew McGeoch, Anders Yeo, Weixiong Zhang, and Alexei Zverovitch. *Experimental Analysis of Heuristics for the ATSP*. Ed. by Gregory Z. Gutin and Abraham P. Punnen. New York, NY, USA, May 2002. doi:10.1007/0-306-48213-4_10. URL: https://archive.dimacs.rutgers.edu/Challenges/TSP/papers/atspchap.pdf (visited on 2026-09-25) (cit. on pp. 305–309, 510).

[123] David Stifler Johnson and Lyle Andrew McGeoch. *Experimental Analysis of Heuristics for the STSP*. Ed. by Gregory Z. Gutin and Abraham P. Punnen. New York, NY, USA, May 2002. doi:10.1007/0-306-48213-4_9. URL: https://archive.dimacs.rutgers.edu/Challenges/TSP/papers/stspchap.pdf (visited on 2026-09-25) (cit. on pp. 305–309, 510).

References XVIII

[124] David Stifler Johnson, Lyle Andrew McGeoch, Fred Glover, and Cesar Rego. *8th DIMACS Implementation Challenge: The Traveling Salesman Problem*. June 1, 2000–Nov. 19, 2013. URL: https://archive.dimacs.rutgers.edu/Challenges/TSP (visited on 2026-09-25) (cit. on pp. 305–309, 510).

[125] Selmer Martin Johnson. "Optimal Two- and Three-Stage Production Schedules with Setup Times Included". *Naval Research Logistics Quarterly* I(1):61–68, Mar. 1954. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0894-069X. doi:10.1002/nav.3800010110. URL: https://www.rand.org/content/dam/rand/pubs/papers/2008/P402.pdf (visited on 2023-12-06) (cit. on p. 501).

[126] Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. "Optimization by Simulated Annealing". *Science Magazine* 220(4598):671–680, May 13, 1983. Washington, D.C., USA: American Association for the Advancement of Science (AAAS). ISSN: 0036-8075. doi:10.1126/science.220.4598.671. URL: https://www.researchgate.net/publication/6026283 (visited on 2025-08-08) (cit. on pp. 202–214, 508).

[127] Robert Klein. *Scheduling of Resource-Constrained Projects*. Vol. 10 of Operations Research/Computer Science Interfaces Series (ORCS). New York, NY, USA: Springer New York, 2000. ISSN: 1387-666X. ISBN: 978-0-7923-8637-7. doi:10.1007/978-1-4615-4629-0 (cit. on pp. 336–350, 502).

[128] Donald Ervin Knuth. "Big Omicron and Big Omega and Big Theta". *ACM SIGACT News* 8(2):18–24, Apr.–June 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0163-5700. doi:10.1145/1008328.1008329 (cit. on p. 514).

[129] Donald Ervin Knuth. *Fundamental Algorithms*. 3rd ed. Vol. 1 of The Art of Computer Programming. Reading, MA, USA: Addison-Wesley Professional, 1997. ISBN: 978-0-201-89683-1 (cit. on p. 514).

[130] Tjalling Charles Koopmans and Martin Beckmann. "Assignment Problems and the Location of Economic Activities". *Econometrica* 25(1):53–76, 1957. New Haven, CT, USA: The Econometric Society and Chichester, West Sussex, England, UK: John Wiley and Sons Ltd. ISSN: 0012-9682. doi:10.2307/1907742 (cit. on pp. 318–324, 507).

[131] Jakob Krarup and Peter Mark Pruzan. "Computer-aided Layout Design". In: *Mathematical Programming in Use*. Ed. by Michel Louis Balinski and Claude Lemaréchal. Vol. 9 of. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1978. Chap. 6, pp. 75–94. ISSN: 0303-3929. ISBN: 978-3-642-00796-5. doi:10.1007/BFb0120827. Uses QAPs as model for designing architectural layouts. (Cit. on pp. 318–324, 507).

References XIX

[132] Mark W. Krentel. "The Complexity of Optimization Problems". *Journal of Computer and System Sciences* 36(3):490–509, June 1988. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0022-0000. Explores the hardness (optimization) problems such TSP, Clique, and Coloring, where computing the optimal value is $\mathcal{NP}$-complete. (Cit. on pp. 299–304, 510).

[133] Edmund Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. Leipzig, Sachsen, Germany: B. G. Teubner, 1909. ISBN: 978-0-8218-2650-8 (cit. on p. 514).

[134] Pedro Larrañaga, Cindy M. H. Kuijpers, Roberto H. Murga, Iñaki Inza, and S. Dizdarevic. "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators". *Artificial Intelligence Review* 13(2):129–170, Apr. 1999. London, England, UK: Springer Nature Limited. ISSN: 0269-2821. doi:10.1023/A:1006529012972 (cit. on pp. 292–297, 499).

[135] Pedro Larrañaga, Cindy M. H. Kuijpers, Mikel Poza, and Roberto H. Murga. "Decomposing Bayesian Networks: Triangulation of the Moral Graph with Genetic Algorithms". *Statistics and Computing* 7(1):19–34, Mar. 1997. London, England, UK: Springer Nature Limited. ISSN: 0960-3174. doi:10.1023/A:1018553211613. This paper introduces AP. (Cit. on pp. 292–297, 499).

[136] Eugene Leighton Lawler, Jan Karel Lenstra, and Alexander Hendrik George Rinnooy Kan. "Recent Developments in Deterministic Sequencing and Scheduling: A Survey". In: *Deterministic and Stochastic Scheduling: Proceedings of the NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems*. June 6–17, 1981, Durham, England, UK. Ed. by Michael Anthony Howe Dempster, Jan Karel Lenstra, and Alexander Hendrik George Rinnooy Kan. Vol. 84 of Nato Science Series C: Mathematical and Physical Science (ASIC). London, England, UK: Springer Nature Limited, Apr. 1982, pp. 35–73. ISBN: 978-90-277-1397-1. doi:10.1007/978-94-009-7801-0_3. The JSSP is $\mathcal{NP}$-hard. (Cit. on pp. 336–350, 504).

[137] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. "Sequencing and Scheduling: Algorithms and Complexity". In: *Production Planning and Inventory*. Ed. by Stephen C. Graves, Alexander Hendrik George Rinnooy Kan, and Paul H. Zipkin. Vol. IV of Handbooks of Operations Research and Management Science. Amsterdam, The Netherlands: Elsevier B.V., 1993. Chap. 9, pp. 445–522. ISSN: 0927-0507. ISBN: 978-0-444-87472-6. doi:10.1016/S0927-0507(05)80189-6. URL: http://alexandria.tue.nl/repository/books/339776.pdf (visited on 2023-12-06). Discusses, amongst others, that the JSSP is $\mathcal{NP}$-hard. (Cit. on pp. 336–350, 504, 514).

References XX

[138] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, England, UK: Wiley Interscience, Sept. 1985. ISSN: 0277-2698. ISBN: 978-0-471-90413-7 (cit. on pp. 128, 510).

[139] Stephen R. Lawrence. "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques". PhD thesis. Pittsburgh, PA, USA: Graduate School of Industrial Administration (GSIA), Carnegie-Mellon University, 1984 (cit. on pp. 386–392, 504).

[140] Kent D. Lee and Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISSN: 1863-7310. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (cit. on p. 507).

[141] Lai-Soon Lee. "A Genetic Algorithm for Two-Dimensional Bin Packing Problem". *MathDigest: Research Bulletin of Institute for Mathematical Research* 2(1):34–39, Dec. 2008. A multi-crossover EA for the 2BP|R|F using a heuristic placing routine. (Cit. on pp. 430–437).

[142] Xueping Li and Kaike Zhang. "Single Batch Processing Machine Scheduling with Two-Dimensional Bin Packing Constraints". *International Journal of Production Economics (IJPE)* 196:113–121, Feb. 2018. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0925-5273. doi:10.1016/j.ijpe.2017.11.015. Mentions some scheduling tasks of Additive Manufacturing jobs can be viewed as instance of the 2BP|R|F. (Cit. on pp. 402–408, 498).

[143] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, Sarah Louise Thomson, and Thomas Weise (汤卫思). "Addressing the Traveling Salesperson Problem with Frequency Fitness Assignment and Hybrid Algorithms". *Soft Computing* 28(17-18):9495–9508, July 2024. London, England, UK: Springer Nature Limited. ISSN: 1432-7643. doi:10.1007/S00500-024-09718-8 (cit. on pp. 501, 506, 510).

[144] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Jörg Lässig, Daan van den Berg, and Thomas Weise (汤卫思). "Solving the Traveling Salesperson Problem using Frequency Fitness Assignment". In: *IEEE Symposium Series on Computational Intelligence (SSCI'2022)*. Dec. 4–7, 2022, Singapore. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2022. ISBN: 978-1-6654-8769-6. doi:10.1109/SSCI51031.2022.10022296 (cit. on pp. 501, 506).

References XXI

[145] Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Matthias Thürer, Markus Wagner, and Thomas Weise (汤卫思). "Generating Small Instances with Interesting Features for the Traveling Salesperson Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 173–180. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888800003837 (cit. on p. 506).

[146] Shen Lin and Brian W. Kernighan. "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". *Operations Research* 21(2):498–516, Mar.–Apr. 1973. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0030-364X. doi:10.1287/opre.21.2.498. URL: https://www.cs.princeton.edu/~bwk/btl.mirror/tsp.pdf (visited on 2026-09-26) (cit. on pp. 310, 505).

[147] Dequan Liu (刘德全) and Hongfei Teng (滕弘飞). "An Improved BL-Algorithm for EAGenetic Algorithm of the Orthogonal Packing of Rectangles". *European Journal of Operational Research* 112(2):413–420, Jan. 1999. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(97)00437-2 (cit. on pp. 409–423, 502).

[148] Andrea Lodi, Silvano Martello, and Michele Monaci. "Two-Dimensional Packing Problems: A Survey". *European Journal of Operational Research* 141(2):241–252, Sept. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(02)00123-6 (cit. on pp. 402–408, 498, 500).

[149] Andrea Lodi, Silvano Martello, and Daniele Vigo. "Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems". *INFORMS Journal on Computing* 11(4):345–357, Aut. 1999. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 1091-9856. doi:10.1287/ijoc.11.4.345 (cit. on p. 498).

[150] Andrea Lodi, Silvano Martello, and Daniele Vigo. "Recent Advances on Two-Dimensional Bin Packing Problems". *Discrete Applied Mathematics* 123(1–3):379–396, Nov. 2002. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0166-218X. doi:10.1016/S0166-218X(01)00347-X (cit. on pp. 402–408, 498, 500).

[151] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter M. Hahn, and Tania Querido. "A Survey for the Quadratic Assignment Problem". *European Journal of Operational Research* 176(2):657–690, 2007. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2005.09.032 (cit. on pp. 318–324, 507).

References XXII

[152] Mariana A. Londe, Luciana S. Pessoa, Carlos E. Andrade, and Mauricio G. C. Resende. *Biased Random-Key Genetic Algorithms: A Review*. arXiv.org: Computing Research Repository (CoRR) abs/2312.00961v2. Ithaca, NY, USA: Cornell Universiy Library, Dec. 6, 2023. doi:10.48550/arXiv.2312.00961. URL: https://arxiv.org/abs/2312.00961v2 (visited on 2024-06-27). arXiv:2312.00961v2 [cs.NE] 06 Dec 2023; See [153] (cit. on p. 482).

[153] Mariana A. Londe, Luciana S. Pessoa, Carlos E. Andrade, and Mauricio G. C. Resende. "Biased Random-Key Genetic Algorithms: A Review". *European Journal of Operational Research* 321(1):1–22, Feb. 2025. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2024.03.030. See [152] (cit. on pp. 482, 500).

[154] Marc Loy, Patrick Niemeyer, and Daniel Leuck. *Learning Java*. 5th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2020. ISBN: 978-1-4920-5627-0 (cit. on p. 503).

[155] Zhipeng Lü and Jin-Kao Hao. "Adaptive Memory-based Local Search for MAX-SAT". *Applied Soft Computing (ASOC)* 12(8):2063–2071, Aug. 2012. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 1568-4946. doi:10.1016/J.ASOC.2012.01.013 (cit. on pp. 267–269, 499).

