Heuristische Lösungsverfahren für Stacking-Probleme mit Transportkosten

---

Tim Bohne

30. Januar 2019

AG Kombinatorische Optimierung

Übersicht

1. Stacking-Probleme
2. Nebenbedingungen
3. Problemstellungen / Testdaten
4. Heuristiken / Vergleiche
5. Ausblick

Stacking-Probleme

- Im Umfeld von Lagerhallen und Container-Terminals
- Eintreffende Items müssen Stacks zugeordnet werden, sodass bestimmte Nebenbedingungen respektiert werden
- Storage Area ist in fixierten Stacks organisiert, die eine limitierte gemeinsame Stack Kapazität besitzen
- Es wird nur der Loading-Prozess betrachtet

Literatur Überblick

- **Kim et al. (2000)**: Deriving decision rules to locate export containers in container yards
- **Kang et al. (2006)**: Deriving stacking strategies for export containers with uncertain weight information
- **Delgado et al. (2012)**: A constraint programming model for fast optimal stowage of container vessel bays
- **Jaehn (2013)**: Positioning of loading units in a transshipment yard storage area
- **Bruns et al. (2015)**: Complexity results for storage loading problems with stacking constraints
- **Le et al. (2016)**: MIP-based approaches for robust storage loading problems with stacking constraints

Stacking-Probleme

Aufbau der Storage Area bestehend aus m fixierten Stacks, die jeweils b Level enthalten:

S_1	S_2	S_3	...	S_m

von oben betrachtet

L_b	$pos_{1,b}$	$pos_{2,b}$	$pos_{3,b}$	...	$pos_{m,b}$
...	...	...	...	...	...
L_2	$pos_{1,2}$	$pos_{2,2}$	$pos_{3,2}$	...	$pos_{m,2}$
L_1	$pos_{1,1}$	$pos_{2,1}$	$pos_{3,1}$	...	$pos_{m,1}$
	S_1	S_2	S_3	...	S_m

von der Seite betrachtet

Stacking-Probleme

Parameter	**Semantik**
n	Anzahl der Items
m	Anzahl der Stacks
b	Stack Kapazität
I	Menge der Items $I := \{1, 2, ..., n\}$

I.d.R. gilt $\boldsymbol{m} < \boldsymbol{n}$, außerdem muss $\boldsymbol{n} \leq \boldsymbol{bm}$ gelten.

Formulierung des Problems

Items: $\boldsymbol{I} = \{1, ..., n\}$
Stacks: $\boldsymbol{Q} = \{1, ..., m\}$
Stack Kapazität: $\boldsymbol{b}$
Stacking Constraints: $\boldsymbol{s_{ij}}$
Placement Constraints: $\boldsymbol{t_{iq}}$

Das Ziel ist, jedes Item $\boldsymbol{i} \in \boldsymbol{I}$ genau einem Stack $\boldsymbol{q} \in \boldsymbol{Q}$ zuzuweisen, wobei die Stacking Constraints $\boldsymbol{s_{ij}}$, die Placement Constraints $\boldsymbol{t_{iq}}$ und die Stack Kapazität $\boldsymbol{b}$ respektiert werden und ggf. eine Zielfunktion optimiert wird.

Stacking Constraints

- Schwerere Items dürfen nicht auf leichteren platziert werden
- Größere Items dürfen nicht auf kleineren platziert werden
- Items bestimmter Materialien / Zielorte dürfen nicht aufeinander gestapelt werden

Sämtliche Stacking Constraints werden in einer Binärmatrix $\boldsymbol{S} = (\boldsymbol{s_{ij}})_{n \times n}$ kodiert, wobei:

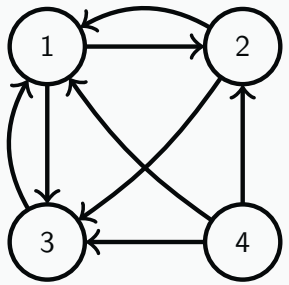
$$\boldsymbol{s_{ij}} = \begin{cases} 1, & \text{wenn } \boldsymbol{i} \text{ direkt auf } \boldsymbol{j} \text{ gestapelt werden darf} \\ 0, & \text{sonst} \end{cases}$$

Stacking Constraints

Diese Matrix lässt sich als gerichteter Graph mit n Knoten $i \in I$ und Kanten $i \to j$ für alle $s_{ij} = 1$ repräsentieren.

