

Machine Learning Inside Decomposition of Scheduling Problems

Přemysl Šůcha, Antonín Novák, Broos Maenhout, Pavlína Koutecká, Roman Václavík, Jan Hůla,
 Michal Bouška, Zdeněk Hanzálek
 (suchap@cvut.cz)

Czech Technical University in Prague
 Czech Institute of Informatics, Robotics and Cybernetics

Ghent University
 Faculty of Economics and Business Administration

$$\min \left((t_u - t_1) + \frac{\sum_{\forall(i,j) \in U: c_{i,j} (\bar{p}_{i,j} - p_{i,j})}{1 + \sum_{\forall(i,j) \in U: c_{i,j} (\bar{p}_{i,j} - p_{i,j})} \right)$$

s.t.

$$t_j - t_i - p_{i,j} \geq 0$$

$$\sum_{u \in \mathcal{A}} w_u = 1$$

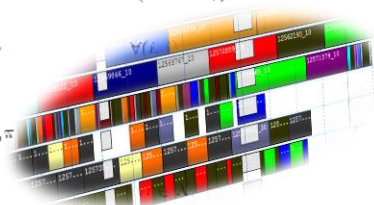
$$W(\omega^{in(u)}) < w_u \pi$$

where

$$t_i \in \mathbb{R}_{\geq 0}$$

$$p_{i,j} \in [p_{i,j}, \bar{p}_{i,j}] \cap \mathbb{Z}_{\geq 0} \quad \forall (i,j) \in U$$

$$w_u \in [0,1] \quad \forall u \in \mathcal{A}$$



Why ML? And why in decomposition approaches?

- There is a lack of scheduling methods **exploiting previously solved instances** for solving new ones.
- Heuristic for scheduling problems often require **fine-tuning of its parameters**.
- It is common that CPU time of scheduling algorithms depends on the **distribution of instance parameters** ($1 \mid \sum w_j U_j$, strongly correlated instances $w_j = p_j$)

***Research question:** is it possible to extract any useful information from the solved instances and use it efficiently to accelerate solving of an unseen instance?*

Why ML? And why in decomposition approaches?

- Challenges/issues of ML:
 - Current ML is not very good in **approximating combinatorial problems**
 - Instance size **scaling** is not trivial for ML
 - Problem **feasibility** is an issue for ML
 - Getting **training instances** is time demanding (NP-hardness of scheduling problems)

***Research hypothesis:** Combination of ML with decomposition methods can mitigate the above stated ML issues.*

Outline

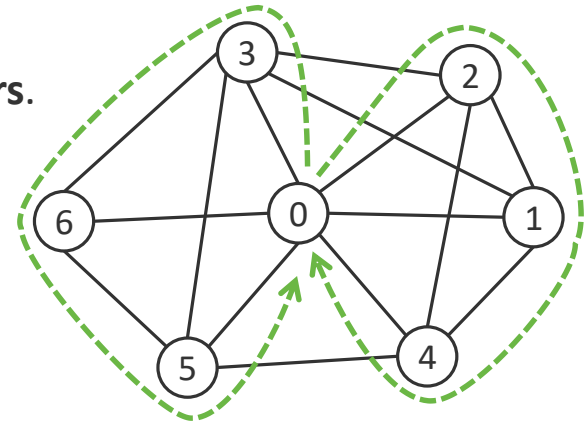
- Branch and price and ML
 - Ranker
 - Estimator
- Lawler's decomposition and ML ($1 \mid \mid \Sigma T_j$)
- Conclusion

Branch and price and ML

Branch and price - Example

- Example: **Vehicle Routing Problem with Time Windows**

- The problem is given by a directed graph $G = (V, A)$,
- Vertex $0 \in V$ is the **depot**, vertices 1 to n represent **customers**.
- A **cost** c_{ij} and a **travel time** t_{ij} are defined for every $(i, j) \in A$.
- Every customer $i \in V \setminus \{0\}$ has a **positive demand** d_i , a **time window** $[a_i, b_i]$ and a positive service time r_i .
- A **fleet** of U vehicles of **capacity** Q is available for serving the customers.
- Vehicles must begin and end their routes at the depot.
- The VRPTW searches for **routes** of vehicles **minimizing the cost**.

