Programmierwettbewerb 2013/2014

Preisverleihung

Wettbewerbs-Team

Mathe-dual e.V.

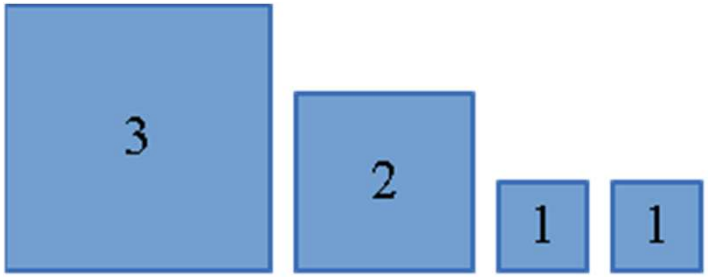
Übersicht

- Problemstellung
- Eingereichte Lösungen
- Lösungen im Detail
- Testumgebung
- Testfälle
- Auswertung
- Ergebnis

Problemstellung

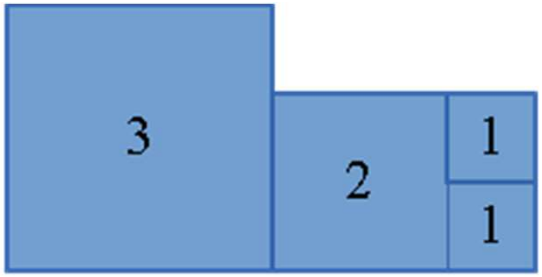
- **Zusammensetzung einer Menge von Quadraten mit Kantenlängen von 1-9 und minimalem Umfang**
- **Laufzeit 10 Minuten**
- **Programmiersprache frei wählbar**

```
3330000000
3330220000
3330220101
```



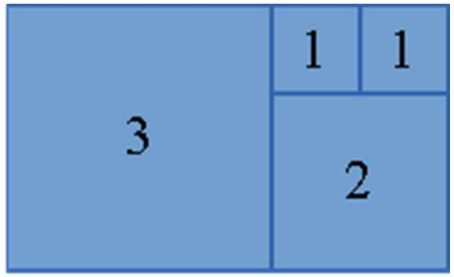
Σ28

```
333000
333221
333221
```



Σ18

```
33311
33322
33322
```



Σ16

Eingereichte Lösungen

- **4 Teilnehmer**
- **Lösungen**
  - → 2 x Java, 2x C / C++
  - → alle eingereichten Lösungen sind funktional und produzieren Ergebnisse
- **Parallelisierung**
  - → 2 Lösungen parallelisiert

