Anytime tree search for combinatorial optimization

Thesis defense

presented by: Luc Libralesso
supervised by: Louis Esperet, Thibault Honegger, Vincent Jost

July, 24, 2020

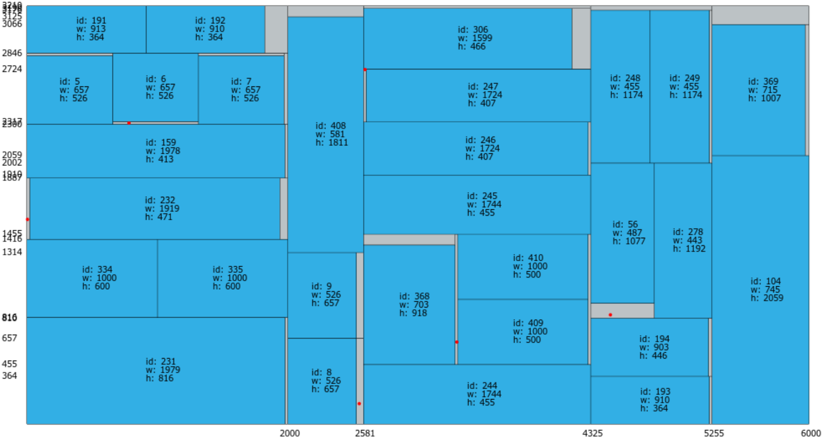
G-SCOP, Grenoble, France
email: luc.libralesso@grenoble-inp.fr

the Traveling Salesman Problem

the Traveling Salesman Problem

Another example (glass window factory)

Given some items, minimize the wasted area (bin packing variant)



These problems are difficult (and we need to solve them)

They are $\mathcal{NP}$-Hard
huge number of solutions:

- if 100 cities: 10^{150} feasible solutions.

These problems are difficult (and we need to solve them)

They are $\mathcal{NP}$-Hard
huge number of solutions:

- if 100 cities: 10^{150} feasible solutions.

And we have to make sure all the “situations” are covered to find the best solution

Exact methods

explore all “situations”
(usually with a branch & bound)

Exact methods

explore all "situations"
(usually with a branch & bound)

pros:
always optimal
cons:
can take a long time

Exact methods

explore all “situations”
(usually with a branch & bound)

pros:
always optimal
cons:
can take a long time

examples:
Branch-and-bound
tree search

Exact methods

explore all “situations”
(usually with a branch & bound)

pros:
always optimal
cons:
can take a long time

examples:
Branch-and-bound
tree search

(meta-) heuristics

explore a promising subset
of solutions

Exact methods

explore all "situations"
(usually with a branch & bound)

pros:
always optimal
cons:
can take a long time

examples:
Branch-and-bound
tree search

(meta-) heuristics

explore a promising subset of solutions

pros:
find quickly a good solution
cons:
not always optimal

Exact methods

explore all “situations”
(usually with a branch & bound)

pros:
always optimal
cons:
can take a long time

examples:
Branch-and-bound
tree search

(meta-) heuristics

explore a promising subset
of solutions

pros:
find quickly a good solution
cons:
not always optimal

examples:
tabu search, evolutionary algorithms
ant colony optimization

from heuristic search / AI planning communities from heuristic search / AI planning communities

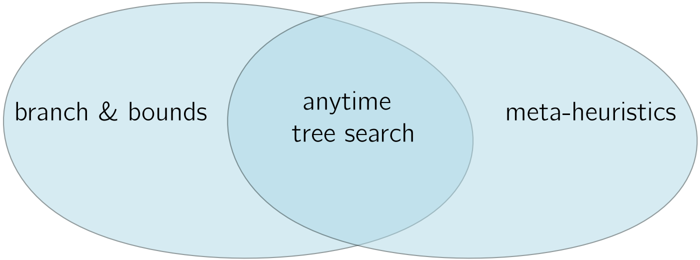
- explore a tree (as branch & bounds)

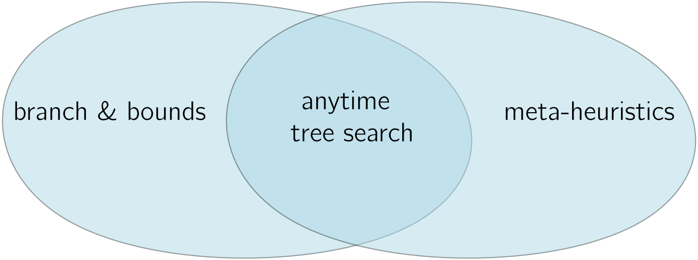
from **heuristic search / AI planning** communities

- explore a tree (as branch & bounds)
- start by the most promising regions (as meta-heuristics)

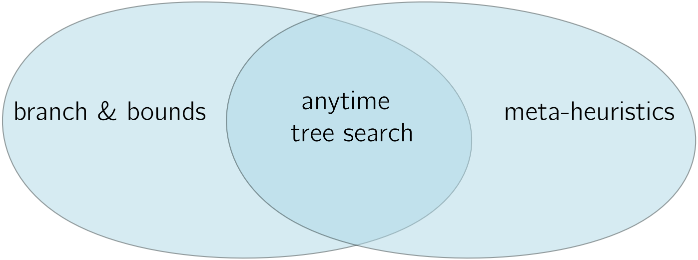
from **heuristic search / AI planning** communities

- explore a tree (as branch & bounds)
- start by the most promising regions (as meta-heuristics)



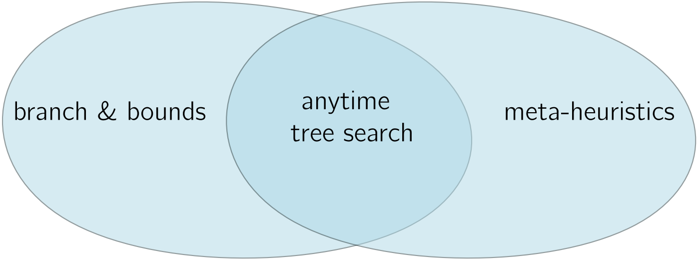


Why could it be interesting?



Why could it be interesting?

- Combine **search-space reductions** from branch & bounds
- and **guidance** strategies from meta-heuristics



Why could it be interesting?