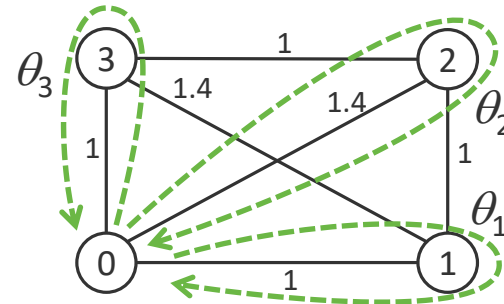


Branch and price - Dantzig-Wolfe decomposition

Example: Vehicle Routing Problem with Time Windows

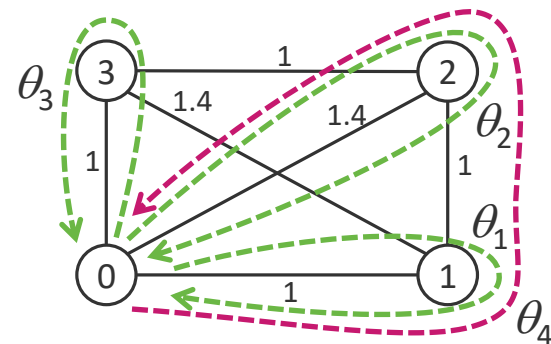
1) minimize $2\theta_1 + 2.8\theta_2 + 2\theta_3$
 subject to

$$\begin{aligned} \theta_1 &\geq 1 \\ \theta_2 &\geq 1 \\ \theta_3 &\geq 1 \\ \theta_1, \theta_2, \theta_3 &\geq 0 \end{aligned}$$



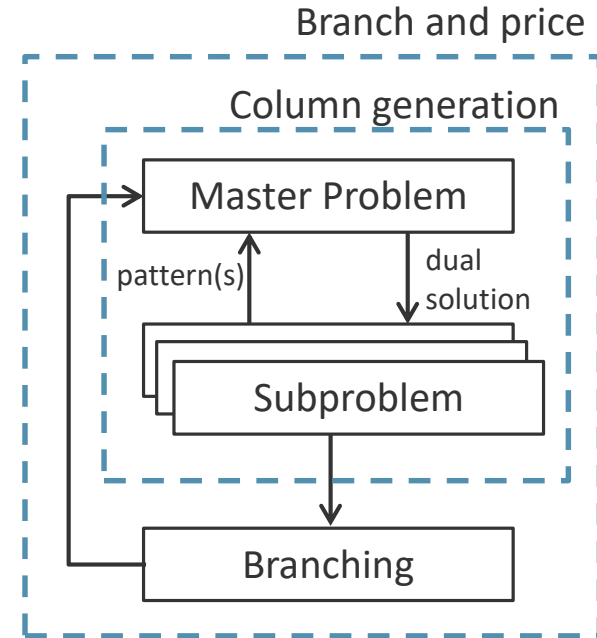
2) minimize $2\theta_1 + 2.8\theta_2 + 2\theta_3 + 3.4\theta_4$
 subject to

$$\begin{aligned} \theta_1 &+ \theta_4 \geq 1 \\ \theta_2 &+ \theta_4 \geq 1 \\ \theta_3 &\geq 1 \\ \theta_1, \theta_2, \theta_3, \theta_4 &\geq 0 \end{aligned}$$



Branch and price

- **Column generation** solver the linear relaxation
- **Branching** guarantees finding optimal integer solution
- Column generation has **Master problem & Subproblem(s)** (or Pricing Problem(s))
- Subproblem(s) generate solutions to decomposed subproblems (patterns) – parametrized by dual solution
- Master Problem combines individual patterns
- Subproblems detects whether column generation can be stopped (**reduced cost**)
- The algorithm provides many opportunities to apply machine learning (interaction among multiple mathematical models, solving similar problems repetitively, ...)



Use of Machine Learning in Branch and Price: Existing Papers

<i>Paper</i>	<i>Focus</i>	<i>Optimal</i>	<i>Branching</i>	<i>Multiple subproblems</i>	<i>Online/offline ML</i>
Václavík et al. (2018)	Subproblem	Yes	Yes	No	Online
Morabit et al. (2021)	Master Problem	Yes (CG)	No	No	Offline
Tahir et al. (2021)	Subproblem	No	No	No	Offline
Shen et al. (2021)	Subproblem	Yes	Yes	No	Offline
Quesnel et al. (2022)	Subproblem	No	Yes	No	Offline
Yuan et al. (2022)	Subproblem	No	Yes	No	Offline
Morabit et al. (2022)	Subproblem	No	Yes	No	Offline
Parmentier (2022)	Subproblem	Yes	Yes	No	Offline
Kraul et al. (2023)	Master Problem	No	No	No	Offline
Koutecká et al. (2024)	Subproblem	Yes	Yes	Yes	Offline