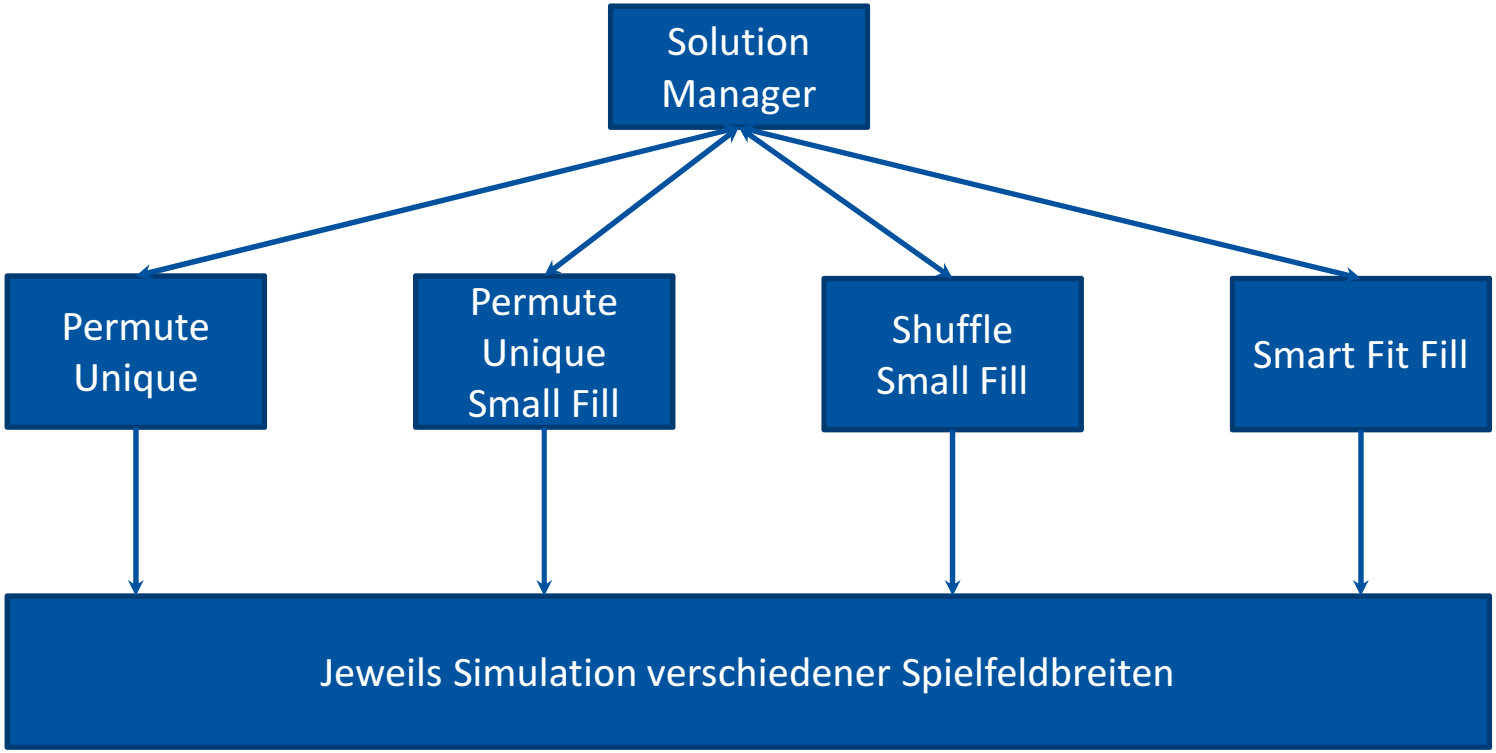
Lösung 1

- **Java (27 Dateien, 1698 Zeilen Sourcecode)**

- **Verschiedene Lösungsansätze laufen parallel (7 implementiert)**
  - → Basis-Algorithmus: Alle Quadrate nebeneinander
  - → Varianten von „Tetris“: Es werden Spielfelder aller möglichen Breiten zwischen einem Lower- und einem Upperbound mit verschiedenen Strategien gespielt
    - → Kombinatorik (Permutationen)
    - → Intelligentes Einfügen kleiner Quadrate
    - → Zufall

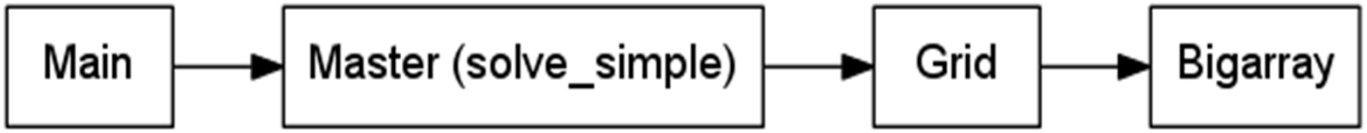
- **Iteratives Rausschreiben der bislang besten Lösung**

