Tree Search for the Recursive Circle Packing Problem

Rui Rei – rui.rei@dcc.fc.up.pt
João P. Pedroso – jpp@fc.up.pt

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

INESC TEC

5th Porto Meeting on Mathematics for Industry
April 10-11, 2014

Outline

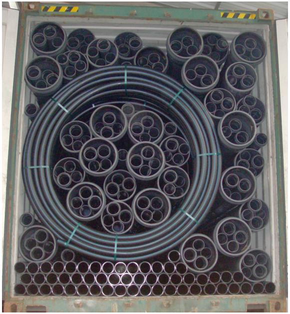
Next up. . .

Origin of the problem: industrial setting

- A company produces hollow tubes of various radii
- Orders are sent to customers in containers
- All tubes have the length of a container
- How should the containers be loaded?

Origin of the problem: industrial setting

- A company produces hollow tubes of various radii
- Orders are sent to customers in containers
- All tubes have the length of a container
- How should the containers be loaded?



Origin of the problem: industrial setting

- Currently: solution is constructed...

Origin of the problem: industrial setting

- Currently: solution is constructed... MANUALLY (what?!? O_o)

Origin of the problem: industrial setting

- Currently: solution is constructed... MANUALLY (what?!? O_o)
- Very tedious and error prone
- Production engineers' time is expensive
- We can surely do better

Problem description (simplified)

Given:

- a set of tubes, where each tube is characterized by
  - external radius
  - internal radius
  - value
- a container with given dimensions (width and height)

Problem description (simplified)

Given:

- a set of tubes, where each tube is characterized by
  - external radius
  - internal radius
  - value
- a container with given dimensions (width and height)

Maximize value of packed tubes, such that:

- tubes may be inside other tubes, but they cannot overlap
- all packed tubes must be completely inside the container

Problem description (simplified)

Given:

- a set of tubes, where each tube is characterized by
  - external radius
  - internal radius
  - value
- a container with given dimensions (width and height)

Maximize value of packed tubes, such that:

- tubes may be inside other tubes, but they cannot overlap
- all packed tubes must be completely inside the container

Solution:

- a list of packed tubes and their positions

A mathematical model: parameters and variables

Parameters for describing an instance

- width W and height H of the rectangular container;
- set of tubes $\mathcal{N}$
- for each tube $i \in \mathcal{N}$:
  - external radius r_i^{ext}
  - internal radius r_i^{int}
  - value v_i

Variables

- $(x_i, y_i) \in \mathbb{R}^2$, position of the center of each tube i
- $p_i \in \{0, 1\}, p_i = 1$ if tube i is placed **directly** in the container
- $q_{ij} \in \{0, 1\}, q_{ij} = 1$ if tube i is placed **directly** inside tube j

A mathematical model: objective function

$$\text{maximize } V = \sum_{i \in \mathcal{N}} v_i \times \left(p_i + \sum_{j \in \mathcal{N}} q_{ij} \right)$$

A mathematical model: constraints

Tubes cannot be in multiple places

$$p_i + \sum_j q_{ij} \leq 1, \quad \forall i \in \mathcal{N}$$

Tubes must stay inside the container

$$r_i^{\text{ext}} \leq x_i \leq W - r_i^{\text{ext}}, \quad \forall i \in \mathcal{N}$$
$$r_i^{\text{ext}} \leq y_i \leq H - r_i^{\text{ext}}, \quad \forall i \in \mathcal{N}$$

A mathematical model: more constraints

Tubes i and j inside the container cannot overlap

$$\|xy_i - xy_j\|_2 \geq r_i^{\text{ext}} + r_j^{\text{ext}} - M \times (2 - p_i - p_j), \quad \forall i, j \in \mathcal{N}$$

Same goes for tubes i and j inside the same larger tube k

$$\|xy_i - xy_j\|_2 \geq r_i^{\text{ext}} + r_j^{\text{ext}} - M \times (2 - q_{ik} - q_{jk}), \quad \forall i, j, k \in \mathcal{N}$$

Tube i inside j must stay within j

$$\|xy_i - xy_j\|_2 \leq r_j^{\text{int}} - r_i^{\text{ext}} + M \times (1 - q_{ij}), \quad \forall i, j \in \mathcal{N}$$

(M disables constraints when any variable inside parenthesis is zero)

A mathematical model: limitations

- Previous model is exact
- Non-linear
- Very hard to solve
  - Quadratic number of variables
  - Cubic number of constraints
- Let's take a look at some practical solutions

Next up...