Branch and price and ML

Idea I. - Ranker

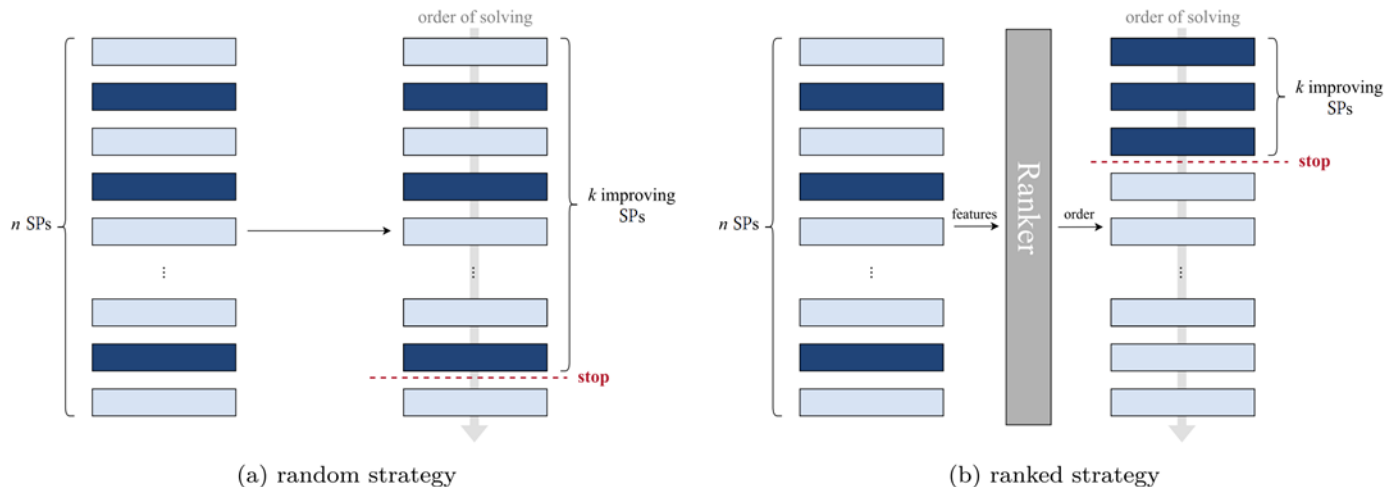
Related paper:

P. Koutecká, P. Šůcha, J. Hula, B. Maenhout: A machine learning approach to rank pricing problems in branch-and-price. Eur. J. Oper. Res. 320(2): 328-342 (2025)

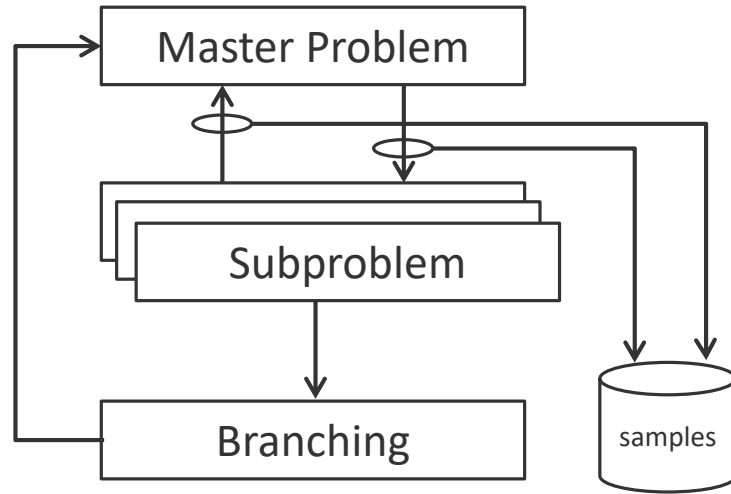
Branch and Price with multiple subproblems

- The majority of the **CPU time** of the branch and price is usually spent in the **subproblem(s)**
- In the case of multiple subproblems (SP) there are multiple:
 - solve all subproblems and add **all patterns** with negative reduced cost
 - solve all subproblems and add pattern with **the most negative reduced cost**
 - stop solving SPs when finding k patterns with negative reduced cost (**partial pricing**)

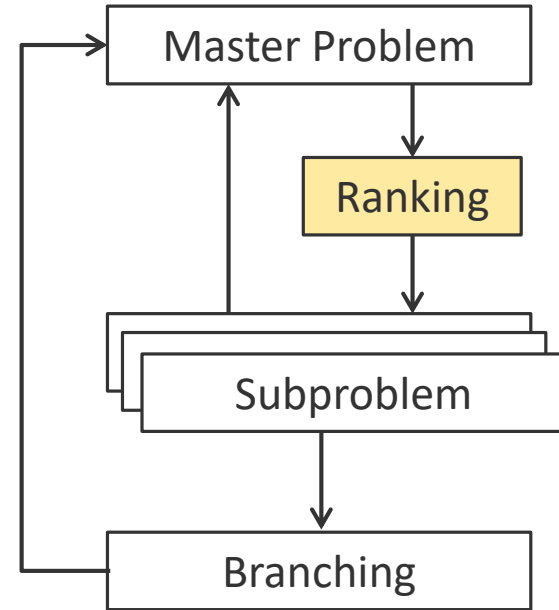
- Partial pricing:



Use of the Ranker in Branch and Price



Acquisition of training samples



Usage of the ranker

Learning to Rank

- Typical ML predicts a class or value for a **single item**
- Learning to rank generates a **relative order of multiple items** (more challenging)
- Learning to rank is defined as:
 - q is a **query** sampled from distribution Q .
 - Each q has n **items** $\{x_1^q, \dots, x_n^q\}$ each represented by m features
 - For each item we have **relevance score** r_1^q , defining the order in q
- Learning to rank **approaches**:
 - pointwise – considers items independently
 - pairwise – considers items pairs
 - listwise – considers items list

Learning to Rank

- The idea was applied to **Operating Room Planning and Scheduling** problem given by
 - set of surgeons and operating rooms
 - set of patients on a waiting list
 - nonlinear objective function penalizing waiting time, overtime, ...
- For each subproblem we defined **17 features** of four types
 - Count (e.g., number of patients)
 - Average (e.g., average surgery time of considered patients)
 - Histogram (e.g., clinical priority of assumed patients)
 - Value (e.g., value of a dual variable)
- Two types of **relevance scores**
 - Binary
 - Graded

Training of Ranking models

- Training data
 - 120 instances of the Operating Room Planning and Scheduling problem (heterogenous, small)
 - 58,408 queries assuming 932,190 subproblems
 - 80% for training, 20% testing
- Training process \leq **5 minutes** (LambdaMART model)

Experimental Results

- We compared multiple **Learning to rank models** ($Precision = \#relevant\ top\ ranked\ items/k$)

<i>Model Type</i>	<i>Ranking model</i>	<i>Precision (k=1)</i>
Pairwise	LambdaRank	0.74
	LambdaMART	0.92
Listwise	ListNet	0.76
Pointwise	Random forest	0.59
	Gradient boosted trees	0.76
	Neural network	0.78

- **Binary relevance** outperformed **graded relevance** regarding ranking metrics but it was worst inside the branch and price
- The ranking procedure **reduces up to 41% of subproblems** ($k=5$), and saves about **10% of the CPU time**.

Branch and price and ML

Idea II. - Estimator

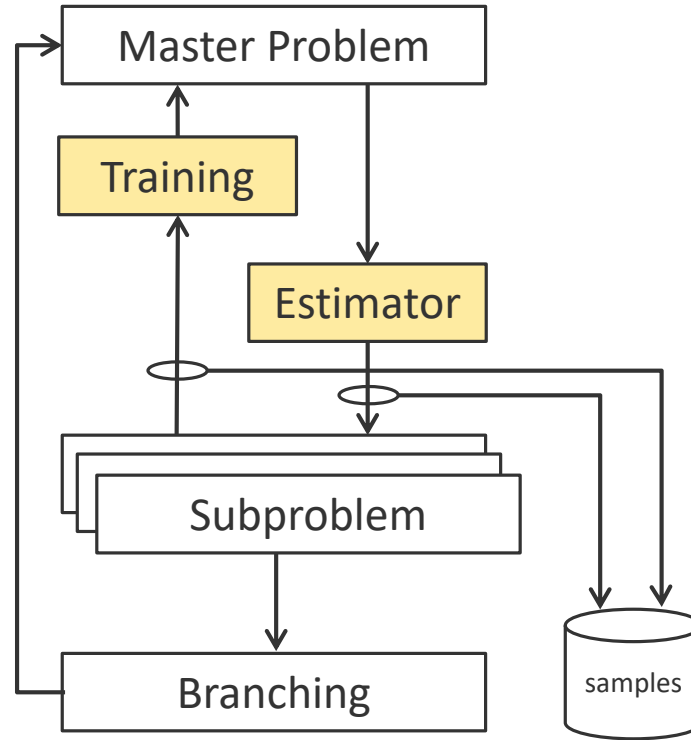
Related paper:

R. Václavík, A. Novák, P. Šůcha, Zdenek Hanzálek: Accelerating the Branch-and-Price Algorithm Using Machine Learning. Eur. J. Oper. Res. 271(3): 1055-1069 (2018)

Branch and Price and Subproblem(s)

- Properties of subproblem(s):
 - Usually **NP-hard** combinatorial problems
 - Are solved **repetitively** (usually differing only in the objective function)
 - Typically consume the majority of the **computation time**
- The idea is to **predict an upper bound** to the subproblem to prune its solution space (assuming minimization form of the objective function)
- Online ML

