# FORMAL MODELING

## Algebra and Logic in Mathematical Modeling

Josef Schicho  Wolfgang Schreiner  Wolfgang Windsteiger
Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
*FirstName.LastName*@risc.jku.at

JOHANNES KEPLER
UNIVERSITY LINZ

1. Introduction

2. Mathematical Models in Kinematics and Mechanics

3. Logical Models of Problems and Computations

4. Modeling Problems in Geometry and Discrete Mathematics
   - Problem 1: How to get from $A$ to $B$?
   - Problem 2: How to efficiently use resources?

5. Organization

# Introduction

What is this course about?

- Application of techniques from symbolic computation.
  - Rooted in computer algebra, algebraic geometry, computational logic.
  - Focus is on correct formalization, precise analysis, exact solving (rather than on fast but numerically approximated computations).
- Modeling and analysis of problems in various application domains.
  - Kinematics and mechanics, geometry and discrete mathematics, programs and computational systems, ...
- Theoretical frameworks and practical tools.
  - Computer algebra and automated reasoning software.

Prerequisites for the algorithmization and automation of mathematics.

# Contents

What are you going to see?

- Mathematical Models in Kinematics and Mechanics.
  - Josef Schicho.
- Logic Models of Problems and Computations.
  - Wolfgang Schreiner.
- Modeling Problems in Geometry and Discrete Mathematics.
  - Wolfgang Windsteiger.

A (non-exhaustive) selection of topics pursued at the RISC institute.

1. Introduction

2. Mathematical Models in Kinematics and Mechanics

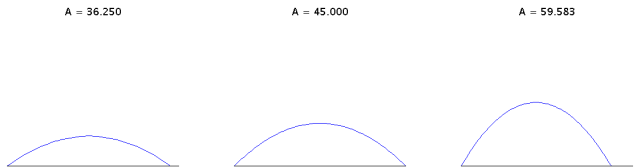
3. Logical Models of Problems and Computations

4. Modeling Problems in Geometry and Discrete Mathematics
   - Problem 1: How to get from $A$ to $B$?
   - Problem 2: How to efficiently use resources?

5. Organization

# The Spear Thrower Problem

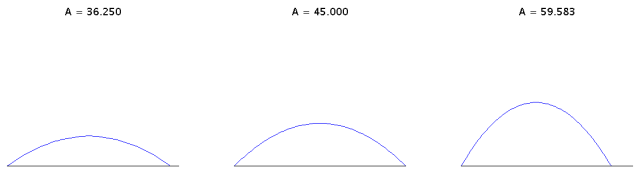
A good spear thrower can give the spear an initial speed of 70km/h. What is the best throwing angle to throw the spear as far as possible?




Idealizations: air resistance can be neglected. The height difference between beginning and end point can be neglected. Only 2D problem because the horizontal direction is irrelevant.

# The Spear Thrower Problem

A good spear thrower can give the spear an initial speed of 70km/h. What is the best throwing angle to throw the spear as far as possible?

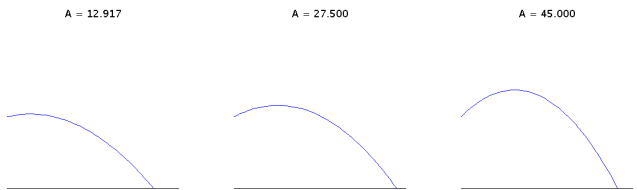



Idealizations: air resistance can be neglected. The height difference between beginning and end point can be neglected. Only 2D problem because the horizontal direction is irrelevant.

Answer: the optimal angle is $45°$. The width is then 38,5m.

# The Problem of the Shot Putter (KugelstoSSer(in))

A good shot putter can give the ball an initial speed of 30 km/h. The ball has then a height of 2m (erased hand). What is the best throwing angle to throw the ball as far as possible?




Still a 2D problem, and still the air resistance can be neglected. But the height difference is now too big to be neglected.