A possible greedy construction

```
def greedy_solution(C, T):
    O = {C}
    while O != {}:
        o = argmin(O, key=free_space)
        repeat:
            if o has no positions:
                O.remove(o)
                break

            t = argmax(T, key=(erad, value, irad))
            P = o.positions_for(t)
            p = argmin(P, key=(y_coord, x_coord))

            o.insert(t, p)
            O.add(t)
            T.remove(t)
            o = t

    return C
```

Generating positions

What is the set of positions for a tube?

In reality, this is an usually an infinite set.

Generating positions

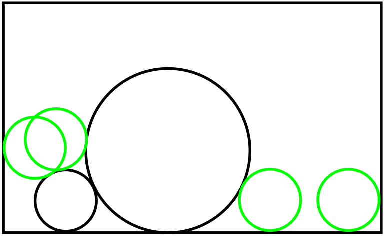
What is the set of positions for a tube?

In reality, this is an usually an infinite set.

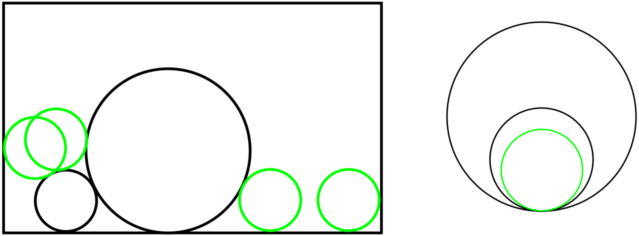
Reduce this to a finite set of possible positions

- the corners of the container
- positions touching another tube and a wall of the container
- positions touching (at least) two other tubes
- the bottom of the outer tube (telescoping; initial tube)

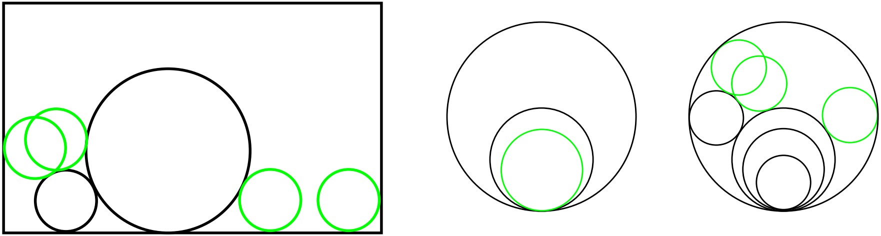
Generating positions



Generating positions



Generating positions



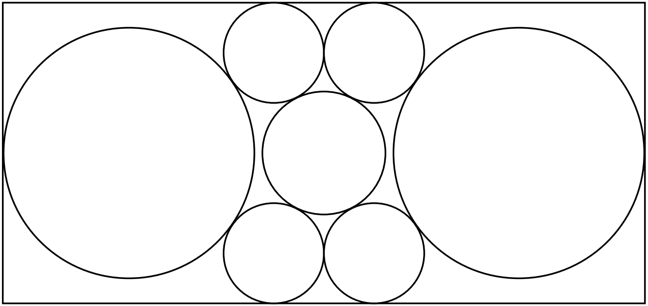
Does this lead to the optimal solution?

Does this lead to the optimal solution?

NO

Does this lead to the optimal solution?

NO



Semi-Greedy construction

- SG = greedy construction + probabilistic choice + repetition

Semi-Greedy construction

- SG = greedy construction + probabilistic choice + repetition
- Probabilistic choice: position of tube

Semi-Greedy construction

- SG = greedy construction + probabilistic choice + repetition
- Probabilistic choice: position of tube
- Optimization by repetition with different RNG seeds

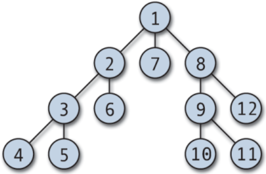
Semi-Greedy construction

- SG = greedy construction + probabilistic choice + repetition
- Probabilistic choice: position of tube
- Optimization by repetition with different RNG seeds
- Could hardly be simpler

Local Search

- Not easy to define a proper (finite) neighborhood
- Moving one tube will likely cause overlaps
- May not be trivial to restore feasibility

Depth-First Tree Search



- Complete enumeration of the search space (if enough time is allowed)
- Avoids repeated solutions
- Very fast
- Low memory requirements

- First decisions are (in most cases) never changed
- Performance **extremely dependent** on a branch ordering heuristic

Monte-Carlo Tree Search

Some facts

- Tree search algorithm mostly employed in game playing
- Asymmetrically constructs a game tree
- Focuses on most promising* branches
- Uses Monte-Carlo simulations to estimate value of nodes
- Each node maintains basic statistics (# of sims and # of wins)
- Requires little/no domain-specific knowledge, but benefits from it
- An iteration consists of. . .