Use of the Estimator in Branch and Price




Usage of the estimator and its training

Estimator in the Branch and Price


Algorithm : Upper bound prediction for the pricing problem.

```
/* column generation */
1 do
  /* master problem part */
  2 ( $\gamma, \boldsymbol{\pi}$ )  $\leftarrow$  current dual solution of the RMP
  /* pricing problem part */
  3  $\boldsymbol{\phi} = \boldsymbol{\phi}(\gamma, \boldsymbol{\pi})$ 
  4  $\hat{f} \leftarrow \mathbf{w}^T \boldsymbol{\phi}$ 
  5  $\mathbf{x} \leftarrow$  solve pricing problem with upper bound  $\hat{f}$ 
  6 if pricing problem is infeasible then
  7   |  $\mathbf{x} \leftarrow$  solve pricing problem with default bounds
  8 end
  /* updating part */
  9 add column  $\mathbf{x}$  into the RMP
  10 add new data point ( $\boldsymbol{\phi}, f(\mathbf{x})$ ) to the model (2)-(5)
  11 obtain new  $\mathbf{w}$  by re-optimizing the model (2)-(5)
12 while column with negative reduced cost exists;
```

estimator
inference



estimator
training



Estimator

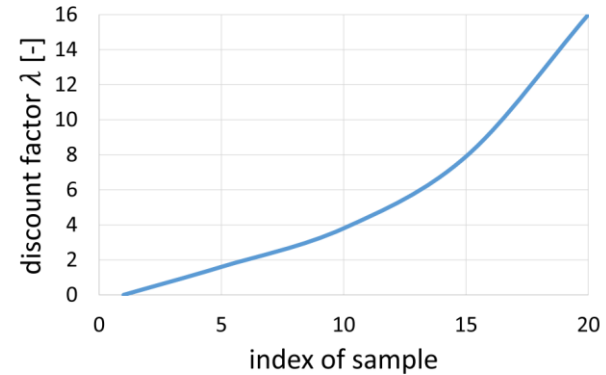
$$\min_{\mathbf{w}, \mathbf{r}} L(\mathcal{D}) = \sum_{i \in \mathcal{D}} \lambda_i [c_i^- r_i^- + c_i^+ \max\{r_i^+ - \epsilon, 0\}]$$

subject to

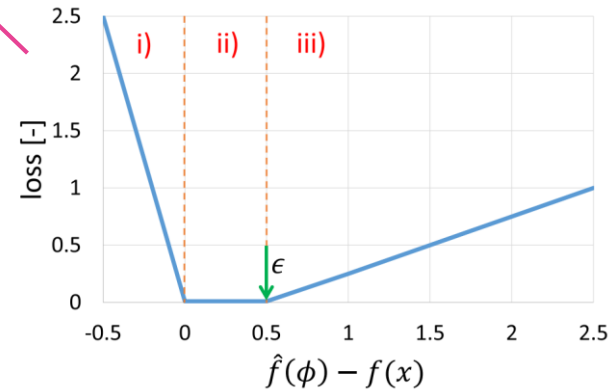
$$\forall i \in \mathcal{D} : \mathbf{w}^T \boldsymbol{\phi}^{(i)} - r_i^+ + r_i^- = f(\mathbf{x}^{(i)})$$

$$\forall i \in \mathcal{D} : r_i^+, r_i^- \geq 0$$

$$\mathbf{w} \in \mathbb{R}^k.$$

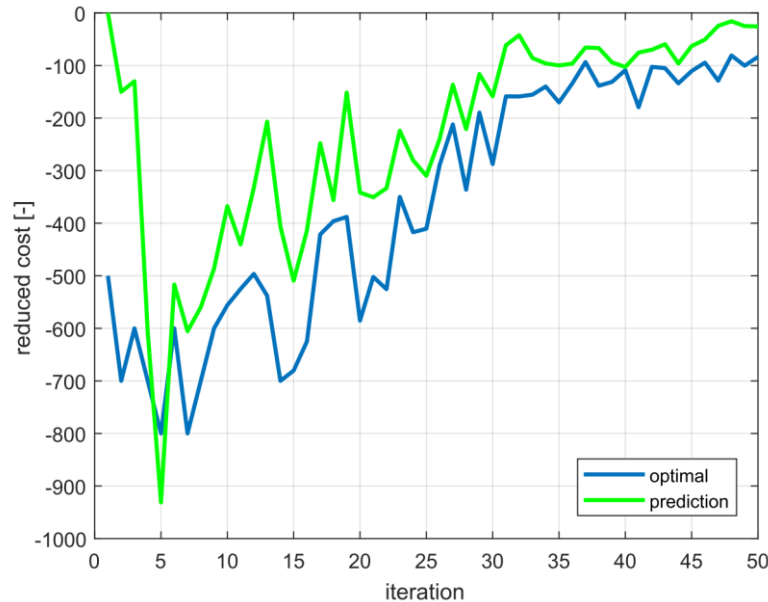


Discounting function for previous samples



Loss function for a single sample

Online learning



Upper bound prediction (of reduced cost) in the root node

- An improvement was also observed on **child nodes** having less iterations than the root node thanks to the reuse of data samples from the superior node.