# First Model

For simplicity, we assume that the acceleration of gravity, the initial speed, and the height is 1. Let $\alpha$ be the throwing angle. The trace of the ball in the vertical $xy$-plane is given by

$$t \mapsto (x_\alpha(t), y_\alpha(t)) = \left(\cos(\alpha)t, h + \sin(\alpha)t - \frac{t^2}{2}\right).$$

# First Model

For simplicity, we assume that the acceleration of gravity, the initial speed, and the height is 1. Let $\alpha$ be the throwing angle. The trace of the ball in the vertical $xy$-plane is given by

$$t \mapsto (x_\alpha(t), y_\alpha(t)) = \left(\cos(\alpha)t, h + \sin(\alpha)t - \frac{t^2}{2}\right).$$

The function $t \mapsto y_\alpha(t)$ has two zeroes, one positive and one negative. The positive one is the time of impact:

$$t_a = \sin(\alpha) + \sqrt{(\sin\alpha)^2 + 2}.$$

# Its Getting Complicated ...

The objective function is $x(t_a)$ as a function of $\alpha$:

$$f : \alpha \mapsto \sin(\alpha)\cos(\alpha) + \sqrt{(\sin\alpha)^2 + 2}\cos(\alpha).$$

# Its Getting Complicated ...

The objective function is $x(t_a)$ as a function of $\alpha$:

$$f : \alpha \mapsto \sin(\alpha)\cos(\alpha) + \sqrt{(\sin\alpha)^2 + 2}\cos(\alpha).$$

Now we have to compute the derivative $f'$ and equate it to zero, to get an equation for $\alpha$.

# Its Getting Complicated ...

The objective function is $x(t_a)$ as a function of $\alpha$:

$$f : \alpha \mapsto \sin(\alpha)\cos(\alpha) + \sqrt{(\sin\alpha)^2 + 2}\cos(\alpha).$$

Now we have to compute the derivative $f'$ and equate it to zero, to get an equation for $\alpha$.

With pencil and paper, this task is already cumbersome. A computer algebra system helps, but even they (Maple) have problems solving the equation with roots and trigonometric functions. In any case, the computation is complicated.

# Second Model

Instead of throwing the ball, we imagine a fountain at the same heigth. Its nozzle simultanuously splashes water drops with initial velocity 1 in all directions (still in 2D, but more or less steep). We compute the set $W$ of all points that are reached by water drops.

$$W = \{(a, b) \mid \exists t, \alpha : \cos(\alpha)t = a, 1 + \sin(\alpha)t - \frac{t^2}{2} = b\}.$$

# Second Model

Instead of throwing the ball, we imagine a fountain at the same heigth. Its nozzle simultanuously splashes water drops with initial velocity 1 in all directions (still in 2D, but more or less steep). We compute the set $W$ of all points that are reached by water drops.

$$W = \{(a, b) \mid \exists t, \alpha : \cos(\alpha)t = a, 1 + \sin(\alpha)t - \frac{t^2}{2} = b\}.$$

The first equation gives an expression for $t$, which we plug into the second equation:

$$\cos(\alpha)^2(b-1) - \cos(\alpha)\sin(\alpha)a + \frac{a^2}{2} = 0.$$

# The Wet Area

This gives again an equation with trigonometric functions in $\alpha$, but it has no roots and is easier to solve. Assuming that the cosine is not zero, it is equivalent to

$$\frac{a^2}{2}(\tan(\alpha))^2 - a\tan(\alpha) + \frac{a^2}{2} + b - 1 = 0.$$

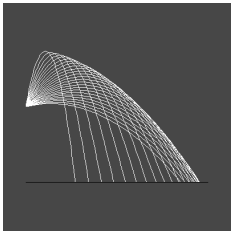
This quadratic equation for $\tan(\alpha)$ has a solution if and only if the discriminant is positive:

$$-a^2 - 2b + 3 \geq 0$$

The set $W$ of wet spots is descriped by the inequality above. It is deliminated by a parabola which open towards the bottom.