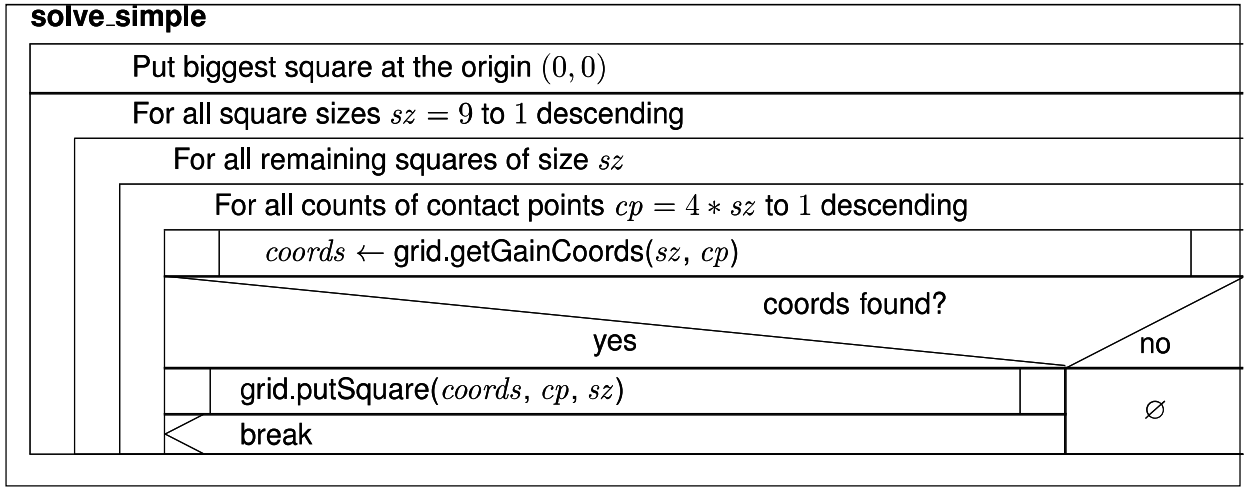
Lösung 1


Solution Manager
Permute Unique
Permute Unique Small Fill
Shuffle Small Fill
Smart Fit Fill
Jeweils Simulation verschiedener Spielfeldbreiten

Lösung 3

- C++
- Eine Lösung wird iterativ nach einem Greedy-Algorithmus zusammengesetzt
- Der Fokus lag in der Erstellung einer Utility Klasse (ursprünglich waren mehrere Lösungen vorgesehen)





Lösung 3

grid.putSquare($coords$**,** cp**,** sz**)**

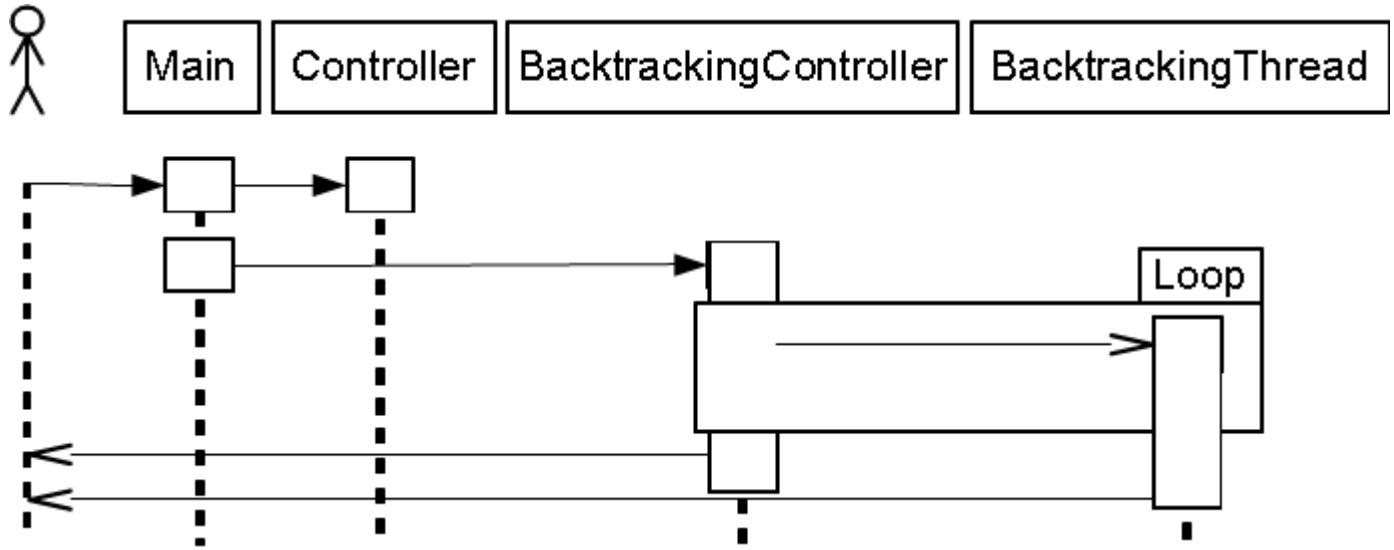
- Mark grid cells located at $coords$ as used (use sz as marker for output)
- Calculate new extent of solution: $ext \leftarrow ext + 4 * sz - 2 * cp$
- Update bounding box of solution
- For all square sizes $sz_x = 1$ to 9 ascending
  - For all relevant cells (x, y) in proximity of $coords$
    - select $list_a$ from lists of candidate coords for square size sz_x and contact point count cp
    - Remove obsolete entry (x, y) from $list_a$
    - Calculate new number of contact points cp_x for a square of size sz_x at cell (x, y)
    - select $list_b$ from lists of candidate coords for square size sz_x and contact point count cp_x
    - Insert new entry (x, y) to $list_b$

