THALES
Building a future we can all trust

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

www.thalesgroup.com

Context

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

Context

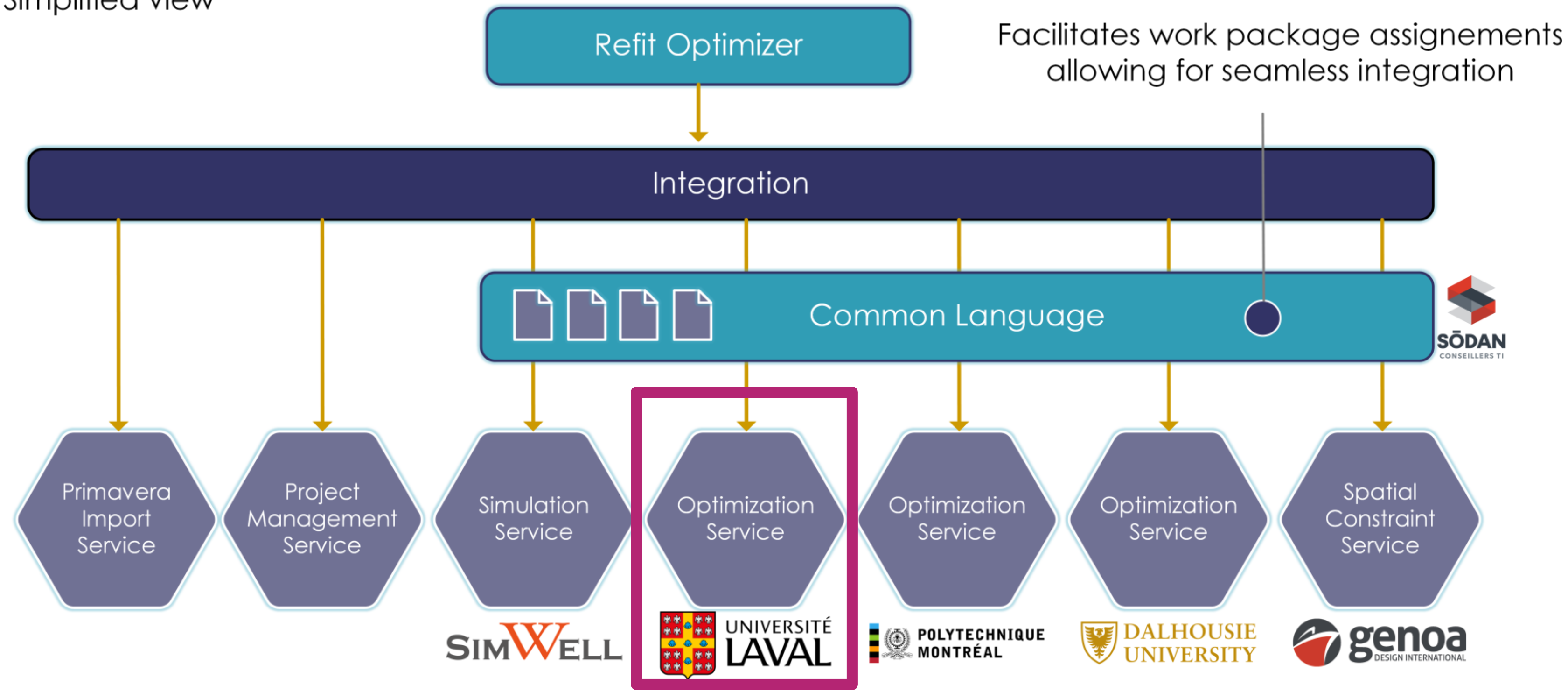
Ship refit planning

- **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

Simplified view



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**)

Instances

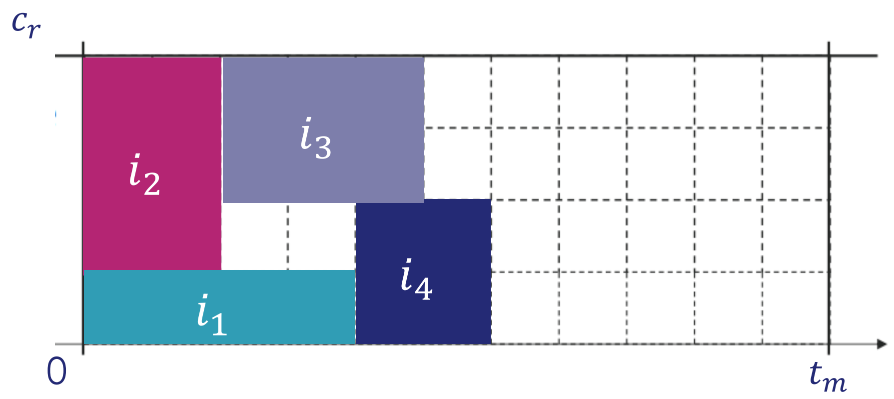
Can be performed in overtime
*Must be idle during weekends

Work areas

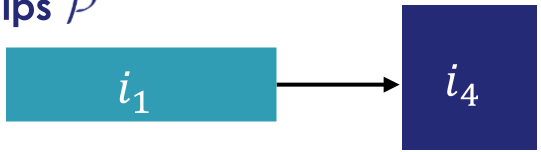
	Instance	Horizon	#Tasks (#O)	Task duration	#Precedence relations	#Resources (#WA)
Artificial	day-yacht21	29 days	21 (20)	1-3 days	32	9 (2)
	hour-yacht21	704 hours	21 (20)	1-8 hours	32	9 (2)
	generic136	178 days	136 (136*)	1-20 days	99	9 (4)
	software138	183 days	138 (138*)	1-10 days	341	8 (0)
Realistic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