[156] Mark Lutz. *Learning Python*. 6th ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2025. ISBN: 978-1-0981-7130-8 (cit. on p. 507).

[157] Rita Macedo, Cláudio Alves, and J. M. Valério de Carvalho. "Arc-Flow Model for the Two-Dimensional Guillotine Cutting Stock Problem". *Computers & Operations Research* 37(6):991–1001, June 2010. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/j.cor.2009.08.005 (cit. on pp. 424–429, 499, 500).

[158] Reinhard Männer and Bernard Manderick, eds. *Parallel Problem Solving from Nature 2 (PPSN-II)*. Sept. 28–30, 1992, Brussels, Belgium. Amsterdam, The Netherlands: Elsevier B.V., 1992.

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

[160] Silvano Martello and Daniele Vigo. "Exact Solution of the Two-Dimensional Finite Bin Packing Problem". 44(3):388–399, Mar. 1998. doi:10.1287/mnsc.44.3.388 (cit. on pp. 424–429, 498, 499).

References XXIII

[161] Stephan Mertens. "Exhaustive Search for Low-Autocorrelation Binary Sequences". *Journal of Physics A: Mathematical and General* 29(18):L473, Sept. 1996. Bristol, England, UK: IOP Publishing Ltd. ISSN: 0305-4470. doi:10.1088/0305-4470/29/18/005 (cit. on pp. 278–281, 504).

[162] Peter Merz and Bernd Freisleben. "A Comparison of MAMemetic Algorithms, TSTabu Search, and ACOAnt Colonies for the QAPQuadratic Assignment Problem". In: *Congress on Evolutionary Computation (CEC'1999)*. July 6–9, 1999, Washington, D.C., USA. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 1999, pp. 2063–2070. doi:10.1109/CEC.1999.785529 (cit. on pp. 331–334, 507).

[163] Zbigniew "Zbyszek" Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 1996. ISBN: 978-3-540-58090-4. doi:10.1007/978-3-662-03315-9 (cit. on pp. 501, 509).

[164] Burkhard Militzer, Michele Zamparelli, and Dieter Beule. "Evolutionary Search for Low Autocorrelated Binary Sequences". *IEEE Transactions on Evolutionary Computation* 2(1):34–39, Apr. 1998. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/4235.728212 (cit. on pp. 278–281, 504).

[165] Alfonsas Misevičius. "A Tabu Search Algorithm for the Quadratic Assignment Problem". *Computational Optimization and Applications* 30:95–111, Jan. 2005. London, England, UK: Springer Nature Limited. ISSN: 0926-6003. doi:10.1007/s10589-005-4562-x (cit. on pp. 326–330, 510).

[166] Alfonsas Misevičius. "A Modified Simulated Annealing Algorithm for the Quadratic Assignment Problem". *Informatica* 14(4):497–514, Jan. 2003. Vilnius, Lithuania: Vilnius University, Institute of Data Science and Digital Technologies (DMSTI), Lithuanian Academy of Sciences. ISSN: 0868-4952. doi:10.15388/Informatica.2003.037 (cit. on pp. 326–330, 508).

[167] Alfonsas Misevičius. *An Implementation of the Iterated Tabu Search Algorithm for the Quadratic Assignment Problem*. Working Paper. Kaunas, Lithuania: Kaunas University of Technology, 2008. See also [168] (cit. on p. 483).

[168] Alfonsas Misevičius. "An Implementation of the Iterated Tabu Search Algorithm for the Quadratic Assignment Problem". *OR Spectrum: Quantitative Approaches in Management. Formerly Operations-Research-Spektrum* 34(3):665–690, July 2012. London, England, UK: Springer Nature Limited. ISSN: 0171-6468. doi:10.1007/s00291-011-0274-z. See also [167] (cit. on pp. 331–334, 483, 507, 510).

References XXIV

[169] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Complex Adaptive Systems. Cambridge, MA, USA: MIT Press, Feb. 1998. ISBN: 978-0-262-13316-6. URL: http://boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf (visited on 2025-08-08) (cit. on pp. 501, 509).

[170] Nenad Mladenović and Pierre Hansen. "Variable Neighborhood Search". *Computers & Operations Research* 24(11):1097–1100, Nov. 1997. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/S0305-0548(97)00031-2 (cit. on p. 512).

[171] Wai Ho Mow, Ke-Lin Du, and Wei Hsiang Wu. "New Evolutionary Search for Long Low Autocorrelation Binary Sequences". *IEEE Transactions on Aerospace and Electronic Systems* 51(1):290–303, Jan. 2015. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 0018-9251. doi:10.1109/TAES.2014.130518 (cit. on pp. 271–275, 282–286, 504).

[172] Heinz Mühlenbein. "How Genetic Algorithms Really Work: Mutation and Hillclimbing". In: *Parallel Problem Solving from Nature 2 (PPSN-II)*. Sept. 28–30, 1992, Brussels, Belgium. Ed. by Reinhard Männer and Bernard Manderick. Amsterdam, The Netherlands: Elsevier B.V., 1992, pp. 15–26. URL: https://www.researchgate.net/publication/284805798 (visited on 2025-08-15) (cit. on pp. 226–237, 506).

[173] Yuichi Nagata and Shigenobu Kobayashi. "A Powerful Genetic Algorithm using Edge Assembly Crossover for the Traveling Salesman Problem". *INFORMS Journal on Computing* 25(2):346–363, Spr. 2013. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 1091-9856. doi:10.1287/ijoc.1120.0506 (cit. on pp. 311–316, 501).

[174] Yuichi Nagata and Shigenobu Kobayashi. "Edge Assembly Crossover: A High-Power Genetic Algorithm for the Traveling Salesman Problem". In: *7th International Conference on Genetic Algorithms (ICGA'1997)*. July 19–23, 1997, East Lansing, MI, USA. Ed. by Thomas Bäck. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 1997, pp. 450–457. ISBN: 978-1-55860-487-2 (cit. on pp. 311–316, 501).

[175] Yuichi Nagata and David Soler. "A New Genetic Algorithm for the Asymmetric Traveling Salesman Problem". *Expert Systems with Applications* 39(10):8947–8953, Aug. 2012. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0957-4174. doi:10.1016/j.eswa.2012.02.029 (cit. on pp. 311–316, 501).

[176] Siegfried Nijssen and Thomas Bäck. "An Analysis of the Behavior of Simplified Evolutionary Algorithms on Trap Functions". *IEEE Transactions on Evolutionary Computation* 7(1):11–22, Feb. 2003. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2002.806169 (cit. on pp. 226–237, 509).

References XXV

[177] Eugeniusz Nowicki and Czesław Smutnicki. "A Fast Taboo Search Algorithm for the Job Shop Problem". *Management Science* 42(6):783–938, June 1996. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.42.6.797 (cit. on pp. 351–360, 393, 394, 510).

[178] Regina O. Obe and Leo S. Hsu. *PostgreSQL: Up and Running*. 3rd ed. Sebastopol, CA, USA: O'Reilly Media, Inc., Oct. 2017. ISBN: 978-1-4919-6336-4 (cit. on p. 507).

[179] Tom Packebusch and Stephan Mertens. "Low Autocorrelation Binary Sequences". *Journal of Physics A: Mathematical and Theoretical* 49(16):165001, Mar. 2016. Bristol, England, UK: IOP Publishing Ltd. ISSN: 1751-8121. doi:10.1088/1751-8113/49/16/165001 (cit. on pp. 271–275, 504).

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

[181] Panos Miltiades Pardalos, Oleg V. Shylo, and Alkis Vazacopoulos. "Solving Job Shop Scheduling Problems Utilizing the Properties of Backbone and "Big Valley"". *Computational Optimization and Applications* 47(1):61–76, Sept. 2010. London, England, UK: Springer Nature Limited. ISSN: 0926-6003. doi:10.1007/s10589-008-9206-5. An algorithm similar to SA. (Cit. on pp. 395–400, 504, 508).

[182] Francisco Parreño, Ramón Alvarez-Valdés, José Fernando Oliveira, and José Manuel Tamarit. "A Hybrid GRASP/VND Algorithm for Two- and Three-Dimensional Bin Packing". *Annals of Operations Research* 179(1):203–220, Sept. 2010. London, England, UK: Springer Nature Limited. ISSN: 0254-5330. doi:10.1007/s10479-008-0449-4 (cit. on pp. 430–437, 498, 502, 511).

[183] Amit Patel. *Amit's A* Pages*. Stanford, CA, USA: Stanford University, 1997–2025. URL: https://theory.stanford.edu/~amitp/GameProgramming (visited on 2026-01-02) (cit. on pp. 50–115, 499).

[184] Bo Peng, Zhipeng Lü, and T. C. Edwin Cheng. "A Tabu Search/Path Relinking Algorithm to Solve the Job Shop Scheduling Problem". *Computers & Operations Research* 53:154–164, Jan. 2015. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/j.cor.2014.08.006. URL: http://arxiv.org/abs/1402.5613 (visited on 2026-09-30) (cit. on pp. 395–400, 504, 510).

References XXVI

[185] Ferdinando Pezzella and Emanuela Merelli. "A Tabu Search Method Guided by Shifting Bottleneck for the Job Shop Scheduling Problem". *European Journal of Operational Research* 120(2):297–310, Jan. 2000. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(99)00158-7. URL: https://www2.cs.sfu.ca/CourseCentral/827/havens/papers/topic%5C%2310(JobShop)/Tabu%5C%20With%5C%20Shifting.pdf (visited on 2026-09-30) (cit. on pp. 393, 394, 510).

[186] Martin Pincus. "Letter to the Editor – A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems". *Operations Research* 18(6):1225–1228, Nov.–Dec. 1970. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0030-364X. doi:10.1287/opre.18.6.1225 (cit. on p. 508).

[187] Marcelo Pinto, Cristóvão Silva, Matthias Thürer, and Samuel Moniz. "Survey in Operations Research and Management Science. Nesting and Scheduling Optimization of Additive Manufacturing Systems: Mapping the Territory". *Computers & Operations Research* 165(106592), Feb. 2024. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0305-0548. doi:10.1016/j.cor.2024.106592. Mentions some scheduling tasks of Additive Manufacturing jobs can be viewed as instance of the 2BP|R|F. (Cit. on pp. 402–408, 498).

[188] Alexander Podlich, Thomas Weise (汤卫思), Manfred Menze, and Christian Gorldt. "Intelligente Wechselbrückensteuerung für die Logistik von Morgen". *Electronic Communications of the EASST (ECEASST)* 17(Communication in Distributed Systems), Feb. 27, 2009. The Netherlands: European Association of Software Science and Technology (EASST). ISSN: 1863-2122. doi:10.14279/tuj.eceasst.17.205. In the proceedings of *Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009* (WowKiVS 2009), 3 6, 2009, Kassel, Hessen, Germany (cit. on pp. 20–25).

[189] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., Mar. 2024 (cit. on p. 507).

[190] S. D. Prestwich. *Improved Branch-and-Bound for Low Autocorrelation Binary Sequences*. arXiv.org: Computing Research Repository (CoRR) abs/1305.6187. Ithaca, NY, USA: Cornell Universiy Library, Apr. 3, 2024. doi:10.48550/arXiv.1305.6187. URL: https://arxiv.org/abs/1305.6187 (visited on 2013-07-23). arXiv:1305.6187v2 [cs.AI] 23 Jul 2013 (cit. on p. 504).

[191] *Programming Languages – C, Working Document of SC22/WG14*. International Standard ISO/IEC9899:2017 C17 Ballot N2176. Geneva, Switzerland: International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), Nov. 2017. URL: https://files.lhmouse.com/standards/ISO%20C%20N2176.pdf (visited on 2024-06-29) (cit. on p. 500).

References XXVII

[192] Sanatan Rai and George Vairaktarakis. "NP-Complete Problems and Proof Methodology". In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos Miltiades Pardalos. 2nd ed. Boston, MA, USA: Springer, Sept. 2008, pp. 2675–2682. ISBN: 978-0-387-74758-3. doi:10.1007/978-0-387-74759-0_462 (cit. on p. 514).