grid.getGainCoords(sz**,** cp**)**

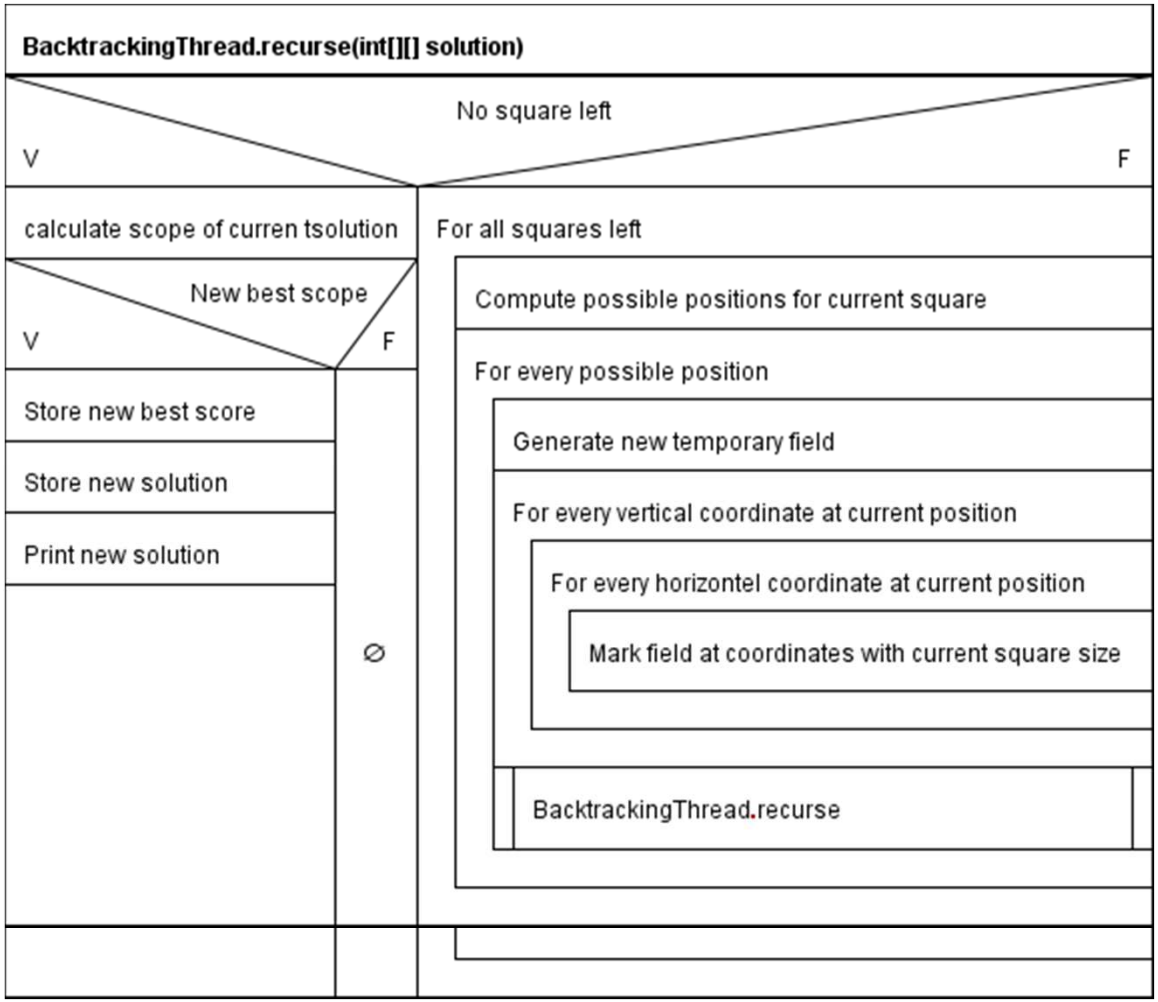
- select $list$ from lists of candidate coordinates for square size sz and contact point count cp
- $coords_{best} \leftarrow$ first coordinates entry of $list$ or $\varnothing$ if $list$ is empty
- $bbdiff_{best} \leftarrow$ change of the bounding box' side length when inserting the new square at $coords_{best}$
- $vdiff_{best} \leftarrow$ change in the number of vertices of the resulting bounding geometry when inserting the new square at $coords_{best}$
- For all other coordinates entries (x, y) of $list$
  - $bbdiff \leftarrow$ change of the bounding box' side length when inserting the new square at (x, y)
  - $vdiff \leftarrow$ change in the number of vertices of the resulting bounding geometry when inserting the new square at (x, y)
  - $(bbdiff < bbdif_{best}) \vee (bbdiff = bbdif_{best} \wedge vdiff < vdiff_{best})$?
    - yes:
      - $coords_{best} \leftarrow (x, y)$
      - $bbdiff_{best} \leftarrow bbdiff$
      - $vdiff_{best} \leftarrow vdiff$
    - no: $\varnothing$
- return $coords_{best}$