$$S = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Constraint Matrix



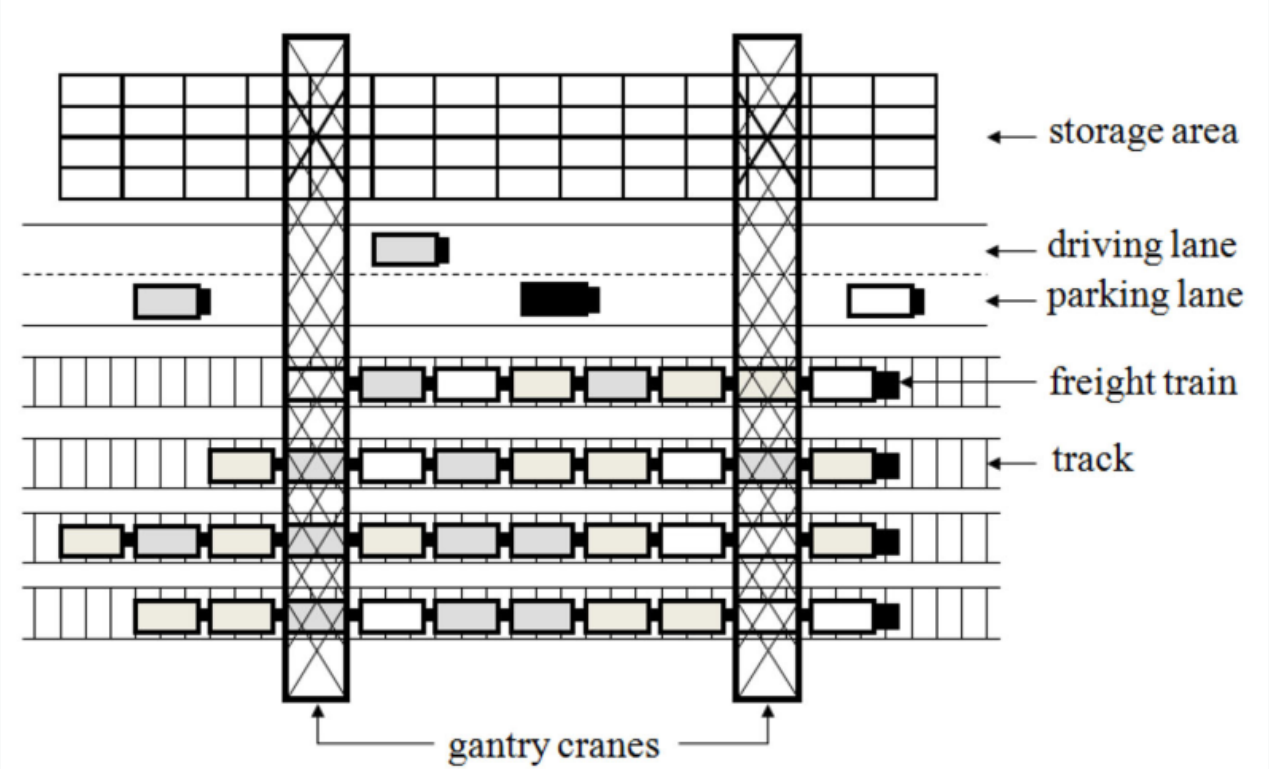
Resultierender Graph

Transportkosten

- Kranbetriebskosten, Wartezeiten, Arbeitszeiten, ...
- Jeder Stack $\boldsymbol{q}$ hat eine fixierte Position $\boldsymbol{F_q}$ in der Storage Area
- Jedes Item $\boldsymbol{i}$ hat eine geg. Originalposition $\boldsymbol{O_i}$ auf dem Fz.

Die Kosten werden in einer Matrix $\boldsymbol{C} = (\boldsymbol{c_{iq}})_{n \times m}$ kodiert.

$\boldsymbol{c_{iq}} := d_{man}(\boldsymbol{O_i}, \boldsymbol{F_q})$



Placement Constraints

- Länge / Gewicht der Items und Eigenschaften der Stacks
- Spezielle Anforderungen der Items (Kühlcontainer, ...)
- Tendenziell mehr erlaubt als verboten (rand(0, 1) zu restriktiv)
- Über sehr hohe Kostenwerte umgesetzt

In Binärmatrix $\boldsymbol{T} = (\boldsymbol{t_{iq}})_{n \times m}$ kodiert, wobei:

$$\boldsymbol{t_{iq}} = \begin{cases} 1, & \text{wenn Item } \boldsymbol{i} \text{ in Stack } \boldsymbol{q} \text{ platziert werden darf} \\ 0, & \text{sonst} \end{cases}$$

Für die Transportkosten-Matrix $\boldsymbol{C} = (\boldsymbol{c_{iq}})_{n \times m}$ gilt:

$$\boldsymbol{c_{iq}} = \begin{cases} d_{man}(\boldsymbol{O_i}, \boldsymbol{F_q}), & \text{wenn } \boldsymbol{t_{iq}} = 1 \\ \infty, & \text{sonst} \end{cases}$$

Problemstellungen

Stack-Kapazitäten: $\boldsymbol{b} = 2, 3, (4)$

- Zulässigkeitsproblem $\boldsymbol{b = 2}$ mit $\boldsymbol{s_{ij}}$ und $\boldsymbol{t_{iq}}$ NP-vollständig
- Zulässigkeitsproblem $\boldsymbol{b = 3}$ mit $\boldsymbol{s_{ij}}$ NP-vollständig

Instanzgrößen:

- klein (**s**) ($\leq$ 100 Items)
- mittel (**m**) ($\approx$ 300 Items)
- groß (**l**) ($\approx$ 500 Items)

Ziel: Sämtliche Items sollen möglichst günstig eingelagert werden.
Zielfunktion: Minimierung der Transportkosten

Zulässigkeit und geringe Laufzeit haben Priorität.

Testdaten-Generierung

- Anzahl der Instanzen: **20**

- Anzahl der Items: n
- Stack Kapazität: b
- Anzahl der Stacks: $m = \lceil n/b \rceil +$ **20%**

- Stacking Constraints: Zwei Varianten **(V2)**
- Placement Constraints: **70%** erlaubt
- Kosten: **Manhattan-Metrik**
  - Item- und Stackpositionen (x, y)
  - Item- und Stack-Längen, Item- und Stack-Breiten
  - Distanz zwischen Storage Area und Fahrzeug

Bin-Packing-Formulierung

$$\textbf{min} \quad \sum_{i \in I} \sum_{q \in Q} c_{iq} x_{iq} \tag{1}$$

$$\textbf{s.t.} \quad \sum_{q \in Q} x_{iq} = 1 \qquad \forall i \in I \tag{2}$$

$$\sum_{i \in I} x_{iq} \leq b \qquad \forall q \in Q \tag{3}$$

$$x_{iq} + x_{jq} \leq 1 \qquad \forall \{i,j\} \notin A \tag{4}$$

$$x_{iq} \in \{0,1\} \qquad \forall i \in I, q \in Q \tag{5}$$

3-Index-Formulierung

$$\textbf{min} \quad \sum_{i \in I} \sum_{q \in Q} \sum_{l \in L} c_{iq} x_{iql} \tag{6}$$

$$\textbf{s.t.} \quad \sum_{q \in Q} \sum_{l \in L} x_{iql} = 1 \qquad \forall i \in I \tag{7}$$

$$\sum_{i \in I} x_{iql} \leq 1 \qquad \forall q \in Q, l \in L \tag{8}$$

$$\sum_{j \in I | i \to j} x_{jq(l-1)} - x_{iql} \geq 0 \qquad \forall i \in I, q \in Q, l \in L \backslash \{1\} \tag{9}$$

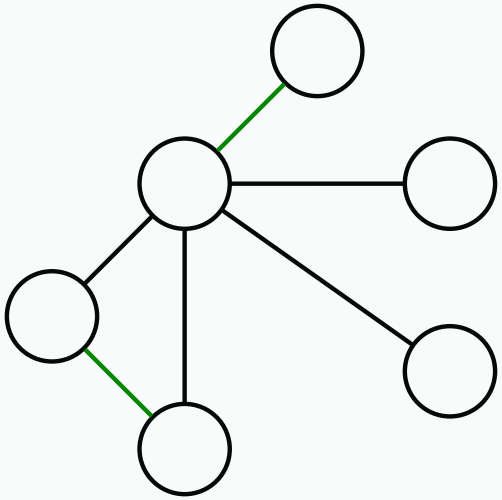
$$x_{iql} \in \{0,1\} \qquad \forall i \in I, q \in Q, l \in L \tag{10}$$