Monte-Carlo Tree Search

Some facts

- Tree search algorithm mostly employed in game playing
- Asymmetrically constructs a game tree
- Focuses on most promising* branches
- Uses Monte-Carlo simulations to estimate value of nodes
- Each node maintains basic statistics (# of sims and # of wins)
- Requires little/no domain-specific knowledge, but benefits from it
- An iteration consists of...

  Selection starting from the root, pick a node to expand

Monte-Carlo Tree Search

Some facts

- Tree search algorithm mostly employed in game playing
- Asymmetrically constructs a game tree
- Focuses on most promising* branches
- Uses Monte-Carlo simulations to estimate value of nodes
- Each node maintains basic statistics (# of sims and # of wins)
- Requires little/no domain-specific knowledge, but benefits from it
- An iteration consists of...

  Selection starting from the root, pick a node to expand

  Expansion create one (or more) children of the selected node

Monte-Carlo Tree Search

Some facts

- Tree search algorithm mostly employed in game playing
- Asymmetrically constructs a game tree
- Focuses on most promising* branches
- Uses Monte-Carlo simulations to estimate value of nodes
- Each node maintains basic statistics (# of sims and # of wins)
- Requires little/no domain-specific knowledge, but benefits from it
- An iteration consists of...

  Selection starting from the root, pick a node to expand
  Expansion create one (or more) children of the selected node
  Simulation make a simulation from each new node

Monte-Carlo Tree Search

Some facts

- Tree search algorithm mostly employed in game playing
- Asymmetrically constructs a game tree
- Focuses on most promising* branches
- Uses Monte-Carlo simulations to estimate value of nodes
- Each node maintains basic statistics (# of sims and # of wins)
- Requires little/no domain-specific knowledge, but benefits from it
- An iteration consists of...

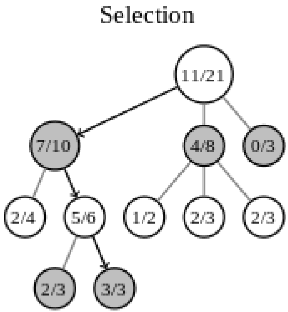
  Selection starting from the root, pick a node to expand

  Expansion create one (or more) children of the selected node

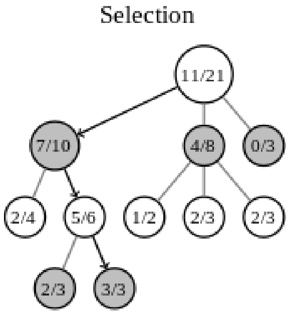
  Simulation make a simulation from each new node

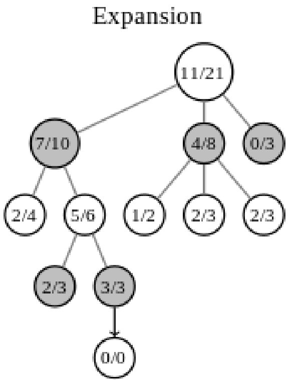
  Backpropagation using the results of the simulations, update the statistics on each node in the path up to the root