Lösung 2

- Java
- Lösungen werden mittels Backtracking gesucht
- Parallelisierung mittels der von Thread abgeleiteten Klasse BacktrackingThread



Lösung 2


BacktrackingThread.recurse(int[][] solution)
No square left
V
F
calculate scope of curren tsolution
For all squares left
New best scope
V
F
Compute possible positions for current square
For every possible position
Store new best score
Generate new temporary field
Store new solution
For every vertical coordinate at current position
Print new solution
For every horizontel coordinate at current position
Mark field at coordinates with current square size
∅
BacktrackingThread.recurse

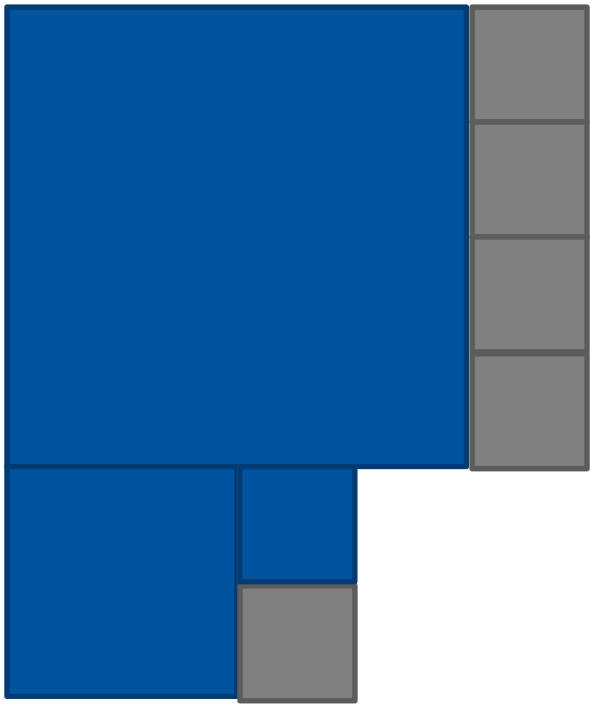
Lösung 4

- 587 Zeilen, eine Datei, C

- Lösung wird iterativ generiert (Greedy)
- Parallelisierung mit OpenMP
- Sonderbehandlung von 1x1 zum Auffüllen von Lücken

- Quadrate werden der Größe nach platziert, sodass möglichst große Teile der Kanten an anderen Quadraten anliegen

