Deep learning and matheuristics: variable selection and decomposition approaches

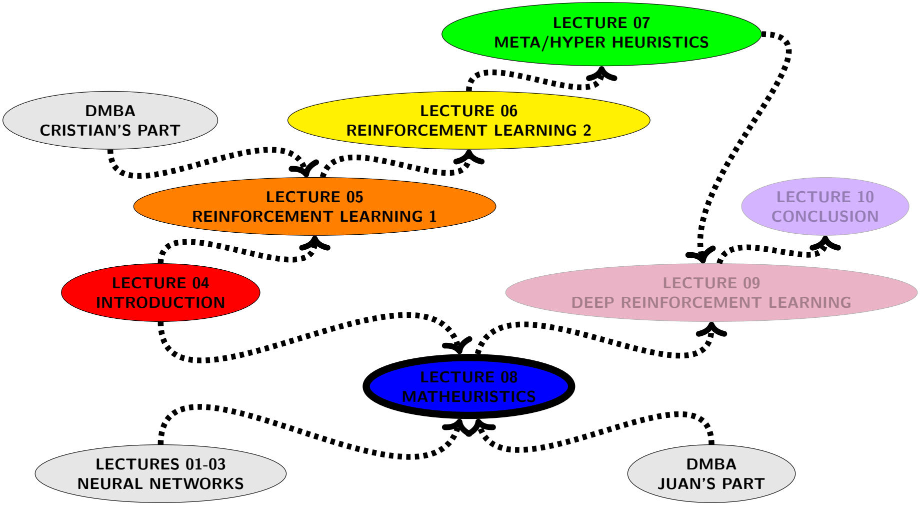
Maxence Delorme

Department of Econometrics and Operations Research
Tilburg University

`m.delorme@tilburguniversity.edu`

OR & ML Lecture 8

Our journey


LECTURE 07
META/HYPER HEURISTICS
DMBA
CRISTIAN'S PART
LECTURE 06
REINFORCEMENT LEARNING 2
LECTURE 05
REINFORCEMENT LEARNING 1
LECTURE 10
CONCLUSION
LECTURE 04
INTRODUCTION
LECTURE 09
DEEP REINFORCEMENT LEARNING
LECTURE 08
MATHEURISTICS
LECTURES 01-03
NEURAL NETWORKS
DMBA
JUAN'S PART

Outline

Plan

Introduction to matheuristics

Important aspects of matheuristics (Boschetti at al.)

- Matheuristics are heuristic algorithms made by the **interoperation** of **metaheuristics** and **mathematical programming (MP)** techniques
- They use features derived from **the mathematical model** of the problem of interest
- Matheuristics have **evolved with MILP solvers** and are not necessarily embedded within metaheuristics anymore
- A **typology** of matheuristics was proposed by Ball who identified 4 types of matheuristics
  - Decomposition approaches
  - Improvement heuristics
  - Mathematical programming to generate approximate solutions
  - Relaxation-based approaches

Introduction to matheuristics

Decomposition approaches (Archetti and Grazia Speranza)

- The problem is **divided** into **smaller** and **simpler** subproblems
- A **specific solution method** is applied to each subproblem
- In a certain class of matheuristics, some or all these subproblems are solved to optimality (or near optimality) with mathematical programming

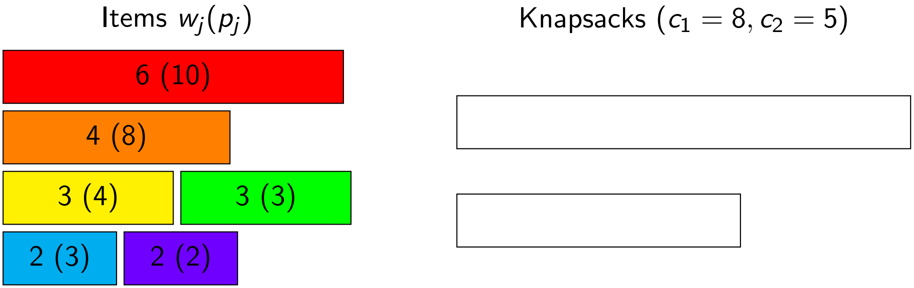
Example

Multiple Knapsack Problem (MKP)

Given a set of m knapsacks with capacity $c_i (i = 1, \ldots, m)$ and a set of n items with profit p_j and weight $w_j (j = 1, \ldots, n)$, the Multiple Knapsack Problem consists in finding a set of items with maximum profit that can be packed within the knapsacks.

Introduction to matheuristics

Non-decomposed MKP

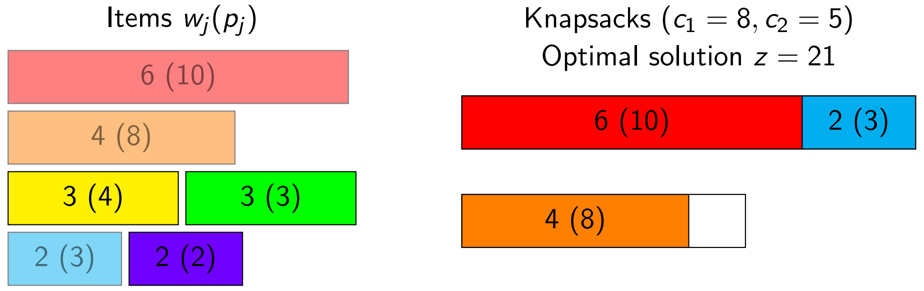


Advantages and drawbacks

- The solution found is optimal
- It is (empirically) much harder to solve the MKP than the classical 0-1 knapsack problem

Introduction to matheuristics

Non-decomposed MKP

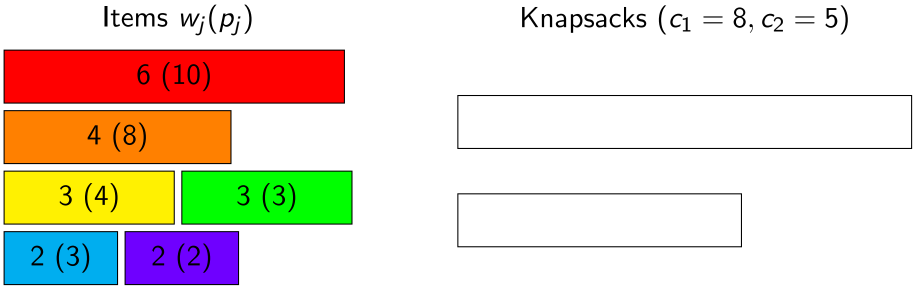


Advantages and drawbacks

- The solution found is optimal
- It is (empirically) much harder to solve the MKP than the classical 0-1 knapsack problem

Introduction to matheuristics

Decomposed MKP

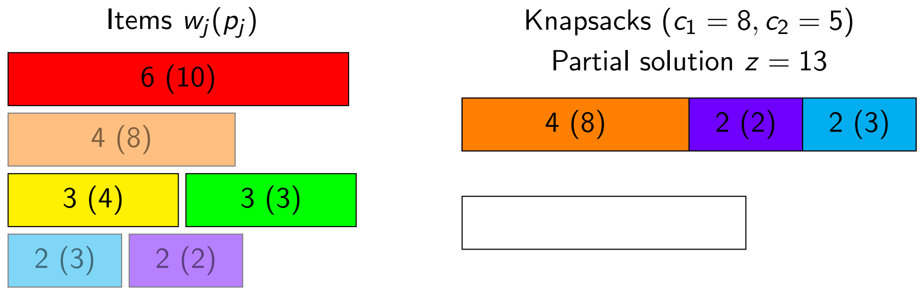


Advantages and drawbacks

- The solution found is not guaranteed to be optimal
- It is (empirically) much easier to solve the 0-1 knapsack problem than the MKP