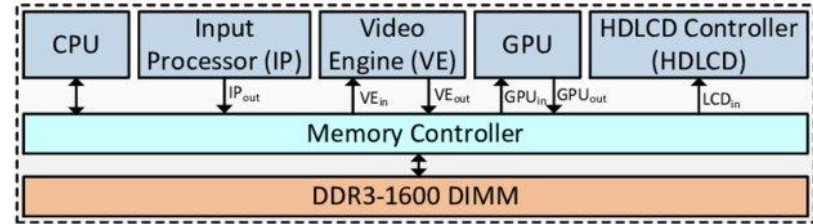
Experimental Results

- The estimator was tested on two problem types

1) Nurse Rostering Problem (NRP)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Nurse 1	N			D	D	D			N	N	N	N			D	D	D			N	N	N			D	D	D	
Nurse 2			D	D	D			N	N	N	N				D	D	D			N	N	N			D	D	D	
Nurse 3	D			N	N	N				D	D	D			N	N	N			D	D	D			D	D	N	
Nurse 4		N	N	N			D	D	D			N	N	N			D	D	D	D				N	N	N	N	
Nurse 5	N	N	N			D	D	D	D			N	N	N			D	D	D				N	N	N			N
Nurse 6	D	D	D			N	N	N				D	D	D	D			N	N	N				D	D	D	D	
Nurse 7		N	N	N			D	D	D			N	N	N			D	D	D				N	N	N			D
Nurse 8	D	D	D			N	N	N			D	D	D			N	N	N				D	D	D	N			
Nurse 9	N	N			D	D	D			N	N	N			N	N	N					D	D	D			N	N
Nurse 10	D	D			N	N	N			D	D	D			N	N	N					D	D	D			N	N

2) Time-Division Multiplexing scheduling (TDM)



- The estimator reduced the entire CPU of the branch and price by:
 - 40%** in the case of the NRP
 - 22%** in the case of the TDM

Conclusions

- **two ML-based method** to speedup branch and price
- exploit advantages of the **decomposition** and preserves optimality
- efficient way to generate **training data** (one instance = multiple samples)
- method was applied to a **practical** planning and scheduling problems

Related papers:

- *P. Koutecká, P. Šůcha, J. Hula, B. Maenhout: A machine learning approach to rank pricing problems in branch-and-price. Eur. J. Oper. Res. 320(2): 328-342 (2025)*
- *R. Václavík, A. Novák, P. Šůcha, Zdenek Hanzálek: Accelerating the Branch-and-Price Algorithm Using Machine Learning. Eur. J. Oper. Res. 271(3): 1055-1069 (2018)*

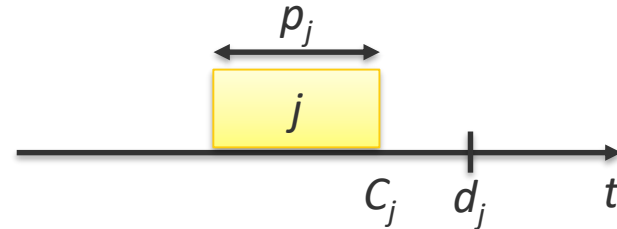
Lawler's decomposition and ML

Related paper:

M. Bouška, P. Šůcha, A. Novák, Z. Hanzálek: Deep learning-driven scheduling algorithm for a single machine problem minimizing the total tardiness. Eur. J. Oper. Res. 308(3): 990-1006 (2023)

Scheduling Problem 1 | $|\Sigma T_j$

- the problem is given by a set of n jobs $J=\{1,\dots,n\}$
- each job $j \in J$ is defined using two non-negative integer parameters:
 - **processing time** p_j , and
 - **due date** d_j .