[193] Soraya B. Rana and L. Darrell Whitley. "Genetic Algorithm Behavior in the MAXSAT Domain". In: *5th International Conference on Parallel Problem Solving from Nature (PPSN V)*. Sept. 27–30, 1998, Amsterdam, The Netherlands. Ed. by Ágoston Endre Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel. Lecture Notes in Computer Science (LNCS). 1998, pp. 785–794. ISSN: 0302-9743. ISBN: 978-3-540-65078-2. doi:10.1007/BFb0056920 (cit. on p. 505).

[194] Gerhard Reinelt. "TSPLIB – A Traveling Salesman Problem Library". *ORSA Journal on Computing* 3(4):376–384, Aut. 1991. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0899-1499. doi:10.1287/ijoc.3.4.376 (cit. on pp. 305–309, 510).

[195] Gerhard Reinelt. *TSPLIB95*. Tech. rep. Heidelberg, Baden-Württemberg, Germany: Universität Heidelberg, Institut für Angewandte Mathematik, 1995–Apr. 28, 2016. URL: https://github.com/shredderzwj/TSPLIB (visited on 2026-09-25). The original Uniform Resource Locator (URL) of the repository has become defunct in 2026. Therefore, we provide the repository listed as main URL which contains the data. Additionally, you may also check https://github.com/thomasWeise/tspSuite of our own project *TSPSuite*, which contains all the TSPLib instances as well as Java code to load them. (Cit. on pp. 305–309, 510).

[196] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*. Vol. 35 of Forschungsergebnisse zur Informatik. Hamburg, Germany: Verlag Dr. Kovač, 1997. ISSN: 1435-6260. ISBN: 978-3-86064-554-3 (cit. on pp. 226–237, 505).

[197] Sartaj Sahni and Teofilo Francisco Gonzalez Arce. "P-complete Approximation Problems". *Journal of the ACM (JACM)* 23(3):555–565, July 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0004-5411. doi:10.1145/321958.321975. URL: https://sites.cs.ucsb.edu/~teo/papers/JACM-PCom.pdf (visited on 2026-02-21) (cit. on pp. 318–324, 507).

[198] Guntram Scheithauer. *Introduction to Cutting and Packing Optimization: Problems, Modeling Approaches, Solution Methods*. Vol. 263 of International Series in Operations Research & Management Science (ISOR). Cham, Switzerland: Springer, 2018. ISSN: 0884-8289. ISBN: 978-3-319-64402-8. doi:10.1007/978-3-319-64403-5 (cit. on pp. 20–25, 500).

References XXVIII

[199] Bart Selman, Henry A. Kautz, and Bram Cohen. "Noise Strategies for Improving Local Search". In: *12th National Conference on Artificial Intelligence*. Vol. 1. July 31–Aug. 4, 1994, Seattle, WA, USA. Ed. by Barbara Hayes-Roth and Richard E. Korf. Washington, D.C., USA: AAAI Press and Cambridge, MA, USA: MIT Press, 1994, pp. 337–343. ISBN: 978-0-262-61102-2. URL: http://www.aaai.org/Library/AAAI/1994/aaai94-051.php. Introduces WalkSAT. (Cit. on pp. 264–266, 512).

[200] Guoyong Shi, Hitoshi Iima, and Nobuo Sannomiya. "New Encoding Scheme for Solving Job Shop Problems by Genetic Algorithm". In: *35th IEEE Conference on Decision and Control (CDC'96)*. Vol. 4. Dec. 11–13, 1996. Kōbe, Hyōgo-ken, Japan: Institute of Electrical and Electronics Engineers (IEEE), 1997, pp. 4395–4400. ISBN: 978-0-7803-3590-5. doi:10.1109/CDC.1996.577484. URL: https://www.researchgate.net/publication/224238934 (visited on 2026-09-30). Proposes a permutations-with-repetitions based encoding for the JSSP. (Cit. on pp. 351–360, 474, 504, 513).

[201] Oleg V. Shylo. *Job Shop Scheduling (Personal Homepage)*. Knoxville, TN, USA: Oleg V. Shylo, Aug. 2019. URL: http://optimizizer.com/jobshop.php (visited on 2026-09-30). Contains many JSSP benchmark instances. (Cit. on pp. 386–389, 488, 504).

[202] Oleg V. Shylo and Hesam Shams. *Boosting Binary Optimization via Binary Classification: A Case Study of Job Shop Scheduling*. arXiv.org: Computing Research Repository (CoRR) abs/1808.10813. Proposes a TS using regression to adapt the tabu tenures for each variable of the solution vector of a JSSP separately. Many results are available in the GitHub repository https://github.com/quasiquasar/gta-jobshop-data; other results are published Shylo's website http://optimizizer.com/jobshop.php. Ithaca, NY, USA: Cornell Universiy Library, Aug. 31, 2018. doi:10.48550/arXiv.1808.10813. URL: https://arxiv.org/abs/1808.10813 (visited on 2026-09-30) (cit. on pp. 395–400, 504, 510).

[203] Hossein M. Soroush. "Scheduling in Stochastic Bicriteria Single Machine Systems with Set-up Times". *International Journal of Planning and Scheduling* 1(1–2):109–145, Jan. 2011. Geneva, Switzerland: Inderscience Publishers. ISSN: 2044-494X. doi:10.1504/IJPS.2011.044605. A scheduling problem is phrased as QAP. (Cit. on pp. 318–324, 507).

[204] Leon Steinberg. "The Backboard Wiring Problem: A Placement Algorithm". *SIAM Review* 3(1):37–50, Jan. 1961. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics Publications (SIAM). ISSN: 0036-1445. doi:10.1137/1003003. The problem is modeled as a QAP (cit. on pp. 318–324, 507).

References XXIX

[205] Robert H. Storer, S. David Wu, and Renzo Vaccari. "New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling". *Management Science* 38(10):1495–1509, Oct. 1992. Catonsville, MD, USA: The Institute for Operations Research and the Management Sciences (INFORMS). ISSN: 0025-1909. doi:10.1287/mnsc.38.10.1495 (cit. on pp. 386–392, 504).

[206] Thomas Strecker and Leonhard Hennig. "Automatic Layouting of Personalized Newspaper Pages". In: *Operations Research Proceedings 2008: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR)*. Sept. 3–5, 2008, Augsburg, Bayern, Germany: University of Augsburg. Ed. by Bernhard Fleischmann, Karl-Heinz Borgwardt, Robert Klein, and Axel Tuma. Vol. 2008 of Operations Research Proceedings (ORP), Gesellschaft für Operations Research e.V., (GOR). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2009, pp. 469–474. ISSN: 0721-5924. doi:10.1007/978-3-642-00142-0_76 (cit. on pp. 402–408, 498).

[207] Thomas Stützle. "Iterated Local Search for the Quadratic Assignment Problem". *European Journal of Operational Research* 174(3):1519–1539, Nov. 2006. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/j.ejor.2005.01.066 (cit. on pp. 318–324, 326–330, 503, 507).

[208] Dirk Sudholt. "Crossover Speeds up Building-Block Assembly". In: *14th Genetic and Evolutionary Computation Conference (GECCO'2012)*. July 7–11, 2012, Philadelphia, PA, USA. Ed. by Terence Soule and Jason Hall Moore. New York, NY, USA: Association for Computing Machinery (ACM), 2012, pp. 689–702. ISBN: 978-1-4503-1177-9. doi:10.1145/2330163.2330260. This paper is on the Greedy $(2+1)$ Genetic Algorithm (Greedy $(2+1)$ GA), see also [209] (cit. on pp. 243–246, 489, 502).

[209] Dirk Sudholt. "How Crossover Speeds up Building Block Assembly in Genetic Algorithms". *Evolutionary Computation* 25(2):237–274, 2017. Cambridge, MA, USA: MIT Press. ISSN: 1063-6560. doi:10.1162/EVCO_a_00171. This paper is on the Greedy $(2+1)$ GA, see also [208] (cit. on pp. 243–246, 489, 502).

[210] Éric D. Taillard. "Benchmarks for Basic Scheduling Problems". *European Journal of Operational Research* 64(2):278–285, Jan. 1993. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/0377-2217(93)90182-M. URL: https://arodes.hes-so.ch/record/8224/files/Preprint.pdf (visited on 2026-02-20) (cit. on pp. 386–392, 504).

[211] Éric D. Taillard. "Comparison of Iterative Searches for the Quadratic Assignment Problem". *Location Science. Now incorporated into Computers & Operations Research* 3(2):87–105, Aug. 1995. Amsterdam, The Netherlands: Elsevier B.V. doi:10.1016/0966-8349(95)00008-6 (cit. on pp. 326–330, 510).

References XXX

[212] Éric D. Taillard. "Robust Taboo Search for the Quadratic Assignment Problem". *Parallel Computing* 17(4–5):443–455, July 1991. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0167-8191. doi:10.1016/S0167-8191(05)80147-4 (cit. on pp. 326–334, 472, 490, 507, 510).

[213] Éric D. Taillard and Luca M. Gambardella. *Adaptive Memories for the Quadratic Assignment Problem*. Tech. rep. IDSIA-87-97. Lugano, Switzerland: Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale (IDSIA), Sept. 10, 1997. URL: https://mistic.iict-heig-vd.ch/taillard/articles.dir/TaillardGambardella1997.pdf (visited on 2026-09-28). A MA using TS from [212] as well as a ACO-based approach refining its solutions with a very simple SLS are presented. (Cit. on pp. 331–334, 507).

[214] Alkin Tezuysal and Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, July 2024. ISBN: 978-1-80323-347-5 (cit. on p. 507).

[215] Dirk Thierens, David Edward Goldberg, and Angela Guimarães Pereira. "Domino Convergence, Drift, and the Temporal-Salience Structure of Problems". In: pp. 535–540. doi:10.1109/ICEC.1998.700085 (cit. on pp. 226–237, 497).

[216] Sarah Louise Thomson, Gabriela Ochoa, Daan van den Berg, Tianyu Liang (梁天宇), and Thomas Weise (汤卫思). "Entropy, Search Trajectories, and Explainability for Frequency Fitness Assignment". In: *Parallel Problem Solving from Nature (PPSN XVIII)*. Vol. 1. Sept. 14–18, 2024, Hagenberg, Mühlkreis, Austria. Ed. by Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tušar, Penousal Machado, and Thomas Bäck. Vol. 15148 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer. ISSN: 0302-9743. ISBN: 978-3-031-70054-5. doi:10.1007/978-3-031-70055-2_23. Explores RLS, RLS with FFA (FRLS), and SA on the QAP. (Cit. on pp. 501, 506).

[217] Ulrich Wilhelm Thonemann and Andreas Bölte. *An Improved Simulated Annealing Algorithm for the Quadratic Assignment Problem*. Working Paper. Paderborn, Nordrhein-Westfalen, Germany: School of Business, Department of Production and Operations Research, University of Paderborn, 1994 (cit. on pp. 326–330, 508).

[218] Renato Tinós, Keld Helsgaun, and L. Darrell Whitley. "Efficient Recombination in the Lin-Kernighan-Helsgaun Traveling Salesman Heuristic". In: *15th International Conference on Parallel Problem Solving from Nature (PPSN XV)*. Vol. 1. Sept. 8–12, 2018, Coimbra, Portugal. Ed. by Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and L. Darrell Whitley. Vol. 11101 of Lecture Notes in Computer Science (LNCS). Cham, Switzerland: Springer, 2018, pp. 95–107. ISSN: 0302-9743. ISBN: 978-3-319-99252-5. doi:10.1007/978-3-319-99253-2_8 (cit. on pp. 311–316, 502).

References XXXI

[219] Marco Tomassini. "Complex Networks Analysis of the Energy Landscape of the Low Autocorrelation Binary Sequences Problem". *Physica A: Statistical Mechanics and its Applications* 577:126089, Sept. 1, 2021. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0378-4371. doi:10.1016/j.physa.2021.126089. Lists the LABS as $\mathcal{NP}$-hard. (Cit. on pp. 271–275, 504).