Exkurs: Maximum-Cardinality-Matching (MCM)

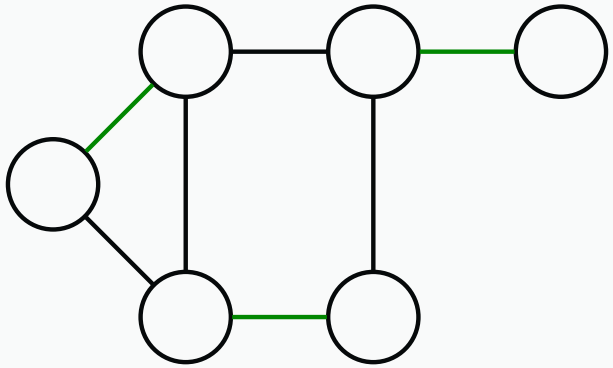
Ungerichteter Graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$

Eine Menge $\boldsymbol{M} \subseteq \boldsymbol{E}$ heißt Matching, wenn keine zwei Kanten aus $\boldsymbol{M}$ einen Knoten gemeinsam haben.

Falls $\boldsymbol{M}$ eine maximale Kardinalität unter allen Matchings von $\boldsymbol{G}$ hat, wird dies als MCM bezeichnet.



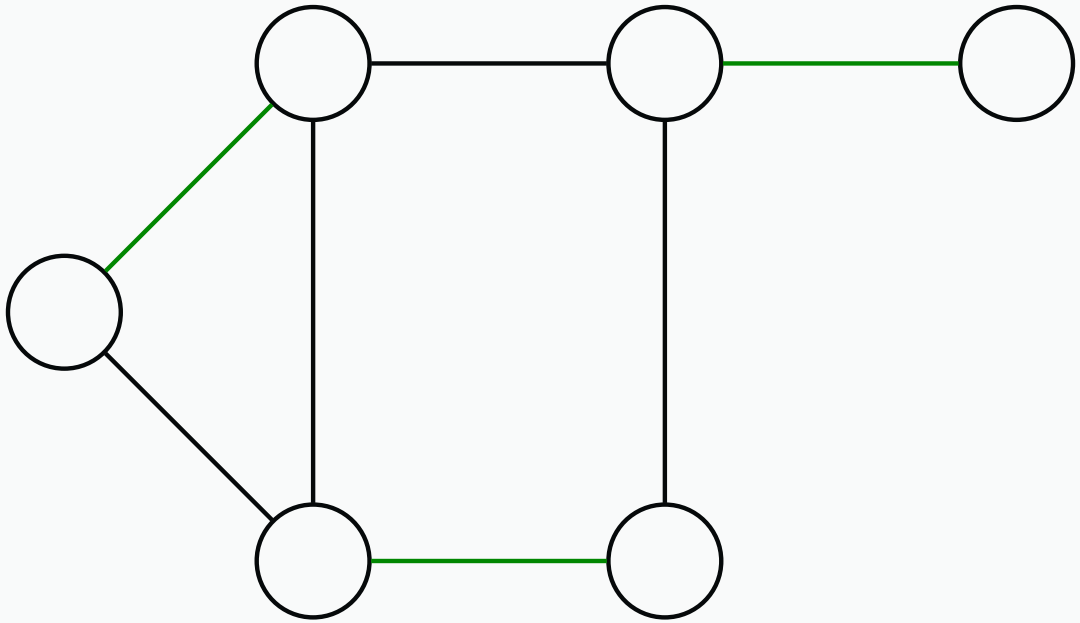
$|\mathrm{MCM}| = 2$



$|\mathrm{MCM}| = 3$

Exkurs: Minimum-Weight-Perfect-Matching (MWPM)

- Matching, welches sämtliche Knoten enthält
- Jeder Knoten ist inzident zu genau einer Kante des Matchings
- Günstigstes Perfect Matching basierend auf den Kantenkosten

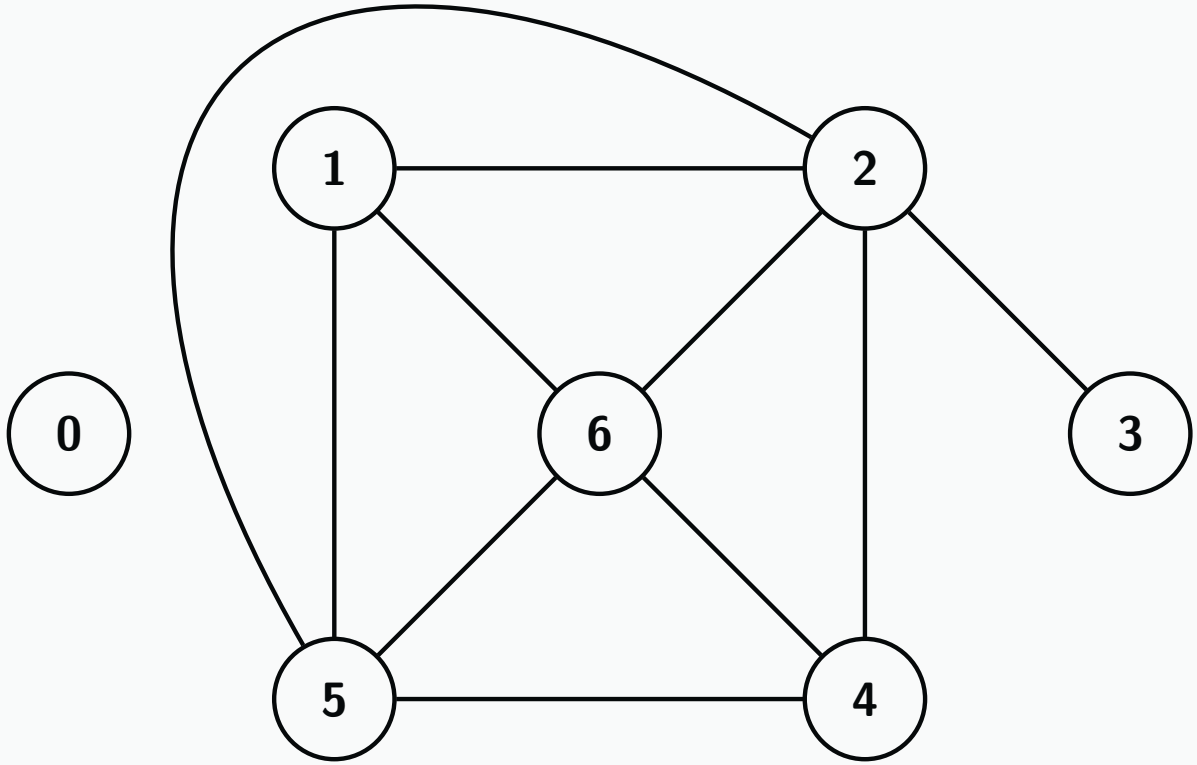


Konstruktive Heuristik ($b = 2$)