- Combine **search-space reductions** from branch & bounds
- and **guidance** strategies from meta-heuristics

Not present in Operations Research

We study anytime tree search algorithms for classical OR problems

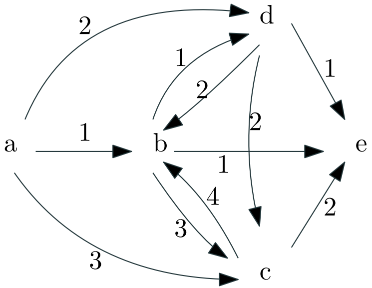
Anytime tree search algorithms

About the implementation

The sequential ordering problem

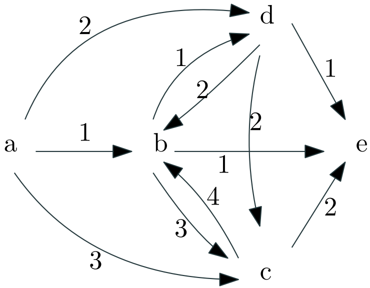
EURO/ROADEF challenge 2018

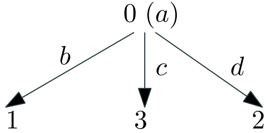
Anytime tree search algorithms

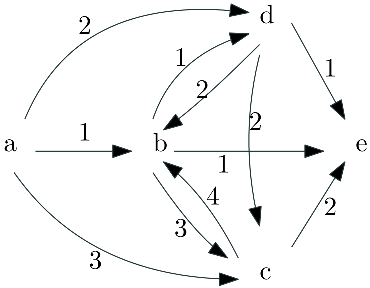


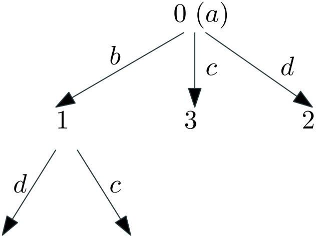
$0\ (a)$

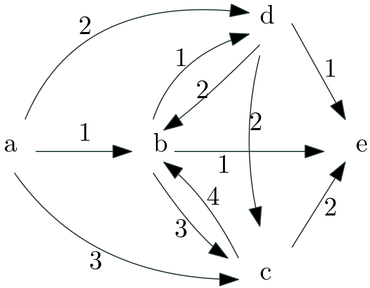
Example of a branch & bound

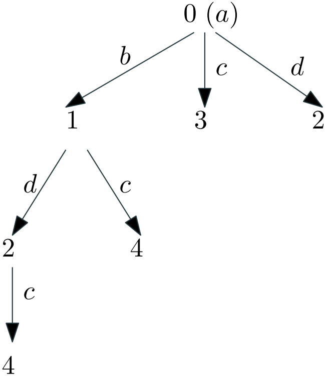


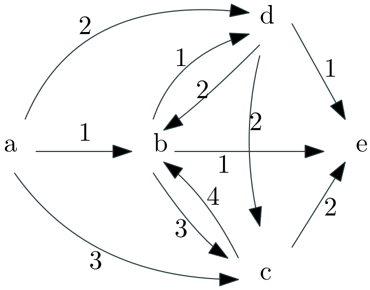


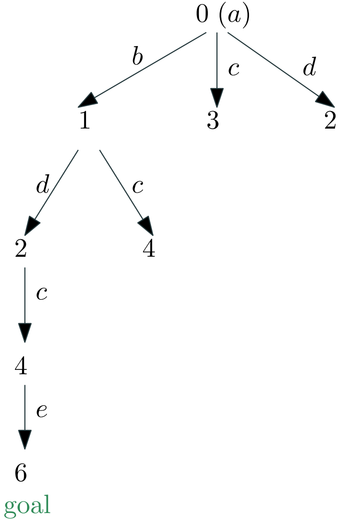

2
d
1
2
1
2
a
1
b
e
1
4
2
3
3
c

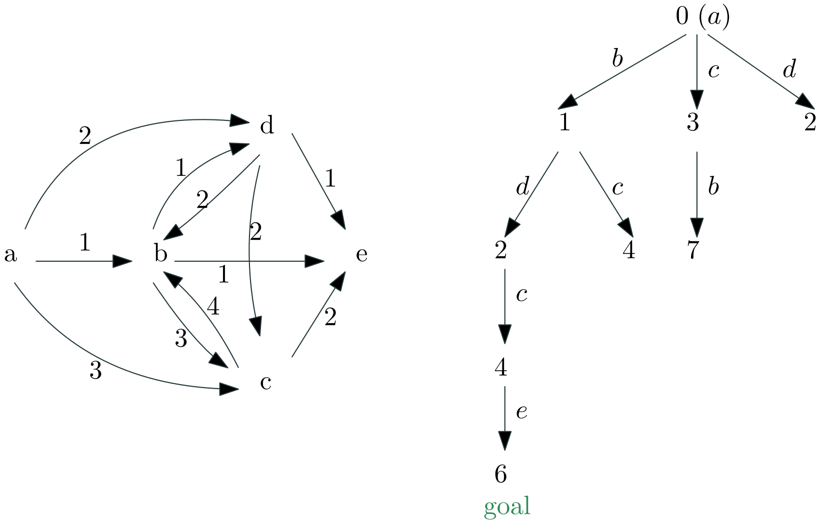

0 (a)
b
c
d
1
3
2
d
c


a
b
c
d
e
1
2
3
1
2
2
1
1
4
3
2

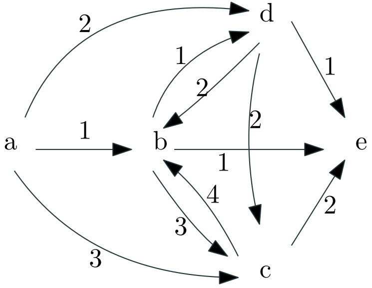

0 (a)
b
c