Introduction to matheuristics

Decomposed MKP

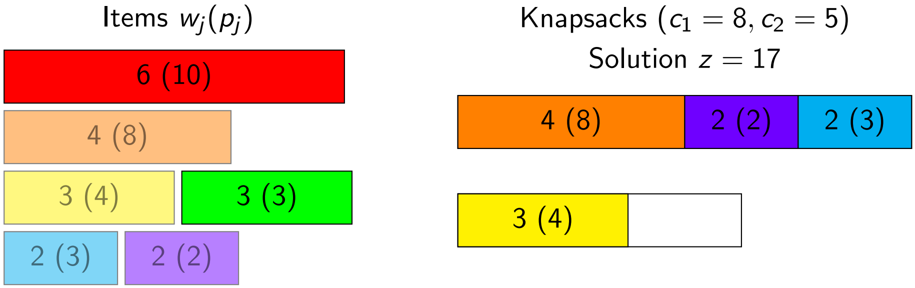


Advantages and drawbacks

- The solution found is not guaranteed to be optimal
- It is (empirically) much easier to solve the 0-1 knapsack problem than the MKP

Introduction to matheuristics

Decomposed MKP



Advantages and drawbacks

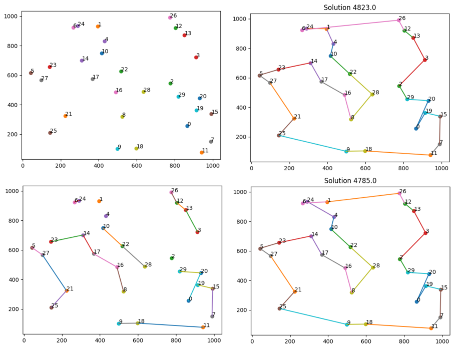
- The solution found is not guaranteed to be optimal
- It is (empirically) much easier to solve the 0-1 knapsack problem than the MKP

Introduction to matheuristics

Improvement heuristics (Archetti and Grazia Speranza)

- Mathematical programming models are used to **improve a solution** found by a different heuristic approach
- They can be applied to whatever heuristic/matheuristic is used to obtain a solution, even itself (recursivity)

TSP example



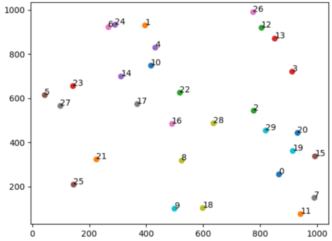
Introduction to matheuristics

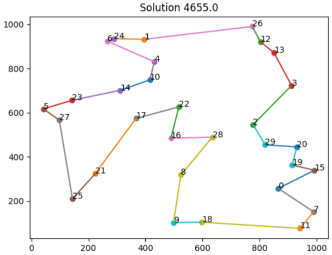
Math. prog. to generate approximate solutions (Ball)

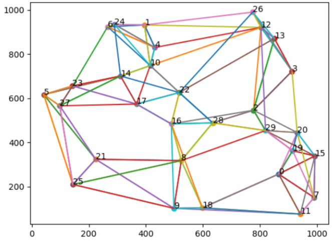
- An exact optimization algorithm terminates with **an optimal solution** and **a proof of optimality**
- In many cases, a significant portion of the total solution time is spent **proving that a solution found (quickly) is optimal**
- Another common scenario is that a large amount of computing time is spent going **from a near optimal** solution **to an optimal** one
- Exact mathematical programming algorithms can be **modified** to generate a **very good (but not necessarily optimal)** solution

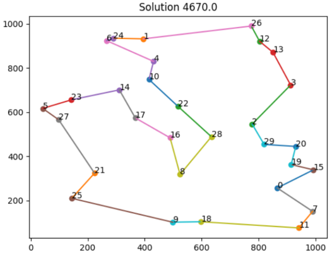
Introduction to matheuristics

TSP example for math. prog. to generate approximate solutions







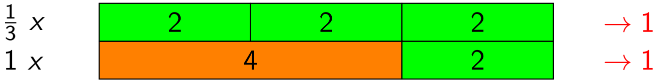


Introduction to matheuristics

Relaxation-based approaches (Ball)

- While a problem may be very difficult, a **certain relaxation** to that problem may be **efficiently solvable** (e.g., the LP-relaxation)
- Such relaxations can serve as a basis for effective heuristics
- Two general approaches are used:
  - The solution to a relaxation is modified to generate a feasible solution to the problem of interest (e.g., by rounding the solution of an LP-relaxation to obtain an integer solution, see Wäscher and Gau)
  - The second class of relaxation makes use of the dual information provided by the LP-relaxation in another heuristic

BPP example ($c = 6$, $w = [4, 2, 2]$)



To sum-up

Important aspects of matheuristics

- The name comes from a contraction between **metaheuristic** and **mathematical programming**
- Nowadays, matheuristics do not necessarily incorporate metaheuristic components anymore
- We identified **4 kinds** of matheuristics
  - Decomposition approaches (solve the sub-problems one after the other)
  - Improvement heuristics (ILP based local search)
  - ILP approximate solutions (solve ILP model with a subset of variables)
  - Relaxation-based approaches (use LP to build a solution)

Matheuristics in practice

- Matheuristics are extremely powerful in practice, especially for local search (but harder to publish)
- RL can help in some aspects of matheuristics (e.g., in the local search)
- Deep Learning can also help

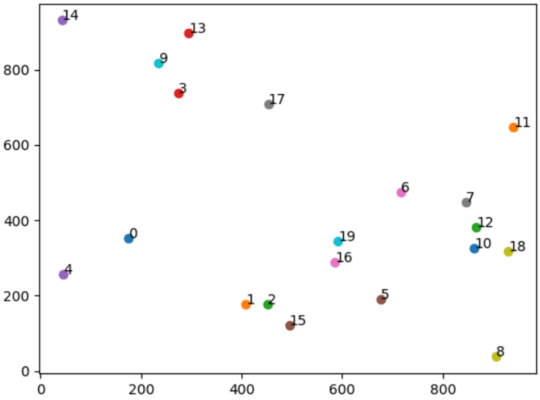
Let's study a practical case

Plan

Travelling Salesman Problem

The Travelling Salesman Problem

Given a weighted graph $G = (V, E)$ where c_{ij} indicates the weight of edge (i, j), the Travelling Salesman Problem (TSP) consists in selecting a set of edges forming an Hamiltonian cycle with minimum total weight.