Lösung 4



Testumgebung

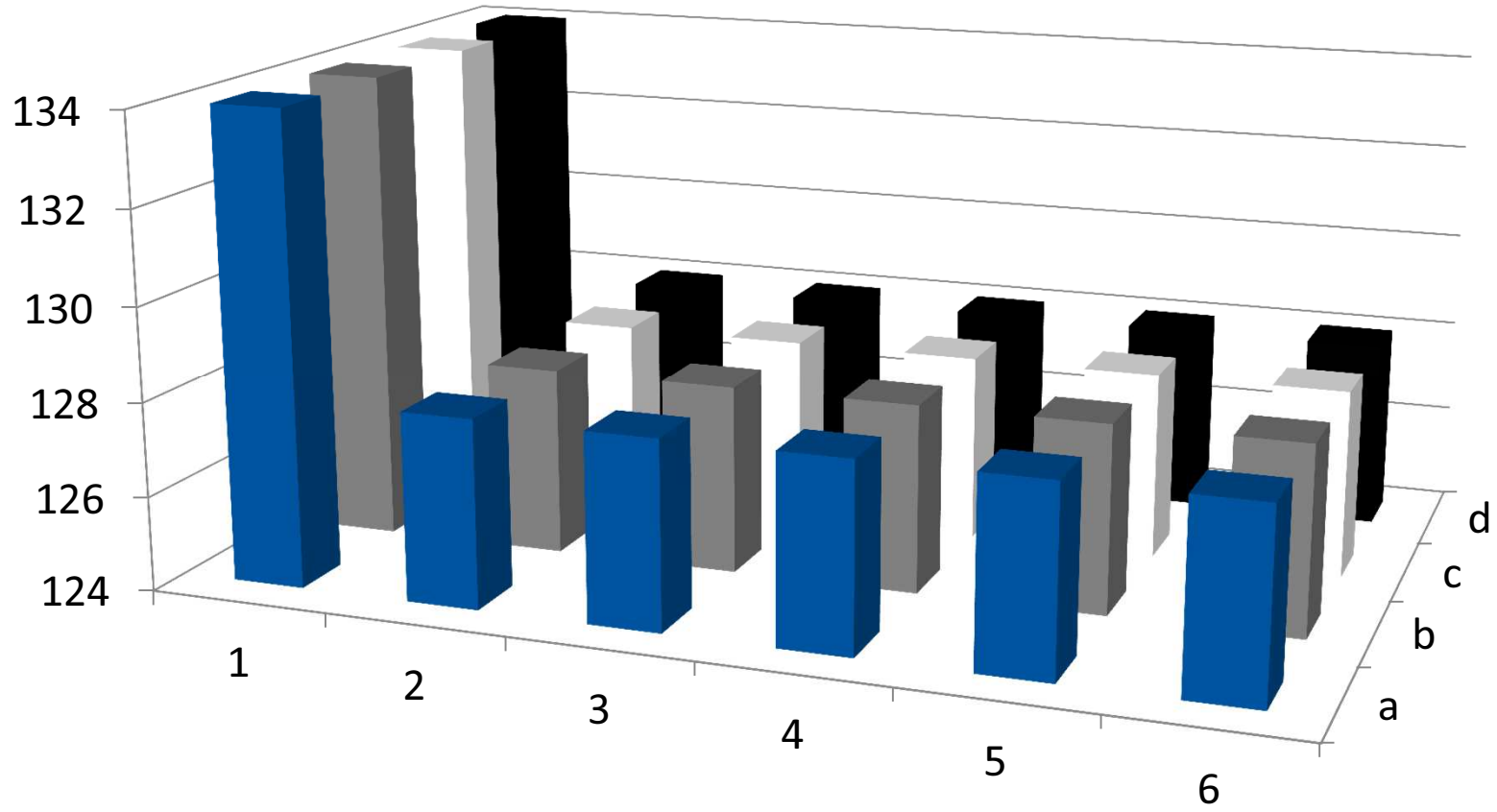
- **Hardware**
  - → Fujitsu RX-200
  - → Intel Xeon CPU E5450, 4 Cores, 3.00GHz
  - → 16 GB RAM, L2 cache 12MB, L1 cache 32KB
- **Software**
  - → Scientific Linux release 6.4 (Carbon)
  - → Java(TM) SE Runtime Environment (build 1.7.0_51-b13)
  - → Nutzbare Cores bzw. OMP-Threads begrenzt auf 4

Testfälle

- **Testfälle verschiedener Kategorien und verschiedener Problemgrößen**
  - → Mit bekannter, optimaler Lösung
    - → z.B. 1024 Quadrate der Seitenlänge 1
  - → Mit “Muster”
    - → z.B. 1 Quadrat der Seitenlänge 1, 2 Quadrate der Seitenlänge 2, usw.
  - → Zufällig

