# Justification of research and implementation

Daan Derks en Ferre van Oort

21-10-2024

## Contents

## Problem definition

- Scheduling problem

- Objective:
  Minimize both setup times (from color switches) and penalties (from late orders) when scheduling car part painting on multiple machines.

- Input:
  - Orders (O): Each order $o_i$ has:
    - Surface Area ($A_i$): Determines processing time based on machine speed.
    - Color ($C_i$): Orders need specific colors; switching colors incurs setup time.
    - Deadline ($D_i$): Time by which the order must be completed.
    - Penalty ($P_i$): Cost per unit of lateness if $t_{end} > D_i$
  - Machines (M):
    Each machine $M_j$ has:
    - Speed ($S_j$): Determines the time required to paint an order.
    - Setup Time ($T_{xy}$): Time to switch from color $C_x$ to $C_y$.

- Solution:
  A schedule S that assigns orders to machines, specifying:
  - Start Time $t_{start}$ , End Time $t_{end}$, and Color for each order.
  - Minimize total penalties and setup times for each machine.
- Constraints:
  - Each machine processes one order at a time, no partial processing allowed.
  - Orders must respect machine availability and deadlines.
- Objective Function: Minimize:

$$Total\ penalty + \alpha * Total\ Setup\ Time$$

Where $\alpha$ is a weighting factor to balance penalties and setup costs.

- Feasible Solution:
  A valid assignment of orders to machines with respect to deadlines, processing times, and color setup constraints.

**Sets**

$M \triangleq$ machines

$O \triangleq$ orders

$H \triangleq$ colours

**Indices**

$m \in M$

$o, o_1, o_2 \in O$

$h, h_1, h_2 \in H$

**Parameters** (FOLLOW FROM EXCEL INPUT)

$s_o \triangleq$ surface for order $o \in O$

$colour_o \triangleq$ colour for order $o \in O$, $colour_o \in O$

$d_o \triangleq$ deadline for order $o \in O$

$c_o \triangleq$ penalty cost for order $o \in O$

$v_m \triangleq$ speed of machine $m$

$\overrightarrow{t^{colour}_{h_1,h_2}} \triangleq$ set-up time from colour $h_1 \in H$ to $h_2 \in H$

Derived input

$t^{order}_{o_1,o_2} \triangleq$ set-up time $o_1 \to o_2$; $= t^{colour}_{colour_{o_1}, colour_{o_2}}$

$p_{om} \triangleq$ processing time $o$ on $m$

$= s_o / v_m$

**Small instance** $H = \{green, yellow, blue\}$

| $o$ | $s_o$ | $colour_o$ | $d_o$ | $c_o$ |
|---|---|---|---|---|
| O1 | 150 | green | 18 | 10 |
| O2 | 200 | yellow | 28 | 12 |
| O3 | 180 | blue | 12 | 8 |

$O$

| $m$ | $v_m$ |
|---|---|
| M1 | 20 |
| M2 | 25 |

$M$

| $h_1$ \ $h_2$: | green | yellow | blue |
|---|---|---|---|
| green | 0 | 5 | 3 |
| yellow | 2 | 0 | 11 |
| blue | 4 | 10 | 0 |

$= t^{colour}_{h_1,h_2}$

$H$

| $o_1$ | $o_2$ | $t^{order}_{o_1,o_2}$ | |
|---|---|---|---|
| O1 | O2 | 5 | green → yellow |
| O1 | O3 | 3 | green → blue |
| O2 | O1 | 2 | yellow → green |
| O2 | O3 | 11 | yellow → blue |
| O3 | O1 | 4 | blue → green |
| O3 | O2 | 10 | blue → yellow |

| $o$ | $m$ | $p_{o,m}$ | |
|---|---|---|---|
| O1 | M1 | 7.5 | 150/20 |
| O1 | M2 | 6 | 150/25 |
| O2 | M1 | 10 | 200/20 |
| O2 | M2 | 8 | 200/25 |
| O3 | M1 | 9 | 180/20 |
| O3 | M2 | 7,2 | 180/25 |

$O_m \triangleq$ orders assigne to m

Source: (Willemsen, R., 2024)

**Decision variables**

$n_m \triangleq$ number orders executed on $m = |O_m|$

$machine_o \triangleq$ machine on which $o$ is executed $machine_o \in M$.