# Creative Modeling Pays Off

The largest achievable width is determined by intersecting the parabola $-a^2 - 2b + 3 = 0$ with the coordinate axes $b = 0$, hence $a = \sqrt{3}$.
The optimal throwing angle is easily determined by the quadratic equation with discriminant zero: $\alpha = \arctan\left(\sqrt{\frac{1}{3}}\right) = 30°$.



The second model leads to an easier computation. Additionally, it carries much further: now we can also optimize throwing distances on a slanted plane. All we have to do is to intersect the parabola with the a slanted line.

1. Introduction

2. Mathematical Models in Kinematics and Mechanics

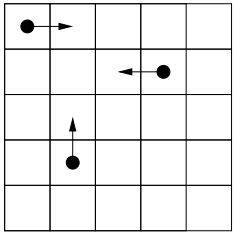
**3. Logical Models of Problems and Computations**

4. Modeling Problems in Geometry and Discrete Mathematics
   - Problem 1: How to get from $A$ to $B$?
   - Problem 2: How to efficiently use resources?

5. Organization

# Example: A Robotic System

An grid in which multiple robots move around.



- System: each robot moves one cell in a selected direction.
- Safety: the robots shall not collide with the walls or with each other.

Our task is to model an adequate control software for each robot: given the current situation of the system, compute a safe direction for the movement of the robot.

# A Model of the System

An operational description of the system.

```
proc system(x0: Positions, y0: Positions): ()
  requires init(x0, y0);
{
  var x: Positions = x0;
  var y: Positions = y0;
  for i:ℕ[N] do
  {
    choose r: Robot;
    choose d: Direction with nextDir(x, y, r, d);
    x := moveX(x, r, d);
    y := moveY(y, r, d);
    assert noCollision(x, y);
  }
}
```

The control software is implicitly specified by predicate `nextDir()`.

# Auxiliary Definitions

```
val R:ℕ; // number of robots
val P:ℕ; // number of positions
axiom notzero ⇔ R ≥ 1 ∧ P ≥ 1;

type Robot = ℕ[R-1];
type Position = ℕ[P-1];
enumtype Direction = Stop | Left | Right | Up | Down;
type Positions = Array[R,Position];

fun moveX(x:Positions, r:Robot, dr: Direction):Positions =
  match dr with
  {
    Left  -> x with [r] = x[r]-1;
    Right -> x with [r] = x[r]+1;
    Stop  -> x; Up -> x; Down  -> x;
  };
fun moveY(y:Positions, r:Robot, dr: Direction):Positions = ...
```

# Core Predicates

```
// the desired safety property of the system:
// no two robots are at the same position
pred noCollision(x:Positions, y:Positions) ⇔
  ∀r1:Robot, r2:Robot with r1 < r2. x[r1] ≠ x[r2] ∨ y[r1] ≠ y[r2];

// the initial state condition of the system:
// robots are at different positions
pred init(x:Positions, y:Positions) ⇔
  noCollision(x, y);
```

# The Control Software

```
// any robot different from r may move to position xr, yr
pred anyOtherAt(x:Positions, y:Positions, r:Robot, xr:Position, yr:Position) ⇔
  ∃r0: Robot with r0 ≠ r. xr = x[r0] ∧ yr = y[r0];

// the relation between the current system state and the new direction d of robot r
pred nextDir(x:Positions, y:Positions, r:Robot, d:Direction) ⇔
  (d = Direction!Stop)
∨ (d = Direction!Left  ∧ x[r] > 0 ∧ ¬anyOtherAt(x, y, r, x[r]-1, y[r]))
∨ (d = Direction!Right ∧ x[r] < P-1 ∧ ¬anyOtherAt(x, y, r, x[r]+1, y[r]))
∨ (d = Direction!Up    ∧ y[r] > 0 ∧ ¬anyOtherAt(x, y, r, x[r], y[r]-1))
∨ (d = Direction!Down  ∧ y[r] < P-1 ∧ ¬anyOtherAt(x, y, r, x[r], y[r]+1))
;
```

The robot may move within the grid to any unoccupied position.

