



Tree Search and the EURO/ROADEF challenge

Luc Libralesso, Florian Fontan

May 10, 2019

G-SCOP

Table of contents

1. Glass Cutting Challenge ?
2. Branching Scheme
3. Anytime Algorithms & Tree Search
4. Results and Conclusion

Glass Cutting Challenge ?

- Presented by the French and European *Operations Research* societies
- International competition

- Presented by the French and European *Operations Research* societies
- International competition
- A challenge every two years
 - 2012: Google
 - 2014: SNCF (state-owned railway company)
 - 2016: Air Liquide

- Presented by the French and European *Operations Research* societies
- International competition
- A challenge every two years
 - 2012: Google
 - 2014: SNCF (state-owned railway company)
 - 2016: Air Liquide
 - **2018: Saint Gobain**



- Founded in 1665
- produces pipes, mirrors, mortars and glass



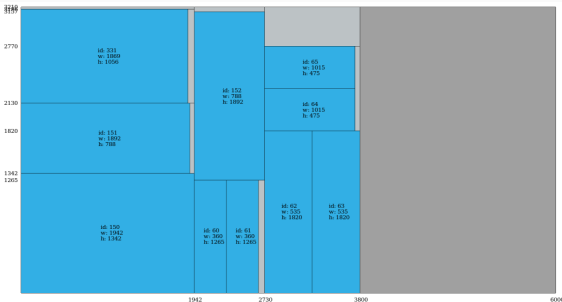
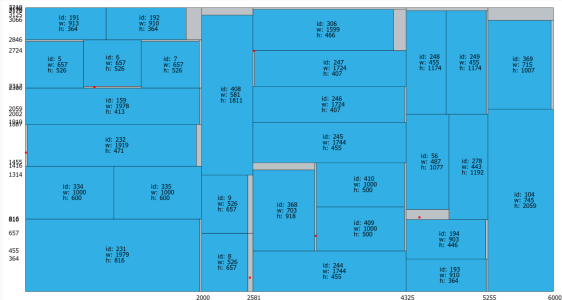
- Founded in 1665
- produces pipes, mirrors, mortars and glass

Cut rectangular glass items from big glass plates (Plates)

How to make glass



One of our solutions



Glass cutting Problem

OBJECTIVE:

minimize waste

Glass cutting Problem

OBJECTIVE:

minimize waste

DATA:

- Items (defined width and height, rotation possible)

Glass cutting Problem

OBJECTIVE:

minimize waste

DATA:

- Items (defined width and height, rotation possible)
- Stacks (chain precedence constraints)

Glass cutting Problem

OBJECTIVE:

minimize waste

DATA:

- Items (defined width and height, rotation possible)
- Stacks (chain precedence constraints)
- 100 Plates (6m x 3m) with defects

Glass cutting Problem

OBJECTIVE:

minimize waste

DATA:

- Items (defined width and height, rotation possible)
- Stacks (chain precedence constraints)
- 100 Plates (6m x 3m) with defects

CONSTRAINTS:

- **guillotine constraint**

Guillotine constraint

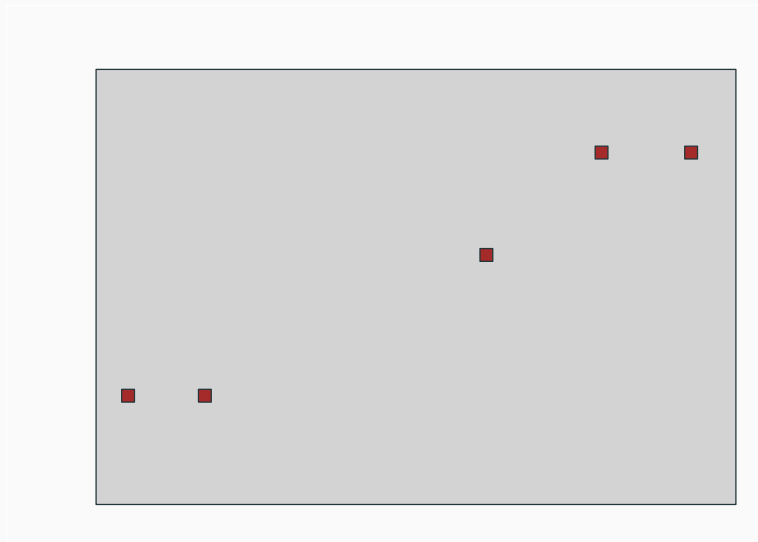


Figure 1: Example of a solution

Guillotine constraint

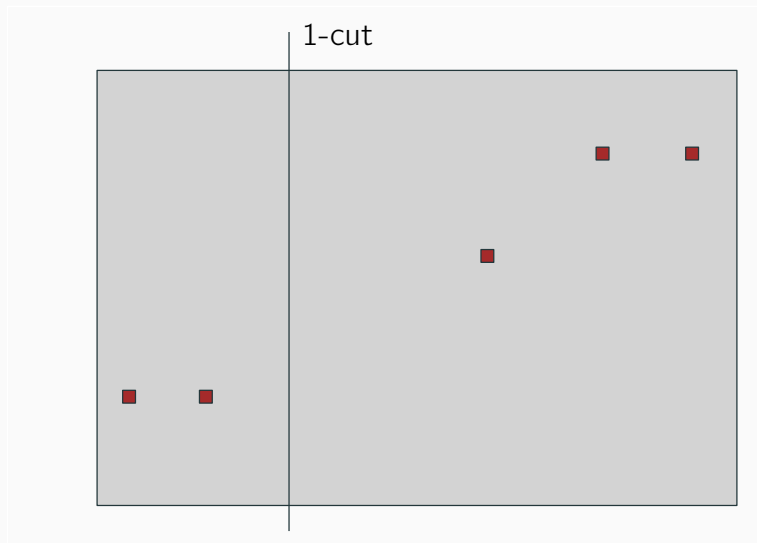


Figure 2: Example of a solution

Guillotine constraint

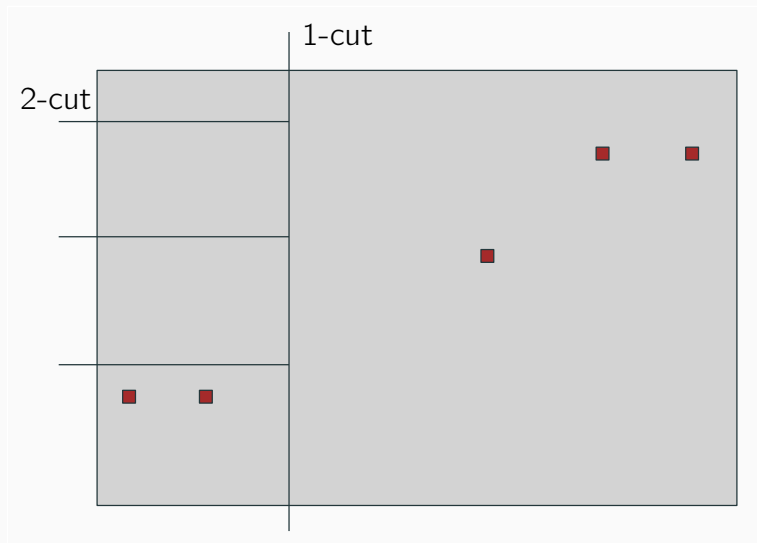


Figure 3: Example of a solution

Guillotine constraint

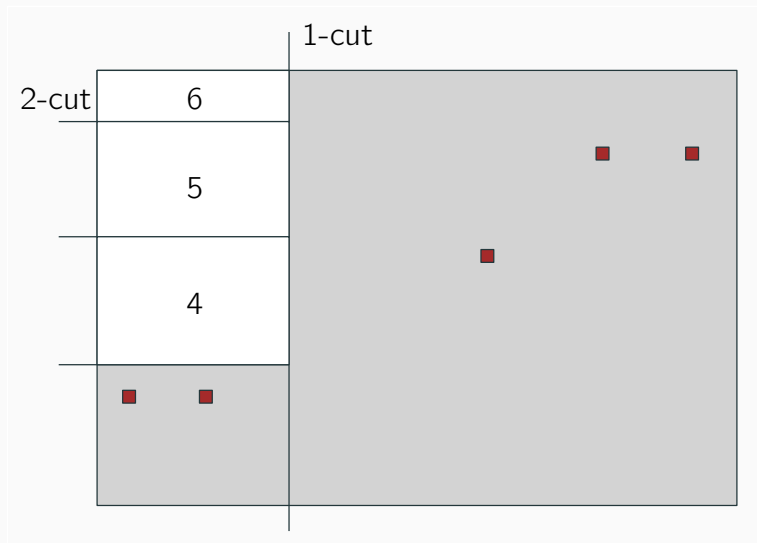


Figure 4: Example of a solution

Guillotine constraint

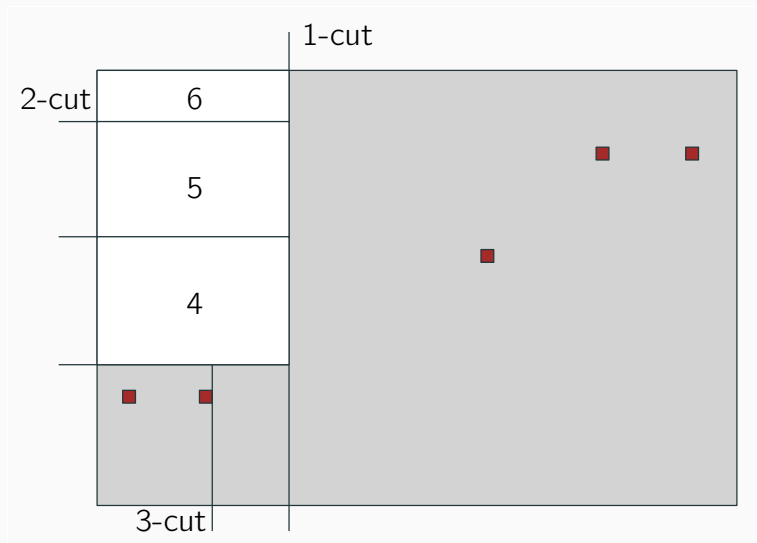


Figure 5: Example of a solution

Guillotine constraint

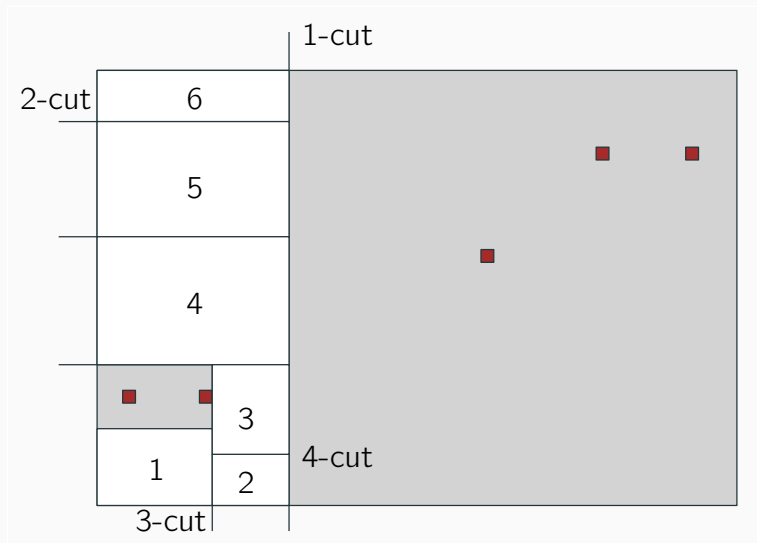
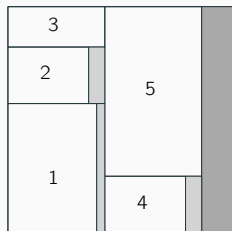


