Solving scheduling problems with Constraint Programming and Graph Neural Networks
ANITI Days – 16-17 November 2023

Guillaume Povéda, Florent Teichteil-Koenigsbuch
Cutting-edge algorithms to optimize the nominal manufacturing workflow

- Multi-skill worker allocation
- Versatile time constraint modeling between tasks
- Non-renewable and renewable resource sharing

Deep learning rescheduling strategies for fast disruption management & adaptation

- Pre-trained rescheduling policy
- Nearly instantaneous adaptation of current schedule to disruption events
- Possibility to improve adapted schedule as time allows

Massive uncertain multi-scenario analysis & robust decision-making

- Fast scenario evaluation and prediction by inferring the deep learning rescheduling policy
- Robust continuous scheduling based on real-time statistical analysis and aggregation of multiple scenarii and criteria
Part I
Large-Scale Scheduling with Limited Resources and Complex Relations in Practice
Scheduling in industry

Problems usually look like classical JSP/RCPSP but with **additional** constraints: preemption of tasks, calendar shifts and multiskill workforce.

**Job shop scheduling problem**
- Unitary resource
- Precedence constraint between subpart of jobs

**Flexible shop scheduling problem**
- Job shop + flexibility of resource allocation for some jobs.

**Resource constrained project scheduling problem (RCPSP)**
- Job shop with “cumulative” resource (≠1), modeling workforce, tool, area, energy..

**Multi-skill RCPSP**
- Adding worker skills modelisation + skills requirements to each activities: increasing realism of real problems.
Limitation of academical scheduling problems

Jobshop and RCPSP fail to model (multi) skills individual workers
1. Preemption of tasks is usually impossible in such optimisation models
2. Generalized precedence constraints are usually not considered in classical model

Contributions:
- Efficient constraint programming modeling
- Generic Large Neighborhood Search
- Capitalisation in open-source libraries

More:
Partially Preemptive Multi Skill/Mode Resource-constrained Project Scheduling with Generalized Precedence Relations and Calendars, CP2023, Povéda, Alvarez, Artigues
Constraint programming for complex scheduling

CPMODEL
array[1..9, 1..9] of var 1..9: grid;
forall i in row:
    allDifferent(grid[i,:]);
forall i in column:
    allDifferent(grid[:,i]);

...
Algorithm 1 Generic Large Neighborhood Search Algorithm

Begin
1: \( Y^* = \infty, X^* = \text{None} \)
2: \((X^0, Y^0) = (X^*, Y^*) = \text{initial solution}(P)\)
3: \( \text{iter} = 0 \)
4: repeat
5: \( \text{RMP} = \text{buildsubproblem}(\text{MP}, X^\text{iter}) \)
6: \( X^{\text{iter+1}}, Y^{\text{iter+1}} = \text{solve}(\text{RMP}) \)
7: if \( Y^{\text{iter+1}} \leq Y^* \) then
8: \( X^* \leftarrow X^{\text{iter+1}} \)
9: \( Y^* \leftarrow Y^{\text{iter+1}} \)
10: \( \text{iter} \leftarrow \text{iter} + 1 \)
11: until stop criterion is met
12: return \( X^*, Y^* \)
Performance of LNS solver with different subproblem methods have been tested:
- random and cut methods taken individually
- portfolio of previous methods (called Mixing in the results table)

Mixing method achieved the most consistent performance (best or second best results) on our few testing instance
One library to **capitalize**/benchmark different solving methods for discrete optimisation problems.

**Easy example of use:**

```python
crcsp_problem = parse(file)
results = solve(rcrpsp_problem, solver=CPSolver)
```

Now used in 3 publications around scheduling:

- "An Empirical Evaluation of Permutation-Based Policies for Stochastic RCPSP", Olivier Regnier-Coudert, Guillaume Povéda, GECCO 2021

- "Fast and Robust Resource-Constrained Scheduling with Graph Neural Networks"  Teichteil-Königsbuch, F., Povéda, G., González de Garibay Barba, G., Luchterhand, T., & Thiébaux, S., ICAPS 2023

- ‘Partially Preemptive Multi Skill/Mode Resource-constrained Project Scheduling with Generalized Precedence Relations and Calendars’, Povéda, Alvarez, Artigues, CP2023,

[https://github.com/airbus/discrete-optimization](https://github.com/airbus/discrete-optimization)
[https://github.com/airbus/scikit-decide](https://github.com/airbus/scikit-decide)
Main interest:
1) Benchmark solvers on the same problem but from different communities (LP, CP, Metaheuristics, soon ML)
2) Combine easily solvers in some more complex pipeline (→ such as the LNS we describe)
3) Educational purpose for combinatorial optimization introduction

Main problem implemented:
Workforce allocation problems, routing, scheduling (JSP, RCPSP and variants...).