# Verifying the Safety of the System

Checking all possible executions for all possible choices with $N + 1$ moves.

```
Using R=2.
Using P=5.
Computing the truth value of notzero...
Using N=2.
Type checking and translation completed.
Executing system(Array[ℤ],Array[ℤ]) with all 625 inputs.
PARALLEL execution with 4 threads (output disabled).
235 inputs (158 checked, 9 inadmissible, 0 ignored, 68 open)...
368 inputs (289 checked, 11 inadmissible, 0 ignored, 68 open)...
502 inputs (418 checked, 15 inadmissible, 0 ignored, 69 open)...
625 inputs (569 checked, 20 inadmissible, 0 ignored, 36 open)...
Execution completed for ALL inputs (8326 ms, 600 checked, 25 inadmissible).
```

The system is safe for three steps.

# An Alternative Model of the System

An logical description of the system.

```
// the initial state condition of the system
pred init(x:Positions, y:Positions) ⇔
  noCollision(x, y);

// the relationship between the prestate of the system and its poststate
pred next(x:Positions, y:Positions, x0:Positions, y0:Positions) ⇔
  ∃r:Robot, d:Direction with nextDir(x, y, r, d).
    x0 = moveX(x, r, d) ∧ y0 = moveY(y, r, d);
```

The system is also uniquely described by an initial state condition and a state transition relation.

# Verifying the Safety of the System

How to ensure safety for infinitely many steps?

```
// the system invariant
pred inv(x:Positions, y:Positions) ⇔ noCollision(x, y);

// the system invariant implies the desired safety property
theorem invIsStrongEnough(x:Positions, y:Positions) ⇔
  inv(x, y) ⇒ noCollision(x, y);

// the system invariant is inductive
theorem invHoldsInitially(x:Positions, y:Positions) ⇔
  init(x, y) ⇒ inv(x, y);
theorem invIsPreserved(x:Positions, y:Positions) ⇔
  inv(x, y) ⇒
  ∀x0:Positions, y0:Positions.
    next(x, y, x0, y0) ⇒ inv(x0, y0);
```

By induction, the theorems imply safety for infinitely many steps.

# Verifying the Validity of the Theorems

Checking the theorems that imply safety for infinitely many steps.

```
Using R=2.
Using P=5.
Computing the truth value of notzero...
Using N=2.
Type checking and translation completed.
Executing invIsStrongEnough(Array[ℤ],Array[ℤ]) with all 625 inputs.
Execution completed for ALL inputs (101 ms, 625 checked, 0 inadmissible).
Executing invHoldsInitially(Array[ℤ],Array[ℤ]) with all 625 inputs.
Execution completed for ALL inputs (82 ms, 625 checked, 0 inadmissible).
Executing invIsPreserved(Array[ℤ],Array[ℤ]) with all 625 inputs.
PARALLEL execution with 4 threads (output disabled).
365 inputs (288 checked, 0 inadmissible, 0 ignored, 77 open)...
625 inputs (572 checked, 0 inadmissible, 0 ignored, 53 open)...
Execution completed for ALL inputs (4400 ms, 625 checked, 0 inadmissible).
```

The system is safe for infinitely many steps.