[220] Daan van den Berg, Florian Braam, Mark Moes, Emiel Suilen, and Sandjai Bhulai. "Almost Squares in Almost Squares: Solving the Final Instance". In: *5th International Conference on Data Analytics (DATA ANALYTICS'2016)*. Oct. 9–13, 2016, Venice, Venezia, Veneto, Italy. Ed. by Sandjai Bhulai and Ivana Semanjski. Wilmington, DE, USA: International Academy, Research, and Industry Association (IARIA), 2016, pp. 69–74. ISSN: 2308-4464. ISBN: 978-1-61208-510-4. URL: https://math.vu.nl/~sbhulai/publications/data_analytics2016b.pdf (visited on 2026-10-01). This poses a very interesting 2BP|R|F problem, where "almost squars," i.e., rectangles with sizes $(i, i+1)$, must be placed in an almost square. (Cit. on pp. 424–429, 498, 500).

[221] Clarissa Van Hoyweghen, David Edward Goldberg, and Bart Naudts. "From Twomax To The Ising Model: Easy And Hard Symmetrical Problems". In: *4th Genetic and Evolutionary Computation Conference (GECCO'2002)*. July 9–13, 2002, New York, NY, USA. Ed. by William B. Langdon, Erick Cantú-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant G. Honavar, Günter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, Alan C. Schultz, Julian F. Miller, Edmund K. Burke, and Natasa Jonoska. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 2002, pp. 626–633. ISBN: 978-1-55860-878-8 (cit. on pp. 226–237, 511).

[222] Dieter Vandesande, Christoph Jabs, Jeremias Berg, Johannes Fichte, George Katsirelos, Ruben Martins, André Schidler, and Dominik Schreiber. *MaxSAT Evaluation 2026: Solvers, Benchmarks & Results*. v1. Geneva, Switzerland: zenodo (hosted by European Organization for Nuclear Research (CERN)), July 14, 2026. doi:10.5281/zenodo.21358014. See also https://maxsat-evaluations.github.io. (Cit. on pp. 257–259, 506).

[223] Kristian Verduin, Sarah Louise Thomson, and Daan van den Berg. "Too Constrained for Genetic Algorithms. Too Hard for Evolutionary Computing. The Traveling Tournament Problem". In: *15th International Joint Conference on Computational Intelligence (IJCCI'23)*. Nov. 13–15, 2023, Rome, Italy. Ed. by Niki van Stein, Francesco Marcelloni, H. K. Lam, Marie Cottrell, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2023, pp. 246–257. ISSN: 2184-3236. ISBN: 978-989-758-674-3. doi:10.5220/0012192100003595 (cit. on p. 511).

References XXXII

[224] Petr Vilím, Philippe Laborie, and Paul Shaw. "Failure-Directed Search for Constraint-Based Scheduling". In: *12th International Conference on the Integration of AI and OR Techniques in Constraint Programming (CPAIOR'2015)*. Ed. by Laurent Michel. Vol. 9075 of Theoretical Computer Science and General Issues (LNTCS), sub-series of Lecture Notes in Computer Science (LNCS). Barcelona, Spain: Springer, 2015, pp. 437–453. ISSN: 2512-2010. ISBN: 978-3-319-18007-6. doi:10.1007/978-3-319-18008-3_30. URL: https://www.researchgate.net/publication/280223626 (visited on 2026-09-30) (cit. on pp. 395–400, 504).

[225] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2025. URL: https://thomasweise.github.io/databases (visited on 2025-01-05) (cit. on pp. 458, 501).

[226] Thomas Weise (汤卫思). *Global Optimization Algorithms – Theory and Application*. self-published, 2009. URL: https://www.researchgate.net/publication/200622167 (visited on 2025-07-25) (cit. on pp. 238–242, 501, 505, 506, 508, 509).

[227] Thomas Weise (汤卫思). *jsspInstancesAndResults: Results, Data, and Instances of the Job Shop Scheduling Problem*. San Francisco, CA, USA: GitHub Inc, 2019–2020. URL: https://github.com/thomasWeise/jsspInstancesAndResults (visited on 2025-08-20) (cit. on pp. 386–389, 393, 394, 504).

[228] Thomas Weise (汤卫思). *Programming with C*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2026. URL: https://thomasweise.github.io/programmingWithC (visited on 2026-05-10) (cit. on p. 500).

[229] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), 2024–2026. URL: https://thomasweise.github.io/programmingWithPython (visited on 2025-01-05) (cit. on pp. 457, 507).

[230] Thomas Weise (汤卫思), Raymond Chiong, Jörg Lässig, Ke Tang (唐珂), Shigeyoshi Tsutsui, Wenxiang Chen (陈文祥), Zbigniew "Zbyszek" Michalewicz, and Xin Yao (姚新). "Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem". *IEEE Computational Intelligence Magazine (CIM)* 9(3):40–52, Aug. 2014. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE). ISSN: 1556-603X. doi:10.1109/MCI.2014.2326101. Applies a multitude of algorithms including SLSes, EAs, MAs, ACO, and EDAs to the TSPLib instances. (Cit. on pp. 305–309, 498, 510, 512).

References XXXIII

[231] Thomas Weise (汤卫思), Xinlu Li (李新路), Yan Chen (陈岩), and Zhize Wu (吴志泽). "Solving Job Shop Scheduling Problems without using a Bias for Good Solutions". In: *23rd Genetic and Evolutionary Computation Conference (GECCO'2021), Companion Volume*. July 10–14, 2021, Lille, France. Ed. by Krzysztof Krawiec. New York, NY, USA: Association for Computing Machinery (ACM), 2021, pp. 1459–1466. ISBN: 978-1-4503-8351-6. doi:10.1145/3449726.3463124. Applies FFA to the JSSP. (Cit. on p. 513).

[232] Thomas Weise (汤卫思), Stefan Niemczyk, Hendrik Skubch, Roland Reichle, and Kurt Geihs. "A Tunable Model for Multi-Objective, Epistatic, Rugged, and Neutral Fitness Landscapes". In: *10th Genetic and Evolutionary Computation Conference (GECCO'2008)*. July 12–16, 2008, Atlanta, GA, USA. New York, NY, USA: Association for Computing Machinery (ACM), 2008, pp. 795–802. ISBN: 978-1-60558-130-9. doi:10.1145/1389095.1389252. The original paper introducing the W-Model. (Cit. on pp. 226–237, 512).

[233] Thomas Weise (汤卫思), Alexander Podlich, and Christian Gorldt. "Solving Real-World Vehicle Routing Problems with Evolutionary Algorithms". In: *Natural Intelligence for Scheduling, Planning and Packing Problems*. Ed. by Raymond Chiong and Sandeep Dhakal. Vol. 250 of Studies in Computational Intelligence (SCI). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Sept. 2009. Chap. 2, pp. 29–53. ISSN: 1860-949X. ISBN: 978-3-642-04038-2. doi:10.1007/978-3-642-04039-9_2 (cit. on pp. 20–25).

[234] Thomas Weise (汤卫思), Alexander Podlich, Manfred Menze, and Christian Gorldt. "Optimierte Güterverkehrsplanung mit Evolutionären Algorithmen". *Industrie Management – Zeitschrift für industrielle Geschäftsprozesse* 10(3):37–40, June 2009. Berlin, Germany: GITO mbH Verlag (cit. on pp. 20–25).

[235] Thomas Weise (汤卫思), Alexander Podlich, Kai Reinhard, Christian Gorldt, and Kurt Geihs. "Evolutionary Freight Transportation Planning". In: *Applications of Evolutionary Computing (EvoWorkshops 2009): Proceedings of EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG*. Apr. 15–17, 2009, Tübingen, Baden-Württemberg, Germany. Ed. by Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni A. Caro, Anikó Ekárt, Anna Isabel Esparcia-Alcázar, Muddassar Farooq, Andreas Fink, and Penousal Machado. Vol. 5484 of Theoretical Computer Science and General Issues (LNTCS), sub-series of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Apr. 2009, pp. 768–777. ISSN: 2512-2010. doi:10.1007/978-3-642-01129-0_87 (cit. on pp. 20–25).

[236] Thomas Weise (汤卫思), Mingxu Wan (万明绪), Pu Wang (王璞), Ke Tang (唐珂), Alexandre Devert, and Xin Yao (姚新). "Frequency Fitness Assignment". *IEEE Transactions on Evolutionary Computation* 18(2):226–243, Apr. 2014. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2013.2251885. The article introducing FFA. (Cit. on p. 501).

References XXXIV

[237] Thomas Weise (汤卫思) and Zhize Wu (吴志泽). "Replicable Self-Documenting Experiments with Arbitrary Search Spaces and Algorithms". In: *25th Conference on Genetic and Evolutionary Computation (GECCO'2023), Companion Volume*. July 15–19, 2023, Lisbon, Portugal. Ed. by Sara Silva and Luís Paquete. New York, NY, USA: Association for Computing Machinery (ACM), 2023, pp. 1891–1899. ISBN: 979-8-4007-0120-7. doi:10.1145/3583133.3596306. Introduces the experiment execution and data collection facilities of moptipy. (Cit. on pp. 459, 506).

[238] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路), and Yan Chen (陈岩). "Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value". *IEEE Transactions on Evolutionary Computation* 25(2):307–319, Apr. 2021. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2020.3032090 (cit. on pp. 243–246, 497, 501, 504–506, 509, 511).

[239] Thomas Weise (汤卫思), Zhize Wu (吴志泽), Xinlu Li (李新路), Yan Chen (陈岩), and Jörg Lässig. "Frequency Fitness Assignment: Optimization without Bias for Good Solutions can be Efficient". *IEEE Transactions on Evolutionary Computation* 27(4):980–992, 2023. Los Alamitos, CA, USA: IEEE Computer Society. ISSN: 1089-778X. doi:10.1109/TEVC.2022.3191698. Applies many FFA-based algorithm variants to problems such as OneMax, LeadingOnes, LinHarm, TwoMax, Trap, Jump, Plateau, Ising Model, and N-Queens. (Cit. on pp. 243–246, 497, 501, 504, 509, 511).

[240] Thomas Weise (汤卫思) and Zijun Wu (吴自军). "Difficult Features of Combinatorial Optimization Problems and the Tunable W-Model Benchmark Problem for Simulating Them". In: *20th Genetic and Evolutionary Computation Conference (GECCO'2018)*. July 15–19, 2018, Kyōto, Kyōto-fu, Japan. Ed. by Hernán E. Aguirre and Keiki Takadama. New York, NY, USA: Association for Computing Machinery (ACM), 2018. ISBN: 978-1-4503-5618-3. doi:10.1145/3205651.3208240 (cit. on pp. 226–237, 512).

[241] Thomas Weise (汤卫思), Yuezhong Wu (吴越钟), Raymond Chiong, Ke Tang (唐珂), and Jörg Lässig. "Global versus Local Search: The Impact of Population Sizes on Evolutionary Algorithm Performance". *Journal of Global Optimization* 66(3):511–534, Feb. 2016. London, England, UK: Springer Nature Limited. ISSN: 0925-5001. doi:10.1007/s10898-016-0417-5 (cit. on pp. 498, 510).

[242] L. Darrell Whitley. "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best". In: *3rd International Conference on Genetic Algorithms (ICGA'1989)*. June 1989, Fairfax, VA, USA: George Mason University. Ed. by J. David Schaffer. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 1989, pp. 116–123. ISBN: 978-1-55860-066-9. URL: https://www.researchgate.net/publication/2527551 (visited on 2025-08-08) (cit. on pp. 226–237, 505, 509).

References XXXV

[243] L. Darrell Whitley, Doug Hains, and Adele Howe. "A Hybrid Genetic Algorithm for the Traveling Salesman Problem using Generalized Partition Crossover". In: *11th International Conference on Parallel Problem Solving from Nature (PPSN XI)*. Vol. 1. Sept. 11–15, 2010, Kraków, Poland. Ed. by Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph. Vol. 6238 of Lecture Notes in Computer Science (LNCS). Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, 2010, pp. 566–575. ISSN: 0302-9743. ISBN: 978-3-642-15843-8. doi:10.1007/978-3-642-15844-5_57. URL: https://www.cs.colostate.edu/~howe/cs640/papers/GpxREV.pdf (visited on 2026-09-26) (cit. on pp. 311–316, 502).

[244] James M. Wilson. "Gantt Charts: A Centenary Appreciation". *European Journal of Operational Research* 149(2):430–437, Sept. 2003. Amsterdam, The Netherlands: Elsevier B.V. ISSN: 0377-2217. doi:10.1016/S0377-2217(02)00769-5 (cit. on pp. 336–350, 502).