ILP model for the TSP

ILP model for the TSP (exponential number of constraints)

$$
\begin{array}{llll}
\min & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \text{weight of selected assignments} \\
\text{s.t.} & \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\ldots,n & \text{an edge from every city} \\
& \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,n & \text{an edge to every city} \\
& \sum_{i\in S}\sum_{\substack{j\in S\\ i\neq j}} x_{ij} \leq |S|-1 & \forall S \supseteq \{1,\ldots,n\} & \text{no sub-tours} \\
& & 2 \leq |S| \leq n-1 & \\
& x_{ij} \in \{0,1\} & i = 1,\ldots,n & \text{domain constraints} \\
& & j = 1,\ldots,n &
\end{array}
$$

ILP model for the TSP

ILP model for the TSP (polynomial number of constraints)

$$
\begin{array}{lllr}
\min & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \text{weight of selected assignments} \\
\text{s.t.} & \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & j = 1,\dots,n & \text{an edge from every city} \\
& \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1,\dots,n & \text{an edge to every city} \\
& u_i - u_j + nx_{ij} \le n-1 & i = 2,\dots,n & \text{cities visited in incr. index} \\
& & j = 2,\dots,n, j \neq i & \\
& u_i \in \{1,\dots,n-1\} & i = 2,\dots,n & \text{index range} \\
& x_{ij} \in \{0,1\} & i = 1,\dots,n & \text{domain constraints} \\
& & j = 1,\dots,n &
\end{array}
$$

ILP model for the TSP

Comments on the ILP model with an exponential number of constraints

- In practice, it is **impossible** to generate every constraint prohibiting sub-tours for medium-size instances
- Usually, people solve a **relaxation** of the problem **without** these constraints and check **afterwards** whether or not the solution has any sub-tour
  - If it does **not**, then the solution is optimal
  - If it does, then the problem should be **solved again** with **an additional cut** to prevent the solution found previously to be generated again
- Machine Learning could be used to **add the cuts**

Comments on the ILP model with a polynomial number of constraints

- In practice, the model size grows **quadratically** with the number of cities
- In addition, the LP-relaxation of the model is **weak** (i.e., it is often far away from the best integer solution)
- ILP solvers are very powerful nowadays but struggle to find good quality solutions for large models with weak LP-relaxations
- Machine Learning could be used to **reduce the model size**

ILP model for the TSP

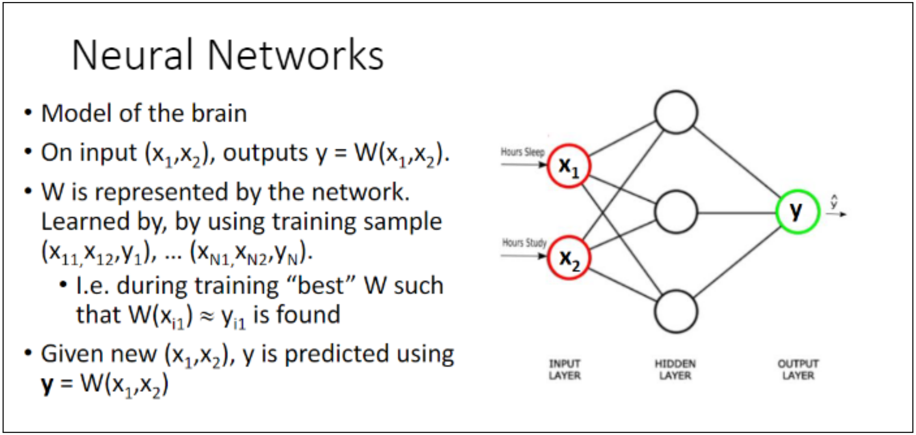
ILP model for the TSP (polynomial number of constraints)

$$\begin{array}{llll}
\min & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \text{weight of selected assignments} \\
\text{s.t.} & \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & j = 1, \ldots, n & \text{an edge from every city} \\
 & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1, \ldots, n & \text{an edge to every city} \\
 & u_i - u_j + nx_{ij} \leq n - 1 & i = 2, \ldots, n & \text{cities visited in incr. index} \\
 & & j = 2, \ldots, n, j \neq i & \\
 & u_i \in \{1, \ldots, n-1\} & i = 2, \ldots, n & \text{Index range} \\
 & x_{ij} \in \{0, 1\} & i = 1, \ldots, n & \text{domain constraints} \\
 & & j = 1, \ldots, n &
\end{array}$$

If one can predict that an edge will not be used in the optimal solution, then the model size can be reduced by one variable and one constraint

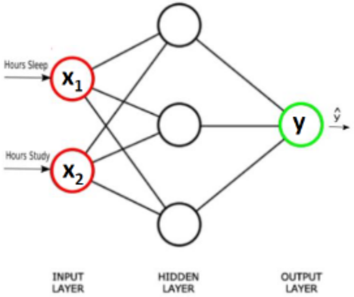
A matheuristic for the TSP

A brief recap



Neural Networks

- Model of the brain
- On input (x_1,x_2), outputs $y = W(x_1,x_2)$.
- W is represented by the network. Learned by, by using training sample $(x_{11,}x_{12},y_1)$, … $(x_{N1,}x_{N2},y_N)$.
  - I.e. during training "best" W such that $W(x_{i1}) \approx y_{i1}$ is found
- Given new (x_1,x_2), y is predicted using **y** = $W(x_1,x_2)$



Slide taken from Juan Vera (DMBA 2021/2022)

Activity

We want a model to predict, based on certain features, whether or not an edge will be contained in the optimal solution. We need to identify the input layers, compute the output layers, and train our neural network

Discuss 5 minutes with your neighbour about the features that could serve as input layer (e.g., the length of the edge), and how we could compute the associated output layer

Step 1 - Gathering the data

Information for edge (i, j) to serve as input layer

- Is c_{ij} among the shortest/longest edges including i or j?
- What is the rank of (i, j) w.r.t. the other edges including i or j and w.r.t. all the edges?
- Among all the edges crossing (i, j), how many are smaller/bigger?
- How many cities k are there such that c_{ik} and c_{jk} are smaller/higher than c_{ij}?
- How many cities k are there "between" i and j? Is there one city on top and one city below (i, j)?
- What is the value of the variable associated with (i, j) in the LP-relaxation? And its reduced cost?

Step 1 - Gathering the data

Tips

- Think about normalizing your data
- Think about having a balanced number of 0s and 1s
  - In every instance, n edges are selected (so $n^2 - n$ are not selected)
  - If the model is trained on a data set with large n, then a naive model with high accuracy will simply tell you to not select any edge

Step 2 - Training the Neural Network

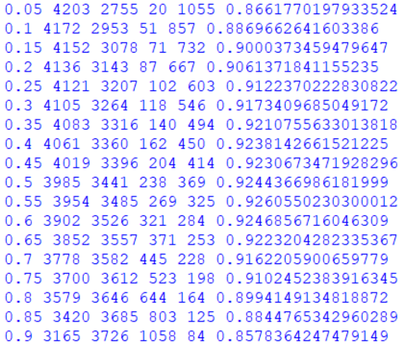
Things to take into account before training the Neural Network

- How much time do you have to train your model
- Size of the training set / testing set
- How many hidden layers
- How many neurons
- What kind of learning algorithm should be selected (SGD, Adam)
- What kind of neurons should be selected (ReLU, sigmoid)
- The loss function, the batch size, the number of epochs,

The choice of parameters is not as "easy" as it is for Q-learning! You might find some tips in specialized books/websites, but the best parameters will probably be achieved with trial and error and experience!