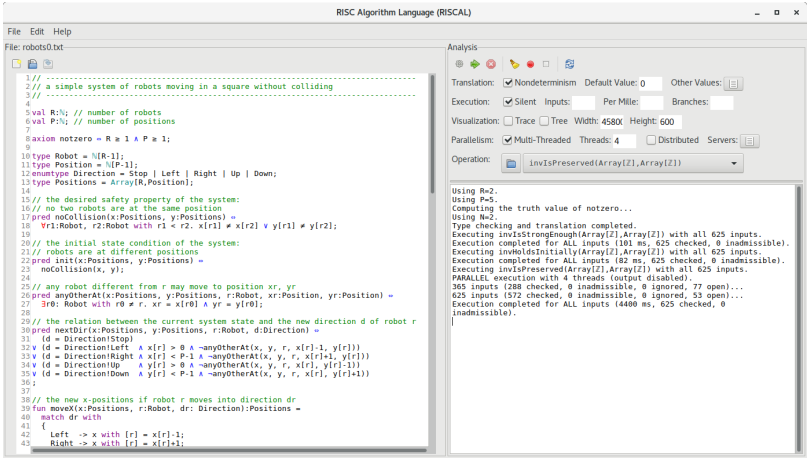
# Another Form of Robots

What if each robot has to choose its direction already in the previous step?

```
proc system(x0: Positions, y0: Positions, d0: Directions): ()
  requires init(x0, y0, d0);
{
  var x: Positions = x0; var y: Positions = y0; var d: Directions = d0;
  for i:ℕ[N] do
  {
    choose r: Robot;
    x := moveX(x, r, d[r]);
    y := moveY(y, r, d[r]);
    assert noCollision(x, y);
    choose dr: Direction with nextDir(x, y, d, r, dr);
    d[r] := dr;
  }
}
```

A more complex control software and a stronger system invariant are needed.

# Software: the RISC Algorithm Language (RISCAL)




A language and checker for mathematical models and algorithms.

# Course Contents

- Logical specification of computational problems.
  - Pre- and post-conditions.
  - Validation of specifications according to various criteria.
  - Computation of results by logical solving.
- Logical modeling of computational systems.
  - Initial state conditions, transition relations.
  - Modeling safety and liveness properties.
  - Simulation of execution by logical solving.

Software: RISCAL and (optionally) Leslie Lamport's TLA+ toolbox.

1. Introduction

2. Mathematical Models in Kinematics and Mechanics

3. Logical Models of Problems and Computations

**4. Modeling Problems in Geometry and Discrete Mathematics**

- Problem 1: How to get from $A$ to $B$?
- Problem 2: How to efficiently use resources?

5. Organization

# MODELING PROBLEMS IN GEOMETRY AND DISCRETE MATHEMATICS

## PROBLEM 1: HOW TO GET FROM $A$ TO $B$?

# Navigation Systems (trivial approach)

What is the mathematics behind navigation systems in modern cars?

# Navigation Systems (trivial approach)

What is the mathematics behind navigation systems in modern cars?

**Given:** start address $A$, destination address $B$.

**Find:** “shortest route” from $A$ to $B$.

# Navigation Systems (trivial approach)

What is the mathematics behind navigation systems in modern cars?

**Given:** start address $A$, destination address $B$.

**Find:** "shortest route" from $A$ to $B$.

Real world: $A$ and $B$ are given by geographical coordinates (2D or even 3D).

# Navigation Systems (trivial approach)

What is the mathematics behind navigation systems in modern cars?

**Given:** start address $A$, destination address $B$.

**Find:** "shortest route" from $A$ to $B$.

Real world: $A$ and $B$ are given by geographical coordinates (2D or even 3D).

Solution: if no further restrictions are given, the solution is trivial: the shortest connection from $A$ to $B$ is the straight line from $A$ to $B$, the "shortest route" is given by $(A, B)$ with length

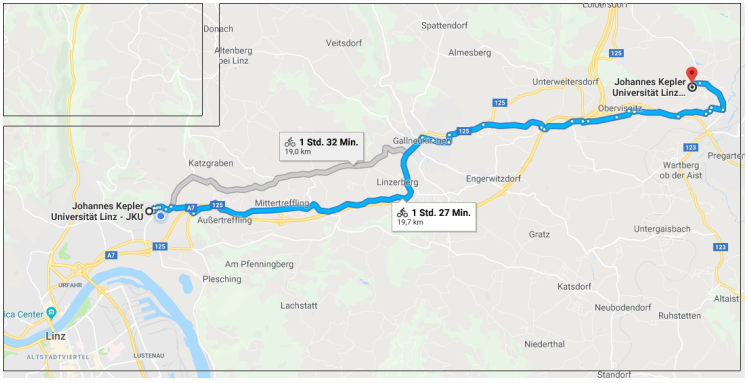
$$d_{\min} = \|B - A\|$$

with some appropriate norm $\|.\|$.