[245] L. Wong and Lai-Soon Lee. "Heuristic Placement Routines for Two-Dimensional Bin Packing Problem". 5(4):334–341, Dec. 2009. doi:/10.3844/jmssp.2009.334.341. URL: https://www.thescipub.com/pdf/jmssp.2009.334.341 (visited on 2026-10-02). A heuristic for the 2BP|R|F and the 2BP|O|F (cit. on p. 465).

[246] CAO Xiang (曹翔), Zhize Wu (吴志泽), Daan van den Berg, and Thomas Weise (汤卫思). "Randomized Local Search vs. NSGA-II vs. Frequency Fitness Assignment on The Traveling Tournament Problem". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 38–49. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012891500003837 (cit. on pp. 499, 506, 511, 513).

[247] Takeshi Yamada and Ryohei Nakano. "A Genetic Algorithm Applicable to Large-Scale Job-Shop Instances". In: *Parallel Problem Solving from Nature 2 (PPSN-II)*. Sept. 28–30, 1992, Brussels, Belgium. Ed. by Reinhard Männer and Bernard Manderick. Amsterdam, The Netherlands: Elsevier B.V., 1992, pp. 281–290. URL: https://www.researchgate.net/publication/220701684 (visited on 2026-09-30) (cit. on pp. 386–392, 504).

[248] Kinza Yasar and Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., June 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (visited on 2025-01-11) (cit. on p. 501).

References XXXVI

[249] François Yergeau. *UTF-8, A Transformation Format of ISO 10646*. Request for Comments (RFC) 3629. Wilmington, DE, USA: Internet Engineering Task Force (IETF), Nov. 2003. URL: https://www.ietf.org/rfc/rfc3629.txt (visited on 2025-02-05). See Unicode and [113] (cit. on p. 511).

[250] Jiazheng Zeng (曾嘉政), Thomas Weise (汤卫思), Zhize Wu (吴志泽), and Markus Wagner. "When Frequency Fitness Assignment Fails: Trapped States in Frequency-Guided Local Search". In: *28th Genetic and Evolutionary Computation Conference (GECCO'2026)*. July 13–17, 2026, San José, Costa Rica. Ed. by Leonardo Trujillo and Ting Hu. New York, NY, USA: Association for Computing Machinery (ACM), 2026, pp. 233–242. ISBN: 979-8-4007-2487-9. doi:10.1145/3795095.3805098. Explores $(1+1)$ EA, $(1+1)$ EA with FFA ($(1+1)$ FEA), RLS, and FRLS as well as hybrid algorithms on a variety of discrete benchmarks such as OneMax, LeadingOnes, LinHarm, BinInt, TwoMax, Trap, Jump, Plateau, Ising Model, N-Queens and also applies them to the search for LABS. (Cit. on pp. 221–225, 243–246, 497, 501, 503–509, 511).

[251] Chaoyong Zhang, Xinyu Shao, Yunqing Rao, and Haobo Qiu. "Some New Results on Tabu Search Algorithm Applied to the Job-Shop Scheduling Problem". In: *Tabu Search*. Ed. by Wassim Jaziri. London, England, UK: IntechOpen Limited, Aug. 1, 2008. Chap. 8. ISBN: 978-3-902613-34-9. doi:10.5772/5593. URL: https://www.intechopen.com/chapters/4596 (visited on 2026-09-30) (cit. on pp. 395–400, 504, 510).

[252] Rui Zhao (赵睿), Tianyu Liang (梁天宇), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, and Thomas Weise (汤卫思). "Randomized Local Search on the 2D Rectangular Bin Packing Problem with Item Rotation". In: *26th Genetic and Evolutionary Computation Conference (GECCO'2024)*. July 14–18, 2024, Melbourne, VIC, Australia. Ed. by Xiaodong Li and Julia Handl. New York, NY, USA: Association for Computing Machinery (ACM), 2024, pp. 235–238. ISBN: 979-8-4007-0494-9. doi:10.1145/3638530.3654139 (cit. on pp. 20–25, 125–127, 409–423, 430–437, 498, 502, 506).

[253] Rui Zhao (赵睿), Zhize Wu (吴志泽), Daan van den Berg, Matthias Thürer, Tianyu Liang (梁天宇), Ming Tan (檀明), and Thomas Weise (汤卫思). "Randomized Local Search for Two-Dimensional Bin Packing and a Negative Result for Frequency Fitness Assignment". In: *16th International Joint Conference on Computational Intelligence (IJCCI'24)*. Nov. 20–22, 2024, Porto, Portugal. Ed. by Francesco Marcelloni, Kurosh Madani, Niki van Stein, and Joaquim Filipe. Porto, Portugal: SciTePress: Science and Technology Publications, Lda, 2024, pp. 15–26. ISSN: 2184-3236. ISBN: 978-989-758-721-4. doi:10.5220/0012888500003837 (cit. on pp. 20–25, 125–127, 409–423, 430–437, 498, 500, 502, 506).

Glossary I

$(1+1)$ EA The $(1+1)$ EA is a local search algorithm that retains the best solution x_c discovered so far during the search [40,72]. In each step, it applies a unary search operator to this best-so-far solution x_c and derives a new solution x_n. If the new solution x_n is *better or equally good* when compared with x_c, i.e., not worse, then it replaces it, i.e., is stored as the new x_c. If the search space are bit strings of length n, then the $(1+1)$ EA uses a unary search operator that flips each bit independently with probability m/n, where usually $m = 1$. This operator is the main difference to randomized local search (RLS). The $(1+1)$ EA is a special case of the $(\mu+\lambda)$ EA where $\mu = \lambda = 1$.

$(1+1)$ FEA The $(1+1)$ EA with FFA plugged in [238,239,250]. On the discrete bit-string domains, this algorithm has a very specific structure: Its unary search operator flips each bit with probability $1/n$, where n is the dimension of the search space, distinguishing it from the unary operator in FRLS which flips exactly one bit. On other domains, such as permutations, the term is used synonymously with FRLS.

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

BinInt The *Binary Integer Problem* is a benchmark problem for bit-string based black-box search [215,250]. It is very similar to OneMax: It tries to maximize the number of 1 bits in the bit string of a chosen dimension n. Different from OneMax, where each bit has the same weight in the objective function, the bits of BinInt have a exponentially increasing weight: The bit string is treated as a binary-encoded integer whose value should be maximized. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = 2^n - 1 - \sum_{i=1}^{n} 2^{i-1} \cdot x_i$. The problem is unimodal, i.e., has only a single global optimum and no local optimum. It is fully separable, i.e., we can optimize each variable separately.

LinHarm The Linear Harmonic Function is a benchmark problem for bit-string based black-box search [66,72,250]. It is very similar to OneMax: It tries to maximize the number of 1 bits in the bit string of a chosen dimension n. Different from OneMax, where each bit has the same weight in the objective function, the bits of LinHarm have a linear increasing weight. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = \frac{1}{2}n(n+1) - \sum_{i=1}^{n} i \cdot x_i$. The problem is unimodal, i.e., has only a single global optimum and no local optimum. It is fully separable, i.e., we can optimize each variable separately. The $(1+1)$ EA can solve this problem with an expected runtime in $\Theta(n \ln n)$ [66,72].

Glossary II

TSP Suite is a open source benchmarking framework for the TSP written in Java[230]. It uses the instances of the TSPLib and offers both automated experiment execution as well as several metrics to measure runtime. It is available at https://thomasweise.github.io/tspSuite and has been used in many research works, such as [241]. It can be viewed as a predecessor of moptipy.

2BP In the two-dimensional Bin Packing Problem (BP), the goal is to pack rectangles into rectangular bins without overlap and such that their edges are parallel to the edges of the bins [114,148,150]. The 2BP is strongly $\mathcal{NP}$-hard [150]. The variants of this problem include the 2BP|O|F, 2BP|R|F, 2BP|O|G, and 2BP|R|G [149,150]. Different objective function for the 2BP|O|F and 2BP|R|F are presented in [252]. 2BPs have many practical applications, including cutting (wood, glas) [150], packing (transportation, warehouse) [150], and newspaper page layout [206].

2BP|O|F In the oriented 2BP with free cuts, the rectangles to pack have fixed orientations, i.e., cannot be rotated, but can otherwise be arranged freely [149,150,160,253]. Permutation-based search spaces can be used for this problem by using the IBL encoding. The SOTA for this problem comprises [25,30,47,97,182,253].

2BP|O|G In the oriented 2BP with guillotine cuts, the rectangles to pack have fixed orientations, i.e., cannot be rotated, and must be arranged such that they can be extracted by edge-to-edge cuts [149,150].

2BP|R|F In this variant of the 2BP with free cuts, the rectangles to pack can be rotated by 90° if need be and can freely be arranged [21,149,150,252,253]. Besides packing and cutting, this problem also occurs in scheduling and build formation in additive manufacturing [142,187]. The Asqas problem [220] is also a 2BP|R|F. Search spaces based on signed permutations can be used for this problem by using the IBL encoding. The SOTA for this problem comprises [25,47,97,252,253].

2BP|R|G In this variant of the 2BP with guillotine cuts, the rectangles to pack can be rotated by 90° if need be, but must be arranged such that they can be extracted by edge-to-edge cuts [149,150].

Glossary III

2DPackLib is a library of benchmark instances for the two-dimensional Bin Packing Problem (2BP) [115,116].The website of this benchmark set is https://site.unibo.it/operations-research/en/research/2dpacklib [115]. The benchmark set contains two-dimensional instances of the Strip Packing, BP, CSP, and Knapsack problems. The 2BP instances are from three groups: The 10 instances of type *beng* [21] have both bin and item dimensions drawn from uniform distributions, between 20 and 200 items, and the largest bin size is $(40, 25)$.The *class* instances [22,160] are divided into 10 classes based on the bin and item dimensions, the former of which ranges from $(10, 10)$ to $(300, 300)$. Each class is divided into five groups with $\{20, 40, 60, 80, 100\}$ items with 10 instances each. The set *A* [157] offers 43 instances with 13 to 809 items and bin sizes of either $(2750, 1220)$, $(2550, 2100)$, or $(2470, 2080)$.

A* Algorithm The A* Algorithm is an a greedy best-first-first search for finding the shortest path between two locations [105,183]. The algorithm iteratively constructs the shortest path. It maintains a list of current candidate paths. For choosing the next candidate path whose end node should be expanded, it computes a value $f(p)$ of these paths p. $f(p)$ is the sum of the cost $g(p)$ so far incurred by the path as well as a heuristic $h(p)$ predicting the cost from the current end of the path to the goal. $h(p)$ must never overestimate that cost.

ACO *Ant Colony Optimization* is a nature-inspired optimization method for combinatorial problems where solutions are generated by "ants" that move from node to node in a graph choosing edges based on (1) the simulated pheromone on the edges and (2) a per-edge heuristic value [68–70]. If an ant produced a good solution, "pheromone" is distributed over the edges it visited, making it more likely to be re-visited by other ants.

AMLS an *Adaptive Memory-based Local Search* algorithm, i.e., a SLS, for solving MaxSAT problems by Lü and Hao [155]. It inherits the diversification mechanism of WalkSAT.

AP *Alternating Position Crossover* is a binary operator for EAs using a permutation-based representation [134,135]. It simply creates an offspring by selecting alternately the next element of the first parent and the next element of the second parent, omitting the elements already present in the offspring. For example, if parent 1 is $(1, 2, 3, 4, 5, 6, 7, 8)$ and parent 2 is $(3, 7, 5, 1, 6, 8, 2, 4)$, AP yields $(1, 3, 2, 7, 5, 4, 6, 8)$ and exchanging the two parents would yield $(3, 7, 5, 1, 6, 8, 2, 4)$ [134]. A generalized version of this operator will randomly choose a parent in each step, thus increasing the diversity of the results [246].