- Stacking-Constraint-Graph generieren
- **MCM** berechnen, Kanten als Item-Paare interpretieren
- Bipartiten Graph generieren:
  1. Items (Item-Paare, Unmatched-Items)
  2. Stacks
- **MWPM** berechnen, Kanten als Stackzuweisungen interpretieren
- Ggf. Reihenfolge der Items innerhalb der Stacks anpassen

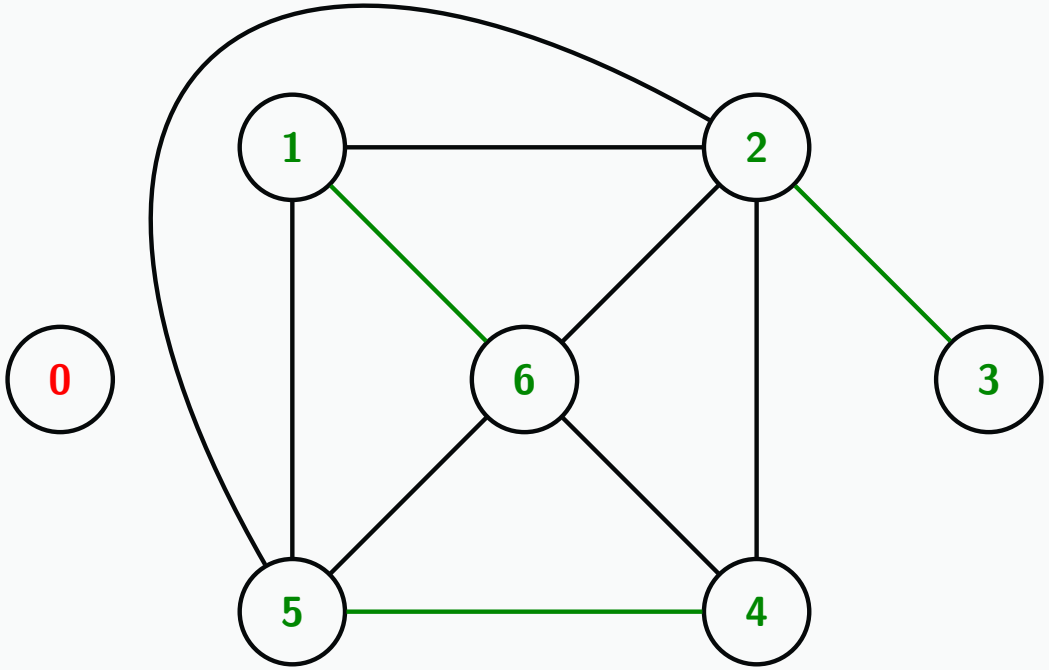
Beispiel $b = 2$

$\boldsymbol{I} := \{0, 1, 2, 3, 4, 5, 6\}$ $\boldsymbol{b} := 2$ $\boldsymbol{m} := 4$



Stacking-Constraint-Graph

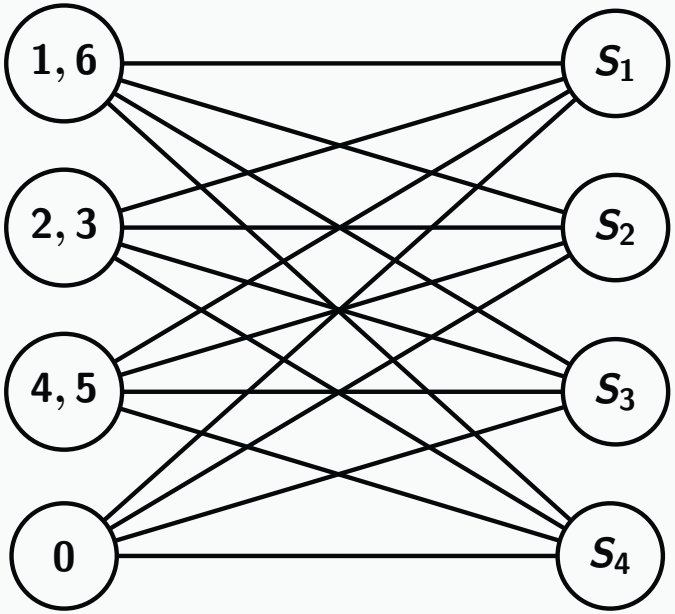
Beispiel $b = 2$



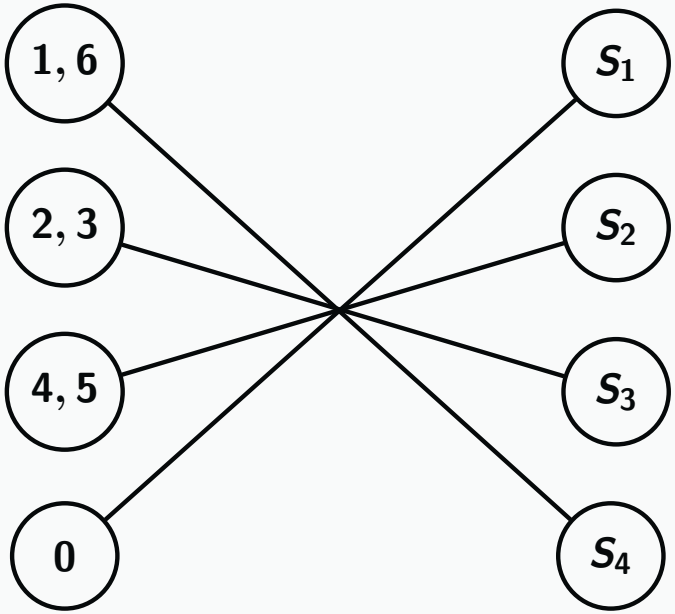
MCM (Item 0 unmatched)