d
1
3
2
d
c
2
4
c
4

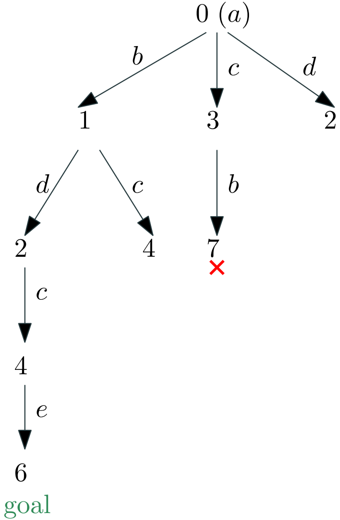

a
b
c
d
e
1
2
3
1
2
2
1
1
4
3
2

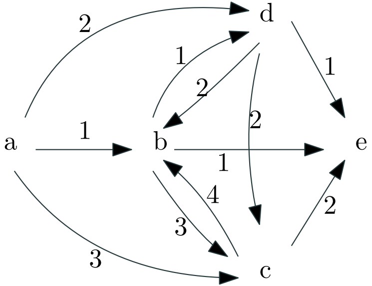

0 (a)
b
c
d
1
3
2
d
c
2
4
c
4
e
6
goal

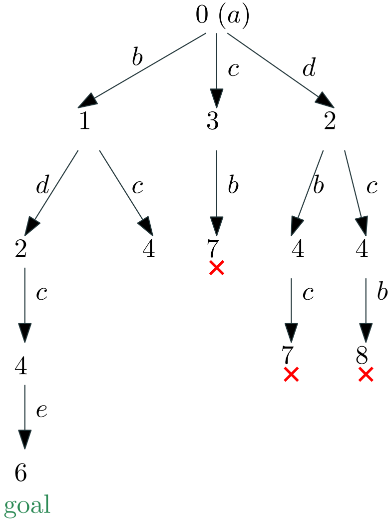

2
d
1
1
2
2
a
1
b
e
1
4
2
3
3
c
0 (a)
b
c
d
1
3
2
d
c
b
2
4
7
c
4
e
6
goal

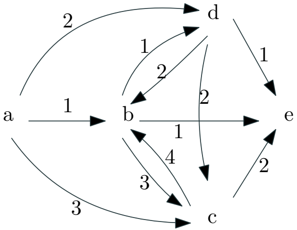
Example of a branch & bound

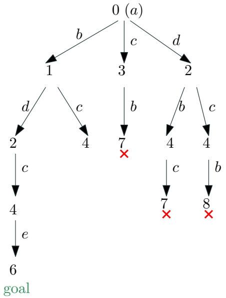





d
a
b
e
c
2
1
2
1
1
1
2
2
3
4
3
2

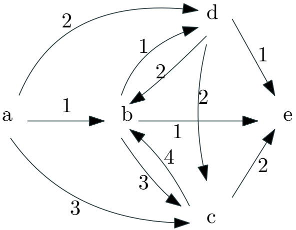

0 (a)
b
c
d
1
3
2
d
c
b
b
c
2
4
7
4
4
c
c
b
4
7
8
e
6
goal

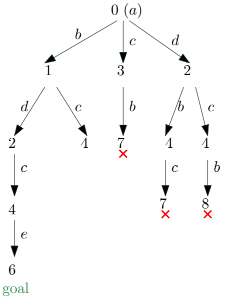




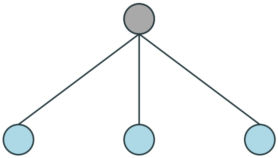
1. define the **search tree**
2. define a **bound** (or **guidance** strategy)

THE ALGORITHM-DESIGN METHODOLOGY

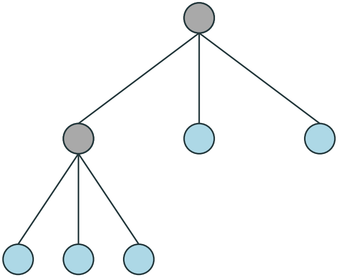




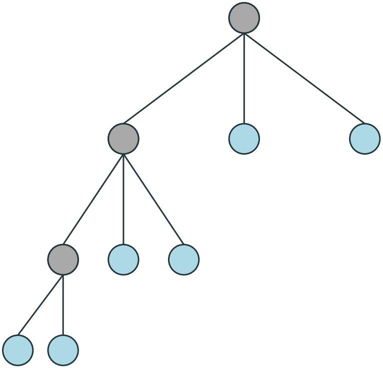
1. define the **search tree**
2. define a **bound** (or **guidance** strategy)
3. **search** the resulting tree (generic part)



