A Constraint Programming Approach to Ship Refit Project Scheduling

Raphaël Boudreault, Thales Digital Solutions (Québec, Canada)
Vanessa Simard, NQB.ai (Québec, Canada)
Daniel Lafond, Thales Digital Solutions (Québec, Canada)
Claude-Guy Quimper, Université Laval (Québec, Canada)

CP 2022
August 2, 2022
What is a ship refit?

- Important shipyard event where all ship’s activities are suspended
- Objective is to restore, customize, modify or modernize part of a ship
- Made of several hundred (or thousand) tasks
- Can span over several weeks, months (or over a year)
- Longer refit = higher costs
- Time window must be planned years in advance
- When exceeded, the dock must be cleared
Ship refit planning

- Complex and tedious
- Initial planning (free of conflicts) can take up to 120 days
- Day-to-day re-planning is difficult and time-consuming
- Typical software (Primavera P6, Microsoft Project) have limited optimization capabilities (not exact, only resources leveling, etc.)

Refit Optimizer

- Prototype solution for multi-objective optimization in the ship refit domain
- Generic architecture for other scheduling contexts
- Key motivation: Challenges identified in the Arctic and Offshore Patrol Ship and Joint Support Ship In-Service Support (AJISS) program with the Royal Canadian Navy
Operational and deployed on a secured cloud platform (Thales TrustNest)
Elements to consider

- Planning **horizon**
- Planning **granularity** (days or hours)
- Tasks depend on **capacity-limited resources** (human/material)
- Maximum number of workers simultaneously in some **work areas**
- **Precedence relationships** between tasks
- **Date constraints** (e.g. milestones)
- Some tasks must be **idle during weekends**
- Some tasks can be performed in **overtime**

Objectives

1. Minimize the refit total duration (**makespan**)
2. Minimize the costs associated with overtime labor (**overtime**)
3. Minimize the risk, planning the overtime as early as possible (**robustness**)

Problem description
<table>
<thead>
<tr>
<th>Instance</th>
<th>Horizon</th>
<th>#Tasks (#O)</th>
<th>Task duration</th>
<th>#Precedence relations</th>
<th>#Resources (#WA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>day-yacht21</td>
<td>29 days</td>
<td>21 (20)</td>
<td>1-3 days</td>
<td>32</td>
<td>9 (2)</td>
</tr>
<tr>
<td>hour-yacht21</td>
<td>704 hours</td>
<td>21 (20)</td>
<td>1-8 hours</td>
<td>32</td>
<td>9 (2)</td>
</tr>
<tr>
<td>generic136</td>
<td>178 days</td>
<td>136 (136*)</td>
<td>1-20 days</td>
<td>99</td>
<td>9 (4)</td>
</tr>
<tr>
<td>software138</td>
<td>183 days</td>
<td>138 (138*)</td>
<td>1-10 days</td>
<td>341</td>
<td>8 (0)</td>
</tr>
<tr>
<td>navy253</td>
<td>728 hours</td>
<td>253 (253*)</td>
<td>1-8 hours</td>
<td>246</td>
<td>92 (87)</td>
</tr>
<tr>
<td>cruise510</td>
<td>268 days</td>
<td>510 (464*)</td>
<td>1-15 days</td>
<td>550</td>
<td>32 (24)</td>
</tr>
<tr>
<td>navy830</td>
<td>6200 hours</td>
<td>830 (830*)</td>
<td>1-200 hours</td>
<td>816</td>
<td>146 (128)</td>
</tr>
</tbody>
</table>

*Must be idle during weekends

Can be performed in overtime

Artificial

Realistic

Work areas
**Standard definition (Pritsker et al., 1969)**

- **Set of tasks** $\mathcal{I}$
- **Timeline** $\mathcal{T} = \{0, 1, \ldots, t_m\}$, horizon $t_m$

- **Set of resources** $\mathcal{R}$
- Task $i \in \mathcal{I}$ **requires** $h_{i,r}$ of resource $r \in \mathcal{R}$, for its whole duration

- **Each resource** $r \in \mathcal{R}$:
  - **Capacity** $c_r$
  - **Renewable** (fully available at all time)
  - **Cumulative** (more than one task can use a resource at a time)

- **Set of precedence relationships** $\mathcal{P}$

- **Objective**: Find a schedule with the **minimal makespan**
Significant efforts in the CP community to solve scheduling problems with resources

Cumulative global constraint (Aggoun & Beldiceannu, 1993)

$$\sum_{s_i \leq t < s_i + d_i} h_{i,r} \leq c_r \quad \forall t \in T$$

Usage of a resource is at most its capacity for each time point in the timeline

Many filtering rules developed and improved
- Time-Tabling
- Time-Table Edge Finding (TTEF)
- Energetic Reasoning...