Activity

Let us assume that we have trained our Neural Network and the results are as follows (threshold, TP, TN, FN, FP, accuracy)



```
0.05 4203 2755 20 1055 0.8661770197933524
0.1 4172 2953 51 857 0.8869662641603386
0.15 4152 3078 71 732 0.9000373459479647
0.2 4136 3143 87 667 0.9061371841155235
0.25 4121 3207 102 603 0.9122370222830822
0.3 4105 3264 118 546 0.9173409685049172
0.35 4083 3316 140 494 0.9210755633013818
0.4 4061 3360 162 450 0.9238142661521225
0.45 4019 3396 204 414 0.9230673471928296
0.5 3985 3441 238 369 0.9244366986181999
0.55 3954 3485 269 325 0.9260550230300012
0.6 3902 3526 321 284 0.9246856716046309
0.65 3852 3557 371 253 0.9223204282335367
0.7 3778 3582 445 228 0.9162205900659779
0.75 3700 3612 523 198 0.9102452383916345
0.8 3579 3646 644 164 0.8994149134818872
0.85 3420 3685 803 125 0.8844765342960289
0.9 3165 3726 1058 84 0.8578364247479149
```

Discuss 5 minutes with your neighbour about the severity of

- Classifying as 0 an edge that should have been 1 (FN)
- Classifying as 1 an edge that should have been 0 (FP)

Based on your discussion, determine an appropriate cut-off value

Step 3 - Testing the Neural Network

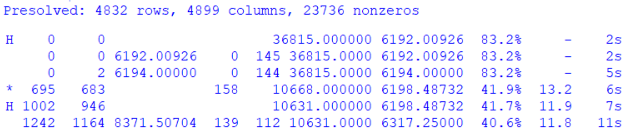
Things to take into account before testing the Neural Network

- What is the threshold?
- What should we compare the model against? (heuristic, ILP with time limit)
- How many instances are we going to use?
- What kind of instances? The same as those used for the training?
- Could we test the model on larger instances?

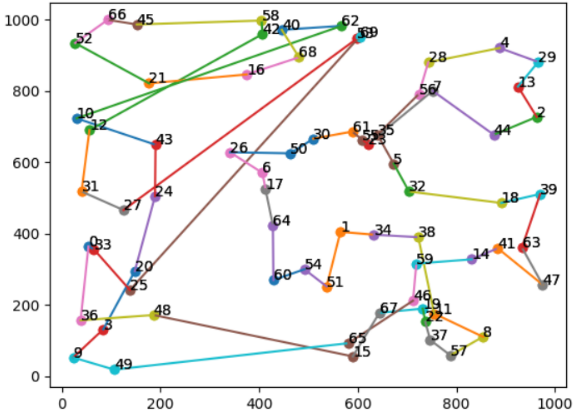
Let us see what happens with a NN trained with instances composed of 20 cities when applied to instances composed of 70 cities (threshold at 0.15)

Step 3 - Testing the Neural Network

Solution after 15 seconds of ILP solver

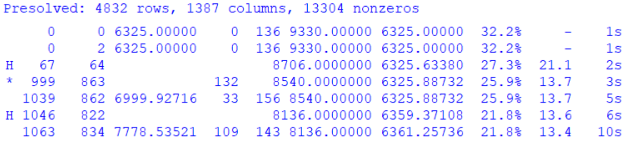


```
Presolved: 4832 rows, 4899 columns, 23736 nonzeros

H     0     0                    36815.000000 6192.00926  83.2%     -    2s
      0     0 6192.00926    0  145 36815.0000 6192.00926  83.2%     -    2s
      0     2 6194.00000    0  144 36815.0000 6194.00000  83.2%     -    5s
*   695   683             158    10668.000000 6198.48732  41.9%  13.2    6s
H  1002   946                    10631.000000 6198.48732  41.7%  11.9    7s
   1242  1164 8371.50704  139  112 10631.0000 6317.25000  40.6%  11.8   11s
```

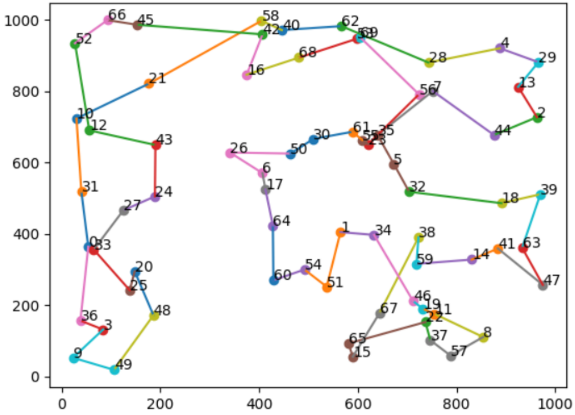


Step 3 - Testing the Neural Network

Solution after 15 seconds of ILP solver



```
Presolved: 4832 rows, 1387 columns, 13304 nonzeros

     0     0 6325.00000    0  136 9330.00000 6325.00000  32.2%     -    1s
     0     2 6325.00000    0  136 9330.00000 6325.00000  32.2%     -    1s
H   67    64                    8706.0000000 6325.63380  27.3%  21.1    2s
*  999   863             132    8540.0000000 6325.88732  25.9%  13.7    3s
  1039   862 6999.92716   33  156 8540.00000 6325.88732  25.9%  13.7    5s
H 1046   822                    8136.0000000 6359.37108  21.8%  13.6    6s
  1063   834 7778.53521  109  143 8136.00000 6361.25736  21.8%  13.4   10s
```



To sum-up

Important aspects of matheuristics based on variable fixing

- The first step is to **gather the data** (normalized and balanced)
- The second step is to **train the neural network** (parameters, training/testing set)
- The third step is to **test the model** (competitor, threshold)

Variable fixing in practice

- Gathering the data and computing the value of the output layer is really challenging
- The process rarely leads to an optimal solution because it only requires one false negative to lose the optimum ($0.99^{100} \approx 0.37$)
- In most cases, the proof of optimality is lost
- Nevertheless, variable fixing can be useful when ILP models have too many variables

Fancy techniques are not always the best, decision trees can also help with variable fixing while being more interpretable!

Plan

ILP model for the TSP

ILP model for the TSP (exponential number of constraints)

$$
\begin{array}{llll}
\min & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \text{weight of selected assignments} \\
\text{s.t.} & \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & j = 1,\ldots,n & \text{an edge from every city} \\
 & \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,n & \text{an edge to every city} \\
 & \displaystyle\sum_{i\in S}\sum_{\substack{j\in S\\ i\neq j}} x_{ij} \leq |S| - 1 & \forall S \supseteq \{1,\ldots,n\} & \text{no sub-tours} \\
 & & 2 \leq |S| \leq n-1 & \\
 & x_{ij} \in \{0,1\} & i = 1,\ldots,n & \text{domain constraints} \\
 & & j = 1,\ldots,n &
\end{array}
$$