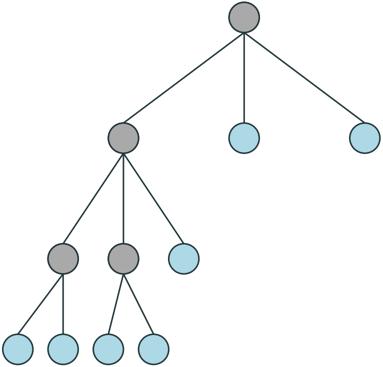
Depth First Search

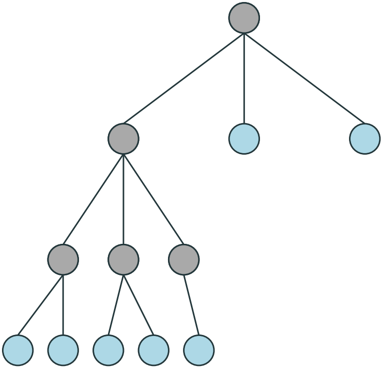


Depth First Search

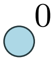


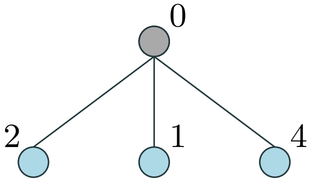
Depth First Search

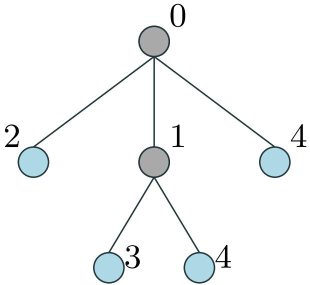




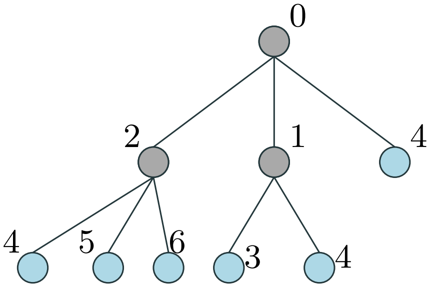
Best (bound) First / A*



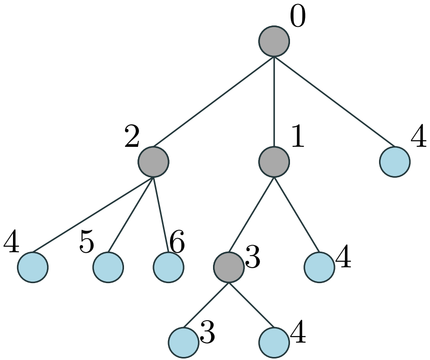

0
2
1
4

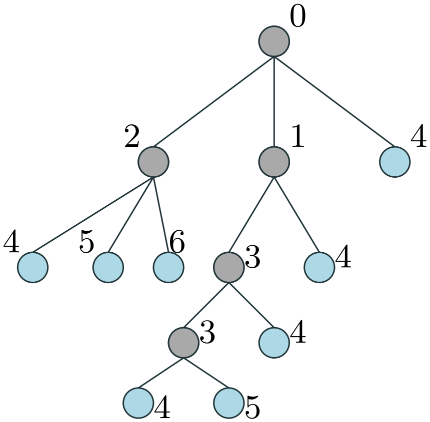

0
2
1
4
3
4

Best (bound) First / A*



Best (bound) First / A*




0
2
1
4
4
5
6
3
4
3
4
4
5

Advantages and Drawbacks

	Depth First Search	A*/Best First
Pros		
Cons		

Advantages and Drawbacks

	Depth First Search	A*/Best First
Pros	• Anytime • Memory Bounded	
Cons		

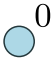
Advantages and Drawbacks

	Depth First Search	A*/Best First
Pros	• Anytime • Memory Bounded	
Cons	• suffers from early bad decisions	

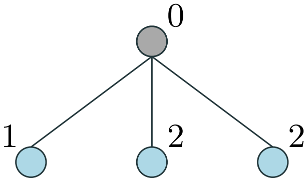
Advantages and Drawbacks

	Depth First Search	A*/Best First
Pros	• Anytime • Memory Bounded	• opens less nodes to close the instance
Cons	• suffers from early bad decisions	

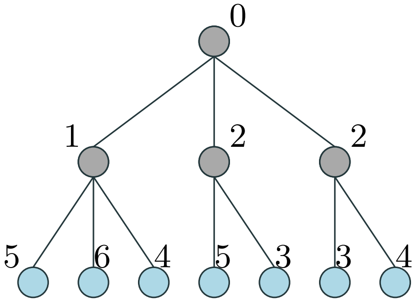
	Depth First Search	A*/Best First
Pros	• Anytime • Memory Bounded	• opens less nodes to close the instance
Cons	• suffers from early bad decisions	• not anytime • Can use too much memory


0

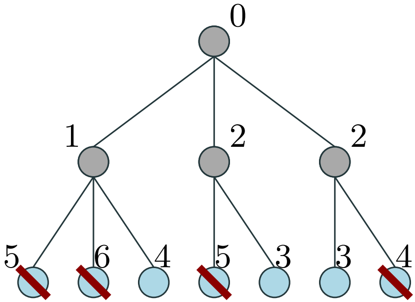
Beam Search (D=3)



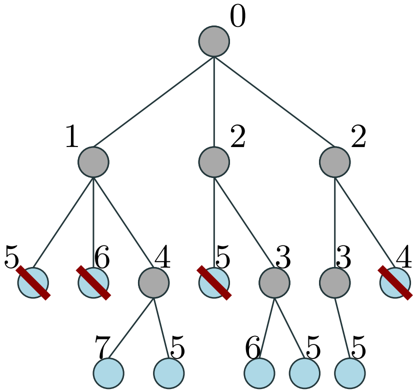
Beam Search (D=3)



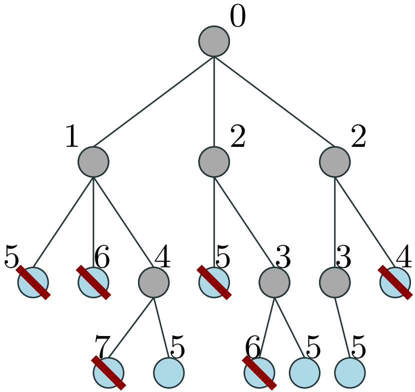
Beam Search (D=3)



Beam Search (D=3)



Beam Search (D=3)



- starts a beam search of size $D = 1$ (greedy)
- then a beam search of size $D = 2$
- then 4, 8, *etc.*

- starts a beam search of size $D = 1$ (greedy)
- then a beam search of size $D = 2$
- then 4, 8, *etc.*

A few properties:

- a complete/exact algorithm when the beam is wide enough

Iterative Beam Search

- starts a beam search of size $D = 1$ (greedy)
- then a beam search of size $D = 2$
- then 4, 8, *etc.*

A few properties:

- a **complete/exact** algorithm when the beam is wide enough
- the algorithm may **open a node multiple times**...

Iterative Beam Search

- starts a beam search of size $D = 1$ (greedy)
- then a beam search of size $D = 2$
- then 4, 8, *etc.*

A few properties:

- a **complete/exact** algorithm when the beam is wide enough
- the algorithm may **open a node multiple times**...
- **but not that much** given some conditions (theorem)
- in average a node is reopened only once

How bad re-openings are?

In our algorithms, we open about a million nodes per second, thus:

How bad re-openings are?

In our algorithms, we open about a million nodes per second, thus:

- re-opening a node is **almost free**

How bad re-openings are?

In our algorithms, we open about a million nodes per second, thus:

- re-opening a node is **almost free**
- data-structures storing nodes usually cost an additional time

In our algorithms, we open about a million nodes per second, thus:

- re-opening a node is **almost free**
- data-structures storing nodes usually cost an additional time
- storing unexplored nodes **saturates** the memory (fast!)

In our algorithms, we open about a million nodes per second, thus:

- re-opening a node is **almost free**
- data-structures storing nodes usually cost an additional time
- storing unexplored nodes **saturates** the memory (fast!)

Thus, we believe it is an efficient strategy

About the implementation

Collaboration with ***Abdel-Malik Bouhassoun***

A large number of tree search algorithms

DFS, A*, Beam Search and many others...