Kanten bilden Item-Paare

Beispiel $b = 2$



VOLLSTÄNDIG BIPARTIT

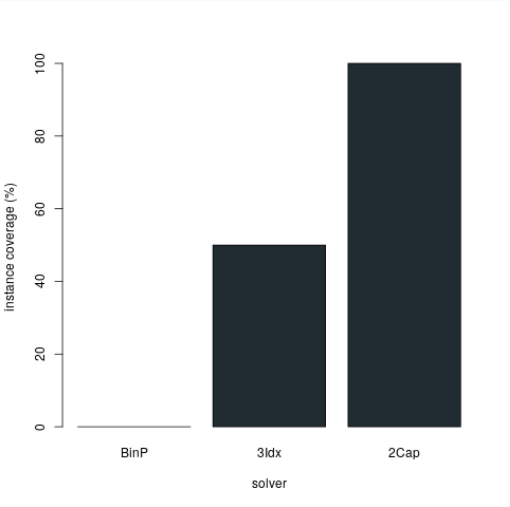


MWPM

L_2		5	3	6
L_1	0	4	2	1
	S_1	S_2	S_3	S_4

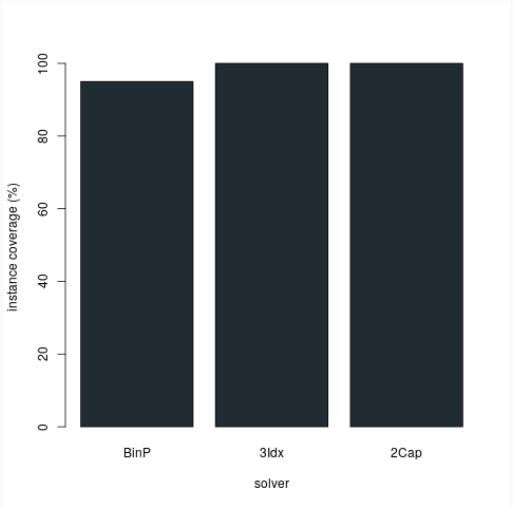
ZULÄSSIGE ZUWEISUNGEN (GGF. ITEM-REIHENFOLGE KORRIGIEREN)

Vergleich der $b = 2$ Solver (s)



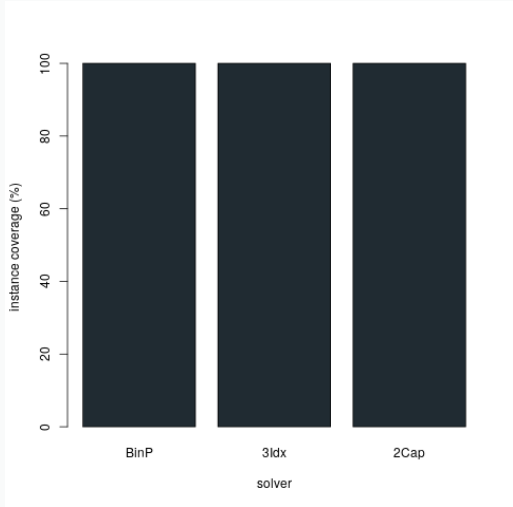
ZEITLIMIT 1s

	BinP	3Idx
Optimal	0%	35%
Laufzeit	– – –	Ø 0.9s
Abweichung	– – –	Ø 2.0%



ZEITLIMIT 3s

	BinP	3Idx
Optimal	10%	100%
Laufzeit	Ø 3.0s	Ø 1.1s
Abweichung	Ø 4.7%	Ø 0.0%



ZEITLIMIT 5s

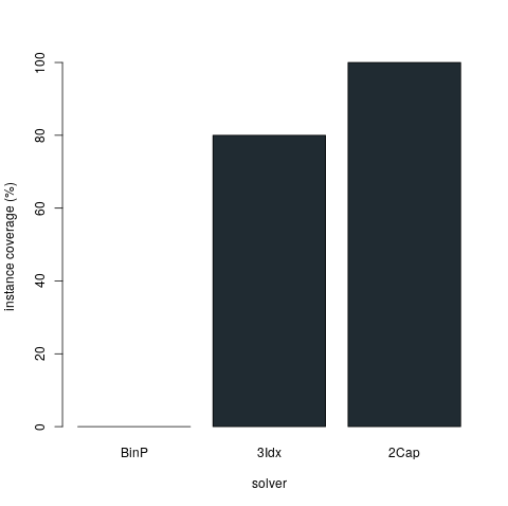
	BinP	3Idx
Optimal	90%	100%
Laufzeit	Ø 3.9s	Ø 1.1s
Abweichung	Ø 0.4%	Ø 0.0%

2Cap-Heuristik

Abweichung vom Optimum: Ø **2.0%**

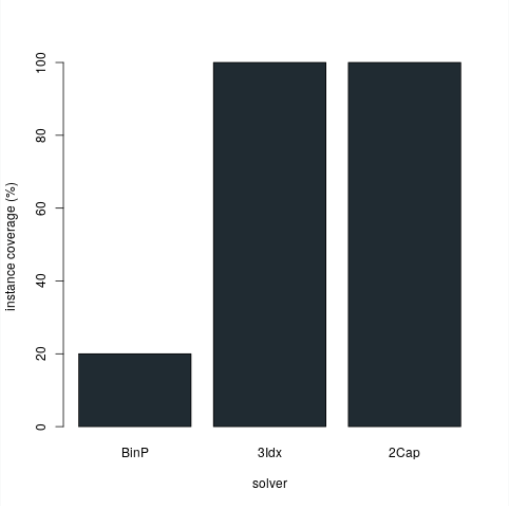
Laufzeit: Ø **0.02s**

Vergleich der $b = 2$ Solver (m)



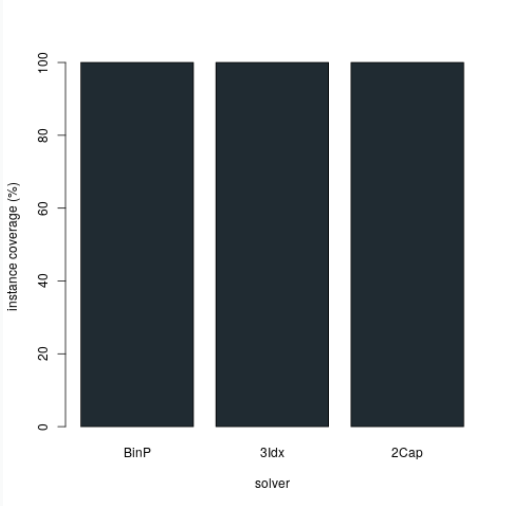
ZEITLIMIT 1*min*

	BinP	3Idx
Optimal	0%	50%
Laufzeit	– – – –	Ø 55.4*s*
Abweichung	– – – –	Ø 0.8%



ZEITLIMIT 10*min*

	BinP	3Idx
Optimal	10%	100%
Laufzeit	Ø 581.6*s*	Ø 73.7*s*
Abweichung	Ø 0.6%	Ø 0.0%