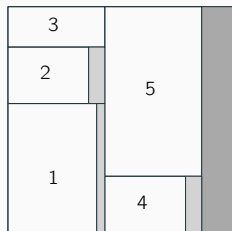
Figure 6: Example of a solution

guillotine cuts and not allowed cuts

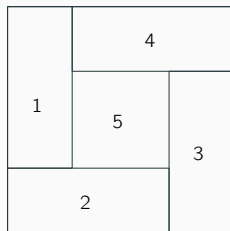


guillotine

guillotine cuts and not allowed cuts

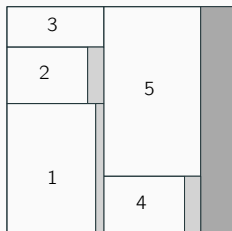


guillotine

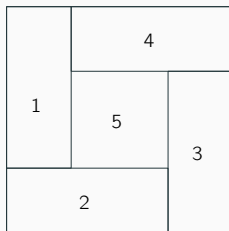


non-guillotine

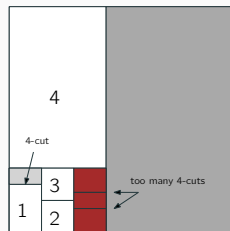
guillotine cuts and not allowed cuts



guillotine



non-guillotine



too many 4-cuts

Precedence constraints

OBJECTIVE:

minimize waste

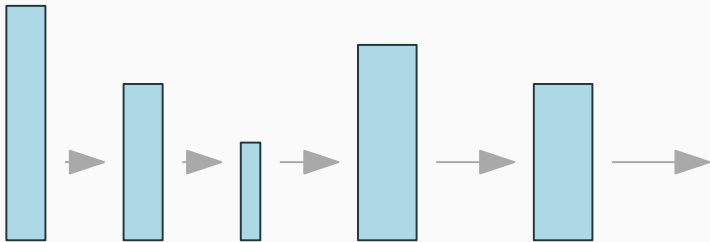
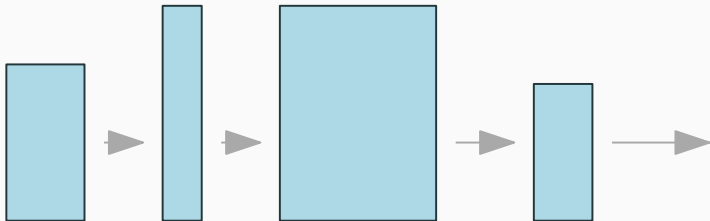
DATA:

- Items (defined width and height, rotation possible)
- Stacks (chain precedence constraints)
- 100 Plates (6m x 3m) with defects

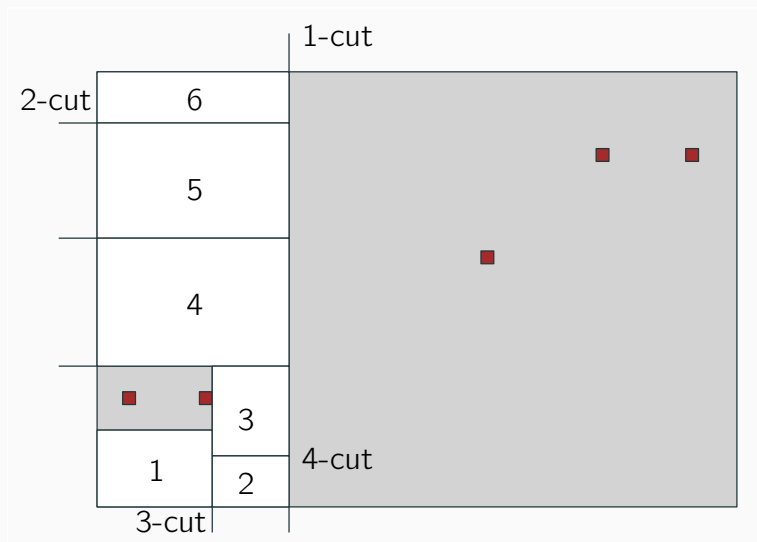
CONSTRAINTS:

- guillotine constraint
- **all items produced in a valid order**

Precedence Constraint



Precedence Constraint



Defect avoidance

OBJECTIVE:

minimize waste

DATA:

- Items (defined width and height, rotation possible)
- Stacks (chain precedence constraints)
- 100 Plates (6m x 3m) with defects

CONSTRAINTS:

- guillotine constraint
- all items produced in a valid order
- **no defects in items**
- **no cut on a defect**

minimum/maximum cut size

OBJECTIVE:

minimize waste

DATA:

- Items (defined width and height, rotation possible)
- Stacks (chain precedence constraints)
- 100 Plates (6m x 3m) with defects

CONSTRAINTS:

- guillotine constraint
- all items produced in a valid order
- no defects in items
- no cut on a defect
- **min/max constraints on cuts and waste**

Min-waste constraint

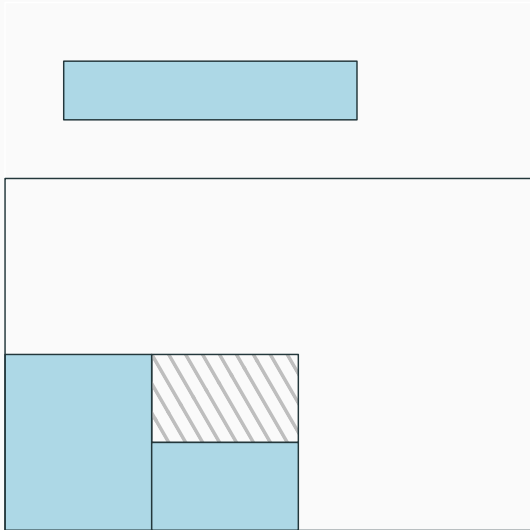


Figure 7: Min waste: easy case

Min-waste constraint

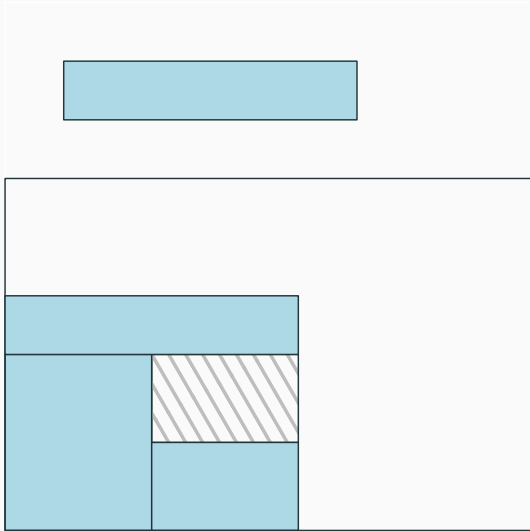


Figure 8: Min waste: easy case

Min-waste constraint

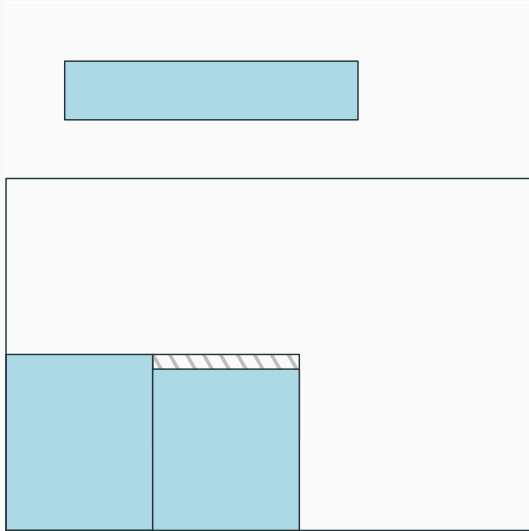


Figure 9: Min waste: more difficult

Min-waste constraint

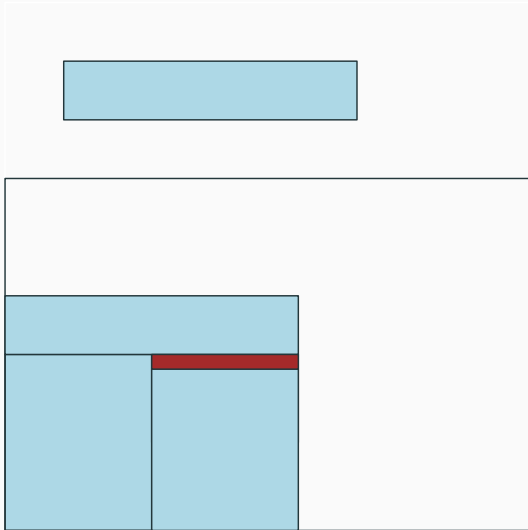


Figure 10: Min waste: more difficult

Min-waste constraint

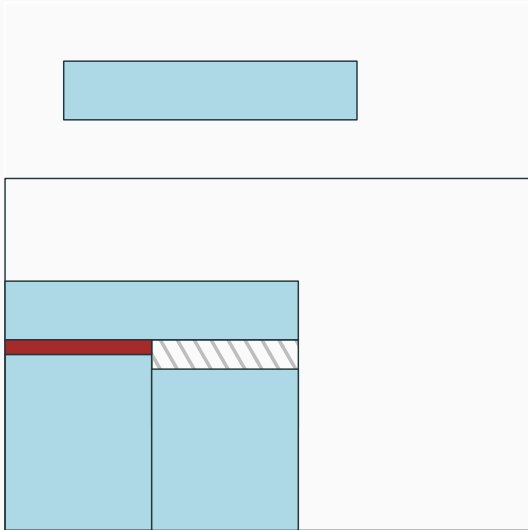


Figure 11: Min waste: more difficult

Min-waste constraint

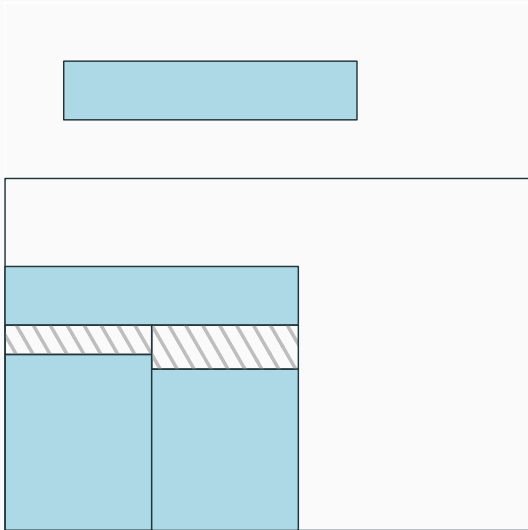


Figure 12: Min waste: more difficult

Glass cutting Problem

The problem is \mathcal{NP} -Hard.