A decomposition approach for the TSP

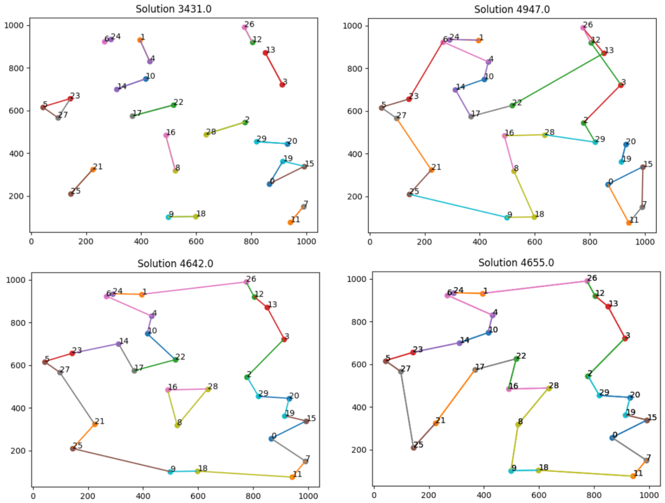
Master problem

$$
\begin{array}{llll}
\min & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \text{weight of selected assignments} \\
\text{s.t.} & \displaystyle\sum_{i=1}^{n} x_{ij} = 1 & j = 1, \ldots, n & \text{an edge from every city} \\
& \displaystyle\sum_{j=1}^{n} x_{ij} = 1 & i = 1, \ldots, n & \text{an edge to every city} \\
& \displaystyle\sum_{i \in S}\sum_{\substack{j \in S \\ i \neq j}} x_{ij} \leq |S| - 1 & \forall S \in \mathcal{C} & \text{no sub-tours} \\
& x_{ij} \in \{0, 1\} & i = 1, \ldots, n & \text{domain constraints} \\
& & j = 1, \ldots, n &
\end{array}
$$

where $\mathcal{C}$ is the set of sub-tours spotted in the solutions provided by the master problem so far

A general decomposition framework

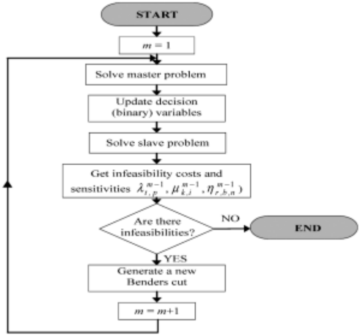
Example in which $\mathcal{C} = \emptyset$ initially



Run BC_TSP.py

A decomposition approach for the TSP

“Benders’ decomposition” flowchart (see Khodr et al.)


START
$m = 1$
Solve master problem
Update decision (binary) variables
Solve slave problem
Get infeasibility costs and sensitivities $\lambda_{t,p}^{m-1}, \mu_{k,j}^{m-1}, \eta_{r,b,n}^{m-1}$)
Are there infeasibilities?
NO
END
YES
Generate a new Benders cut
$m = m+1$

Implementation details

In practice

- Solving the **master problem** from scratch every time is not efficient
- Researchers use the `callback` function of the ILP solver that allows the user to add a cut **during** the search procedure at some key moments (e.g., after a new integer solution, called an **"incumbent"**, has been found)
- Such cuts are called **"lazy constraints"**
- Forbidding a solution is easy when the variables are binary, we simply have to add a so-called "**no-good cut**" (e.g., $x_1 + x_2 + x_3 \leq 2$ prevents any solution with $x_1 = x_2 = x_3 = 1$)

Implementation details

In practice

- When the variables are integer, finding a suitable cut is more difficult: how would you forbid a solution where $x_1 = 1$, $x_2 = 1$, and $x_3 = 2$?
- Some solvers offer the option to **reject** a solution (such as Cplex) while others (such as Gurobi) do not

Adding a no-good cut

- Depending of the difficulty of the **slave problem**, there are several strategies
    - If there is only one slave problem (or several trivial slave problems) to solve and the master problem has a reasonable size, add every violated cut
    - If there are many difficult slave problems to solve or if the master problem is already huge, add a subset of cuts (one cut is enough to forbid the current master problem solution)

Let's study a practical case

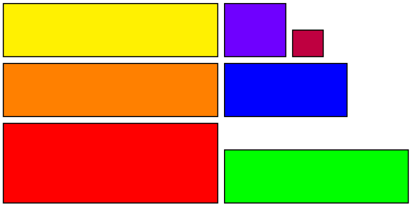
Plan

2D-BPP

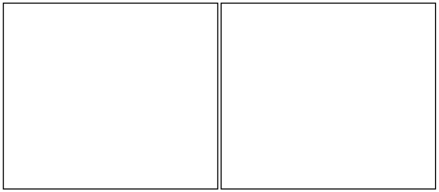
The 2-Dimensional Bin Packing Problem

Given n rectangular items of width w_j and height h_j ($j = 1, \ldots, n$) and an unlimited number of identical bins with fixed width W and high H, the 2-Dimensional Bin Packing Problem (2D-BPP) consists in packing all the items using the minimum number of bins. Items must be packed with their edges parallel to those of the strip and cannot be rotated.

Items



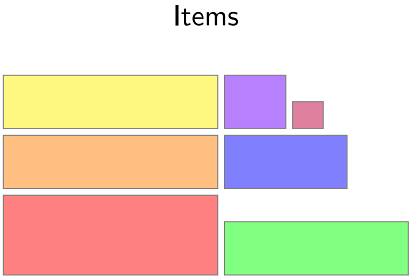
Bins

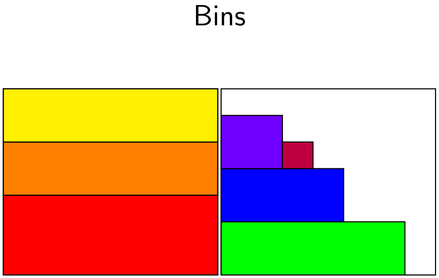


2D-BPP

The 2-Dimensional Bin Packing Problem

Given n rectangular items of width w_j and height h_j $(j = 1, \ldots, n)$ and an unlimited number of identical bins with fixed width W and high H, the 2-Dimensional Bin Packing Problem (2D-BPP) consists in packing all the items using the minimum number of bins. Items must be packed with their edges parallel to those of the strip and cannot be rotated.





ILP model for the 2D-BPP

ILP model for the 2D-BPP

$$\begin{array}{llll}
\min & \displaystyle\sum_{k=1}^{K} y_k & & \text{number of open bins} \\
\text{s.t.} & \displaystyle\sum_{i=0}^{W-w_l}\sum_{j=0}^{H-h_l}\sum_{k=1}^{K} x_{lijk} = 1 & l = 1,\dots,n & \text{every item is packed in a bin} \\
& \displaystyle\sum_{l=1}^{n}\sum_{i'=i-w_l+1}^{i}\sum_{j'=j-h_l+1}^{j} x_{li'j'k} \le y_k & i = 0,\dots,W-1 & \text{no overlaps} \\
& & j = 0,\dots,H-1 & \\
& & k = 1,\dots,K & \\
& y_k \in \{0,1\} & k = 1,\dots,K & \text{domain constraints} \\
& x_{lijk} \in \{0,1\} & \forall l,i,j,k & \text{domain constraints}
\end{array}$$

This ILP model does not work particularly well because of its large size and its poor LP-relaxation

A decomposition approach for the 2D-BPP

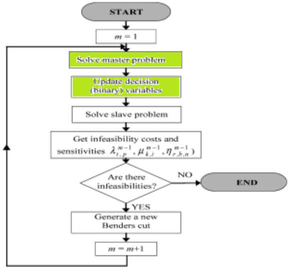
Master problem (see Côté et al.)

$$
\begin{array}{llll}
\min & \sum_{k=1}^{K} y_k & & \text{number of open bins} \\
\text{s.t.} & \sum_{k=1}^{K} x_{lk} = 1 & l = 1, \ldots, n & \text{every item is packed in a bin} \\
 & \sum_{l=1}^{n} w_l \cdot h_l \cdot x_{lk} \leq W \cdot H \cdot y_k & k = 1, \ldots, K & \text{capacity constraints} \\
 & \sum_{l \in s} x_{lk} \leq |s| - 1 & s \in \mathcal{S} & \text{no-good cuts} \\
 & & k = 1, \ldots, K & \\
 & y_k \in \{0,1\} & k = 1, \ldots, K & \text{domain constraints} \\
 & x_{lk} \in \{0,1\} & l = 1, \ldots, n & \text{domain constraints} \\
 & & k = 1, \ldots, K &
\end{array}
$$

This ILP model is much easier to solve!