- registering search statistics measuring

- registering search statistics measuring
- dynamic-programming dominance pruning

A large number of possible modifications for each strategy

- registering search statistics measuring
- dynamic-programming dominance pruning
- online learning (ACO-style)

- registering search statistics measuring
- dynamic-programming dominance pruning
- online learning (ACO-style)
- probing strategy
- *etc.*

For a total of...

- 15 search algorithms
- 4 possible variations

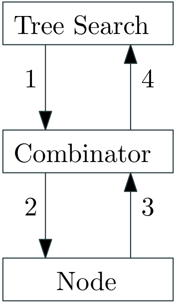
For a total of...

- 15 search algorithms
- 4 possible variations
- thus $15 \times 2^4 = 240$ possible combinations!

- 15 search algorithms
- 4 possible variations
- thus $15 \times 2^4 = 240$ possible combinations!

we need a clever way to implement all of these variants tree search algorithm

Problem specific tree


Tree Search
1
4
Combinator
2
3
Node

Introducing...

the CATS framework:
(Combinator-based Anytime Tree Search)

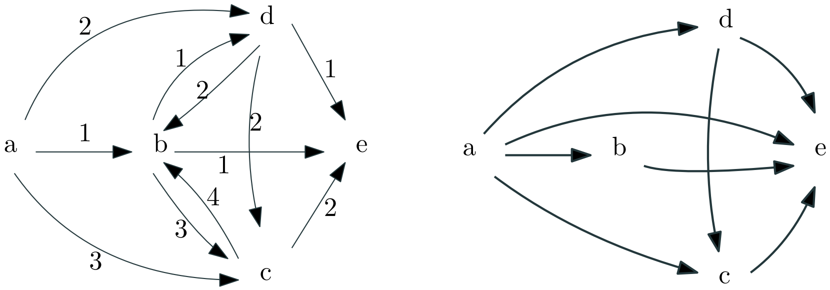
the CATS framework:
(Combinator-based Anytime Tree Search)

- implemented in C++ (efficient)
- 15+ tree search algorithms
- 5 combinators
- GNU/GPL license

The sequential ordering problem

Collaboration with ***Abdel-Malik Bouhassoun*** *and* ***Hadrien Cambazard***

Asymmetric Traveling Salesman Problem with precedence constraints



The benchmark: SOPLIB

- Standard benchmark, proposed in 2006 (“large” instances)

- Standard benchmark, proposed in 2006 (“large” instances)
- Some instances are almost precedence free
- Some are heavily constrained
- “in the middle” instances remain open (7 instances)

Many methods implemented during the 30 last years to solve SOP

Exact methods:

- Branch and cuts
- Decision diagrams + CP
- Branch & Bounds with advanced bounds/prunings

Literature

Many methods implemented during the 30 last years to solve SOP

Exact methods:

- Branch and cuts
- Decision diagrams + CP
- Branch & Bounds with advanced bounds/prunings

Meta-heuristics:

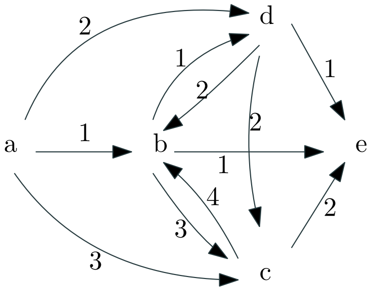
- Local search (3-opt)
- Ant Colony Optimization (using a 3-opt move)
- others (GA, ABC, parallel roll-out, LKH …)

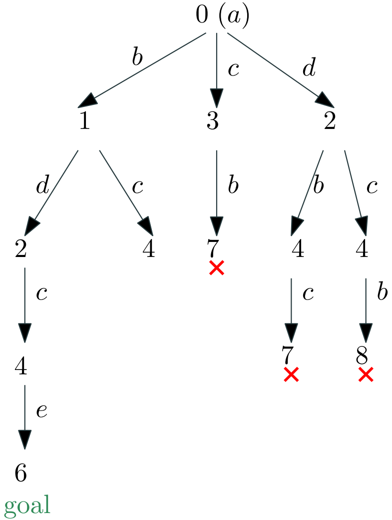
Many methods implemented during the 30 last years to solve SOP

Exact methods:
- Branch and cuts
- Decision diagrams + CP
- Branch & Bounds with advanced bounds/prunings

Meta-heuristics:
- Local search (3-opt)
- Ant Colony Optimization (using a 3-opt move)
- others (GA, ABC, parallel roll-out, LKH ...)

- Exact methods tend to build stronger bounds
- meta-heuristics strongly rely on 3-opt (local search)


a
b
c
d
e
1
2
3
1
2
2
1
1
3
4
2


0 (a)
b
c
d
1
3
2
d
c
b
b
c
2
4
7
4
4
c
c
b
4
7
8
e
6
goal

Dynamic Programming inspired prunings

Example, two equivalent partial solutions:

1. **a,b,c,d** cost 10
2. **a,c,b,d** cost 12

Example, two equivalent partial solutions:

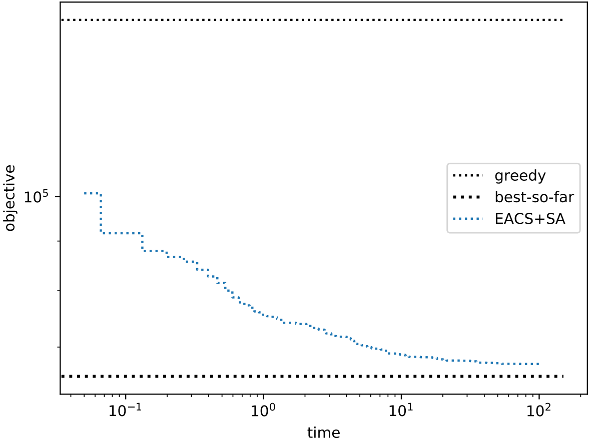
1. **a,b,c,d** cost 10
2. **a,c,b,d** cost 12

Discard (2) as it is “dominated” by (1).