Glass cutting Problem

The problem is \mathcal{NP} -Hard.

Difficult problem and big instances

Glass cutting Problem

The problem is \mathcal{NP} -Hard.

Difficult problem and big instances

We use anytime algorithms (meta-heuristics)

In this talk

We generate an implicit search tree. (next section)

It is called **Branching Scheme**

In this talk

We generate an implicit search tree. (next section)
It is called **Branching Scheme**

We explore this search tree cleverly (section after)
we use **anytime tree searches**

In this talk

We generate an implicit search tree. (next section)
It is called **Branching Scheme**

We explore this search tree cleverly (section after)
we use **anytime tree searches**

Results & conclusions & perspectives

Branching Scheme

Branching scheme

- Root node (initial solution): empty solution, no item placed.



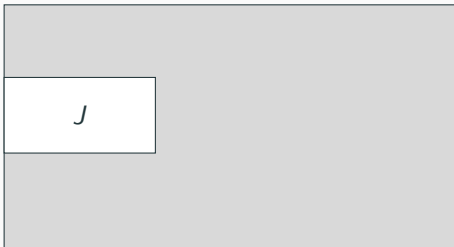
Branching scheme

- Root node (initial solution): empty solution, no item placed.
- Children: where to place items?



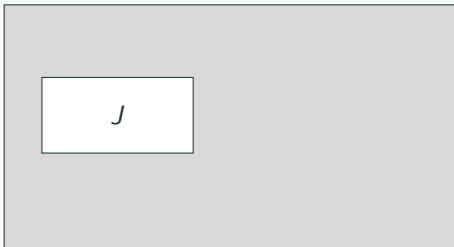
Branching scheme

- Root node (initial solution): empty solution, no item placed.
- Children: where to place items?



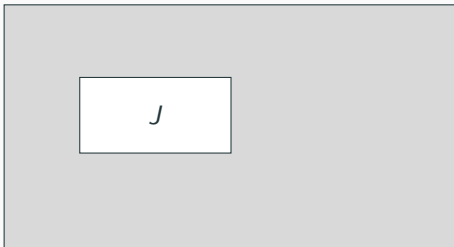
Branching scheme

- Root node (initial solution): empty solution, no item placed.
- Children: where to place items?



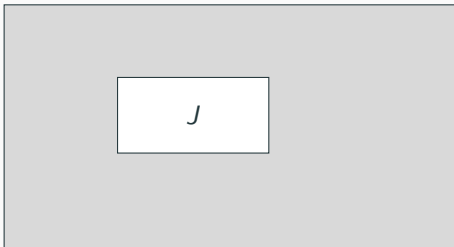
Branching scheme

- Root node (initial solution): empty solution, no item placed.
- Children: where to place items?



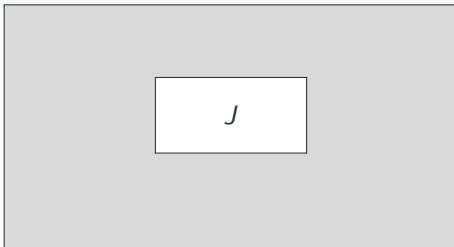
Branching scheme

- Root node (initial solution): empty solution, no item placed.
- Children: where to place items?



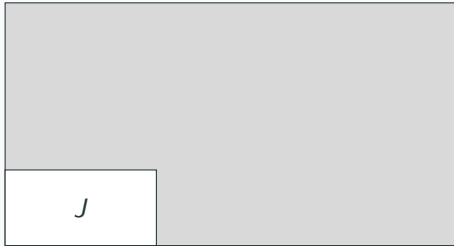
Branching scheme

- Root node (initial solution): empty solution, no item placed.
- Children: where to place items?



Branching scheme

- Root node (initial solution): empty solution, no item placed.
- Children: where to place items?
- In a corner?



Dominant sets

- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



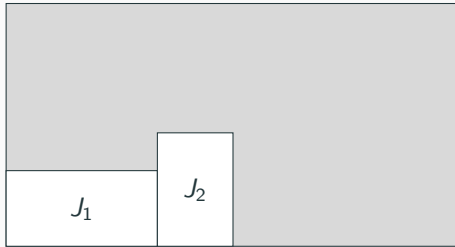
Dominant sets

- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



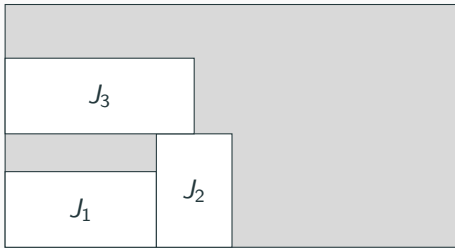
Dominant sets

- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



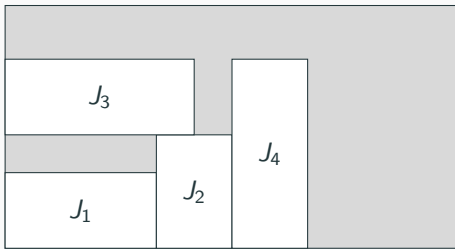
Dominant sets

- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



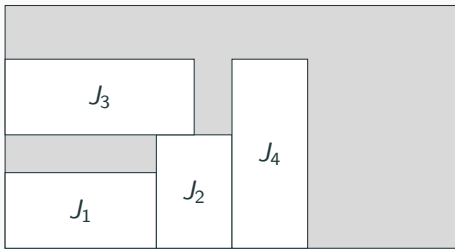
Dominant sets

- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



Dominant sets

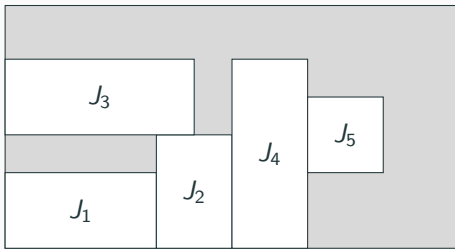
- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



- There exists an optimal solution such that for each item of the solution, its left and its bottom touch either another item or a border.

Dominant sets

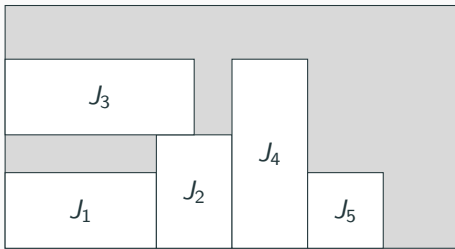
- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



- There exists an optimal solution such that for each item of the solution, its left and its bottom touch either another item or a border.

Dominant sets

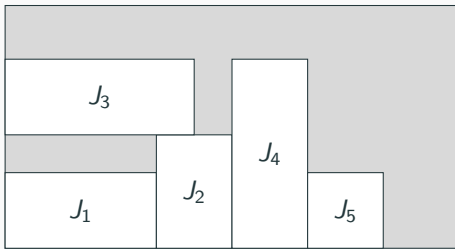
- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



- There exists an optimal solution such that for each item of the solution, its left and its bottom touch either another item or a border.

Dominant sets

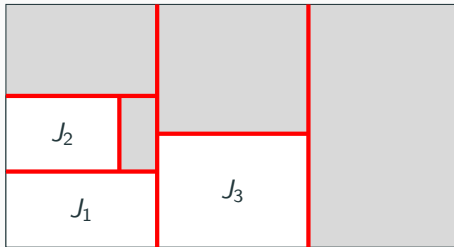
- For the classical two-dimensional packing problem, it is dominant to place items in a corner.



- There exists an optimal solution such that for each item of the solution, its left and its bottom touch either another item or a border.
- Does the property hold in our problem?

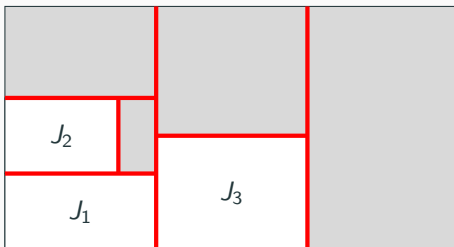
Dominant sets

- The property holds for the guillotine two-dimensional packing problem.



Dominant sets

- The property holds for the guillotine two-dimensional packing problem.



- There exists an optimal solution such that every left side of its vertical cuts and every bottom sides of its horizontal cuts touch an item.

Dominant sets

- The border of an item corresponds to one unique cut (or border)

possible



not possible



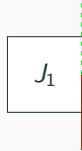
Dominant sets

- The border of an item corresponds to one unique cut (or border)

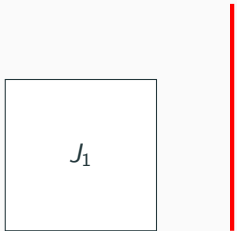
possible



not possible



- Then the proof is trivial



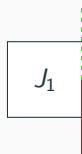
Dominant sets

- The border of an item corresponds to one unique cut (or border)

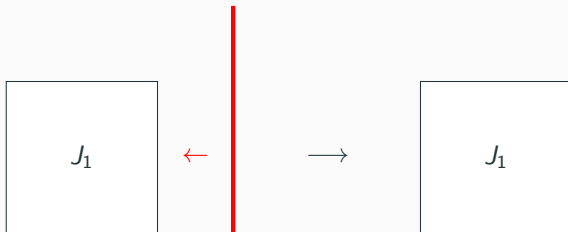
possible



not possible



- Then the proof is trivial



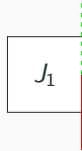
Dominant sets

- The border of an item corresponds to one unique cut (or border)

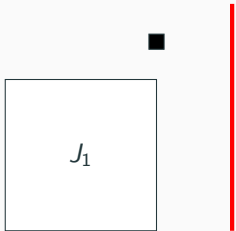
possible



not possible



- Then the proof is trivial



- And with defects?

Dominant sets

- The border of an item corresponds to one unique cut (or border)

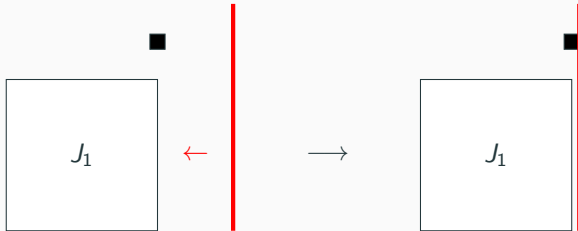
possible



not possible



- Then the proof is trivial



- And with defects?

Dominant sets

- The border of an item corresponds to one unique cut (or border)

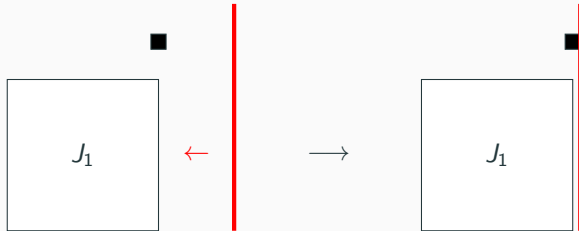
possible



not possible



- Then the proof is trivial



- And with defects? And with precedences?