ZEITLIMIT 20*min*

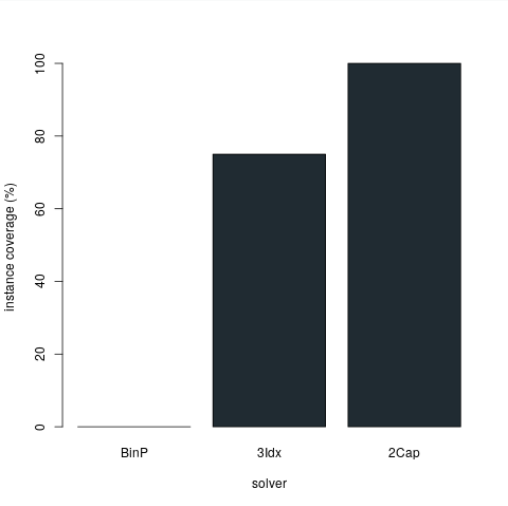
	BinP	3Idx
Optimal	100%	100%
Laufzeit	Ø 869.6*s*	Ø 107.8*s*
Abweichung	Ø 0.0%	Ø 0.0%

2Cap-Heuristik

Abweichung vom Optimum: Ø **0.8%**

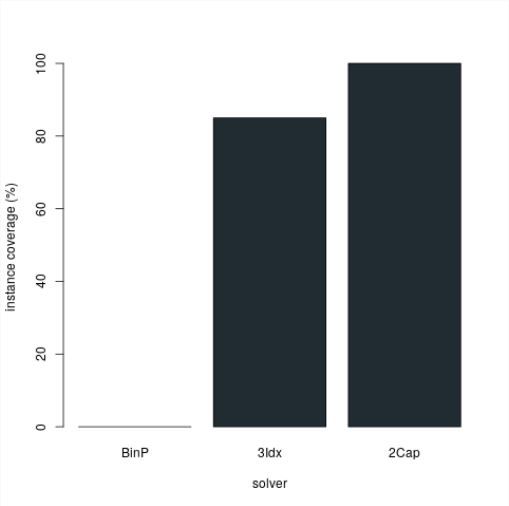
Laufzeit: Ø **0.1*s***

Vergleich der $b = 2$ Solver (I)



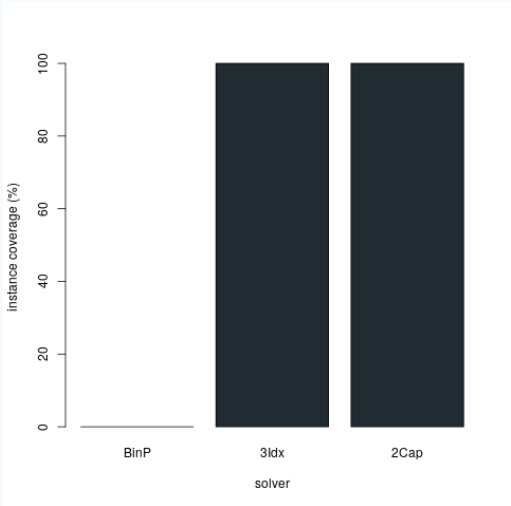
ZEITLIMIT 15*min*

	BinP	3Idx
Optimal	0%	70%
Laufzeit	– – –	Ø 728.0*s*
Abweichung	– – –	Ø 0.53%



ZEITLIMIT 30*min*

	BinP	3Idx
Optimal	0%	85%
Laufzeit	– – –	Ø 977.2*s*
Abweichung	– – –	Ø 0.0%



ZEITLIMIT 45*min*

	BinP	3Idx
Optimal	0%	100%
Laufzeit	– – –	Ø 1185.9*s*
Abweichung	– – –	Ø 0.0%

2Cap-Heuristik

Abweichung vom Optimum: Ø **0.6%**

Laufzeit: Ø **0.7*s***

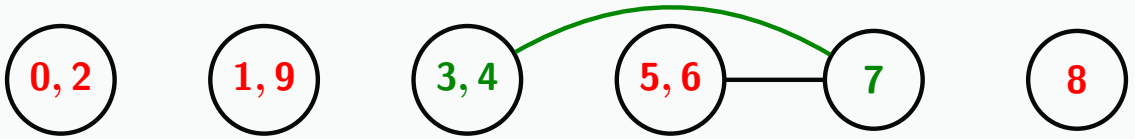
Konstruktive Heuristik ($b = 3$)

- Item-Paare bilden (**MCM**(ITEMS))
- Item-Tripel bilden (**MCM**(PAIRS, UNMATCHEDITEMS))
- Verbleibende Item-Paare zu Tripeln mergen
- Item-Paare bilden (**MCM**(REMAININGITEMS))
- Verbleibende Items als „unmatched“ betrachten
- Bipartiten Graph generieren:
  1. Items (Item-Tripel, Item-Paare, Unmatched-Items)
  2. Stacks
- **MWPM** berechnen, Kanten als Stackzuweisungen interpretieren
- Ggf. Reihenfolge der Items innerhalb der Stacks anpassen

Beispiel $b = 3$

$\boldsymbol{I} := \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ $\boldsymbol{b} := 3$ $\boldsymbol{m} := 4$