A decomposition approach for the 2D-BPP

“Benders’ decomposition” flowchart (see Khodr et al.)


START
$m = 1$
Solve master problem
Update decision (binary) variables
Solve slave problem
Get infeasibility costs and sensitivities $\lambda_{i,p}^{m-1}, \mu_{k,j}^{m-1}, \eta_{r,b,n}^{m-1}$)
Are there infeasibilities?
NO
END
YES
Generate a new Benders cut
$m = m+1$

What information does the master problem return?

A decomposition approach for the 2D-BPP

Slave problem (to solve for every bin k and candidate set of items I_k)

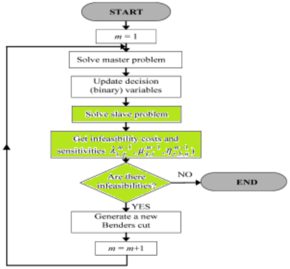
$$
\begin{array}{llll}
\min & 0 & & \text{decision problem} \\
\text{s.t.} & \displaystyle\sum_{i=0}^{W-w_l}\sum_{j=0}^{H-h_l} x_{lij} = 1 & l \in I_k & \text{every item is packed in the bin} \\
& \displaystyle\sum_{l\in I_k}\sum_{i'=i-w_l+1}^{i}\sum_{j'=j-h_l+1}^{j} x_{li'j'} \le 1 & i = 0,\dots,W-1 & \text{no overlaps} \\
& & j = 0,\dots,H-1 & \\
& x_{lij} \in \{0,1\} & \forall l,i,j & \text{domain constraints}
\end{array}
$$

Advantages of this ILP model

- Its reduced size as $|I_k| << n$ and $1 << K$ (it is usually faster to solve K ILP models with $O(n)$ variables than solving one ILP model with $O(Kn)$ variables)
- A cut can be added to the master problem as soon as one of the slave problems is infeasible

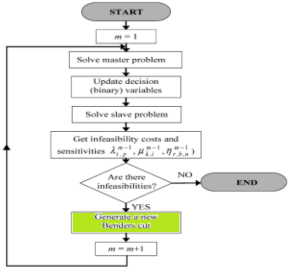
A decomposition approach for the 2D-BPP

“Benders’ decomposition” flowchart (see Khodr et al.)


START
$m = 1$
Solve master problem
Update decision (binary) variables
Solve slave problem
Get infeasibility costs and sensitivities $\lambda_{i,p}^{m-1}, \mu_{k,j}^{m-1}, \eta_{r,b,n}^{m-1}$)
Are there infeasibilities?
NO
END
YES
Generate a new Benders cut
$m = m+1$

A decomposition approach for the 2D-BPP

“Benders’ decomposition” flowchart (see Khodr et al.)


START
$m = 1$
Solve master problem
Update decision (binary) variables
Solve slave problem
Get infeasibility costs and sensitivities $\lambda_{i,p}^{m-1}, \mu_{k,j}^{m-1}, \eta_{r,b,n}^{m-1}$)
Are there infeasibilities?
NO
END
YES
Generate a new Benders cut
$m = m+1$

A decomposition approach for the 2D-BPP

In practice

- The slave problem can be very slow and we might only be interested in an approximate solution
- Could ML techniques help solving the slave problem quicker?

We want a model to predict, based on certain features, whether or not a set of items will fit into a bin. If the model predicts a success, then the slave problem is called to confirm it. If the model predicts a failure, then a no-good cut is directly added to the master. We need to identify the input layers, compute the output layers, and train our neural network.

Discuss 5 minutes with your neighbour about the features that could serve as input layer (e.g., the number of items), and how we could compute the associated output layer

Step 1 - Gathering the data

Information for set I_k to serve as input layer

- Number of items in set $|I_k|$
- Proportion of the bin covered by items
- Number of items fitting in category $1, 2, \ldots, C$

Information about categories

- Category 1 includes items with width of 1 and height of 1
- Category 2 includes items with width of 1 and height of 2
- Category C includes items with width of w_{max} and height of h_{max}

Tips and remarks

- This model will only be valid for a fixed W and a fixed H, so data normalization is not as important
- Think about having a balanced number of 0s and 1s
- Generating 15,000 instances for the training set took me 5 days, so use my training data set!

Step 2 - Training the Neural Network

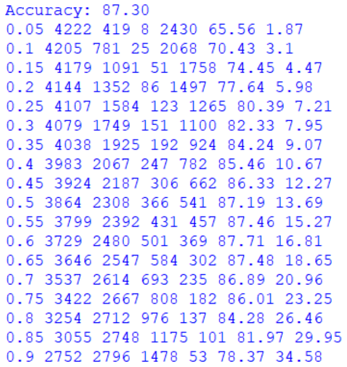
Things to take into account before training the Neural Network

- How much time do you have to train your model
- Size of the training set / testing set
- How many hidden layers
- How many neurons
- What kind of learning algorithm should be selected (SGD, Adam)
- What kind of neurons should be selected (ReLU, sigmoid)
- The loss function, the batch size, the number of epochs,

The architecture of the network will probably not be the same as the one we used for the TSP

Activity

Let us assume that we have trained our Neural Network and the results are as follows (threshold, TP, TN, FN, FP, accuracy)



```
Accuracy: 87.30
0.05 4222 419 8 2430 65.56 1.87
0.1 4205 781 25 2068 70.43 3.1
0.15 4179 1091 51 1758 74.45 4.47
0.2 4144 1352 86 1497 77.64 5.98
0.25 4107 1584 123 1265 80.39 7.21
0.3 4079 1749 151 1100 82.33 7.95
0.35 4038 1925 192 924 84.24 9.07
0.4 3983 2067 247 782 85.46 10.67
0.45 3924 2187 306 662 86.33 12.27
0.5 3864 2308 366 541 87.19 13.69
0.55 3799 2392 431 457 87.46 15.27
0.6 3729 2480 501 369 87.71 16.81
0.65 3646 2547 584 302 87.48 18.65
0.7 3537 2614 693 235 86.89 20.96
0.75 3422 2667 808 182 86.01 23.25
0.8 3254 2712 976 137 84.28 26.46
0.85 3055 2748 1175 101 81.97 29.95
0.9 2752 2796 1478 53 78.37 34.58
```

According to you, what is the severity of

- Classifying as 0 an item set that should have been 1 (FN)
- Classifying as 1 an item set that should have been 0 (FP)