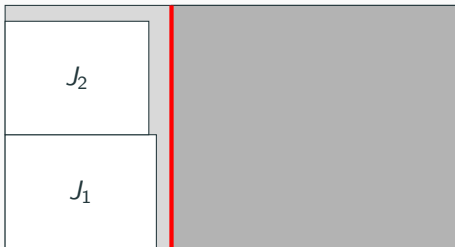
- No defects, no precedences, but minimum waste.

Dominant sets

- No defects, no precedences, but minimum waste.
- There exists an optimal solution such that every left side of its vertical cuts and every bottom sides of its horizontal cuts touch an item

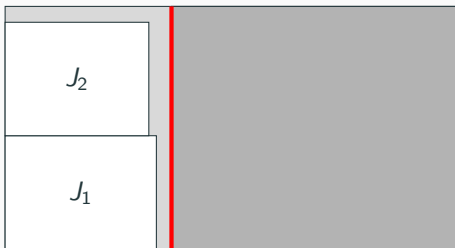
Dominant sets

- No defects, no precedences, but minimum waste.
- There exists an optimal solution such that every left side of its vertical cuts and every bottom sides of its horizontal cuts touch an item

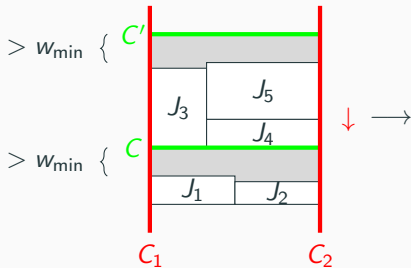


Dominant sets

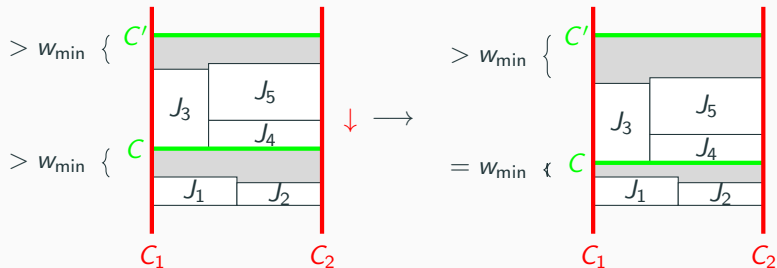
- No defects, no precedences, but minimum waste.
- There exists an optimal solution such that every left side of its vertical cuts and every bottom sides of its horizontal cuts touch an item **or is exactly at w_{\min} from an item.**



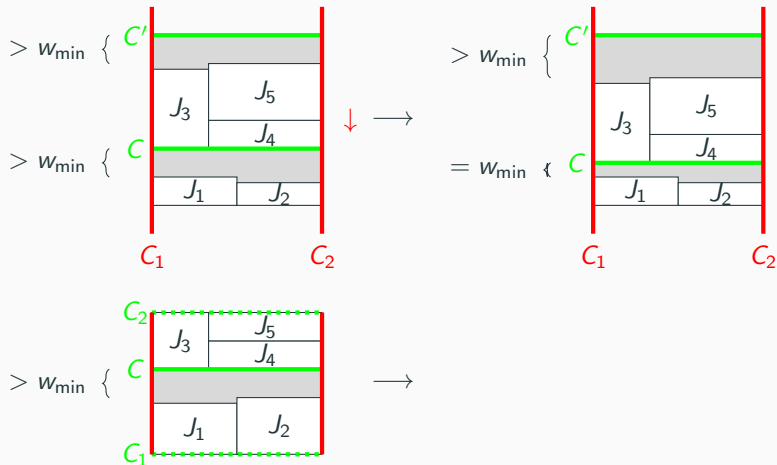
Dominant sets



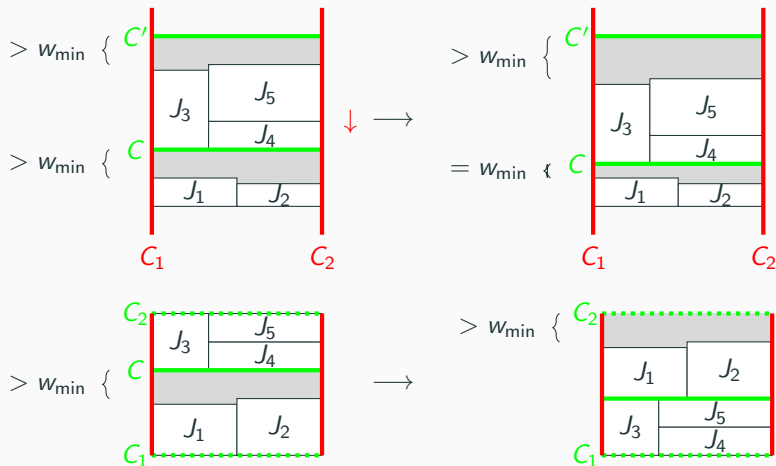
Dominant sets



Dominant sets



Dominant sets



Dominant sets

- But what happens with minimum waste and precedences? or with minimum waste and defects?

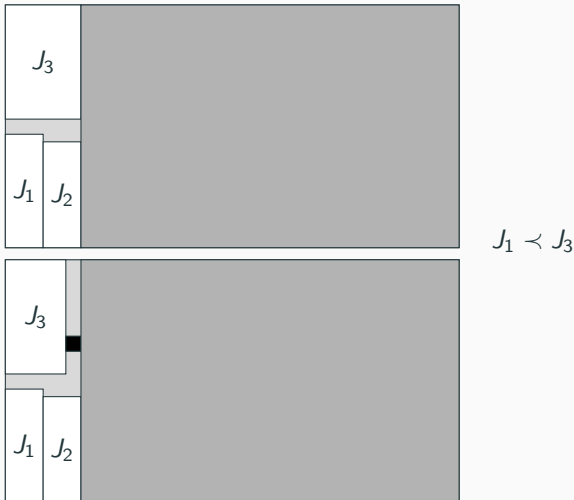
Dominant sets

- But what happens with minimum waste and precedences? or with minimum waste and defects?



Dominant sets

- But what happens with minimum waste and precedences? or with minimum waste and defects?



- Placing items in a corner is not dominant.

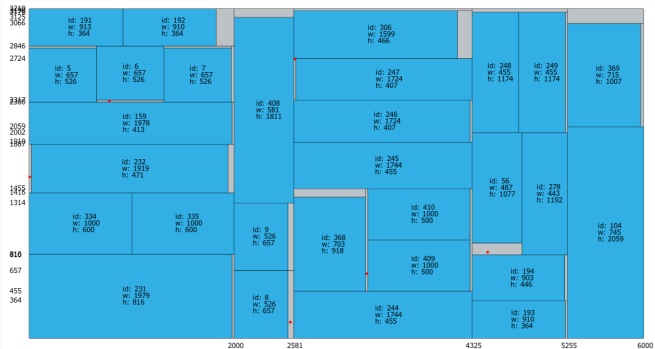
- Placing items in a corner is not dominant.
- Finding a dominant branching scheme that does not increase too much the number of nodes is hard.

- Placing items in a corner is not dominant.
- Finding a dominant branching scheme that does not increase too much the number of nodes is hard.
- Placing items in a corner is still dominant for several subproblems

- Placing items in a corner is not dominant.
- Finding a dominant branching scheme that does not increase too much the number of nodes is hard.
- Placing items in a corner is still dominant for several subproblems
- We are only looking for heuristics and not for exact algorithms.

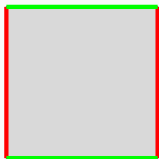
- Placing items in a corner is not dominant.
- Finding a dominant branching scheme that does not increase too much the number of nodes is hard.
- Placing items in a corner is still dominant for several subproblems
- We are only looking for heuristics and not for exact algorithms.

⇒ We base our branching scheme on it anyway.



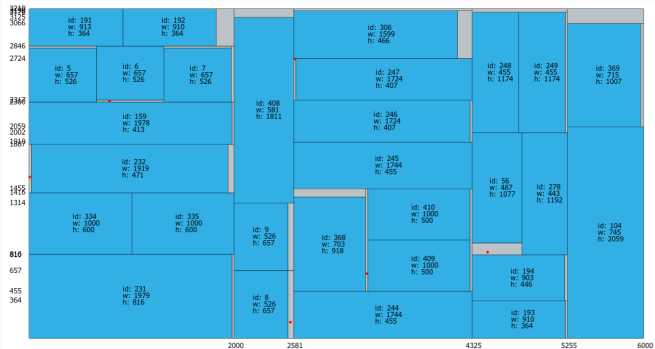
next 2-cut

3-cut



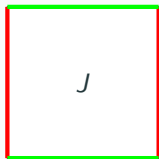
next
3-cut

2-cut



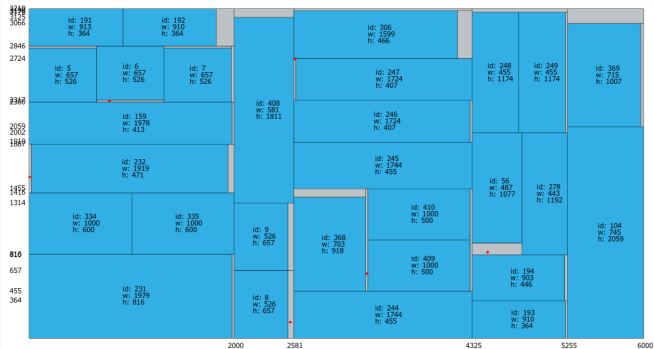
next 2-cut

3-cut

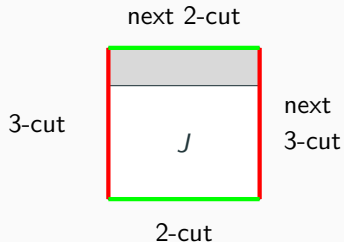


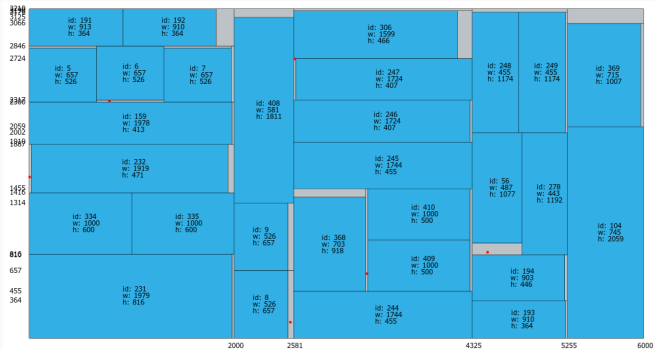
next
3-cut

2-cut



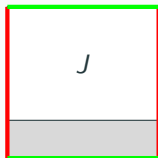
- 1 item at the bottom of the block (231, 334, 335...)





next 2-cut

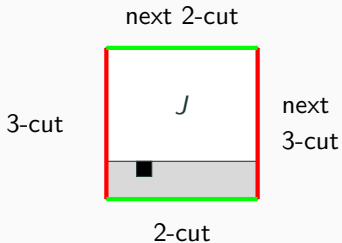
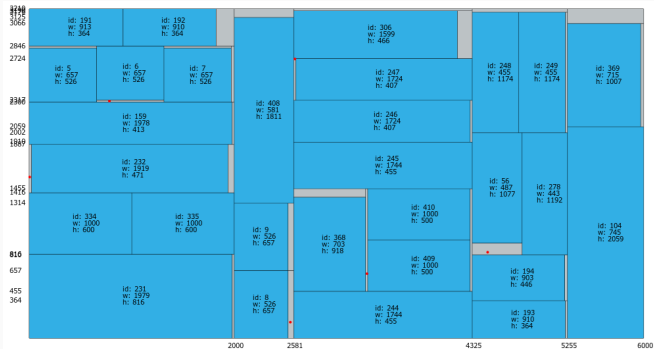
3-cut