# Navigation Systems (realistic approach)

**Given:** start address $A$, destination address $B$, “network” of streets $S$.

**Find:** “shortest route” from $A$ to $B$ on $S$.



# Navigation Systems (realistic approach)

**Given:** start address $A$, destination address $B$, "network" of streets $S$.

**Find:** "shortest route" from $A$ to $B$ on $S$.

Mathematical model: Given $n$ Streets $S_i$ with $i = 1, \ldots, n$. Streets $S_i$ and $S_j$ intersect at crossing $C_{ij}$. Two crossings $c$ and $d$ are adjacent iff $c \neq d$ and there is no crossing between them on the same street. Two adjacent crossings are connected by a street segment.

Network of streets is characterized by

- crossings $V = \{C_{ij} \mid i, j = 1, \ldots, n\}$,
- adjacency relation between crossings $E = \{\{v_1, v_2\} \mid v_1 \text{ and } v_2 \text{ are adjacent}\}$,
- length of street segments $w \colon E \to \mathbb{R}^+$.

# Undirected Weighted Graphs

The triple $G = (V, E, w)$ is called an undirected weighted graph iff

- $V$ is some non-empty finite set,
- $E \subset P(V)$ with $|e| = 2$ for all $e \in E$, and
- $w \colon E \to \mathbb{R}$.

# Undirected Weighted Graphs

The triple $G = (V, E, w)$ is called an undirected weighted graph iff

- $V$ is some non-empty finite set,
- $E \subset P(V)$ with $|e| = 2$ for all $e \in E$, and
- $w \colon E \to \mathbb{R}$.

Our problem now becomes

**Given:** an undirected weighted graph $G = (V, E, w)$, $A, B \in V$.

**Find:** a sequence $P$ of some length $n$ in $V$ such that

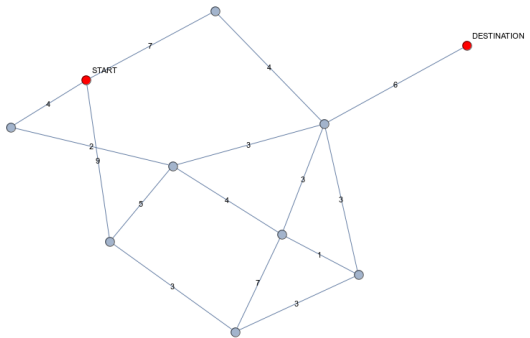
$$P_1 = A, P_n = B$$

$$\forall 1 \leq i \leq n-1 : \{P_i, P_{i+1}\} \in E$$

$$\sum_{i=1}^{n} w(\{P_i, P_{i+1}\}) = \min\{w(Q) \mid Q \text{ is a path from } A \text{ to } B \text{ in } G\}$$

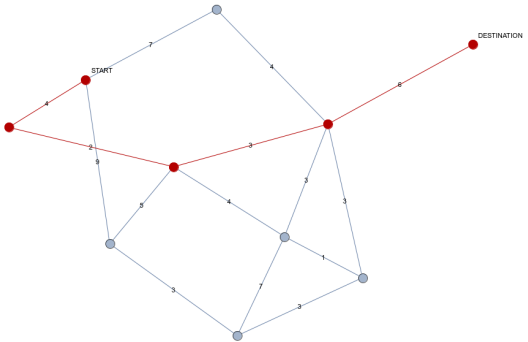
# Solution

The above problem is a well-known and well-studied problem in graph theory called the Shortest Path Problem.



# Solution

The above problem is a well-known and well-studied problem in graph theory called the Shortest Path Problem.




There are several algorithms to solve the Shortest Path Problem, e.g. Dijkstra's Algorithm or the Bellman-Ford-Algorithm.