Results - Performance profiles on R.700.1000.15

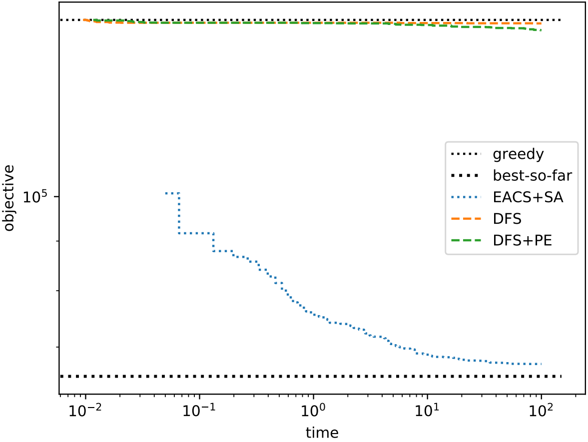
state-of-the-art:

- Enhanced Ant Colony System and Simulated Annealing (EACS+SA)
- best-so-far LKH3 with 100.000 seconds run ($\approx$ 27h)



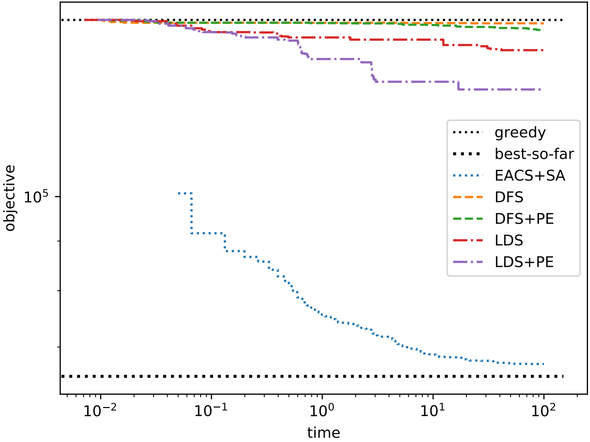
state-of-the-art:

- Enhanced Ant Colony System and Simulated Annealing (EACS+SA)
- best-so-far LKH3 with 100.000 seconds run ($\approx$ 27h)



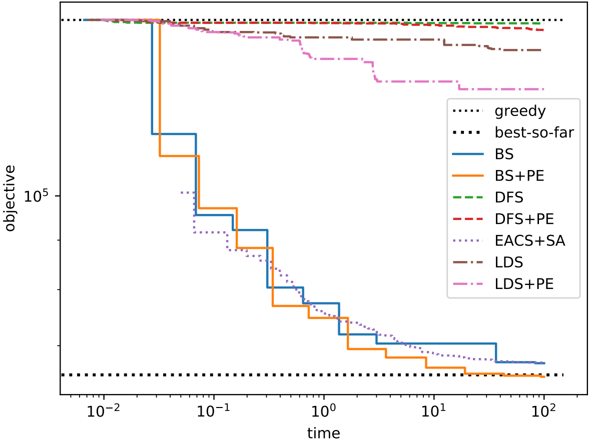
state-of-the-art:

- Enhanced Ant Colony System and Simulated Annealing (EACS+SA)
- best-so-far LKH3 with 100.000 seconds run (≈ 27h)



state-of-the-art:

- Enhanced Ant Colony System and Simulated Annealing (EACS+SA)
- best-so-far LKH3 with 100.000 seconds run (≈ 27h)



6 over 7 new-best-so-far solutions
(the other one is probably optimal)

Instance	best known	BS+PE (600s)
R.500.100.15	5.284	**5.261**
R.500.1000.15	49.504	**49.366**
R.600.100.15	5.472	**5.469**
R.600.1000.15	55.213	**54.994**
R.700.100.15	7.021	**7.020**
R.700.1000.15	65.305	**64.777**

Results - Overview

How this simple tree search behaves on the SOPLIB:

1% precedence constraints: large search space and poor guidance

How this simple tree search behaves on the SOPLIB:

1% precedence constraints: large search space and poor guidance

15% precedence constraints: 5 children in average

Results - Overview

How this simple tree search behaves on the SOPLIB:

1% precedence constraints: large search space and poor guidance

15% precedence constraints: 5 children in average

30% ,60% precedence constraints: proves optimality in a few milliseconds.

How this simple tree search behaves on the SOPLIB:

1% precedence constraints: large search space and poor guidance

15% precedence constraints: 5 children in average

30% ,60% precedence constraints: proves optimality in a few milliseconds.

The SOPLIB mainly contains heavily constrained instances:

- hard for MIPs and local searches
- but (relatively) easy for constructive algorithms
- **thus the need to consider anytime tree searches**

- The search-strategy choice is crucial

- The search-strategy choice is crucial
- (cheap) search space reductions are useful

EURO/ROADEF challenge 2018

Collaboration with ***Florian Fontan***

EURO/ROADEF Challenge

Presented by the French and European *Operations Research* societies

International competition

A challenge every two years:

- 2012: Google
- 2014: SNCF
- 2016: Air Liquide

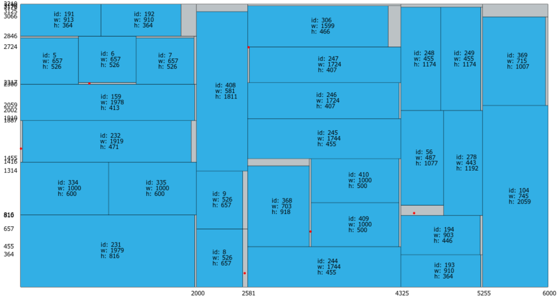
Presented by the French and European *Operations Research* societies

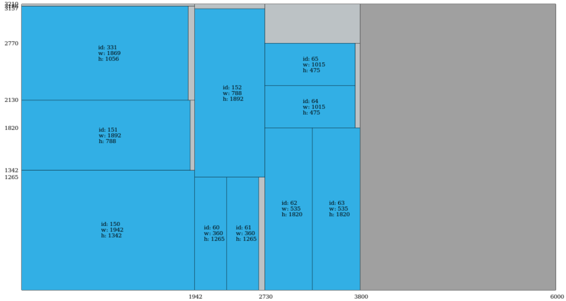
International competition

A challenge every two years:

- 2012: Google
- 2014: SNCF
- 2016: Air Liquide
- **2018: Saint Gobain**

ONE OF OUR SOLUTIONS





- Cutting & packing problem
- variant of the bin-packing

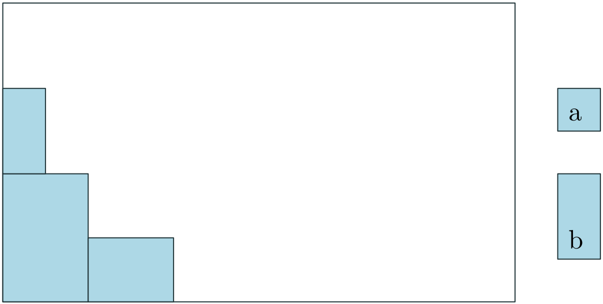
The problem

- Cutting & packing problem
- variant of the bin-packing
- with various constraints, some examples:
    - guillotine cuts
    - Defects
    - precedence constraints
- Large-size instances (up to 700 items)

Place items in the corner rule

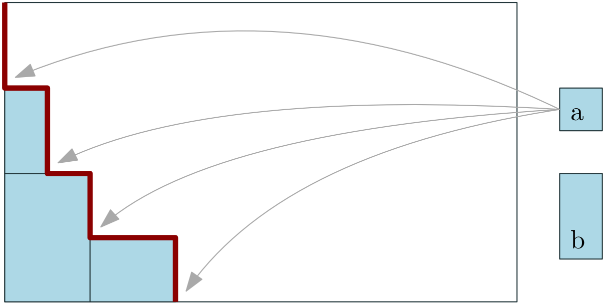
Called a "staircase" representation

Place a remaining item at a possible position



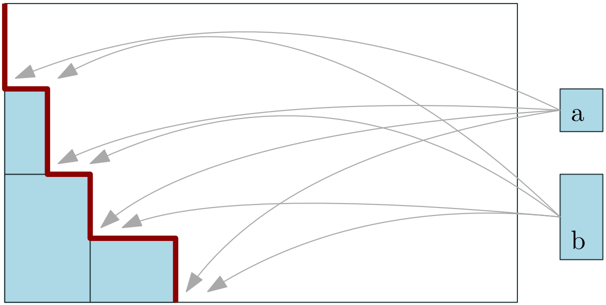
Called a “staircase” representation
Place a remaining item at a possible position

Called a “staircase” representation
Place a remaining item at a possible position



Place items in the corner rule

Called a "staircase" representation
Place a remaining item at a possible position

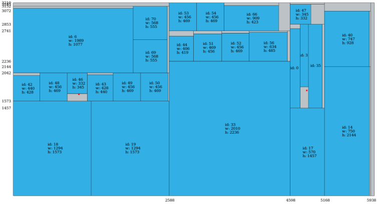


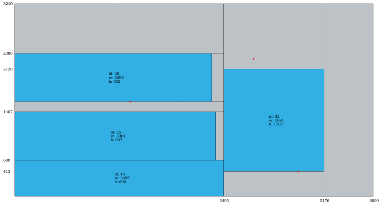
In this case, 8 children for this node

branch & bound ideas:

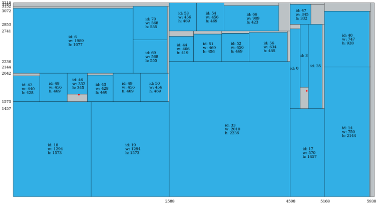
- (pseudo-)dominance rules
- symmetry-breaking rules

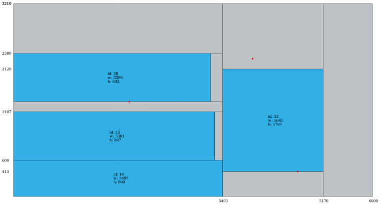
Which one should I keep?



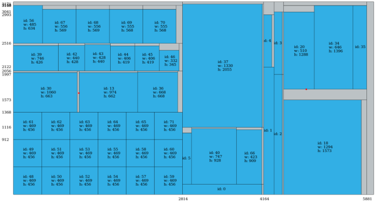


Which one should I keep?

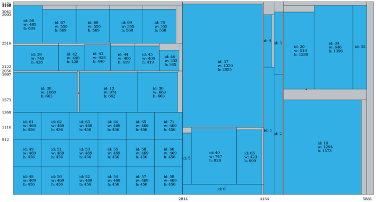


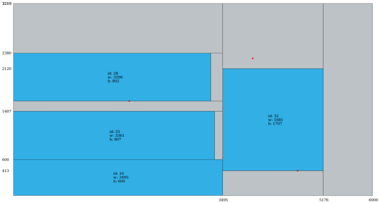


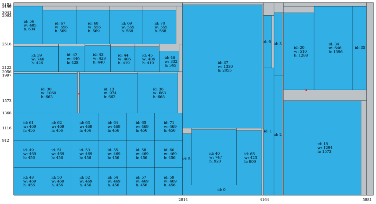
The less waste, the more attractive the partial solution

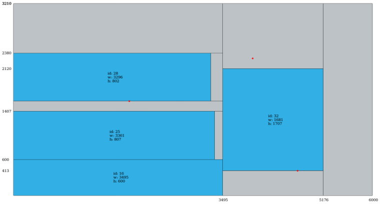

3041
2993
2516
2122
2056
1997
1573
1368
1116
912
id: 56 w: 485 h: 634
id: 67 w: 556 h: 569
id: 68 w: 556 h: 569
id: 69 w: 555 h: 568
id: 70 w: 555 h: 568
id: 39 w: 746 h: 426
id: 42 w: 440 h: 428
id: 43 w: 428 h: 440
id: 44 w: 406 h: 419
id: 45 w: 406 h: 419
id: 46 w: 332 h: 345
id: 30 w: 1060 h: 663
id: 13 w: 974 h: 662
id: 36 w: 668 h: 668
id: 61 w: 469 h: 456
id: 62 w: 469 h: 456
id: 63 w: 469 h: 456
id: 64 w: 469 h: 456
id: 65 w: 469 h: 456
id: 71 w: 469 h: 456
id: 49 w: 469 h: 456
id: 51 w: 469 h: 456
id: 53 w: 469 h: 456
id: 55 w: 469 h: 456
id: 58 w: 469 h: 456
id: 60 w: 469 h: 456
id: 48 w: 469 h: 456
id: 50 w: 469 h: 456
id: 52 w: 469 h: 456
id: 54 w: 469 h: 456
id: 57 w: 469 h: 456
id: 59 w: 469 h: 456
id: 37 w: 1330 h: 2055
id: 5
id: 40 w: 747 h: 928
id: 66 w: 423 h: 909
id: 0
id: 4
id: 3
id: 1
id: 2
id: 20 w: 510 h: 1288
id: 34 w: 646 h: 1396
id: 35
id: 18 w: 1294 h: 1573
2814
4164
5881