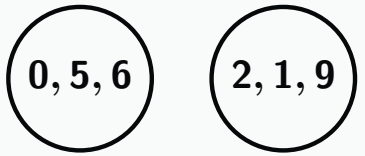
Item-Paare und Unmatched-Items



MCM aus Item-Paaren und Unmatched-Items

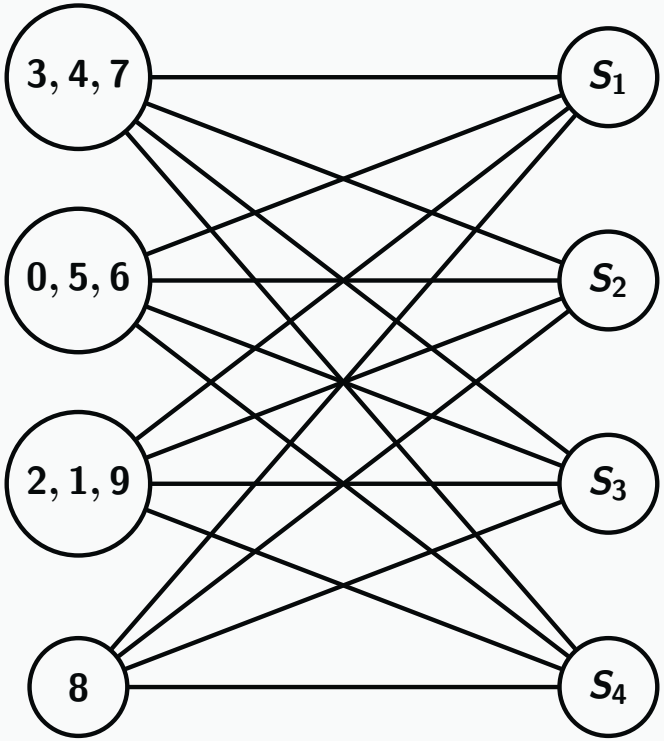
Item-Tripel, Item-Paare und Unmatched-Items

Item-Paare

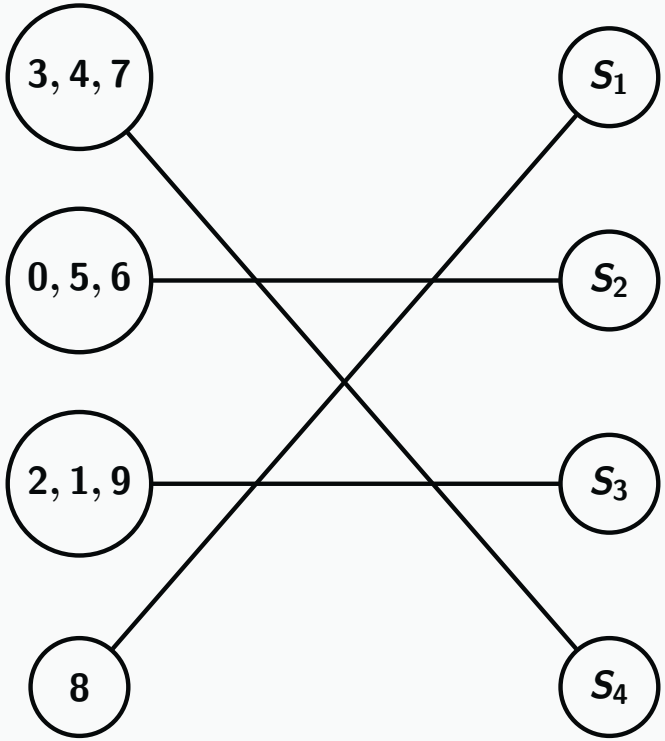


Merge-Ergebnis

Beispiel $b = 3$



Vollständig Bipartit

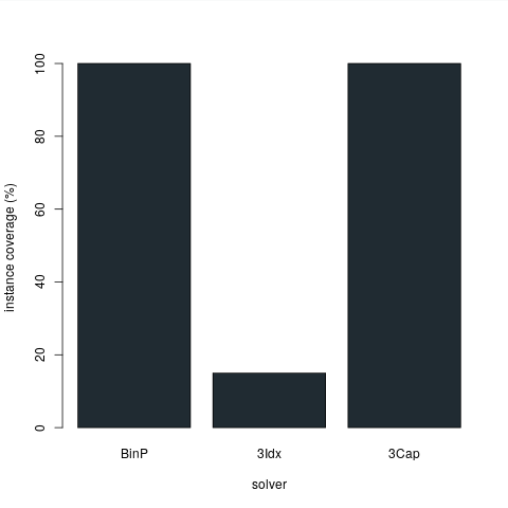


MWPM

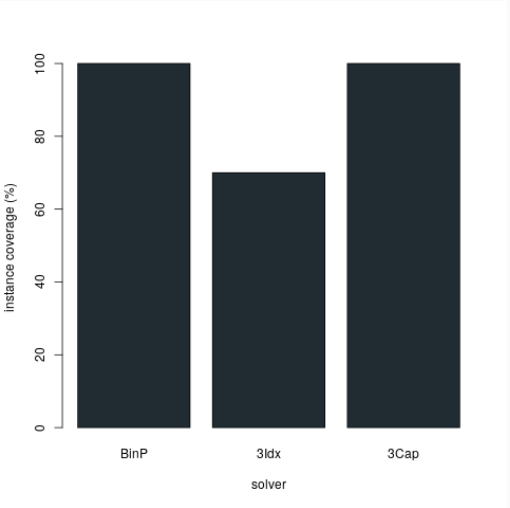
L_3		6	9	7
L_2		5	1	4
L_1	8	0	2	3
	S_1	S_2	S_3	S_4

Zulässige Zuweisungen (ggf. Item-Reihenfolge korrigieren)

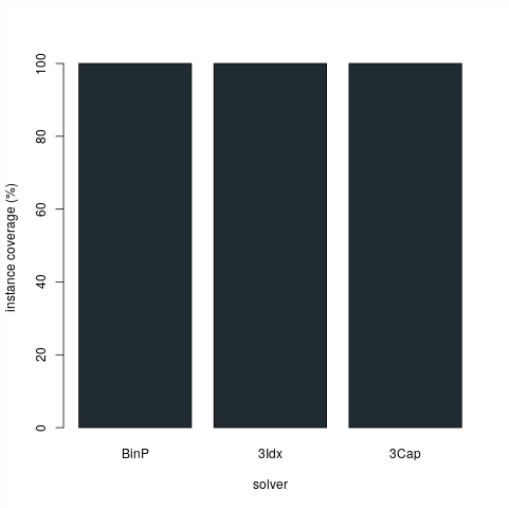
Vergleich der $b = 3$ Solver (s)