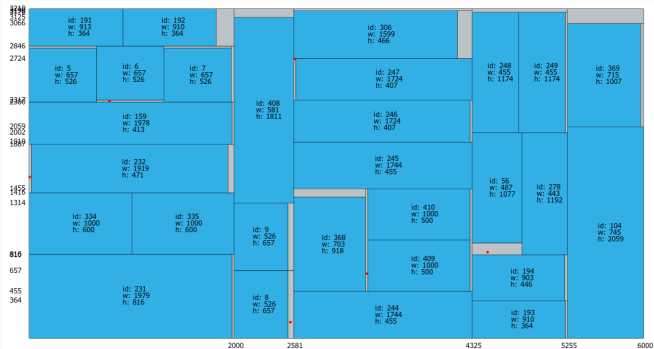
2-cut

next
3-cut

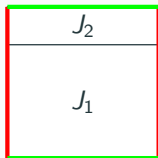
- 1 item at the bottom of the block (231, 334, 335...)
- 1 item at the top of the block (6, 56)



- 1 item at the bottom of the block (231, 334, 335...)
- 1 item at the top of the block (6, 56)



next 2-cut

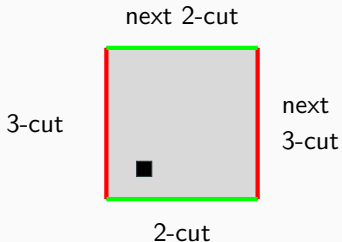
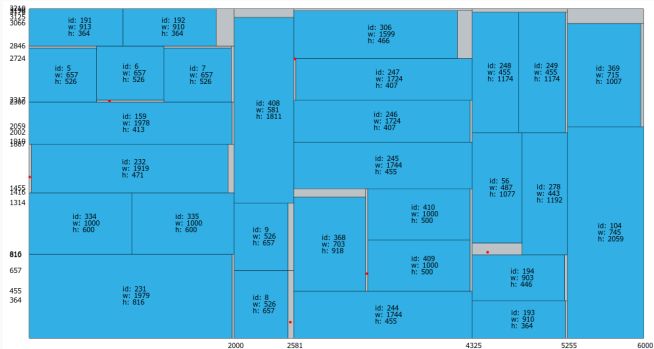


3-cut

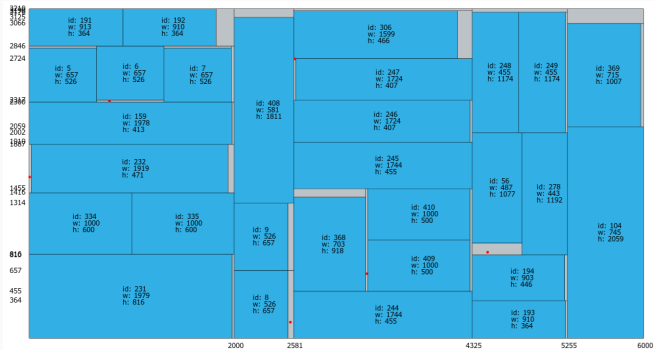
next
3-cut

2-cut

- 1 item at the bottom of the block (231, 334, 335...)
- 1 item at the top of the block (6, 56)
- 2 items (409 and 410)

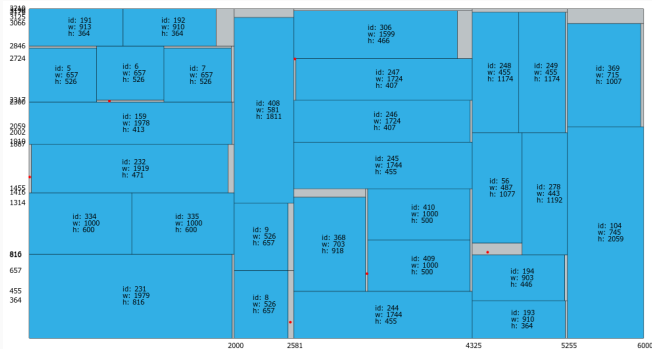


- 1 item at the bottom of the block (231, 334, 335...)
- 1 item at the top of the block (6, 56)
- 2 items (409 and 410)
- a defect (before 232, before 409 and 410, before 247)



Four depths of insertions:

- 0: on a new plate (231)
- 1: new current 1-cut (8, 244, 193, 104)
- 2: new 2-cut (334, 159, defect before 232...)
- 2: new 3-cut (335, 6, 7, defect before 409 and 410...)

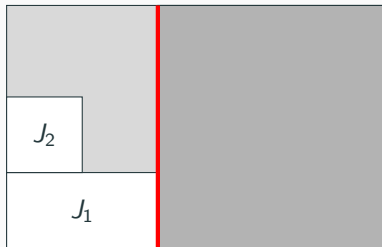


Four depths of insertions:

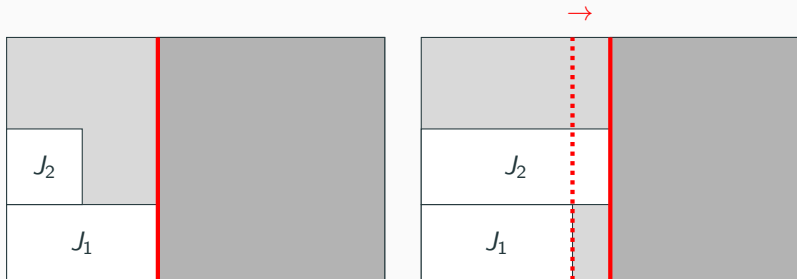
- 0: on a new plate (231)
- 1: new current 1-cut (8, 244, 193, 104)
- 2: new 2-cut (334, 159, defect before 232...)
- 2: new 3-cut (335, 6, 7, defect before 409 and 410...)

Note that items may be rotated.

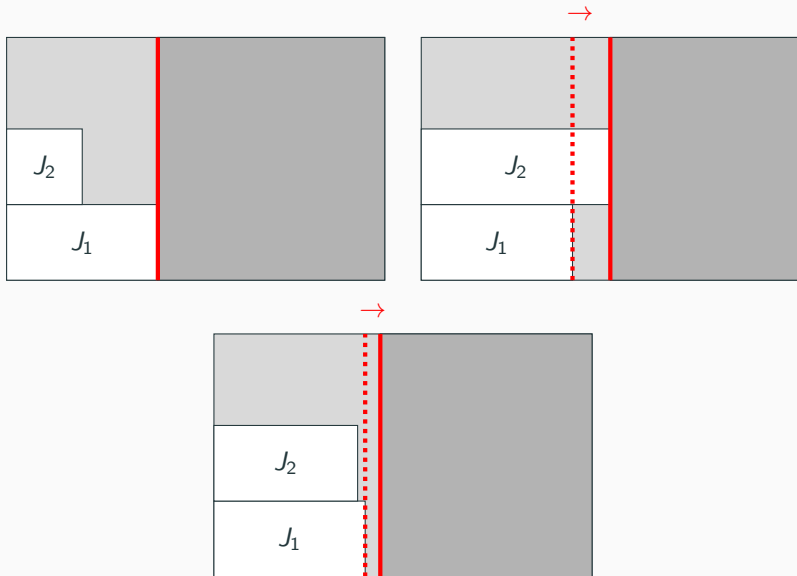
Computing last cuts positions



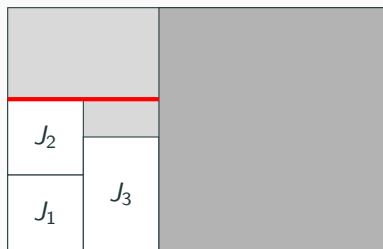
Computing last cuts positions



Computing last cuts positions

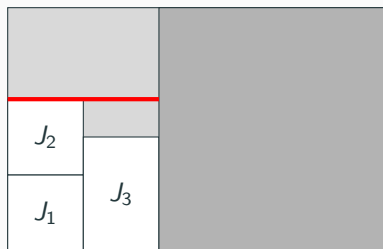


Computing last cuts positions

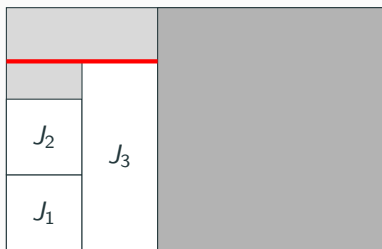


feasible

Computing last cuts positions

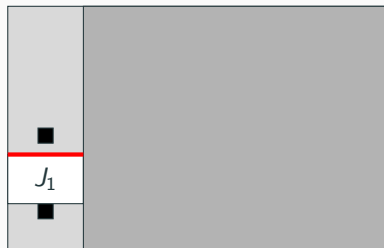


feasible

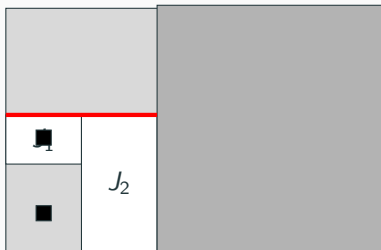
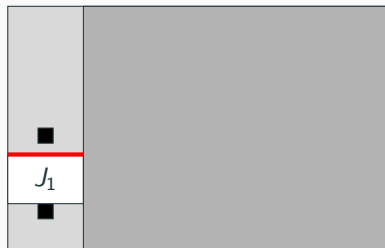