What happens when we use bounds as guides









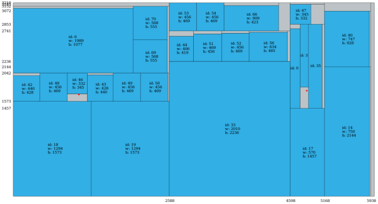
Problem with waste:

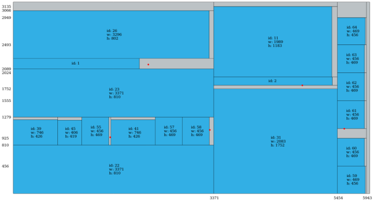
- Small items at the beginning and big items at the end

How to correct this bias?

$$\frac{\text{waste percentage}}{\text{mean item area}}$$

$$\frac{\text{waste percentage}}{\text{mean item area}}$$





“Heuristic” guides

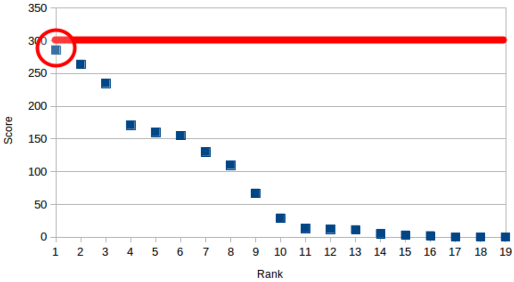
Much more efficient than the bound guide

Much more efficient than the bound guide
cannot be used to prune nodes

Much more efficient than the bound guide
cannot be used to prune nodes
Thus the need to separate the two concepts

Search Strategy

- Variant of Iterative Beam Search
- replace the truncated BrFS by a truncated A*
- Called **Iterative MBA***


350
300
250
200
150
100
50
0
Score
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Rank

Conclusions on the challenge

- anytime tree search algorithm (IMBA*)
- combines exact-methods parts (dominances, *etc.*)
- new “heuristic” guidance strategy

These 3 components are required to provide a competitive algorithm.

- Study over many well-studied variants in the literature
- large number of benchmarks (10+)

Generalizing the algorithm

- Study over many well-studied variants in the literature
- large number of benchmarks (10+)
- We obtain state-of-the-art results on many variants
- and very competitive on other variants

Generalizing the algorithm

- Study over many well-studied variants in the literature
- large number of benchmarks (10+)
- We obtain state-of-the-art results on many variants
- and very competitive on other variants
- open-source software (PackingSolver)

BONUS

TREE SEARCH FOR OTHER PROBLEMS

*Collaboration with **Aurélien Secardin** and **Pablo Andres Focke***

LCS is a famous and well-studied optimization problem.

LCS is a famous and well-studied optimization problem.

We present an **iterative beam search**:

LCS is a famous and well-studied optimization problem.

We present an **iterative beam search**:

- with **Pareto-dominance strategies**

LCS is a famous and well-studied optimization problem.

We present an **iterative beam search**:

- with **Pareto-dominance strategies**
- probability-based **heuristic guidance strategy**

LCS is a famous and well-studied optimization problem.

We present an **iterative beam search**:

- with **Pareto-dominance strategies**
- probability-based **heuristic guidance strategy**
- state-of-the-art (new best-known solutions on many instances)

Well studied problem ($F_m/permu/C_{max}$, and $F_m/permu/\sum C_j$)

Well studied problem ($F_m/permu/C_{max}$, and $F_m/permu/\sum C_j$)

We present an **iterative beam search** (again):

Well studied problem ($F_m/permu/C_{max}$, and $F_m/permu/\sum C_j$)

We present an **iterative beam search** (again):

- with a **search tree from a recent branch & bound** (Gmys et al.)

Well studied problem ($F_m/permu/C_{max}$, and $F_m/permu/\sum C_j$)

We present an **iterative beam search** (again):

- with a **search tree from a recent branch & bound** (Gmys et al.)
- **guidance strategy similar the LR greedy heuristic**

Well studied problem ($F_m/permu/C_{max}$, and $F_m/permu/\sum C_j$)

We present an **iterative beam search** (again):

- with a **search tree from a recent branch & bound** (Gmys et al.)
- **guidance strategy similar the LR greedy heuristic**
- state-of-the-art results on large VRF instances (makespan)
- state-of-the-art results on large Taillard instances (flowtime)

Wrapping-up

Benefits from a large variety of contributions:

- **exact methods** (search space reductions)
- **anytime tree search** (AI/planning)
- **meta-heuristics** (guide functions)

Contributions (anytime tree search algorithms)

Simple and efficient anytime tree search algorithms applied on various problems:

- sequential ordering problem
- EURO/ROADEF challenge
- generalization to Cutting & packing
- longest common subsequence
- permutation flowshop

Perspectives

- Apply anytime tree search on other problems

Perspectives

- Apply anytime tree search on other problems
- Learn guides automatically (ACO, Reinforcement Learning)

Perspectives

- Apply anytime tree search on other problems
- Learn guides automatically (ACO, Reinforcement Learning)
- More search-space reductions:
  - decision diagrams, ng-routes, *etc.*
  - MIP, CP

Anytime tree search for combinatorial optimization

Thesis defense

presented by: Luc Libralesso

supervised by: Louis Esperet, Thibault Honegger, Vincent Jost

July, 24, 2020

G-SCOP, Grenoble, France

email: luc.libralesso@grenoble-inp.fr

Jan Gmys, Mohand Mezmaz, Nouredine Melab, and Daniel Tuyttens. A computationally efficient branch-and-bound algorithm for the permutation flow-shop scheduling problem. 284(3):814–833. ISSN 0377-2217. doi: 10.1016/j.ejor.2020.01.039. URL *http://www.sciencedirect.com/science/article/pii/S037722172030076X*.