Based on your discussion, determine an appropriate cut-off value

Step 3 - Testing the Neural Network

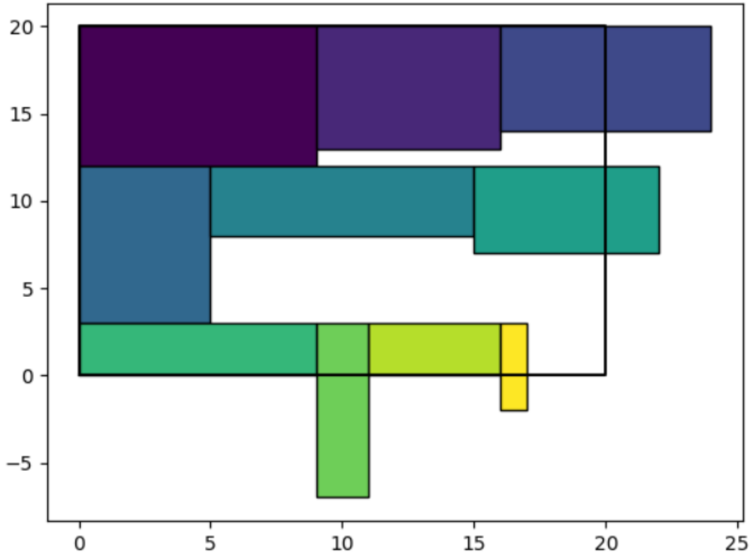
Things to take into account before testing the Neural Network

- What is the threshold?
- What should we compare the model against?
- How many instances are we going to use?
- What kind of instances? The same as those used for the training?
- Could we test the model on larger instances?

Run Math2BPP_TestModel.py

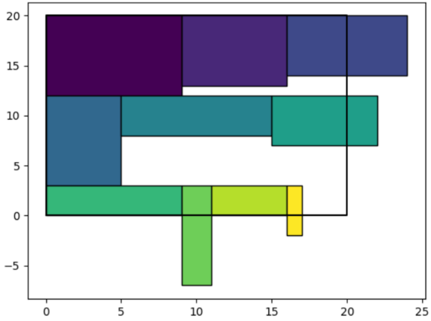
Human vs. machine round 1

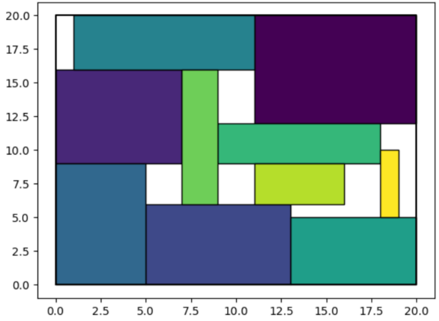
Is this instance feasible?



Human vs. machine round 1

It is!



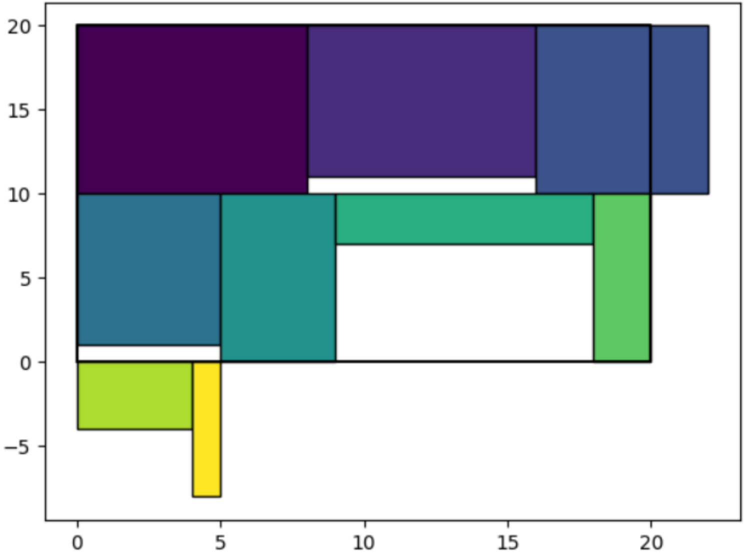


```
start of iteration 0
I think it fits
```

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00000    0   10          -    0.00000      -     -    0s
H    0     0                       0.0000000    0.00000  0.00%     -    0s

Explored 1 nodes (1799 simplex iterations) in 0.31 seconds
Thread count was 4 (of 4 available processors)
```

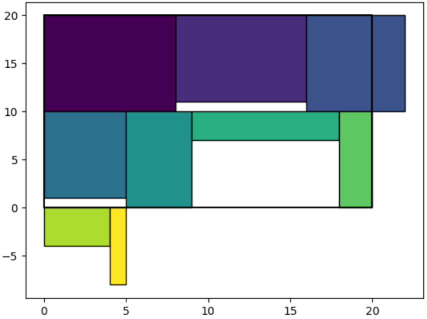
Human vs. machine round 2

Is this instance feasible?



Human vs. machine round 2

It is not!

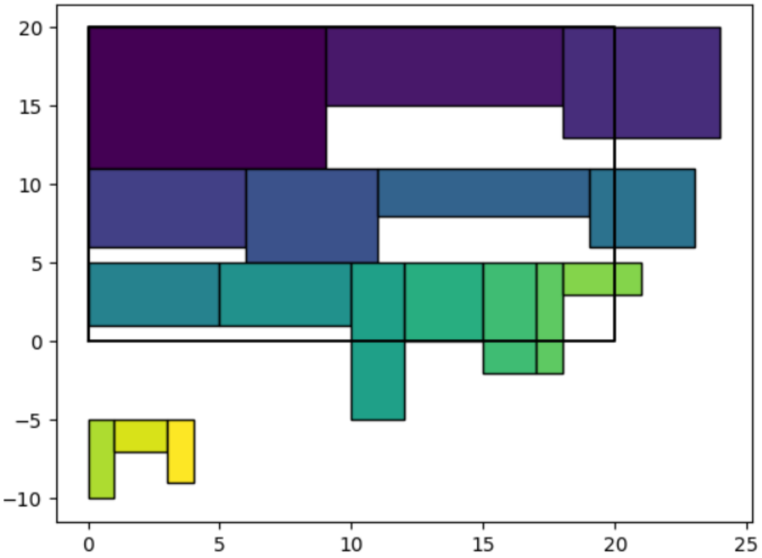


```
start of iteration 3
I think it does NOT fit
```

```
Root relaxation: objective 0.000000e+00, 348 iterations, 0.02 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00000    0   13          -    0.00000      -     -    0s
     0     0    0.00000    0   55          -    0.00000      -     -    0s
     0     0    0.00000    0   45          -    0.00000      -     -    0s
     0     0    0.00000    0   14          -    0.00000      -     -    1s
     0     0    0.00000    0   14          -    0.00000      -     -    1s
     0     0    0.00000    0   10          -    0.00000      -     -    2s
     0     0    0.00000    0   76          -    0.00000      -     -    3s
     0     0    0.00000    0   56          -    0.00000      -     -    3s
     0     0    0.00000    0   12               0.00000                 3s
     0     0    0.00000    0   10          -    0.00000      -     -    3s
     0     2    0.00000    0    8          -    0.00000      -     -    5s
   919     9 infeasible   21               -    0.00000      -   131   10s

Cutting planes:
  Gomory: 2
  Cover: 40
  Clique: 51
  Inf proof: 5
  Zero half: 24