Glossary IV

Asqas An "almost-square" be a rectangle with size $i \times (i+1)$ with $i \in \mathbb{N}_{\mathbf{1}}$. The *almost-squares-in-almost-squares* problem [220] is a special case of the 2BP|R|F. Here, almost-square rectangles of sizes 1×2, 2×3, ..., $n \times (n+1)$ must be placed into an almost-square bin without waste. This requires that there exists a natural number $p \in \mathbb{N}_{\mathbf{1}}$ such that $p(p+1) = \sum_{i=\mathbf{1}}^{n} i(i+1)$ [220]. There are exacty five Asqas instances [220], namely with bin sizes 1×2 for $n = 1$, $p = 1$, 5×4 for $n = 3$, $p = 4$, 15×16 for $n = 8$, $p = 15$, 55×56 for $n = 20$, $p = 55$, and 119×120 for $n = 34$, $p = 119$.

ATSP The general case for TSPs are *asymmetric Traveling Salesperson Problems*, where the $n \times n$ distance matrix A does not need to be symmetric. In other words, it is possible that $\exists a, b \in 1..n$ with $A[a, b] \neq A[b, a]$. Notice that while such index tuple *may* exist, this is not always or necessarily the case. In this case, one may consider the STSP as a special case of the ATSP, while ATSPs may be considered as synonymous to TSPs.

BP The *Bin Packing Problem* [74,198] is one of the classical problems from Operations Research (OR). The goal is to pack a set of objects into containers, using as few containers as possible. Different flavors of BPs can be distinguished, e.g., by the number of dimensions [56,148] that the objects and containers have (see, e.g., the 2BP), and/or whether the objects can be rotated [253], as well as other constraints and limitations. Both the one- and two-dimensional BP are strongly $\mathcal{NP}$-hard [150].

BRKGA The *Biased Random Key Genetic Algorithm* is a EA with a continuous search space $\mathbb{X} \subset [0, 1]^n$ and a decoding function γ translating this space to something else and biased, uniform, and elitist mating strategies [99,153].The *bias* comes from the fact that one of the parents used for the binary operation is biased to be of higher fitness than the other parent [99]. Often, γ is the Random Keys encoding [16,17] procedure translating $\mathbb{X}$ to permutations of the first n natural numbers [98]. Sometimes, it is actually a MA further refining the translation result with a LS [98]. Sometimes the permutations are then further translated to more complex data structures, such as Gantt charts for the JSSP [98], or packing plans for 2BP 3D packing [97].

C is a programming language, which is very successful in system programming situations [60,191,228].

Concorde is an exact solver for the TSP[5,49,50], which could even solve the largest instance of TSPLib (with 85 900 cities) to optimality.

CSP The *Cutting Stock Problem* is about cutting shapes of a larger shape while reducing waste [150,157]. It is closely related to the BP.

Glossary V



DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases* [225].

DBMS A *database management system* is the software layer located between the user or application and the database (DB). The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB [248].

EA An *evolutionary algorithm* is a metaheuristic optimization method that maintains a population of candidate solutions, which undergo selection (where better solutions are chosen with higher probability) and reproduction (where mutation and recombination create a new candidate solution from one or two existing ones, repectively) [9,226].

EAX *Edge assembly crossover* is a binary operator to be used inside a EA for the TSP [173–175]. It belongs to the SOTA on that field.

FFA *Frequency Fitness Assignment* (中文: 频率适应度分配) is a algorithm plugin for optimization methods applied to discrete or combinatorial problems with not-too-many different possible objective values [236,238,239]. It replaces the objective values in all comparisons with their absolute encounter frequency so far during the search. FFA has successfully been applied to the MaxSAT [238,239], the TSP [143,144], and the QAP [43,216], amongst others. Its properties have been intensely researched on discrete benchmark functions such as OneMax, LeadingOnes, Trap, TwoMax, Jump, and Plateau [238,239,250]. The most common realizations of FFA so far are the $(1+1)$ FEA and FRLS.

FSSP The *Flow Shop Scheduling Problem* is a variant of a JSSP where all products go through the same sequence of machines [101,125].

GA A *Genetic Algorithm* is an EA usually for bit-string based search spaces [9,53,95,163,169,226] (although many researchers use the terms EA and GA synonymously). It is a population-based metaheuristic applying a unary and a binary search operater as well as a selection scheme. The original GA is the Standard Genetic Algorithm (SGA), which today is considered outdated. Modern variants are the Greedy $(2+1)$ GA and the Self-Adjusting $(1+(\lambda,\lambda))$ Genetic Algorithm (Self-Adjusting $(1+(\lambda,\lambda))$ GA) as well as bit-string based $(\mu+\lambda)$ EAs.

Glossary VI

Gantt chart Gantt charts are two-dimensional diagrams illustrating the steps of a process [127,244]. Their horizontal axis represents the time, the vertical axis represents work packages, machines, or workers. In the diagram area, bars located at certain vertical locations indicate that the work package, machine, or worker is processed or works at the time range covered by the bar as projected to the horizontal axis.

GPX *Generalized Partition Crossover* (GPX) is a binary operator for EAs for the TSP [243]. It is *respectful*, because the generated tour inherits all edges common to both parents. It is *transmits alleles*, because it only uses edges that occur in at least one parent. It belongs to the SOTA on the TSP, especially if used in conjunction with LKH [218].

GRASP A *Greedy Randomized Adaptive Search Procedure* is a metaheuristic where each cycle consists of two main components [78,80,81]: First, a starting point is generated using a heuristic method (as opposed to being randomly sampled). Then, a LS is applied to refine the solution. Then, another cycle starts and proceeds in the same way. The overall best solution is retained as result. Many examples of applications of GRASPs are listed in [81]. GRASP in combination with VND has been applied to the 2BP|O|F in [182].

Greedy $(2+1)$ GA The Greedy $(2+1)$ GA [39,40,208,209] is an EA for bit-string based search spaces, i.e., a GA. It was shown to be twice as fast as any mutation-only EA on OneMax [208,209].

IBL The *improved bottom left* heuristic is a one-shot algorithm constructing a solution for 2BP [147]. For the 2BP|O|F problem variant, the input of the IBL algorithm is a permutation of the first n natural numbers, each number identifying one rectangle to be packed. For the 2BP|R|F problem flavor, the permutation elements are signed, where a negative sign indicates a rotation by 90° [147]. In [252,253] it is adapted to a multi-bin situation as follows: The algorithm iterates over the (potentially signed) permutation x and places the rectangles into the packing plan in the prescribed order. It first places the rectangle outside on top of the current bin b, with its bottom-right corner onto the top-right corner of the current bin b. It then moves the rectangle downwards and leftwards as far as possible, prioritizing the downward movement whenever possible. If the rectangle cannot be moved any further, it checks if it is completely contained in the bin b. If yes, it can remain there. Otherwise, a new bin is opened ($b \leftarrow b+1$) and the rectangle is placed at its bottom-left corner. Once a new bin is opened, this bin is used for all further insertions. IBL is suitable as a decoding function γ inside an EA [147]. Historically, it improves upon the BL algorithm [120]: That algorithm cannot represent all possible optimal packings using a permutation, but the IBL can [147].

Glossary VII

ILS An *iterated local search* is a variant of the LS which applies cycles of local search and perturbation [207]. First, an initial solution is generated and refined by the local search. Then the cycle begins: A perturbation, i.e., a large search move that likely destroys a good part of the solution, is applied. The resulting new solution is refined by local search. Then, an acceptance criterion decides whether we should use this or the previous solution to continue in the next cycle. ILS has successfully been applied to the QAP.

Ising Model The Ising models are benchmark problems for bit-string based black-box search [66,82,83,250]. Here, a neighborhood is defined around each bit. The bit needs to have the same value as all bits in the neighborhood structure T. Thus, we try to achieve that all bits have the same value. The objective function $f(x)$ of the minimization variant of this problem family can be defined as $f(x,T) = \sum_{(i,j)\in T} |x_i - x_j|$, where T contains the index pairs of neighboring bits and is defined differently for the Ising model instances Ising 1D and Ising 2D. The problems are highly epistatic, as all variables interact with some other variables.

Ising 1D The 1D is a benchmark problem for bit-string based black-box search [66,83,250]. The bits are arranged in a one-dimensional ring and we try to minimize the number of bits that have different values than their neighbors. Hence, the neighborhood structure T is defined as $T_{\mathbf{1}D} = \{(i+1, (i+1 \bmod n)+1) \mid i \in [0..n-1]\}$. The problems are highly epistatic, as all variables interact with some other variables. The $(1+1)$ EA has an expected runtime in $\mathcal{O}(n^{\mathbf{3}})$ on 1-Dimensional Ising Model (Ising 1D) [83].

Ising 2D The 2D is a benchmark problem for bit-string based black-box search [66,82,250]. The bits are arranged in a two-dimensional torus and we try to minimize the number of bits that have different values than their neighbors. Hence, the neighborhood structure $T = T_{\mathbf{2}D}$ is defined as the set of all $\{(\alpha+\beta N, \gamma+\delta N)\}$ pairs with $\alpha,\beta,\gamma,\delta \in 0..N-1$ and either ()$\gamma = (\alpha+1) \bmod N$ and $\delta = \beta$) or ($\gamma = \alpha$ and $\delta = (\beta+1) \bmod N$) with $n = N^{\mathbf{2}}$. The problems are highly epistatic, as all variables interact with some other variables. On a similar class of problems, the $(1+1)$ EA has an exponential expected runtime [82].

Java is another very successful programming language, with roots in the C family of languages [29,154].

Glossary VIII

JSSP The *Job Shop Scheduling Problem* [28,45,101,137] is one of the most prominent and well-studied scheduling tasks. In a JSSP instance, there are m machines and k jobs [210]. Each job must be processed once by each machine in a job-specific sequence and has a job-specific processing time on each machine. The goal is to find an assignment of jobs to machines that results in an overall shortest makespan, i.e., the schedule (Gantt chart) which can complete all the jobs in the shortest time. The JSSP is $\mathcal{NP}$-complete [42,136,137]. The 8 most common sets of JSSP benchmark instances are *abz* [1], *dmu* [57], *ft* [84], *la* [139], *orb* [6], *swv* [205], *ta* [210], and *yn* [247].Their data can be found (sometimes partially) in several repositories, such as the OR-Library [18,19], [201], and [227], the latter of which also contains many results from existing works. Encodings based on permutations with repetition are proposed in [26,27,90,200] and used in [238]. The SOTA of the JSSP includes [98,108,181,184,202,224,251].

Jump The Jump problem is a benchmark problem for bit-string based black-box search [72,87,250]. Like the OneMax, it tries to maximize the number of 1 bits in a bit string of chosen length n. However, the solutions representing the last ω bits around the optimum to flip are deceptive, meaning that they are actually worse and getting worse than surrounding solutions. The objective function $f(x)$ of the minimization variant is $n - \|x\|_1$ if $\|x\|_1 = n$ or $\|x\|_1 \leq n - \omega$ hold, and $\omega + \|x\|_1$ otherwise, where $\|x\|_1 = \sum_{i=1}^{n} x_i$. This means that, the larger ω, the larger the deceptive basin around the optimum that an algorithm needs to cross. An $(1+1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\Theta(n^\omega + n \ln n)$ [72].In experiments, the $(1+1)$ FEA achieves polynomial runtime on Jump, seemingly in $\mathcal{O}(n^2 \ln n)$ [238,239,250].

LABS *Low-Autocorrelation Binary Sequences* are bit strings of length n whose autocorrelation is as low as possible [31,33,34,58,59,88,161,164,171,179,190]. This means that sub-sequences of any length $1 \leq k < n$ should ideally never repeat. The objective function $f : \{0,1\}^n \mapsto \mathbb{N}_0$ represents the energy of a bit string x and is subject to minimization. We can define it as $f(x) = \sum_{k=1}^{n-1} \left(\sum_{i=0}^{n-1-k} s_i \cdot s_{i+k} \right)^2$ with $s_j = 2x_j - 1$. Such sequences are needed for communication engineering [33,34,171], cryptography[58,171], as well as modulation pulses in radar and sonar ranging[179]. The LABS is $\mathcal{NP}$-hard [219]. The SOTA on LABS contains [31,34,58,59,171].

Glossary IX

LeadingOnes The LeadingOnes problem is a benchmark problem for bit-string based black-box search [196,242,250]. It tries to maximize the number of *leading* 1 bits in the bit string of a chosen dimension n. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = n - \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$. Different from OneMax, the contribution to the objective value of the bit at index i thus depends on the state of all the bits *before* this index. The problem is unimodal, i.e., has only a single global optimum and no local optimum. Different from OneMax, it is not separable, i.e., we can optimize each variable separately. The expected runtime of $(1+1)$ EA on LeadingOnes is in $\Theta(n^2)$ [72]. The average runtime $(1+1)$ FEA on LeadingOnes seems to be slower by factor proportional to n, i.e., seems to be in $\mathcal{O}(n^3)$ [238].