Important progress towards solving large-scale RCPSP (Schutt et al., 2011, 2013)

Lazy clause generation (Ohrimenko et al., 2009)
- Hybrid between CP and SAT solvers
- Filtered values recorded with explanations as SAT clauses
- On a failure, learns a nogood
- Solvers: Chuffed, OR-Tools (Google), CP Optimizer (IBM)
- SAT-based branching heuristic: Variable State Independent Decaying Sum (VSIDS)
**Additional parameters**

- $s^u_i, s^l_i, e^u_i, e^l_i$, bounds on start/end times implied by **date constraints**
- $p_i$, processing time (**task duration without overtime**)
- $w^s_r, w^o_r$, **daily standard/overtime usage cost** of resource $r$ ($w^s_r \leq w^o_r$)
- A working day schedule:
  
  - Standard $d_s$  
  - Overtime $d_o$  
  - Total $d_E$

- Set of tasks that can be planned with overtime $\mathcal{I}^* \subset \mathcal{I}$

**Decision variables**

- For each task $i \in \mathcal{I}$
  - **Starting time** $S_i \in [s^l_i, s^u_i]$
  - **Elapsed time** $E_i \in \mathcal{T}$
### Constraints

\[
\text{Cumulative}([S_i \mid i \in \mathcal{I}], [E_i \mid i \in \mathcal{I}], [h_{i,r} \mid i \in \mathcal{I}], c_r) \\
\forall r \in \mathcal{R}
\]

\[
e_i^L \leq S_i + E_i \leq e_i^U \\
\forall i \in \mathcal{I}
\]

\[
S_i + E_i + l \leq S_j \\
\forall (i, j, l) \in \mathcal{P}
\]

\[
E_i = p_i \\
\forall i \in \mathcal{I} \setminus \mathcal{I}^*
\]

\[
\left\lfloor \frac{(d^O - d^S) p_i}{d^E - d^S} \right\rfloor \leq E_i \leq p_i \\
\forall i \in \mathcal{I}^*
\]

### Parameters

- **Standard day**: 8 to 16
- **Overtime day**: 12 to 20

\[
\left\lfloor \frac{8 \times 3}{12} \right\rfloor \leq E_i \leq 3
\]
Objectives

1. **Makespan**

   $$\min \max_{i \in I} (S_i + E_i)$$

   $I^* = \emptyset$

2. **Overtime**

   **Overtime days**

   $$\min \sum_{i \in I^*} (p_i - E_i) \left( \sum_{r \in R} h_{i,r} (w_r^O - w_r^S) \right)$$

3. **Robustness**

   $$\min \sum_{i \in I^*} (p_i - E_i) S_i$$
More types of precedence constraints

\[ X_i + l \leq Y_j \]

Suspension of some tasks during weekends
- Additional variables \( N_i \), non-working (idle) time points
- Included in the elapsed time with specific constraints

Support of planning granularity in hours
- Additional variable \( O_i \), overtime time points
- Constraints for relation with \( N_i \), which includes nights
- Elapsed time is replaced by \( p_i + N_i \)
**Baseline strategy**

- **Makespan**
  
  $S_i$ with smallest value in domain, assigned to that value

  **Focus:** Scheduling as early as possible

- **Overtime and robustness**
  
  1. Choose $i$ such that $S_i$ has smallest value in domain
  2. Assign smallest value to $S_i$
  3. Assign greatest value to $E_i$

  **Focus:** Scheduling as early as possible with as few overtime as possible

*Formulated as a priority search in MiniZinc*
SBPS strategy

- Uses a simple and efficient value selection heuristic
  - Best-Solution (Vion and Piechowiak, 2017)
  - Solution-Based Phase Saving (SBPS) (Demirović et al., 2018)

If $b$ is the value of $X$ in the current best solution, when branching on $X$:
- If $b$ is in domain of $X$, choose $b$
- Else, use a fallback heuristic

- Combined with a restart strategy and a dynamic variable selection heuristic, effectively mimics a Large Neighborhood Search (LNS), without loss of exactness

- We use BASELINE until a first solution

- Then, use SBPS with conflict activity (VSIDS) variable selection and BASELINE as fallback
**Setup**

- Modeled with *MiniZinc*
- SBPS scheme implemented in *Chuffed CP* solver, that we used
- *Cumulative* set to use TTEF checking and filtering
- Timeout: 4 hours
- Constant restart strategy of 100 failures

**Experiments**

- Each instance, each objective, each strategy
- **Overtime/Robustness**: restricted horizon between 2-30% of the best known makespan
  - Not *generic*136, due to special structure
**Table 3** Results on the benchmark instances when considering the makespan objective.