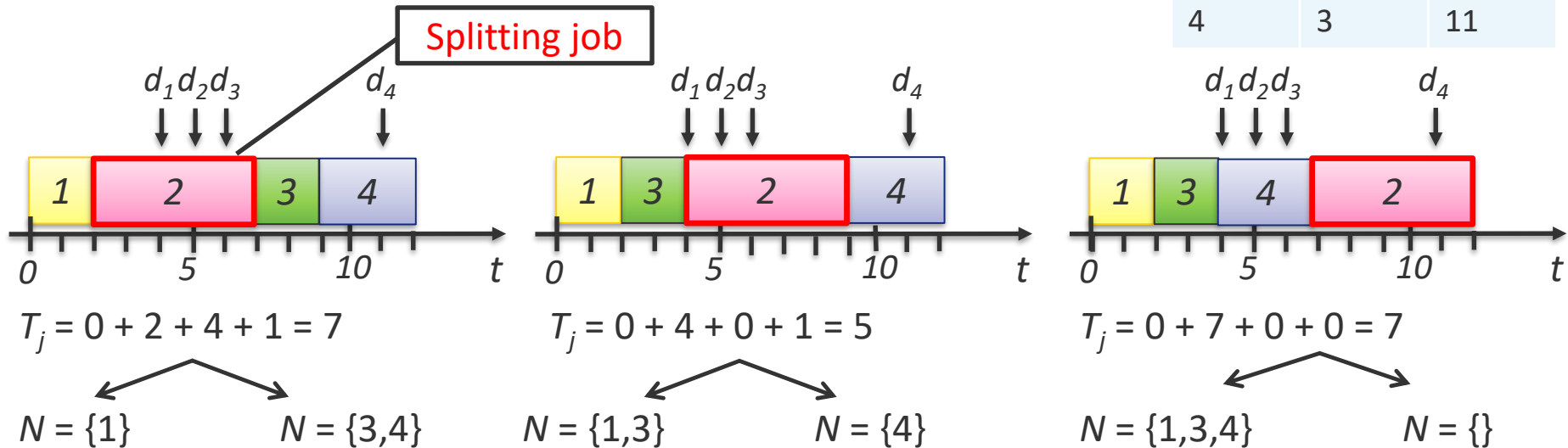


- a solution is a **schedule** - assignment of the jobs to the start times (no overlap, no preemption)
- violation of due date is penalized by **tardiness** $T_j = \max(0, C_j - d_j)$
- the goal is to find a schedule minimizing ΣT_j
- the problem is *NP*-hard.

Lawler's Decomposition for $1 || \sum T_j$

- for this problem we based our approach on **Lawler's decomposition** (EDD) and decomposition proposed by **Della Croce et al.** (SPT)
- an example** Lawler's decomposition assuming 4 jobs, i.e., $N=\{1,2,3,4\}$:

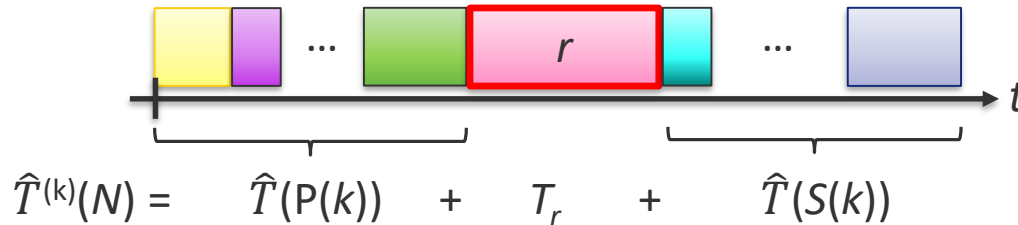
Job j	p_j	d_j
1	2	4
2	5	5
3	2	6
4	3	11



- Eugene L. Lawler, A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness, Annals of Discrete Mathematics, Volume 1, 1977, Pages 331-342.
- Della Croce, F., Tadei, R., Baracco, P., & Grosso, A. A new decomposition approach for the single machine total tardiness scheduling problem. Journal of the Operational Research Society, 49(10), 1101-1106, 1998.

The Solution - Idea

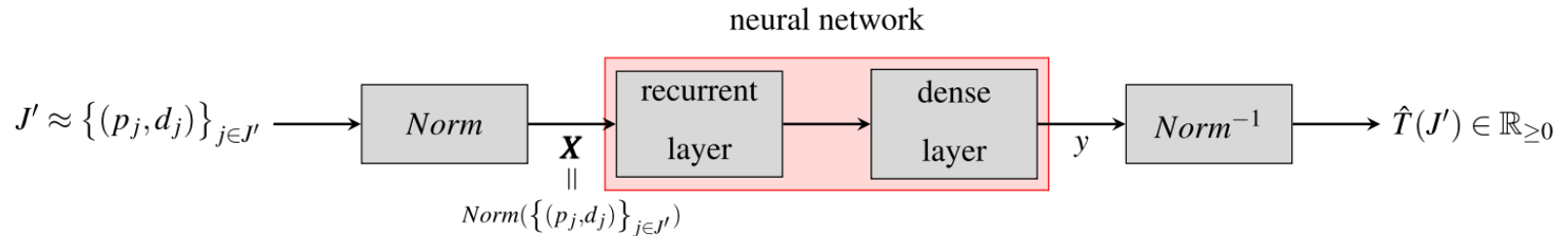
- instead of enumerating all possibilities, we employed ML to **estimate the optimal position** of the splitting job r in the decomposition (heuristic solution)
- estimating the position can be hard, therefore we use a different strategy:
- for every position k of the splitting job we estimate the total tardiness as:



- estimating \hat{T} for $P(k)$ and $S(k)$ is not easy:
 1. varying sizes of $P(k)$ and $S(k)$
 2. combinatorial problem (difficult to approximate, training data)

The Estimation of the Objective Function Value

1. estimation \hat{T} is obtained by a **recurrent neural network (LSTM)**



2. properties leading to **successful design**:

- instead of estimating total tardiness, we estimate **relative deviation** to EDD solution,
- generation of **training data set** using Lawler's decomposition
- ...