infeasible

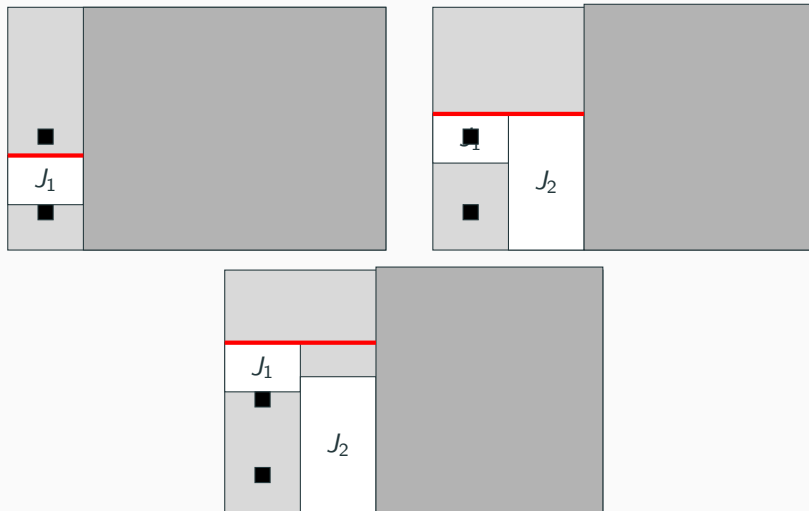
Computing last cuts positions



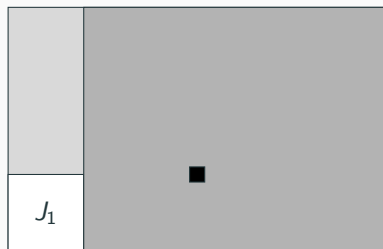
Computing last cuts positions



Computing last cuts positions

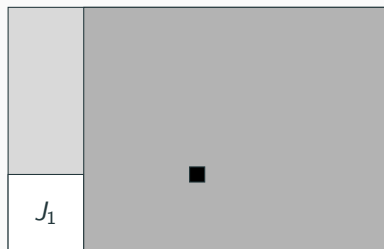


Computing last cuts positions

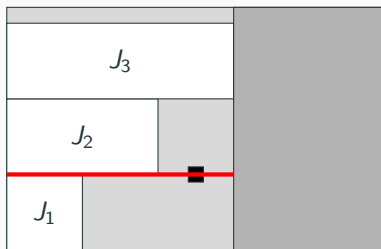


feasible

Computing last cuts positions



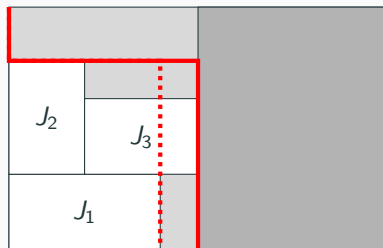
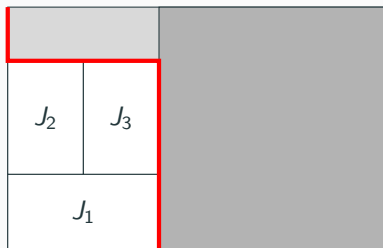
feasible



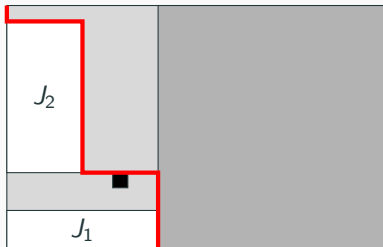
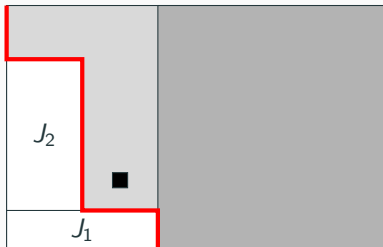
infeasible

Pseudo-dominance

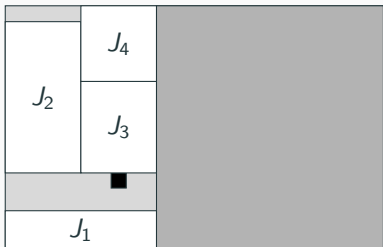
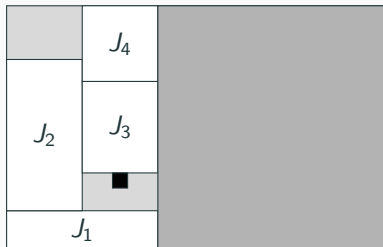
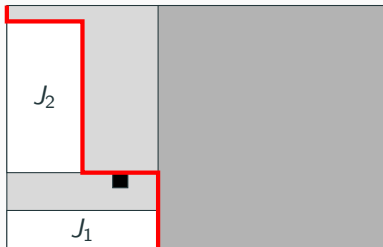
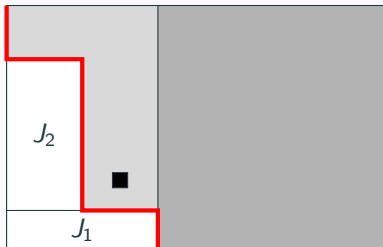
- Dominance rule: if two partial solutions S_1 and S_2 contain the same items and the front of S_1 is before the front of S_2 , then S_1 dominates S_2 .



Pseudo-dominance

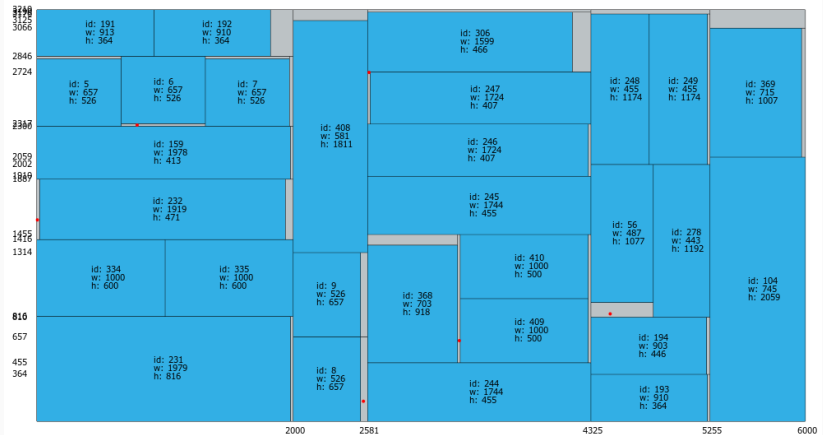


Pseudo-dominance



Symmetry breaking

- Symmetry breaking strategy: for two consecutive blocks, the one with the smallest minimum item id comes before.



Branching scheme

- not dominant, but good compromise

Branching scheme

- not dominant, but good compromise
- all constraints taken into account

Branching scheme

- not dominant, but good compromise
- all constraints taken into account
- very high number of nodes

Branching scheme

- not dominant, but good compromise
- all constraints taken into account
- very high number of nodes
- pseudo-dominance rules and symmetry breaking strategy

Anytime Algorithms & Tree Search

Some formalizations

Two separate parts in *Tree Search*:

Branching Scheme: The implicit search tree. Contains

- root node
- how to generate children from a node
- lower bounds
- guides (estimation of node quality)

Some formalizations

Two separate parts in *Tree Search*:

Branching Scheme: The implicit search tree. Contains

- root node
- how to generate children from a node
- lower bounds
- guides (estimation of node quality)

Tree Search Algorithms: How to visit the tree (Branch & Bound, Greedy, *others* ?)

Anytime algorithms (meta-heuristics) - A landscape

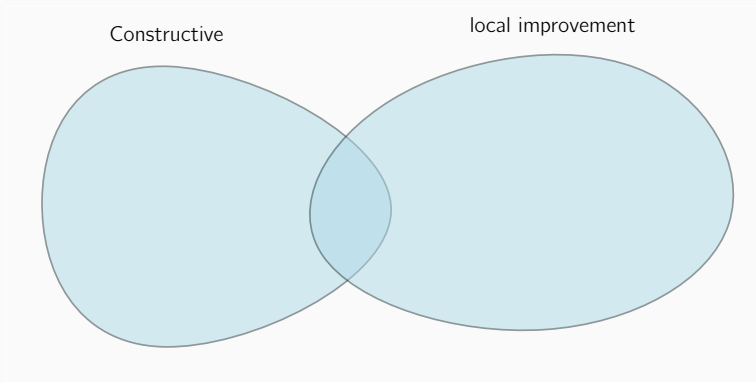


Figure 13: Anytime algorithms: a classification

Anytime algorithms (meta-heuristics) - A landscape

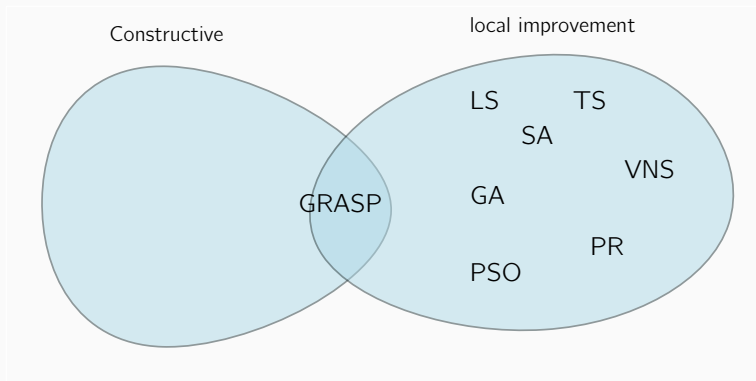


Figure 14: Anytime algorithms: a classification

Anytime algorithms (meta-heuristics) - A landscape

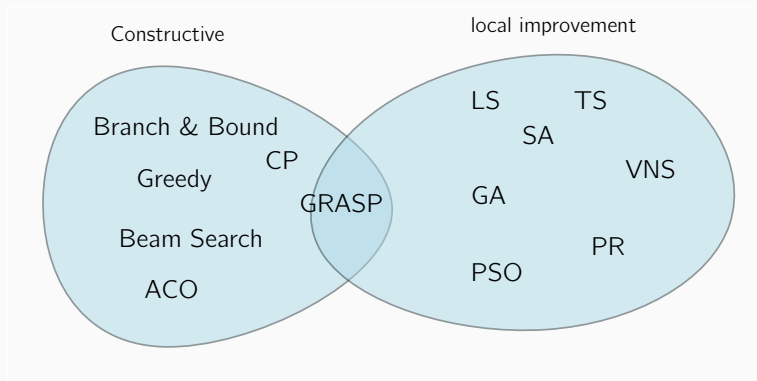


Figure 15: Anytime algorithms: a classification

Anytime algorithms (meta-heuristics)

- Many anytime algorithms are based on **Local Improvement**
- A few are **constructive**

Anytime algorithms (meta-heuristics)

- Many anytime algorithms are based on **Local Improvement**
- A few are **constructive**

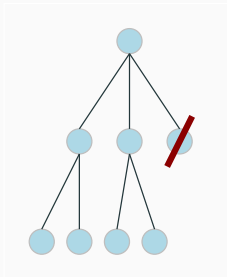
Why ?

Can constructive methods be competitive with Local Searches ?