<table>
<thead>
<tr>
<th>Instance</th>
<th><strong>Baseline</strong></th>
<th></th>
<th></th>
<th><strong>SBPS</strong></th>
<th></th>
<th>Time (s)</th>
<th>Time (s)</th>
<th>improv.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Objective</td>
<td>Time (s)</td>
<td></td>
<td>Objective</td>
<td>Time (s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>day-yacht21</td>
<td>28 days</td>
<td>0.2*</td>
<td></td>
<td>28 days</td>
<td>0.2*</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hour-yacht21</td>
<td>78 hours</td>
<td>0.4*</td>
<td></td>
<td>78 hours</td>
<td>0.4*</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>generic136</td>
<td>178 days</td>
<td>0.7*</td>
<td></td>
<td>178 days</td>
<td>0.7*</td>
<td>0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>software138</td>
<td>144 days</td>
<td>1.4</td>
<td></td>
<td>119 days</td>
<td>41.6</td>
<td>1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy253</td>
<td>389 hours</td>
<td>4.2</td>
<td></td>
<td>389 hours</td>
<td>3.7</td>
<td>3.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cruise510</td>
<td>228 days</td>
<td>14.7</td>
<td></td>
<td>227 days</td>
<td>785.7</td>
<td>229.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy830</td>
<td>5216 hours</td>
<td>18.7</td>
<td></td>
<td>5144 hours</td>
<td>199.7</td>
<td>18.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Best makespan reduced by 5% on average*
Table 4: Results on the benchmark instances when considering the *over time* objective.

<table>
<thead>
<tr>
<th>Instance</th>
<th>\textit{Baseline}</th>
<th></th>
<th></th>
<th>\textit{SBPS}</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Objective</td>
<td>Time (s)</td>
<td>Objective</td>
<td>Time (s)</td>
<td>Time (s)</td>
<td>improv.</td>
<td></td>
</tr>
<tr>
<td>day-yacht21</td>
<td>1560</td>
<td>0.3*</td>
<td>1560</td>
<td>0.3*</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hour-yacht21</td>
<td>485</td>
<td>0.4*</td>
<td>485</td>
<td>0.4*</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>software138</td>
<td>5600</td>
<td>14359.6</td>
<td>2600</td>
<td>153.4</td>
<td>34.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy253</td>
<td>70</td>
<td>4.2</td>
<td>66</td>
<td>5.0</td>
<td>4.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cruise510</td>
<td>26000</td>
<td>11.7</td>
<td>15760</td>
<td>7555.3</td>
<td>5.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy830</td>
<td>227</td>
<td>25.2</td>
<td>36</td>
<td>276.5</td>
<td>26.6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best cost reduced by 48% on average

Table 5: Results on the benchmark instances when considering the *robustness* objective.

<table>
<thead>
<tr>
<th>Instance</th>
<th>\textit{Baseline}</th>
<th></th>
<th></th>
<th>\textit{SBPS}</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Objective</td>
<td>Time (s)</td>
<td>Objective</td>
<td>Time (s)</td>
<td>Time (s)</td>
<td>improv.</td>
<td></td>
</tr>
<tr>
<td>day-yacht21</td>
<td>47</td>
<td>0.3*</td>
<td>47</td>
<td>0.3*</td>
<td>0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hour-yacht21</td>
<td>192</td>
<td>0.4*</td>
<td>192</td>
<td>0.4*</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>software138</td>
<td>900</td>
<td>13571.8</td>
<td>258</td>
<td>320.2</td>
<td>15.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy253</td>
<td>10686</td>
<td>5057.6</td>
<td>3480</td>
<td>1411.9</td>
<td>6.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cruise510</td>
<td>4870</td>
<td>13022.5</td>
<td>842</td>
<td>1321.7</td>
<td>14.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navy830</td>
<td>146794</td>
<td>11208.9</td>
<td>9076</td>
<td>13863.4</td>
<td>41.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best value reduced by 79% on average
Solving time restrictions
- Obtain “good” solutions under 15 min. for < 100 tasks, under 4 hours for > 500 tasks
- In comparison, up to 4 hours to manually “optimize” day-yacht21

Anonymity
- Estimated workforce costs changed to abstract values

Explainability of results
- Input data format, parameter selection, etc.
- Focus on results interpretation and solution selection
- Unsatisfiability during initial planning of real projects
Contributions

- Introduced a CP approach for the ship refit planning problem
- Successfully tested on seven industrial instances
  - Detailed complexity analysis with RCPSP metrics in the paper
  - Three objective functions (makespan, overtime, robustness)
  - Used SBPS value selection to speed-up the search
  - Better solutions found significantly faster than baseline

Next steps

- Complex geospatial constraints and visualization (Lafond et al., 2021)
- Experiments with Mixed-Integer Programming model
- Consider task priority levels
- Further explore simulations for robustness assessment
- Maintenance Optimizer: long-term planning of preventive maintenance over work periods