# MODELING PROBLEMS IN GEOMETRY AND DISCRETE MATHEMATICS

## PROBLEM 2: HOW TO EFFICIENTLY USE RESOURCES?

# A Planning Problem

## Real Life Situation

A factory has 10 production stations with equal capabilities. Each machine can be operated for at most 9 hours per day, production may start at 8:30. Every station needs two workers for operation, if a station stays closed the two employees can be used for other useful tasks. There are 160 orders with different production duration that have to be processed on a certain day. Each order can be processed on any of the stations. The delivery of the final products is scheduled on the night train leaving the factory no earlier than 18:00. Time for packing the products on the train is less than half an hour.

Design a “good” production schedule for that day.

# Problem Analysis

- Every station can be utilized for the whole 9 hours from 8:30–17:30.

# Problem Analysis

- Every station can be utilized for the whole 9 hours from 8:30–17:30.
- Production order does not play a role.

# Problem Analysis

- Every station can be utilized for the whole 9 hours from 8:30–17:30.
- Production order does not play a role.
- Every order has to be processed.

# Problem Analysis

- Every station can be utilized for the whole 9 hours from 8:30–17:30.
- Production order does not play a role.
- Every order has to be processed.
- There is no need to finish production as early as possible, finishing by 17:30 is all that is required so that the train is readily packed by 18:00.

# Problem Analysis

- Every station can be utilized for the whole 9 hours from 8:30–17:30.
- Production order does not play a role.
- Every order has to be processed.
- There is no need to finish production as early as possible, finishing by 17:30 is all that is required so that the train is readily packed by 18:00.
- Fast production is not the criterion for a “good” production schedule, but the number of open production stations.

# Mathematical Model

**Given:** orders $O = \{1, \ldots, n\}$, duration $d \colon O \to \mathbb{R}$, stations $S = \{1, \ldots, m\}$, maximal operation time on stations $D \colon S \to \mathbb{R}$.

**Find:** number of open stations $k$ and assignment of orders to stations $s \colon O \to \{1, \ldots, k\}$ such that

$$k \leq m, \tag{1}$$

$$\forall j \in S : \sum_{\substack{i \in O \\ s(i)=j}} d(i) \leq D(j), \tag{2}$$

$$\forall l < k \nexists t \colon O \to \{1, \ldots, l\} \forall j \in S : \sum_{\substack{i \in O \\ t(i)=j}} d(i) \leq D(j) \tag{3}$$

(2) means assignment obeys limit on every station.

(3) means that no assignment with less stations is possible.

# Solution

The above problem is a well-known and well-studied problem in combinatorial optimization called the Bin Packing Problem.

# Solution

The above problem is a well-known and well-studied problem in combinatorial optimization called the Bin Packing Problem.

There are several algorithms to solve the Bin Packing Problem, e.g. Branch-and-Bound or various Heuristic Approximation Methods, because finding the minimal $k$ can be very time consuming.

1. Introduction

2. Mathematical Models in Kinematics and Mechanics

3. Logical Models of Problems and Computations

4. Modeling Problems in Geometry and Discrete Mathematics
   - Problem 1: How to get from $A$ to $B$?
   - Problem 2: How to efficiently use resources?

**5. Organization**

# Organization

- This course (VO)
  - Grading based on three home assignments (3×100 grade points).
  - Each assignment deals with the elaboration of a small model.
  - Minimum requirement to pass the course: 3×50 grade points.
  - Extra exam: only if the minimum requirements are not met.
- Accompanying proseminar (PS)
  - Deals with the kind of models treated in this course.
  - Additionally discusses the basics of “mathematical practice”.
  - Each participant selects an individual problem to be modeled/analyzed.
  - Requirement is to write a small paper and prepare/give a small presentation.
  - Some topics are also suitable for a bachelor thesis.

This course and the proseminar are not formally linked: they can be independently pursued and are independently graded.

# Moodle Course

Central point of electronic interaction.

- Forum “Discussions”: your questions and answers.
  - Anyone can post a question or an answer.
- Forum “Announcements”: our messages.
  - Only we (the lecturers) can post here.
- Various “Assignments”: your submissions.
  - Email submissions are not accepted.
- Personal messages/emails: only for confidential matters.
  - Everything else all lecturers and students should see.

See the link in the KUSSS page of this course.