LK The *Lin-Kernighan* (LK) algorithm [146] is a SLS method for the TSP. The LK algorithm performs iterative k-opt moves, i.e., in each step, it constructs the two sets X and Y of k edges such that removing the edges in X from the current tour T and then adding the edges from Y to T results in a better tour.The LK algorithm is one of the most classical approaches to the TSP.

LKH The *Lin-Kernighan-Helsgaun* algorithm [50,106,107] is a SLS method for the TSP based on the Lin-Kernighan (LK) algorithm [146]. LKH-1 was proposed in [107] and LKH-2 in [106]. It holds the record in the *World TSP* [51] as of 2025.

LS A *local search* is an algorithm where a single "current" candidate solution is maintained and new solutions are selected from its neighborhood in each step. Most LSes are randomized, i.e., stochastic local search (SLS).

MA A *Memetic Algorithm* is an EA where each new solution is refined by a LS before it enters the population [226]. As LS, we can use, e.g., RLS, but TS [44,85] is maybe a better choise. Some SOTA algorithms in some domains may be classified as MAs, e.g., Lin-Kernighan Helsgaun (LKH) basically uses a population that is refined via a local search. MAs also worked well on the JSSP [44,108] and the TSP [85].

MaxSAT The goal of satisfiability problems is to find an assignment for n Boolean variables that make a given Boolean formula $F : \{0,1\}^n \mapsto \{0,1\}$ become true. In the *Maximum Satisfiability* problem (MaxSAT) [109], F is given in conjunctive normal form, i.e., the variables appear either directly or negated in m "or" clauses, which are all combined into one "and." The objective function $f(x)$, subject to minimization, computes the number of clauses which are false under the variable setting x. If $f(x) = 0$, then all clauses of F are true, which solves the problem. The MaxSAT problem is $\mathcal{NP}$-complete [48] and exhibits low epistasis but deceptiveness [193], making it hard for local searches. Typical benchmark instances can be found in SATLIB, which contains smaller-scale instances, and the up-to-date MSE, providing SOTA benchmarks, algorithms, and results.

Glossary X

metaheuristic
: A *metaheuristic* is an optimization algorithm which performs a cycle of trial and error during which candidate solutions are sampled from the solution space $\mathbb{Y}$ and evaluated using an objective function $f : \mathbb{Y} \mapsto \mathbb{R}$ [93,226]. The results of the objective function are used to decide which solution is promising. The neighborhoods around promising solutions *or* similarly-structured solutions are then sampled in the next cycle. Prominent metaheuristics are SLSes, SA, TS, and EAs.

moptipy
: is the *Metaheuristic Optimization in Python* library [237]. It has been used in several different research works, including [43,143–145,210,246,252,253]. Algorithms implemented by moptipy include EAs, $(1 + 1)$ EA, RLS, SA, and the $(1 + 1)$ FEA, amongst others. Learn more at https://thomasweise.github.io/moptipy and https://thomasweise.github.io/moptipyapps.

MSE
: The *MaxSAT Evaluation* is a regularly updated set of benchmark instances for the MaxSAT [117]. It also includes solver software and benchmark results [222].

N-Queens
: The N-Queens Problem is a benchmark problem for bit-string based black-box search [66,250]. The goal is to arrange N queens on an $N \times N$ chessboard so that none can attack the other. The search space has thus dimension $n = N^2$, with each bit standing for a field and encoding whether or not a queen is placed there. The objective function $f(x)$ of the minimization variant is $N - \left(\sum_{i=1}^{n} x_i\right) - N \sum_{\xi} \max\left\{0, Q_\xi(x) - 1\right\}$ where $Q_\xi(x)$ is the number of queens in the horizontal, vertical, or diagonale ξ. The problem is quite epistatic.

NuWLS
: a SLS algorithm by Chu, Cai, and Luo [46] for solving the (Weighted) Partial MaxSAT problem.

OneMax
: OneMax is the simplest possible benchmark problem for bit-string based black-box search [73,172,250]. The goal is to maximize the number of 1 bits in the bit string of a chosen dimension n. The objective function $f(x)$ of the minimization variant of this problem can be defined as $f(x) = n - \sum_{i=1}^{n} x_i$. The problem is unimodal, i.e., has only a single global optimum and no local optimum. It is fully separable, i.e., we can optimize each variable separately. The expected runtime of $(1 + 1)$ EA on OneMax is in $\Theta(n \ln n)$ [62,172]. The average runtime $(1 + 1)$ FEA on OneMax seems to be slower by factor proportional to n, i.e., seems to be in $\mathcal{O}(n^2 \ln n)$ [238].

Glossary XI

OR Operations Research (or Operational Research) is the application of sciences such as mathematics and computer science to the management and organization of systems, organizations, enterprises, factories, or projects. It encompasses the development and application of problem-solving methods and techniques (such as mathematical optimization, simulation, queueing theory and other stochastic models) with the goal to improve decision-making and efficiency [54].

OR-Library is a library of benchmark instances for various OR problems [18,19] located at https://people.brunel.ac.uk/~mastjjb/jeb/info.html. It contains instances of the JSSP, 1D- and 2D-Packing, Knapsack, periodic TSPs, amongst others.

Plateau The Plateau problem is a benchmark problem for bit-string based black-box search [4,250]. It is structured exactly like Jump, but instead of being deceptive, the last ω bits to flip around the optimum are neutral. The objective function $f(x)$ of the minimization variant is $n - \|x\|_1$ if $\|x\|_1 = n$ or $\|x\|_1 \leq n - \omega$ hold, and ω otherwise, where $\|x\|_1 = \sum_{i=1}^{n} x_i$. This means that, the larger ω, the larger the neutral plane around the optimum that an algorithm needs to cross. An $(1+1)$ EA has an expected runtime of $\Theta(n^\omega)$ on plateaus [4]. The average runtime of the $(1+1)$ FEA on Plateaus is slower by a factor seemingly proportional in n.

PostgreSQL An open source object-relational DBMS [79,178,189,214]. See https://postgresql.org for more information.

Python The Python programming language [112,140,156,229], i.e., what you will learn about in our book [229]. Learn more at https://python.org.

QAP The *Quadratic Assignment Problem* is an optimization problem where the goal is to assign a set of n facilities to a set of n locations [36,43,130,151]. Such an assignment can be represented as a permutation x of the first n natural numbers, where x_i specifies the location where facility i should be placed. For each QAP instance, a distance matrix A is given, where $A[a,b]$ specifies the distance from location a to location b, as well as a flow matrix B, where $B[i,j]$ is the amount of material flowing (e.g., daily) from facility i to facility j. The objective function $f(x)$ then rates a permutation x as $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} A[x_i, x_j] * B[i,j]$. The QAP is $\mathcal{NP}$-complete [71,197] and even among this problem class, is considered as particularly hard for metaheuristics [20,36,151,207]. The QAP has applications such as building layout [52,76,131], keyboard layout [38], circuit design [77], wiring [204], and scheduling [203]. Typical benchmark instances can be found in QAPLIB. SOTA algorithms for the QAP include [20,85,162,168,212,213].

Glossary XII

QAPLIB is a library of 136 benchmark instances for the Quadratic Assignment Problem (QAP) [35,37], 28 of which have not yet been solved to optimality [3] (at least by 2023).

Random Keys encoding is a decoding procedure $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ that maps a continuous search space $\mathbb{X} \subset [0,1]^n$ which is a subset of the n-dimensional real numbers to the space $\mathbb{Y}$ of permutations of the first n natural numbers [16,17]. For example, on $n = 5$, the vector $x = (0.49, 0.91, 0.33, 0.75, 0.51)$ is translated to the permutation $y = (3, 1, 5, 4, 2)$, because $x_{\mathbf{3}} = 0.33$ has the smallest value, putting 3 at the beginning, $x_{\mathbf{1}} = 0.49$ the second-smallest, placing 1 second, ..., and $x_{\mathbf{2}} = 0.91$ being the largest value meaning that the last number in y will be 2 [17]. The advantage of the encoding is that we can search in $\mathbb{X}$ applying arbitrary mathematical operators, crossover, and mutation procedures, but will always get correct permutations. The disadvantage is that $|\mathbb{X}| \gg |\mathbb{Y}|$ and that there actually is no meaning underpinning moves in $\mathbb{X}$ in relation to their projected result in $\mathbb{Y}$.

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

FRLS The RLS with FFA plugged in [250]. On the discrete bit-string domains, this algorithm has a very specific structure: Its unary search operator flips exactly one bit, distinguishing it from the unary operator in the $(1 + 1)$ FEA, which flips each bit with probability $1/n$. On other domains, such as permutations, we use the term synonymously with $(1 + 1)$ FEA.

RTS *Reactive Tabu Search* is a variant of TS where the length of the tabu list is adapted to the problem and to the current evolution of the search, and an escape strategy for diversifying the search is used when that is not sufficient [12–15]. It has successfully been applied to a wide range of problems, from the QAP [13,14], to Knapsack problems [12,14], the N-K landscape [12] to the training of neural networks [15].

SA *Simulated Annealing* is a SLS that sometimes accepts a worse solution [41,119,126,186,226]. The probability to do so decreases over time and with the difference in objective values, i.e.,is the lower the worse the new solution is. SA performed well on a variety of problems, including the QAP [166,217] and the JSSP [181].

Glossary XIII

SATLIB is a library of benchmark instances for the Maximum Satisfiability (MaxSAT) available at https://www.cs.ubc.ca/~hoos/SATLIB/index-ubc.html [110,111].

Self-Adjusting $(1 + (\lambda, \lambda))$ GA The Self-Adjusting $(1 + (\lambda, \lambda))$ GA is a theory-inspired EA for bit string-based search spaces [40]. It has linear runtime on the OneMax problem [61].

SGA The Standard Genetic Algorithm [9,53,95,163,169,226] was the first EA. It maintains a population of solutions and applies mutation and crossover to generate offspring solutions. It uses fitness proportionate selection to choose which solutions should "survive" into the next generation, which today is considered a very bad design choice, see, e.g., [242].

SLS *Stochastic local search* is a local search (LS), one of the simplest metaheuristic search paradigms [109]: A single "current" candidate solution is maintained and new solutions are *stochastically* sampled from its neighborhood, i.e., in a way that involves random numbers. If these solutions are deemed interesting based on their objective value, they replace the "current". The cycle continues. Basic examples are RLS and $(1 + 1)$ EA. More complex examples that add mechanisms to escape local optima on top of the SLS paradigm are TS and SA.

SOTA State-of-the-Art, i.e., the best-performing methods for a field or problem type

STSP A *symmetric Traveling Salesperson Problem* is a special case of the TSP, where the $n \times n$ distance matrix A is symmetric, i.e., $A[a, b] = A[b, a] \forall a, b \in 1..n$.

Trap The Trap problem is a benchmark problem for bit-string based black-box search [72,176,250]. It is structured exactly like OneMax, however the worst and the best solution are swapped. This makes the problem highly deceptive, as following the "gradient" in the search space leads the search away from the global optimum. For bit strings of length n, the objective function $f(x)$ of the minimization variant is 0 if $\|x\|_1 = 0$ and $n - \|x\|_1 + 1$ otherwise, where $\|x\|_1 = \sum_{i=1}^{n} x_i$. An $(1 + 1)$ EA will converge to the resulting local optimum and thus have an expected runtime of $\Theta(n^n)$ [72]. In experiments, the $(1 + 1)$ FEA achieves polynomial runtime on Trap, seemingly in $\mathcal{O}(n^2 \ln n)$ [238,239,250].

Glossary XIV

TS *Tabu Search* is one of the most successful SLSes [91,92,94,121]. In order to avoid returning to previous solutions, it declares certain solutions or search moves as "tabu." For example, if a bit in a discrete problem has been flipped by the search operator, it may be forbidden to flip the bit again for some time, thus preventing the algorithm from getting trapped in a cycle. This rule may be overridden if a new best-so-far solution is discovered that outperforms all previous solutions, in which case it would be accepted (this is the so-called aspiration criterion). TS has performed well on the QAP [13,14,165,168,211,212] and on the JSSP [177,184,185,202,251].