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

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

$$\sum_{\substack{i \in \mathcal{I}: \\ S_i \leq t < S_i + D_i}} h_{i,r} \leq c_r \qquad \forall t \in \mathcal{T}.$$

↔ *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**)

Main CP model

Planning granularity in days

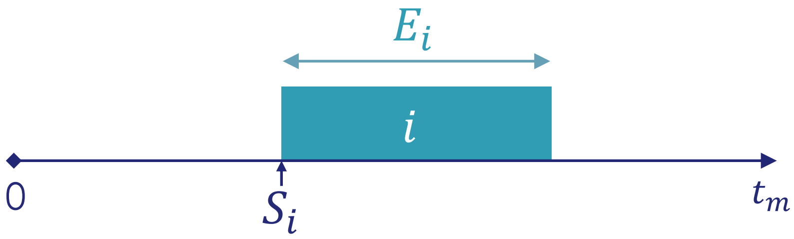
Additional parameters

- $s_i^U, s_i^L, e_i^U, e_i^L$, bounds on start/end times implied by **date constraints**
- p_i, processing time (**task duration without overtime**)
- w_r^S, w_r^O, **daily standard/overtime usage cost** of resource r ($w_r^S \leq w_r^O$)
- A working day schedule:

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

Decision variables

- For each task $i \in \mathcal{I}$
  - **Starting time** $S_i \in [s_i^L, s_i^U]$
  - **Elapsed time** $E_i \in \mathcal{T}$



Constraints

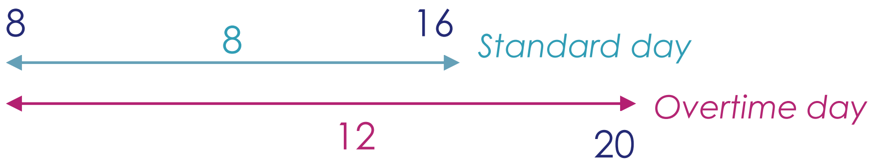
$$\textsc{Cumulative}([S_i \mid i \in \mathcal{I}], [E_i \mid i \in \mathcal{I}], [h_{i,r} \mid i \in \mathcal{I}], c_r) \qquad \forall r \in \mathcal{R}$$

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

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

$$E_i = p_i \qquad \forall i \in \mathcal{I} \setminus \mathcal{I}^*$$

$$\left\lceil \frac{(d^O - d^S) p_i}{d^E - d^S} \right\rceil \leq E_i \leq p_i \qquad \forall i \in \mathcal{I}^*$$



$$\left\lceil \frac{8 * 3}{12} \right\rceil \leq E_i \leq 3$$

Main CP model

Objectives

1. ***Makespan***

$$\min \max_{i \in \mathcal{I}} (S_i + E_i) \qquad \boxed{\mathcal{I}^* = \emptyset}$$

2. ***Overtime***

$$\min \sum_{i \in \mathcal{I}^*} \overbrace{(p_i - E_i)}^{\text{Overtime days}} \left(\sum_{r \in \mathcal{R}} h_{i,r} \overbrace{(w_r^O - w_r^S)}^{\text{Daily overtime cost/resource}} \right)$$

3. ***Robustness***

$$\min \sum_{i \in \mathcal{I}^*} (p_i - E_i) S_i$$

- More types of **precedence constraints**

$$X_i \pm l \leq Y_j$$

- 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 *generic136*, due to special structure

Table 3 Results on the benchmark instances when considering the **makespan** objective.

Instance	BASELINE Objective	BASELINE Time (s)	SBPS Objective	SBPS Time (s)	Time (s) improv.
day-yacht21	**28 days**	0.2*	**28 days**	0.2*	0.2
hour-yacht21	**78 hours**	0.4*	**78 hours**	0.4*	0.4
generic136	**178 days**	0.7*	**178 days**	0.7*	0.7
software138	144 days	1.4	**119 days**	41.6	1.1
navy253	**389 hours**	4.2	**389 hours**	3.7	3.7
cruise510	228 days	14.7	**227 days**	785.7	229.3
navy830	5216 hours	18.7	**5144 hours**	199.7	18.2

Best makespan reduced by 5% on average

Results

Table 4 Results on the benchmark instances when considering the **overtime** objective.

Instance	BASELINE		SBPS		Time (s) improv.
	Objective	Time (s)	Objective	Time (s)	
day-yacht21	**1560**	0.3*	**1560**	0.3*	0.3
hour-yacht21	**485**	0.4*	**485**	0.4*	0.4
software138	5600	14 359.6	**2600**	153.4	34.3
navy253	70	4.2	**66**	5.0	4.0
cruise510	26 000	11.7	**15 760**	7555.3	5.8
navy830	227	25.2	**36**	276.5	26.6

Best cost reduced by 48% on average

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

Instance	BASELINE		SBPS		Time (s) improv.
	Objective	Time (s)	Objective	Time (s)	
day-yacht21	**47**	0.3*	**47**	0.3*	0.3
hour-yacht21	**192**	0.4*	**192**	0.4*	0.4
software138	900	13 571.8	**258**	320.2	15.3
navy253	10 686	5057.6	**3480**	1411.9	6.7
cruise510	4870	13 022.5	**842**	1321.7	14.2
navy830	146 794	11 208.9	**9076**	13 863.4	41.1

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