Focus on constructive algorithms

They explore a tree

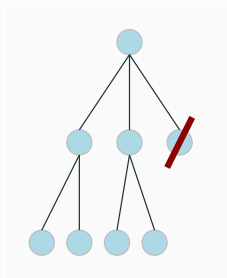
Branch & Bound



Focus on constructive algorithms

They explore a tree

Branch & Bound

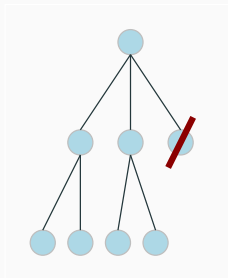


time –
quality +

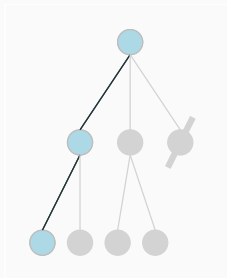
Focus on constructive algorithms

They explore a tree

Branch & Bound



Greedy

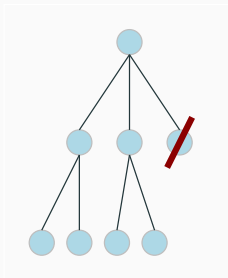


time –
quality +

Focus on constructive algorithms

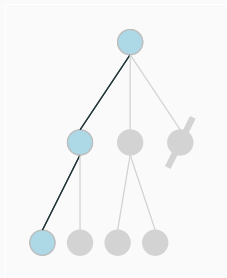
They explore a tree

Branch & Bound



time –
quality +

Greedy

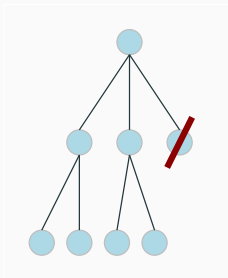


time +
quality –

Focus on constructive algorithms

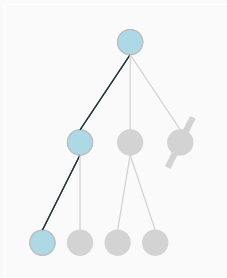
They explore a tree

Branch & Bound



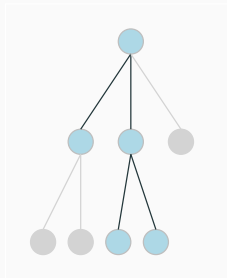
time –
quality +

Greedy



time +
quality –

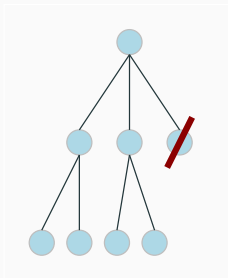
Beam Search



Focus on constructive algorithms

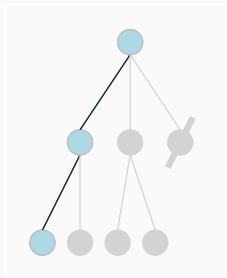
They explore a tree

Branch & Bound



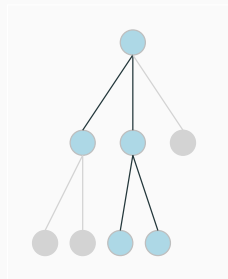
time –
quality +

Greedy



time +
quality –

Beam Search



depends



Figure 16: time vs quality axis

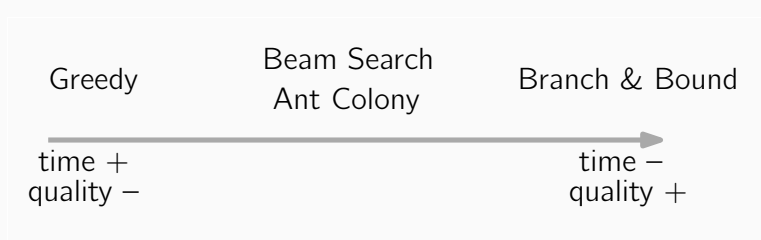


Figure 17: time vs quality axis



Figure 17: time vs quality axis

- *Beam Search* behaves like a BFS when the beam is big.

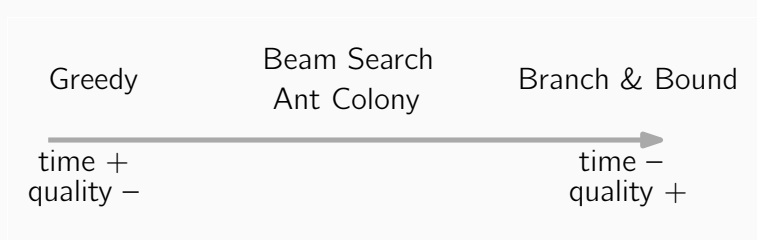


Figure 17: time vs quality axis

- *Beam Search* behaves like a BFS when the beam is big.
- *Ant Colony* depends too much on the structure of the problem



Figure 17: time vs quality axis

- *Beam Search* behaves like a BFS when the beam is big.
- *Ant Colony* depends too much on the structure of the problem

Are there some other algorithms ?

Not in *Meta-heuristics* nor *Operations Research*¹

¹to the best of our knowledge

²and a bit in CP

Not in *Meta-heuristics* nor *Operations Research*¹

But in AI^2

¹to the best of our knowledge

²and a bit in CP

Not in *Meta-heuristics* nor *Operations Research*¹

But in AI^2

We import those methods

¹to the best of our knowledge

²and a bit in CP

Many algorithms usable in Operations Research:

Beam Stack Search ([ZH05]), **Limited Discrepancy Search** ([HG95]),
BULB ([FK05]), others

Many algorithms usable in Operations Research:

Beam Stack Search ([ZH05]), **Limited Discrepancy Search** ([HG95]),
BULB ([FK05]), others

Our Approach (*Memory Bounded A**)

MBA*: *A** or *Best First* with a limit on the number of nodes (like *Beam Search*)