TSP In an instance of the *Traveling Salesperson Problem*, also known as *Traveling Salesman Problem*, a set of n cities or locations as well as the distances between them are given [5,103,138,143,230,241]. The goal is to find the shortest round-trip tour that starts at one city, visits all the other cities one time each, and returns to the origin. The TSP is one of the most well-known $\mathcal{NP}$-hard combinatorial optimization problems [89,103,132]. The TSP has many applications[5], including in transportation, logistics, (circuit) plotting and drilling problems[102], telescope aiming, and genome sequencing. Solutions are often encoded as permutations x of the first n natural numbers, where each city gets a number and the order of numbers in the permutation corresponds to the order of cities in the tour. The distances between the n cities are usually given in a $n \times n$ distance matrix A, where $A[a, b]$ is the distance from city a to city b with $a, b \in 1..n$. Then, the objective function subject to minimization is $f(x) = A[x_n, x_1] + \sum_{i=1}^{n-1} A[x_i, x_{i+1}]$. Typical benchmark instances can be found in TSPLib [194,195], the 8th DIMACS challenge[122–124], and the *World TSP* repository[51].

TSPLib is a library of benchmark instances for the Traveling Salesperson Problem (TSP) [194,195]. Its original URL was available until 2025 and then became defunct. The repository https://github.com/shredderzwj/TSPLIB contains the data. Additionally, you may also check https://github.com/thomasWeise/tspSuite, which contains all the TSPLib instances as well as Java code to load them. The TSPLib contains 110 instances of the symmetric Traveling Salesperson Problems (STSPs) with scales n ranging from 14 to 85 900, but 93 instances have less than 2000 nodes. There also are 19 asymmetric Traveling Salesperson Problems (ATSPs) with scales $n \in 17..443$.

Glossary XV



TTP The *Traveling Tournament Problem* is the combinatorial optimization problem of both efficiently and fairly organizing a tournament of n teams that play against each other in a pairwise fashion [75,246]. The efficient part boils down to arranging the games such that the total travel length is short, which is somewhat similar to the classical TSP. Initially, each team is at its home location. On each day, a team needs to travel if its scheduled game is not at its present location. On the last day, each team may need to travel back home unless their last game is a home game. The total travel length sums up the lengths of all travels over all teams. The fair part is represented in several constraints, such as doubleRoundRobin, compactness, maxStreak, and noRepeat. The TTP is $\mathcal{NP}$-hard [223].

TwoMax The TwoMax problem is a benchmark problem for bit-string based black-box search [87,221,250]. It has a global and a local optimum, both of about the same size, at opposite ends of the search space. For bit strings of length n, the objective function $f(x)$ of the minimization variant is 0 if $\|x\|_1 = n$ and $1 + n - \max\{\|x\|_1, n - \|x\|_1\}$ otherwise, where $\|x\|_1 = \sum_{i=1}^{n} x_i$. An $(1+1)$ EA has a 50/50 chance to converge to the local or global optimum and thus has an expected runtime in $\Theta(n^n)$ [86,87]. In experiments, the $(1+1)$ FEA achieves polynomial runtime seemingly in $\mathcal{O}(n^2 \ln n)$ on TwoMax [238,239,250].

URI A *Uniform Resource Identifier* is an identifier for an abstract or physical resource in the internet [249]. It can be a URL, a name, or both. URIs are supersets of URLs. The connection strings of the PostgreSQL DBMS are examples of Uniform Resource Identifiers (URIs).

URL A *Uniform Resource Locator* identifies a resource in the World Wide Web (WWW) and a way to obtain it by describing a network access mechanism. The most notable example of URLs is the text you write into web browsers to visit websites [24]. URLs are subsets of URIs.

VND *Variable Neighborhood Descend*; see VNS. VND in combination with GRASP has been applied to the 2BP|O|F in [182].

Glossary XVI

VNS *Variable Neighborhood Search* is a metaheuristic framework combining a systematic change of neighborhood with LS [104,170]. A *neighborhood* structure in LS defines which solutions can be reached from a given solution within one search step. Usually, one uses a single unary operator and, hence, has a single neighborhood structure. However, for many problems, we can easily design many operators. For permutations, for example, one operator may swap two elements, another one may reverse a section of the permutation, and yet another one, while yet another one may rotate a subsequence one step to the left or right [230]. In the VNS framework, a multiple neighborhoods are investigated in a systematic fashion. Assume that we have a list of neighborhood structures. Variable Neighborhood Descend (VND) works in a deterministic fashion. It finds the best solution that can be reached in one step from the current solution under the current neighborhood. If this is better than the current solution, we accept it and continue with the first neighborhood. Otherwise, we continue with the next neighborhood in the list. In Basic VNS, instead of picking the best move from the current neighborhood, the first improving move is used. Also, at the beginning of each cycle, a random point from the current neighborhood around the current solution is chosen as starting point.

W-Model The W-Model is a tunable benchmark function for bit-string based black-box search [232,240].It allows you to adjust the amount of ruggedness, deceptiveness, epistasis, and uniform neutrality in the search space.

WalkSAT a SLS algorithm by Selman, Kautz, and Cohen [199] for solving the MaxSAT problem.

WWW World Wide Web [23,55]

$\circ^\star$ the superscript $\star$ symbol denotes that $\circ$ is a global optimum, i.e., there is no better solution than $\circ$.

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

A the $n \times n$ distance matrix. $A[a, b]$ indicates the distance between locations a and b with $a, b \in 1..n$. A distance matrix can be symmetric, in which case $A[a, b] = A[b, a]\ \forall a, b \in 1..n$. Distance matrices represent the problem instance data in TSPs and part of the instance data of QAPs.

Glossary XVII

B the $n \times n$ flow matrix. $B[a, b]$ is the flow of goods between locations a and b with $a, b \in 1..n$. Flow matrices are one half of the problem instance data of QAPs.

$f(y)$ The *objective function* $f : \mathbb{Y} \mapsto \mathbb{R}$ can compute a cost or rating for each candidate solution y in the solution space $\mathbb{Y}$. The smaller this value, the better the solution, i.e., in the context of this work, objective functions are subject to minimization. Objective functions are not necessarily continuous and it is also quite common that they have integer or natural numbers as results, e.g., we often have $f : \mathbb{Y} \mapsto \mathbb{N}_0$ instead of $f : \mathbb{Y} \mapsto \mathbb{R}$.

$\gamma(x)$ The decoding function $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ translates point from the search space $\mathbb{X}$ to one point y in the solution space $\mathbb{Y}$. It is a very common case that $\mathbb{X} = \mathbb{Y}$ and that the decoding function γ is the identity mapping. Examples for encodings that translate from a simple search space $\mathbb{X}$ to a complex solution data structure $\mathbb{Y}$ are the IBL algorithm that translates (potentially signed) permutations to two-dimensional packings on the 2BP, the permutation-based encoding of Gantt charts for the JSSP from [26,27,90,200,231], or the permutation-based encoding of game plans for the Traveling Tournament Problem (TTP) from [246]. Encodings can also translate between types of search spaces. The Random Keys encoding, for example, translates from $[0, 1]^n$ to permutations of the first n natural numbers..

k the number of jobs in an instance of the JSSP.

m the number of machines in an instance of the JSSP.

Glossary XVIII

n We usually use the value $n \in \mathbb{N_0}$ to denote the *scale*, i.e., the size of the problem to solve. This notion is very general in computer science and not limited to optimization. For example, if we sort a list or search in a list, the scale n is the length of the list, i.e., the number of elements in it. In a TSP, this may be the number of cities. In a discrete benchmark problem such as OneMax or Trap, this may be the number of bits over which the problem is defined. In a QAP, this may be the number of facilities and locations. In a MaxSAT problem, this may be the number of variables that are input to the Boolean formula that we want to become true. There are also problems where the scale is not defined straightforwardly like this. For example, in a JSSP, we have a number of machines and a number of jobs, which together define the size of the problem. But even then, the scale n relates to the size of the solution space $\mathbb{Y}$, and, hence, also of the search space $\mathbb{X}$, which usually grow exponentially with n. And this is already the main issue: The larger n, the (exponentially) larger the number of possible solutions and thus, the larger the runtime of algorithms..

$\mathbb{N_0}$ the set of the natural numbers *including* 0, i.e., 0, 1, 2, 3, and so on. It holds that $\mathbb{N_0} \subset \mathbb{Z}$ and that $\mathbb{N_0} = \mathbb{N_1} \cup \{0\}$.

$\mathbb{N_1}$ the set of the natural numbers *excluding* 0, i.e., 1, 2, 3, 4, and so on. It holds that $\mathbb{N_1} \subset \mathbb{Z}$ and that $\mathbb{N_1} = \mathbb{N_0} \setminus \{0\}$.

$\mathcal{NP}$ is the class of computational problems that can be solved in polynomial time by a non-deterministic machine and can be verified in polynomial time by a deterministic machine (such as a normal computer) [100].

$\mathcal{NP}$-complete A decision problem is $\mathcal{NP}$-complete if it is in $\mathcal{NP}$ and all problems in $\mathcal{NP}$ are reducible to it in polynomial time [100,192]. A problem is $\mathcal{NP}$-complete if it is $\mathcal{NP}$-hard and if it is in $\mathcal{NP}$.

$\mathcal{NP}$-hard A problem is $\mathcal{NP}$-hard if all problems in $\mathcal{NP}$ are reducible to it in polynomial time [100]. Algorithms that guarantee to find the correct solutions of $\mathcal{NP}$-hard problems [42,48,137] have a worst-case runtime that grows exponentially with the problem scale.

$\Omega(g(x))$ If $f(x) = \Omega(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $f(x) \geq c * g(x) \geq 0 \forall x \geq x_0$ [128,129]. In other words, $\Omega(g(x))$ describes a lower bound for function growth.

$\mathcal{O}(g(x))$ If $f(x) = \mathcal{O}(g(x))$, then there exist positive numbers $x_0 \in \mathbb{R}^+$ and $c \in \mathbb{R}^+$ such that $0 \leq f(x) \leq c * g(x) \forall x \geq x_0$ [8,128,129,133]. In other words, $\mathcal{O}(g(x))$ describes an upper bound for function growth.

$\Theta(g(x))$ If $f(x) = \Theta(g(x))$, then $f(x) = \mathcal{O}(g(x))$ and $f(x) = \Omega(g(x))$ [128,129]. In other words, $\Theta(g(x))$ describes an exact order of function growth.

Glossary XIX

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

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

$\mathbb{X}$ The search space $\mathbb{X}$ is the space through which an optimization algorithm navigates in order to find good solutions. The search operators of metaheuristics are used to sample points $x \in \mathbb{X}$, for example. On many benchmark problems, the objective functions f directly evaluate the elements $x \in \mathbb{X}$, i.e., are defined as $f : \mathbb{X} \mapsto \mathbb{R}$. However, very often these elements x are not the same data structures that the user is interested in and/or that the objective functions evaluate: It is common that the search space $\mathbb{X}$ is a simple and plain data structure that can be tackled with standard operations or mathematical tools, such as real vectors $\mathbb{R}^n$, permutations, or bit strings, whereas the solution space $\mathbb{Y}$ can be something more complicated. Then, the user only cares about the solutions $y \in \mathbb{Y}$ and not about the search space $\mathbb{X}$. A decoding $\gamma : \mathbb{X} \mapsto \mathbb{Y}$ is employed to translate the $x \in \mathbb{X}$ to $y = \gamma(x)$ with $y \in \mathbb{Y}$.

x a point in the search space $\mathbb{X}$.

$\mathbb{Y}$ the set of candidate solutions of an optimization problem is called the *solution space* $\mathbb{Y}$. The solution space corresponds to the data structure holding the candidate solutions y. These are the elements that are passed to the user after the optimization process is completed. They are also the parameters of the objective function(s) $f : \mathbb{Y} \mapsto \mathbb{R}$.

y a candidate solution $y \in \mathbb{Y}$ is one possible solution to an optimization problem. After the optimization process is completed, one or multiple such solutions are passed to the user.

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