Example of solvers binded:
https://github.com/airbus/discrete-optimization
https://github.com/airbus/scikit-decide
Part II
Frugal Learning of Deep Learning Scheduling Heuristics
With the help of model-based solvers
RCPSP represented as a Graph (Neural Network)

Cumulative (special case: non-overlapping)

If \( RTasks \neq {} \) and \( \sum_{r} rr[r] > rc[r] \) then:

\[
\text{cumulative}(\{ s[i] | i \in RTasks \}, \{ d[i] | i \in RTasks \}, \{ rr[r, i] | i \in RTasks \}, rc[r])
\]

Currently using TransformerConv as NN layers:

- Task node encoding: \([0, 1, 0, \text{duration}]\)
- Resource node encoding: \([1, 0, \#\text{resources}, 0]\)
- Precedence edge encoding: \([1, 0, 0, 0]\)
- Resource edge encoding: \([0, 1, 0, \#\text{consumed}]\)

\[
x'_i = W_1x_i + \sum_{j \in N(i)} \alpha_{i,j} (W_2x_j + W_6e_{ij})
\]

where the attention coefficients \( \alpha_{i,j} \) are now computed via:

\[
\alpha_{i,j} = \text{softmax} \left( \frac{(W_3x_i)^\top (W_4x_j + W_6e_{ij})}{\sqrt{d}} \right)
\]
Hybridizing CP+GNN: our SIREN training algorithm (80% of 2040 RCPSP instances from PSPLIB)

Don't learn to mimic the CP solver but learn to directly produce schedules with a Graph Neural Network structure specific to all RCPSP problems
Testing phase: our SIREN inference algorithm

Idea 1 (not working well)

Idea 2 (working well) SIREN

Testing phase: our SIREN inference algorithm

Idea 1 (not working well)

Idea 2 (working well) SIREN

SGS is way faster than CP!
Protocol: evaluate vanilla CP solver time to get same quality solution as GNN+SGS solver, then compare with GNN+SGS solver time

- In more than **82%** of problems CP-SAT takes more time than SIREN to achieve a solution of comparable quality.
- In over **40%** of the cases, CP-SAT’s computational overhead ranges from 10 times up to over 20,000 times the computation time of SIREN.
3 heuristics: DUM, MDPR, CCPM are all using SGS with a different task ordering

- **DUM**: \([1, 2 \ldots N]\) : order by index of task
- **MDPR**: Order by maximum of descendants in the precedence graph.
- **CCPM**: Order using critical path method outputs.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>relative mean</th>
<th>std</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUM</td>
<td>12.07</td>
<td>10.12</td>
<td>0.0</td>
<td>11.81</td>
<td>20.46</td>
<td>36.79</td>
</tr>
<tr>
<td>MDPR</td>
<td>7.72</td>
<td>7.22</td>
<td>0.0</td>
<td>6.59</td>
<td>12.70</td>
<td>36.29</td>
</tr>
<tr>
<td>CCPM</td>
<td>6.21</td>
<td>7.34</td>
<td>0.0</td>
<td>2.21</td>
<td>11.87</td>
<td>30.95</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td><strong>4.16</strong></td>
<td><strong>6.45</strong></td>
<td><strong>0.0</strong></td>
<td><strong>0.93</strong></td>
<td><strong>5.73</strong></td>
<td><strong>30.32</strong></td>
</tr>
</tbody>
</table>

Table 2: Statistics of relative overcost compared to best CP solutions on the 408 test instances. (% are for percentiles)

*Using ResTransformer with 256 hidden neurons and 50000 epochs*
Final words: thank you ANITI-1.0!

- Innovative methods for solving scheduling problems inspired by Airbus manufacturing applications
  - ICAPS-23 paper: Hybrid DL/CP
  - CP-23 paper: LNS/CP

- Get-To-Know and to work together 🤝
  - TUPLES project
  - ANITI-2 HEROIC chair proposal

- Discussions
  - Seminars
  - Social activities ☕️

- Cross-Fertilization
  - Use Cases
  - Methods

- Knowledge Exchange
  - ANITI Knowledge Compilation Chair
  - Building Trust and Long-Term Relationships
  - Scientific Collaboration