ZEITLIMIT 3s



ZEITLIMIT 5s



ZEITLIMIT 10s

	BinP	3Idx
Optimal	40%	5%
Laufzeit	Ø 2.9s	Ø 2.9s
Abweichung	Ø 1.7%	Ø 7.8%

	BinP	3Idx
Optimal	100%	50%
Laufzeit	Ø 3.1s	Ø 4.0s
Abweichung	Ø 0.0%	Ø 1.2%

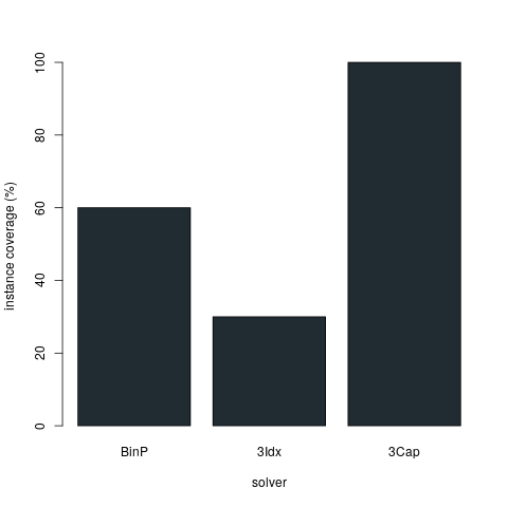
	BinP	3Idx
Optimal	100%	85%
Laufzeit	Ø 3.1s	Ø 5.6s
Abweichung	Ø 0.0%	Ø 0.5%

3Cap-Heuristik

Abweichung vom Optimum: Ø **2.65%**

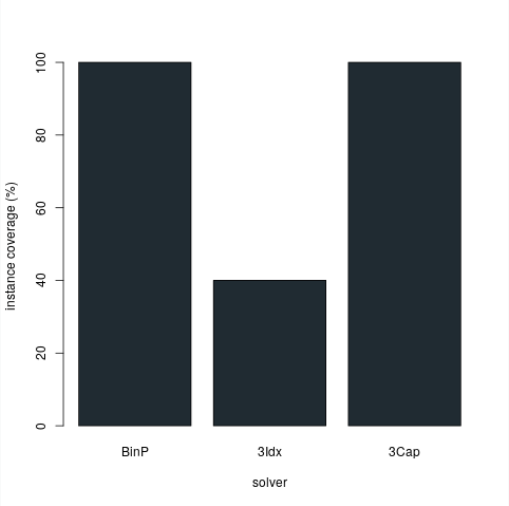
Laufzeit: Ø **0.01s**

Vergleich der $b = 3$ Solver (m)



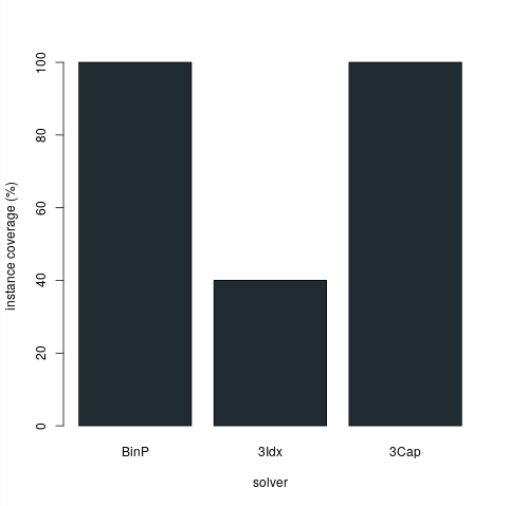
ZEITLIMIT 10*min*

	BinP	3Idx
Optimal	20%	10%
Laufzeit	Ø 583.7*s*	Ø 554.3*s*
Abweichung	Ø 0.2%	Ø 0.01%



ZEITLIMIT 20*min*

	BinP	3Idx
Optimal	100%	30%
Laufzeit	Ø 791.4*s*	Ø 795.4*s*
Abweichung	Ø 0.0%	Ø 0.0%



ZEITLIMIT 30*min*

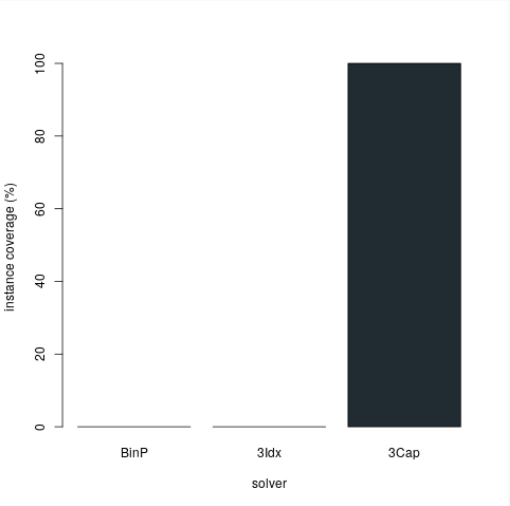
	BinP	3Idx
Optimal	100%	35%
Laufzeit	Ø 766.4*s*	Ø 874.0*s*
Abweichung	Ø 0.0%	Ø 0.0%

3Cap-Heuristik

Abweichung vom Optimum: Ø **1.02%**

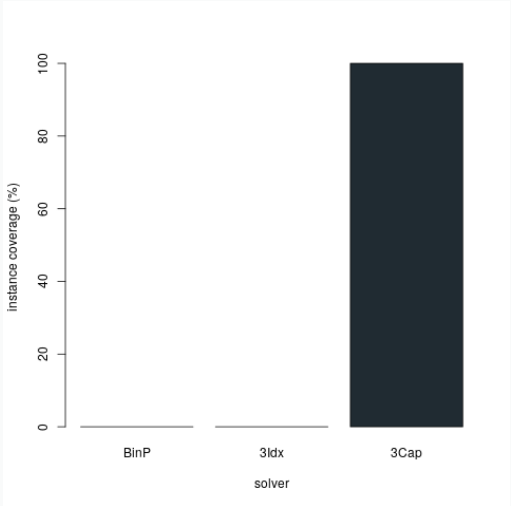
Laufzeit: Ø **0.2*s***

Vergleich der $b = 3$ Solver (I)



ZEITLIMIT 30*min*

	BinP	3Idx
Optimal	0%	0%
Laufzeit	– – –	– – –
Abweichung	– – –	– – –



ZEITLIMIT 45*min*

	BinP	3Idx
Optimal	0%	0%
Laufzeit	– – –	– – –
Abweichung	– – –	– – –

ZEITLIMIT 60*min*

	BinP	3Idx
Optimal	*–tbd–*	*–tbd–*
Laufzeit	*–tbd–*	*–tbd–*
Abweichung	*–tbd–*	*–tbd–*

3Cap-Heuristik

Abweichung vom Optimum: ***tbd***

Laufzeit: ∅ **1.0*s***

Ausblick

- Konstruktive Heuristik für $\boldsymbol{b = 4}$ entwickeln
- Verbesserungsverfahren implementieren
- Realistischere Test-Instanzen basierend auf **Briskorn (2018)**
- Einfluss von $\boldsymbol{m}$ auf die Schwierigkeit des Problems untersuchen (+20%, +30%, ...)
- Evaluation