Explored 2338 nodes (313667 simplex iterations) in 15.04 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Model is infeasible
```

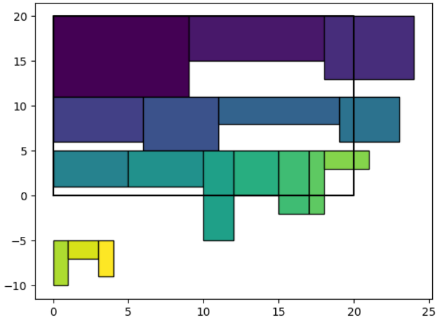
Human vs. machine round 3

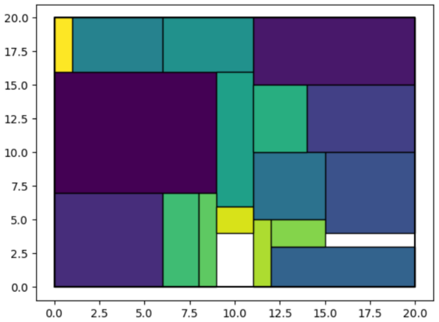
Is this instance feasible?



Human vs. machine round 3

It is!



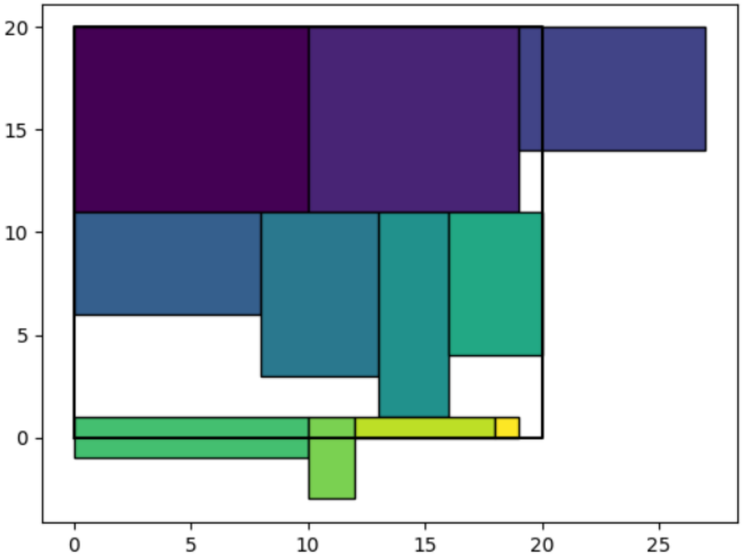


```
start of iteration 4
I think it fits
```

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00000    0   52          -    0.00000      -     -    0s
H    0     0                       0.0000000    0.00000  0.00%     -    0s

Explored 1 nodes (4672 simplex iterations) in 0.97 seconds
Thread count was 4 (of 4 available processors)
```

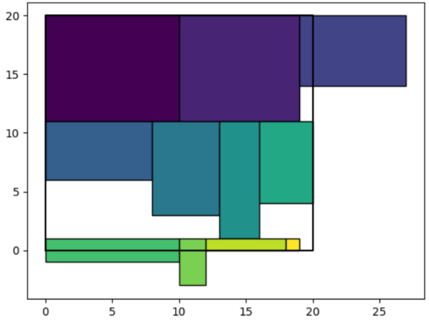
Human vs. machine round 4

Is this instance feasible?



Human vs. machine round 4

It is not!

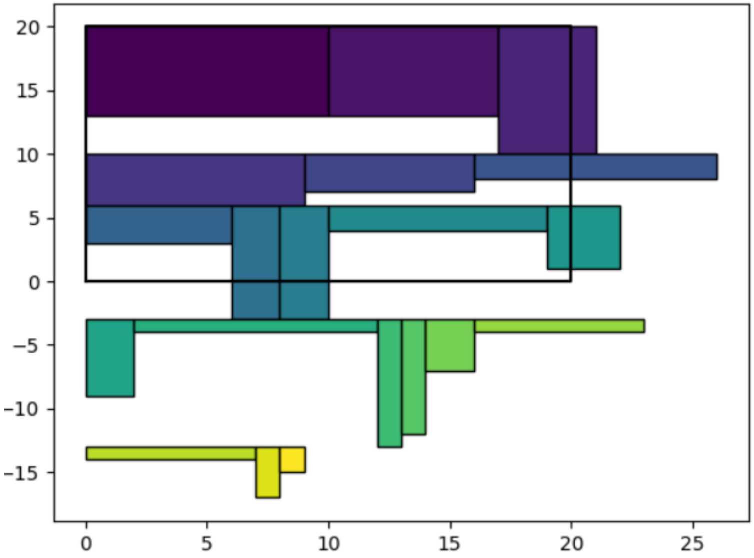


```
Root relaxation: objective 0.000000e+00, 880 iterations, 0.06 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00000    0  111          -    0.00000      -     -    1s
     0     0    0.00000    0  121          -    0.00000      -     -    1s
     0     0    0.00000    0  111          -    0.00000      -     -    1s
     0     0    0.00000    0  145          -    0.00000      -     -    4s
     0     0    0.00000    0  144          -    0.00000      -     -    4s
     0     0    0.00000    0  140          -    0.00000      -     -    5s
     0     0    0.00000    0  139          -    0.00000      -     -    5s
     0     2    0.00000    0   10          -    0.00000      -     -    8s
    61     5    0.00000    5  168          -    0.00000      -   699   10s

Cutting planes:
  Clique: 50
  MIR: 1

Explored 83 nodes (80254 simplex iterations) in 10.78 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Model is infeasible
```

```
start of iteration 6
I think it does NOT fit
```

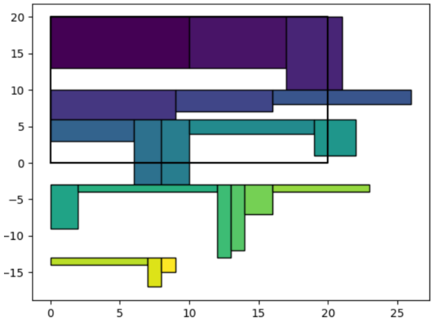
Human vs. machine round 5

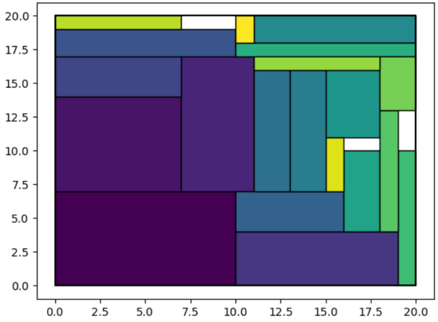
Is this instance feasible?



Human vs. machine round 5

It is!





```
start of iteration 7
I think it fits
```

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

H    0     0                      0.0000000    0.00000  0.00%     -    0s

Explored 0 nodes (323 simplex iterations) in 0.20 seconds
Thread count was 4 (of 4 available processors)
```

To sum-up

Important aspects of Benders' like decomposition approaches

- It is useful for complex problems that can be split
- The master problem should be much easier than the original problem
- The slave problem should be fast to solve (sometimes it is even solvable in polynomial time)
- Attention should be paid to how the no-good cuts are added to the master problem

Benders' like decomposition approaches in practice

- These are extremely fashionable at the moment in operations research
- One reason is because it allows us to solve exactly some very difficult problems
- Machine learning is a good candidate to help solving the slave problem . . . but it might be at the cost of the optimality proof
- Not every decomposition works better than the original formulation
- Some decomposition approaches only work well on a subset of instances