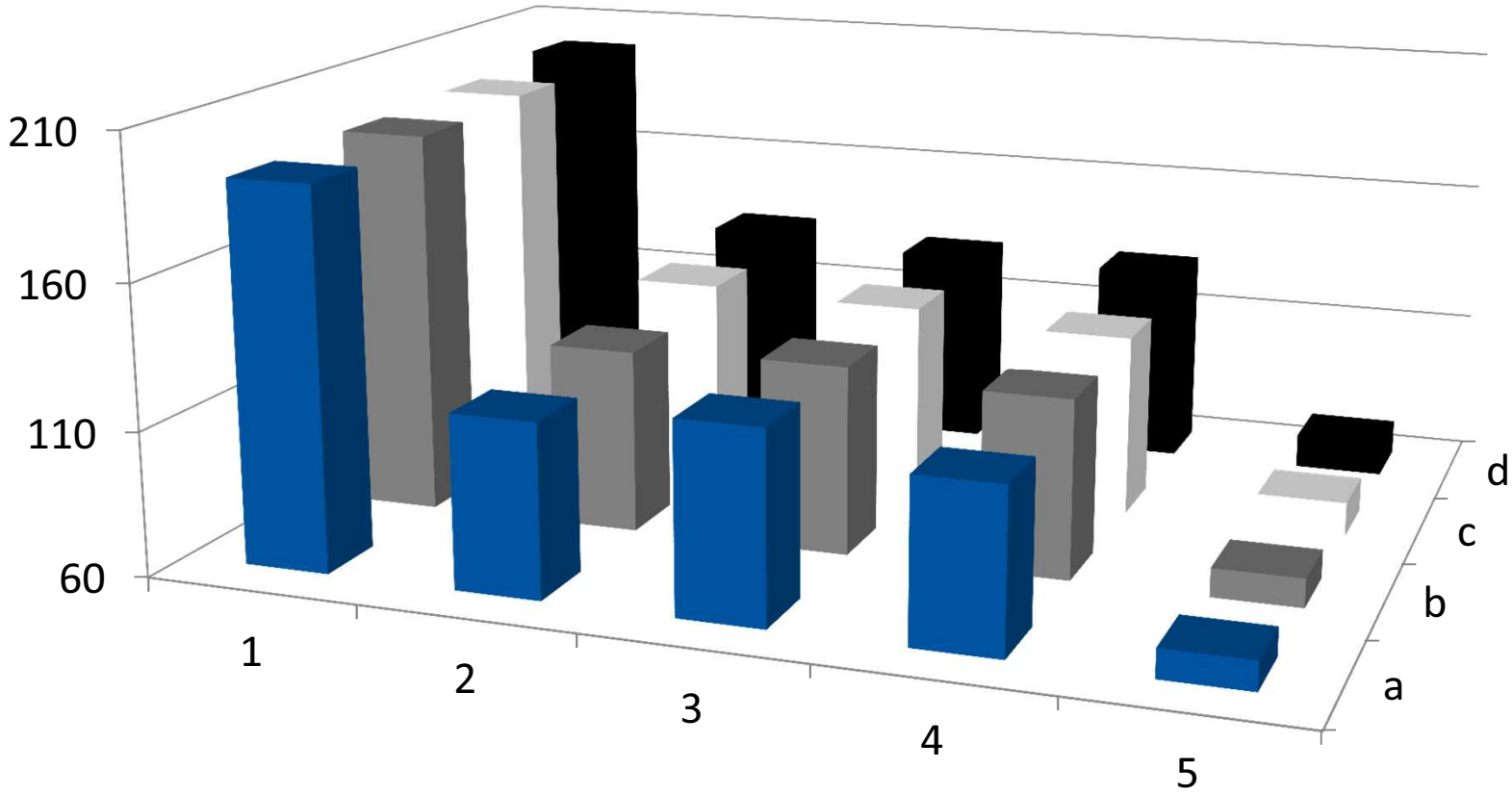
Testfälle – Bekannte Lösung

- **Alle Testfälle wurden von allen Teilnehmern optimal gelöst**



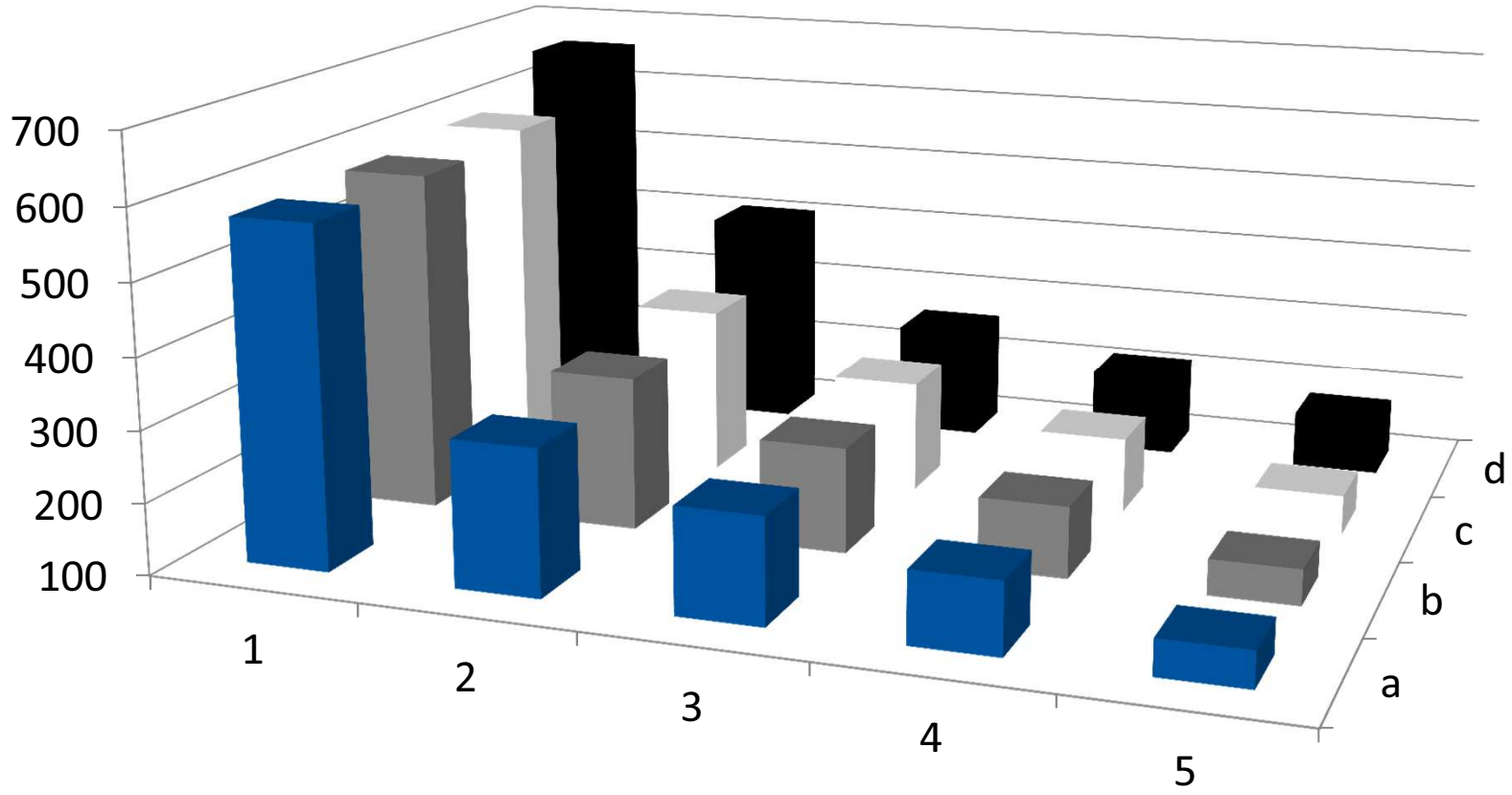
Testfälle – „Muster“

- **Nur kleine Abweichungen bei der Lösung der Testfälle**



Testfälle – Zufällig

- **Abweichung abhängig von der Größe des Problems**
  - → Parallelisierte Programme besser



Preisgelder

Platzierung	1. Platz	2. Platz	3. Platz	4. Platz
Preisgeld	180,00 €	168,00 €	95,00 €	57,00 €

Danke!