Monte-Carlo Tree Search

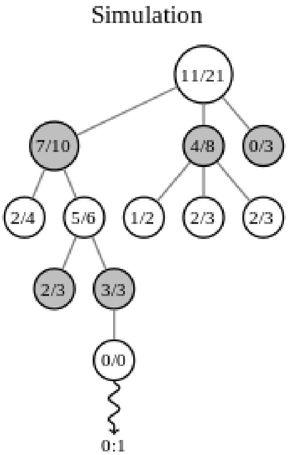


Monte-Carlo Tree Search

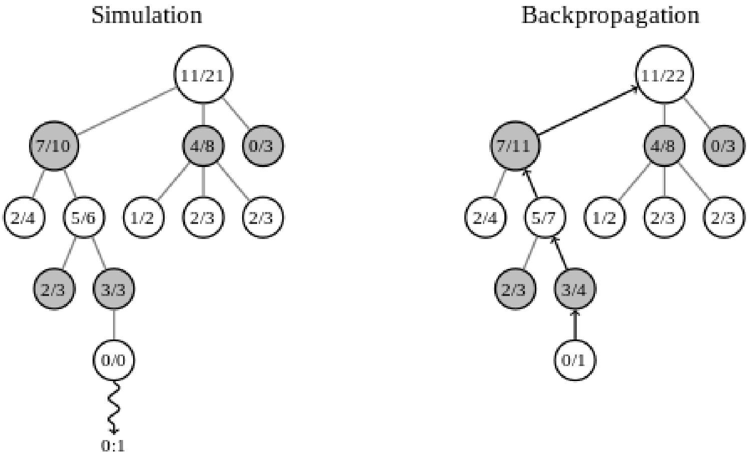




Monte-Carlo Tree Search



Monte-Carlo Tree Search



UCT: Upper Confidence Bound 1 applied to Trees

$$UCT_n = X_n + c.\sqrt{\frac{\ln N_{P_n}}{N_n}}$$

- Formula for selecting the "best" child (selection step)
- Most popular variant of MCTS
- UCT formula consists of two components:

  Exploitation prefers nodes with best known values
  Exploration prefers nodes that have few simulations
- X_n is assumed to be in $[0, 1]$

Adapting UCT for optimization

Normalization

$$X_n = \frac{e^{1-\frac{z_n^*-z^*}{w^*-z^*}} - 1}{e-1}$$

- $X_n \in [0, 1]$ ✓ UCT-approved
- uses both z^* and w^* to assess how good a value is
- avoids scale issues with objective function values

Adapting UCT for optimization

Normalization

$$X_n = \frac{e^{1-\frac{z_n^* - z^*}{w^* - z^*}} - 1}{e - 1}$$

- $X_n \in [0, 1]$ ✓ UCT-approved
- uses both z^* and w^* to assess how good a value is
- avoids scale issues with objective function values

Weighting exploration with $\overline{X_n}$

$$E_n = \overline{X_n}.c.\sqrt{\frac{\ln N_{P_n}}{N_n}}$$

- use mean to help guide the search
- assign less time to branches with worse mean score

Next up. . .

Experiment

- 6 instances
  - 3, 5, and 16 different types of tubes
  - 2 container sizes: large and small (= large/2)
- Competing algorithms

  DFS Depth-First Tree Search
  SG repeated Semi-Greedy construction
  MCTS Monte Carlo Tree Search
- Software implemented in Python 2.X, run in PyPy

Results: large instances

		large03	large05	large16
DFS*		3570033	3720124	18492283
SG**	min	3660028	3810114	22381890
	avg	3660029.3	3822105.7	22548874.4
	max	3660030	3840093	22851844
MCTS**	min	3660029	4050053	24241737
	avg	**3660031.3**	**4098052.3**	**24842685.8**
	max	3660034	4140048	25451624

* result of 1 run of 600s

** results of 10 independent runs of 600s

Results: small instances

		small03	small05	small16
DFS*		900000	1090000	9540056
SG**	min	940000	1090000	9790035
	avg	940000.0	1090000.0	9820032.6
	max	940000	1090000	9840031
MCTS**	min	940000	1120000	10470034
	avg	**956000.1**	**1120000.0**	**10643039.3**
	max	960000	1120000	10700041

* result of 1 run of 600s

** results of 10 independent runs of 600s

Next up...

Conclusion

Contributions

- Definition of the RCPP
- Non-linear formulation: not usable in practice
- Adaptation of MCTS/UCT
  - make X_n independent of problem scale
  - use mean performance to guide exploration
- Interesting results

Conclusion

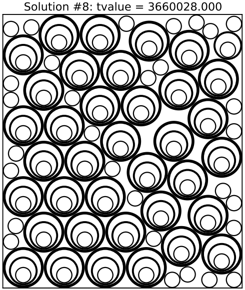
Contributions

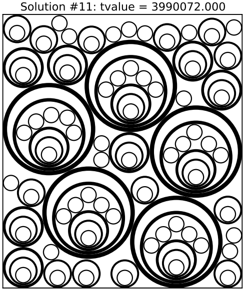
- Definition of the RCPP
- Non-linear formulation: not usable in practice
- Adaptation of MCTS/UCT
  - make X_n independent of problem scale
  - use mean performance to guide exploration
- Interesting results

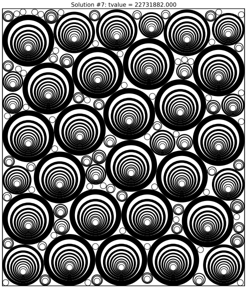
Future work

- Compare with more challenging opponents
- Apply MCTS to other problems, *e.g.*, MIP
- Add some mechanism to discard nodes when the tree grows too large (beam search)

Thank you!


Solution #8: tvalue = 3660028.000


Solution #11: tvalue = 3990072.000


Solution #7: tvalue = 22731882.000