The Algorithm

Function Algorithm(J): **Input:** a set of jobs J , **Output:** schedule π

1. Small instances are solved using an exact approach.
2. Determines the splitting job (r^{EDD}, r^{SPT}) and the position sets (K^{EDD}, K^{SPT}) for both decompositions (assuming filtering rules)
3. Select the decomposition with smaller position set, i.e.,
if $|K^{EDD}| \leq |K^{SPT}|$ then $r = r^{EDD}$, $K = K^{EDD}$ else $r = r^{SPT}$, $K = K^{SPT}$
4. Determine the position k^* in K of the splitting job r using the estimator such that
$$k^* = \arg_{k \in K} \min (\hat{T}(P(k)) + T_r + \hat{T}(S(k)))$$
5. Recursively call the Algorithm both subproblems, i.e.,
$$\pi_p = \text{Algorithm}(P(k^*)); \quad \pi_s = \text{Algorithm}(S(k^*))$$
6. Return joined schedule $\pi = \{\pi_p, r, \pi_s\}$

Training of the Estimator

- Training instances were generated utilizing the standard benchmark generator (Potts & Van Wassenhove, 1991) for fixed rdd and tf
- Training data:
 - For each $n \in [75, 100]$ we generated 20 instances
 - Using decompositions we generated 1.6×10^6 samples (optimal solutions)
 - Generation of samples took 600 seconds (decomposition based generation is **20 times faster**),
 - **Training** took 3 hours

Potts, C., & Van Wassenhove, L. N. (1991). Single machine tardiness sequencing heuristics. IIE Transactions, 23(4), 346–354.

Comparison with SotA

Exact approach

n	Garraffa et al. (2018) [*]		Our approach	
	<i>CPU time [s]</i>	<i>gap [%]</i>	<i>CPU time [s]</i>	<i>gap [%]</i>
100-145	0.34	0	0.46	0.39
200-245	11.09	0.07	1.56	0.45
300-345	15.00	1.02	3.07	0.31
400-450	15.00	2.33	5.01	0.24
...				
750-795	15.00	3.75	13.95	0.11

^{*} Time limit set to 15s

GA

n	Süer et al. (2012) ^{**}		Our approach	
	<i>CPU time [s]</i>	<i>gap [%]</i>	<i>CPU time [s]</i>	<i>gap [%]</i>
10	2.00	0	0	0
20	51.00	0	0.06	0
30	354.00	0	0.013	0.1
50	536.00	2.12	0.018	0.006
100	1083.00	6.32	0.044	0.002

^{**} CPU time was rescaled w.r.t. power of their and our CPU

- Garraffa, M., Shang, L., Della Croce, F., & T'Kindt, V. (2018). An exact exponential branch-and-merge algorithm for the single machine total tardiness problem. *Theoretical Computer Science*, 745, 133–149.
- Süer, G. A., Yang, X., Alhawari, O. I., Santos, J., & Vazquez, R. (2012). A genetic algorithm approach for minimizing total tardiness in single machine scheduling. *International Journal of Industrial Engineering and Management*, 3(3), 163–171.

Conclusions

- a **synergy** between the state-of-the-art OR methods and our NN
- an efficient way to **generate the training data set**
- NN is able to **generalize** the acquired knowledge (training on 100 jobs max)
- our approach provides **near-optimal solutions** very quickly

M. Bouška, P. Šůcha, A. Novák, Z. Hanzálek: Deep learning-driven scheduling algorithm for a single machine problem minimizing the total tardiness. Eur. J. Oper. Res. 308(3): 990-1006 (2023)

Conclusion

Use of ML to solve Combinatorial Problems

- In general, **current ML is not strong enough** to approximate combinatorial problems
- **End-to-end approaches** ignore fundamental properties of scheduling problems
- Scheduling domain should **not ignore** SotA in ML and vice versa
- Decomposition techniques allow many interesting opportunities to apply ML

- Related papers:
 - *R. Václavík, P. Šůcha, Z. Hanzálek: Roster evaluation based on classifiers for the nurse rostering problem. J. Heuristics 22(5): 667-697 (2016)*
 - *R. Václavík, A. Novák, P. Šůcha, Zdenek Hanzálek: Accelerating the Branch-and-Price Algorithm Using Machine Learning. Eur. J. Oper. Res. 271(3): 1055-1069 (2018)*
 - *M. Bouška, P. Šůcha, A. Novák, Z. Hanzálek: Deep learning-driven scheduling algorithm for a single machine problem minimizing the total tardiness. Eur. J. Oper. Res. 308(3): 990-1006 (2023)*
 - *P. Koutecká, P. Šůcha, J. Hula, B. Maenhout: A machine learning approach to rank pricing problems in branch-and-price. Eur. J. Oper. Res. 320(2): 328-342 (2025)*