MBA* - an example

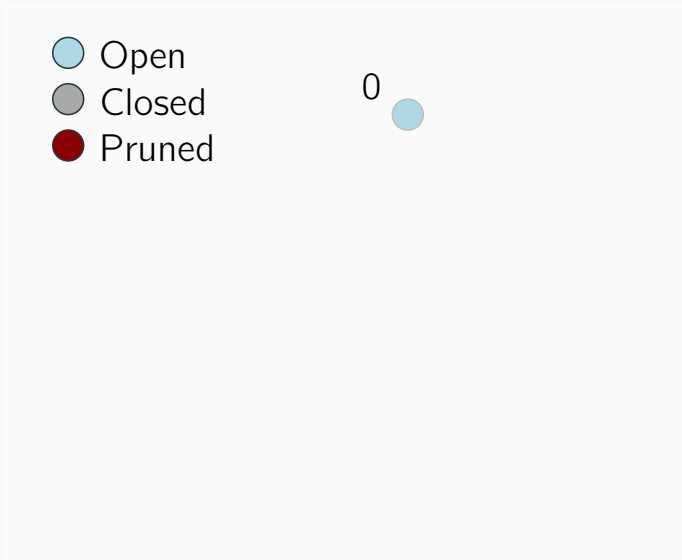


Figure 18: MBA* with a maximum fringe size of 4

MBA* - an example

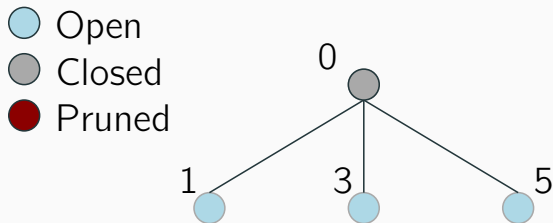


Figure 19: MBA* with a maximum fringe size of 4

MBA* - an example

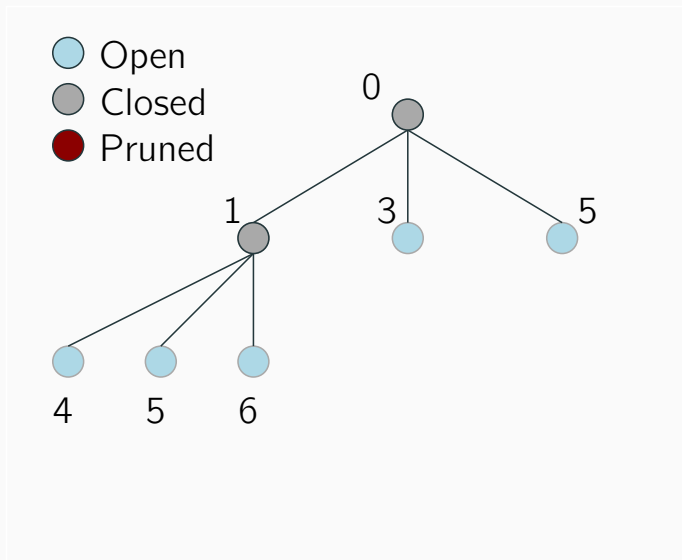


Figure 20: MBA* with a maximum fringe size of 4

MBA* - an example

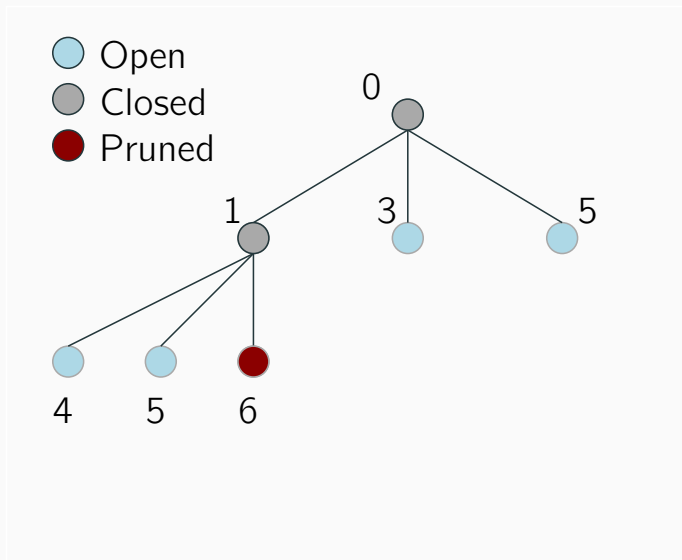


Figure 21: MBA* with a maximum fringe size of 4

MBA* - an example

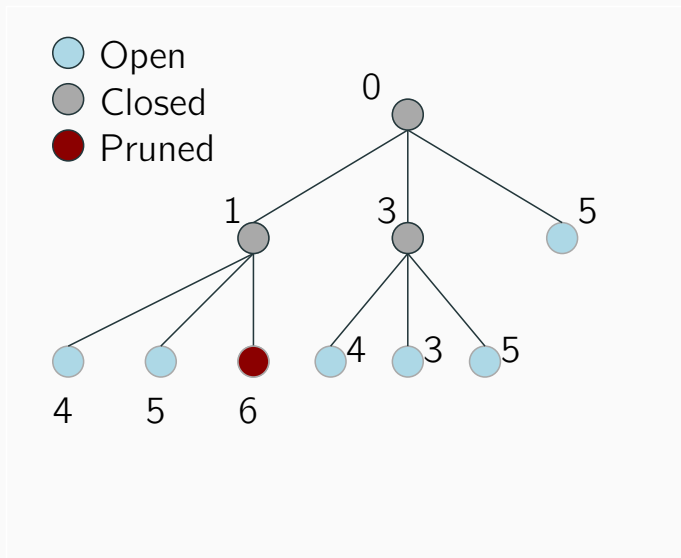


Figure 22: MBA* with a maximum fringe size of 4

MBA* - an example

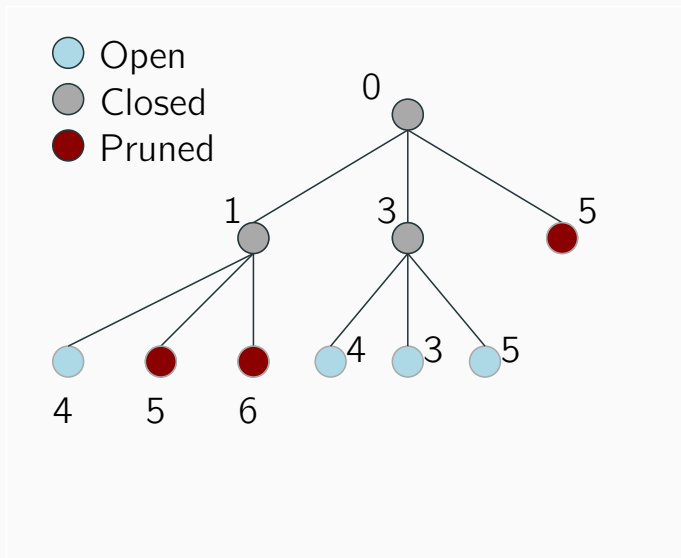


Figure 23: MBA* with a maximum fringe size of 4

MBA* - an example

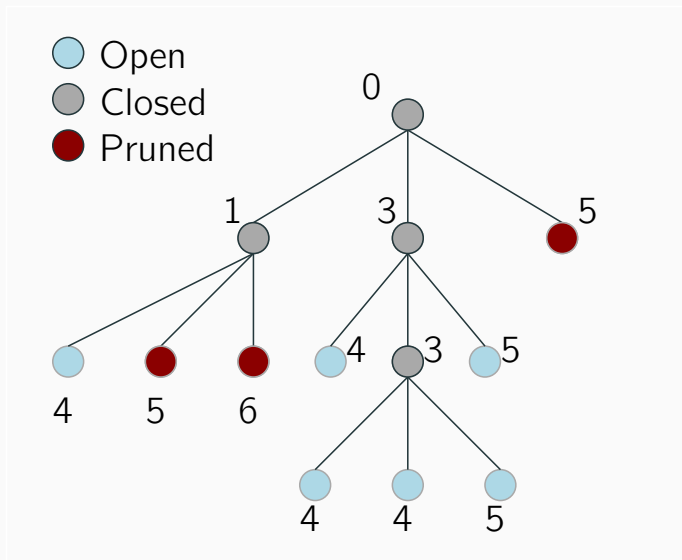


Figure 24: MBA* with a maximum fringe size of 4

MBA* - an example

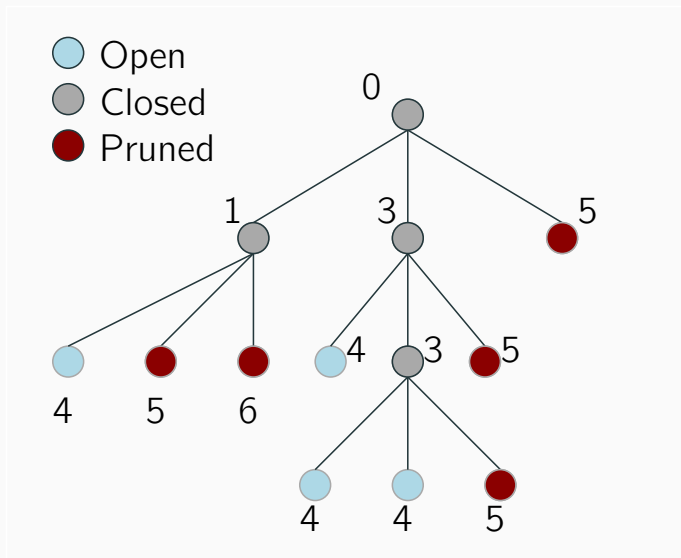
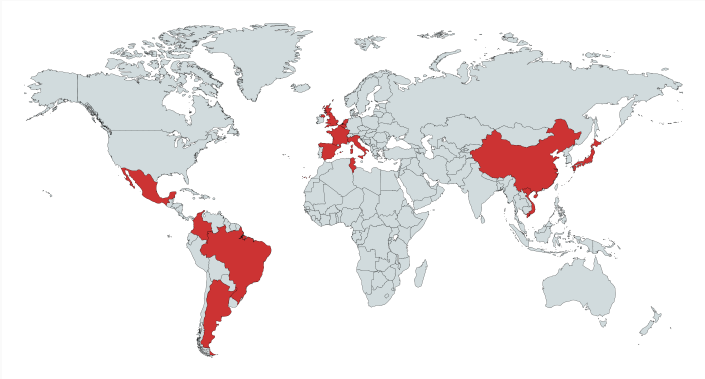


Figure 25: MBA* with a maximum fringe size of 4

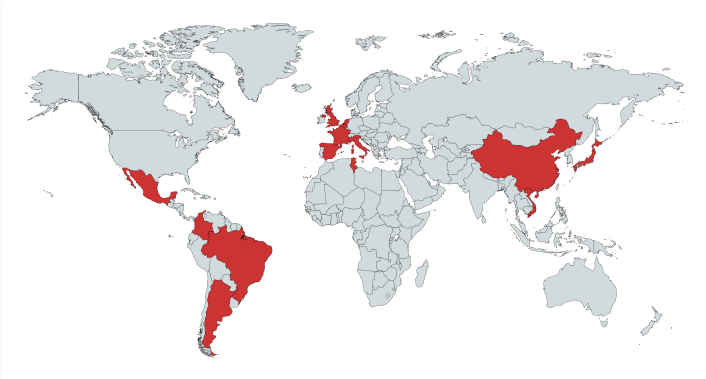
Results and Conclusion

- 20 over 74 international teams qualified for the Final phase.

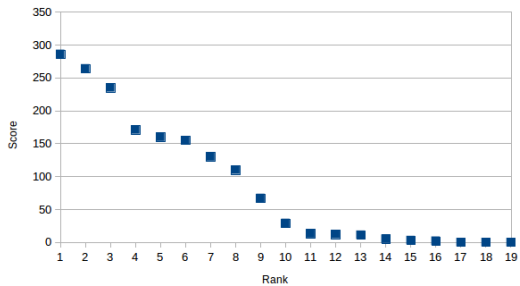


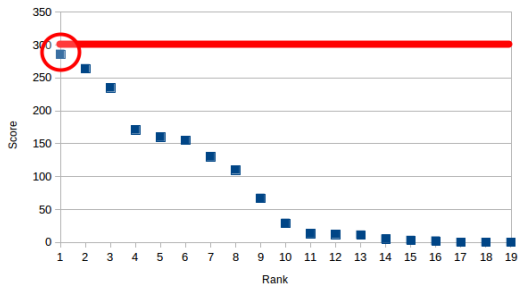
- 30 instances (15 known, 15 unknown), 1 hour running time.

- 20 over 74 international teams qualified for the Final phase.

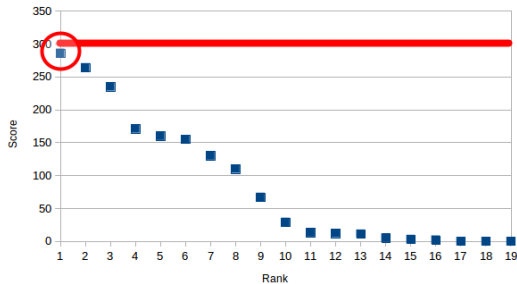


- 30 instances (15 known, 15 unknown), 1 hour running time.
- for each instance, a team earns 10 points minus the number of teams that found a better solution.

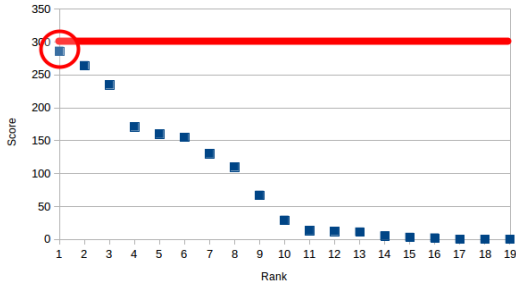




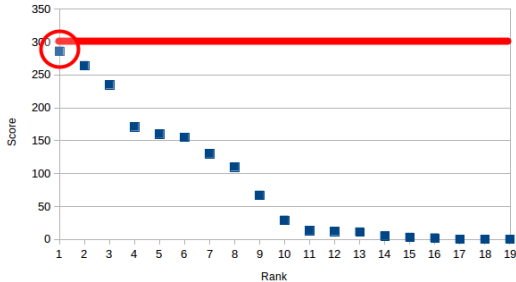
- Ranked first during the Final phase of the challenge.



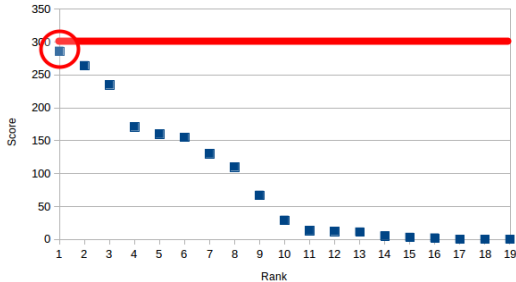
- Ranked first during the Final phase of the challenge.
- Best solutions found on 20 over 30 instances.



- Ranked first during the Final phase of the challenge.
- Best solutions found on 20 over 30 instances.
- Total waste: 493M, Total waste 2nd team: 506M.



- Ranked first during the Final phase of the challenge.
- Best solutions found on 20 over 30 instances.
- Total waste: 493M, Total waste 2nd team: 506M.
- Total waste new version: 469M; better on 26 instances compared to the challenge version.



- Ranked first during the Final phase of the challenge.
- Best solutions found on 20 over 30 instances.
- Total waste: 493M, Total waste 2nd team: 506M.
- Total waste new version: 469M; better on 26 instances compared to the challenge version.
- Anytime algorithm

Conclusion

- Algorithm design: implicit search tree + tree search algorithm.

Conclusion

- Algorithm design: implicit search tree + tree search algorithm.
- New simple and competitive tree search algorithm MBA*

Conclusion

- Algorithm design: implicit search tree + tree search algorithm.
- New simple and competitive tree search algorithm MBA*

Perspectives

- Problem remains open (mean gap to best known solution: 7%).

Conclusion

- Algorithm design: implicit search tree + tree search algorithm.
- New simple and competitive tree search algorithm MBA*

Perspectives

- Problem remains open (mean gap to best known solution: 7%).
- Combining tree search with local searches.

Conclusion

- Algorithm design: implicit search tree + tree search algorithm.
- New simple and competitive tree search algorithm MBA*

Perspectives

- Problem remains open (mean gap to best known solution: 7%).
- Combining tree search with local searches.
- Apply method for classical and industrial problems.

Questions or remarks ?



David Furcy and Sven Koenig.

Limited discrepancy beam search.

In *IJCAI*, pages 125–131, 2005.



William D Harvey and Matthew L Ginsberg.

Limited discrepancy search.

In *IJCAI (1)*, pages 607–615, 1995.



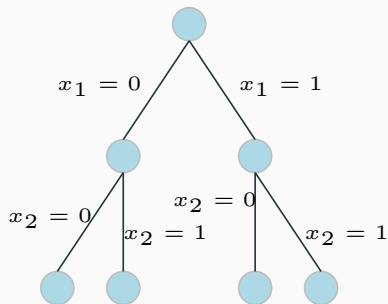
Rong Zhou and Eric A Hansen.

Beam-stack search: Integrating backtracking with beam search.

In *ICAPS*, pages 90–98, 2005.

Trailing vs Copying

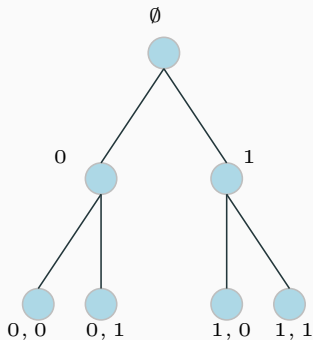
Trailing



(+) more nodes

(-) less tree searches available

Copying



(-) less nodes

(+) more tree search available