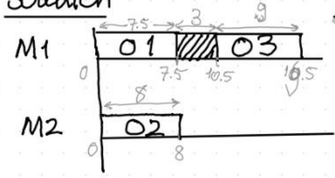
$seqno_o \triangleq$ sequence number of $o \in O$ on $machine_o$

$sequence_m \triangleq$ order in which orders assigned to $m$ are executed

⇝ – "no two orders $o_1, o_2$ assigned to the same machine $m$ have same seqno"

– "all seqno for orders assigned to the same machine have seqno $1, 2, \ldots, n_m$, where $n_m$ is number orders assigned to machine $m$."

**Solution**



| $o$ | $machine_o$ | $seqno_o$ |
|---|---|---|
| O1 | M1 | 1 |
| O2 | M2 | 1 |
| O3 | M1 | 2 |

| $m$ | $sequence_m$: 1 | 2 |
|---|---|---|
| M1 | O1 | O3 |
| M2 | O2 | |

→ derived variables:

$pred_o \triangleq$ predecessor of $o$ on $machine_o$; $pred_o \in O$

$$= \begin{cases} - & \text{if } seqno_o = 1 \\ sequence(machine_o, seqno_o - 1) & \text{if } seqno_o > 1 \end{cases}$$

| $o$ | $pred_o$ |
|---|---|
| O1 | – |
| O2 | – |
| O3 | O1 |

$b_o \triangleq$ start time order $o \in O$

$e_o \triangleq$ end time order $o \in O$

$$b_o = \begin{cases} 0 & \text{if } seqno_o = 1 \\ e_{pred_o} + t^{order}_{pred_o, o} & \text{if } seqno_o > 1 \end{cases}$$

$e_o = b_o + p_{o, machine_o}$

| $o$ | $b_o$ | $e_o$ |
|---|---|---|
| O1 | 0 | 7.5 |
| O2 | 0 | 8 |
| O3 | 10.5 | 19.5 |

$7.5 + 3$

$l_o \triangleq$ lateness order $o \in O$

$l_o = \max\{0, e_o - d_o\}$ $\max\{0, 18-7.5\}$

| $o$ | $l_o$ |
|---|---|
| O1 | 0 |
| O2 | 0 |
| O3 | 7.5 |

$16.5 - 12$

schedule cost:

$\sum_{o \in O} c_o \cdot l_o = 10 \cdot 0 + 12 \cdot 0 + 8 \cdot 7.5 = 60$

import input data from excel
define sets $O, M$
set parameters based on input data $s_o, d_o \ldots$
calculate derived parameters $t^{order}_{o_1,o_2}$, $p_{o,m}$
... manually set solution ...
calculate derived variables $b_o, e_o, \ldots$
calculate costs
output schedule

## Research approach

- Literature
    - R. Rardin (2014), *Optimization in Operations Research*, Pearson New International Edition
    - E.H.L. Aarts and J.K. Lenstra (Editors) (2003), *Local Search in Combinatorial Optimization*, Princeton University Press

- Use of external sources
    - ChatGPT-4o was used as help by implementing the solution method in Python
        - Brainstorming on schedule data-structures
        - Obtaining Python code snippets for improving search heuristics
        - Advice on the best metaheuristic to use

## Design heuristic based on Improving Search

- Chosen neighborhood structure
    - Move: Swap 2 orders on the same or a different machine to search for potential improvements

- Heuristic solution methods
    - Constructive heuristic
        - Start with an initial schedule by assigning orders based on deadlines and availability
        - Ensure minimal colour switches and setup times

    - 2-opt Improving search
        - Iteratively swap 2 orders and evaluate the new schedule
        - If the swap lowers total penalty and setup time, accept the new schedule
        - Repeat until no further improvement

    - Tabu search
        - Iteratively swap 2 orders and evaluate the new schedule
        - Even if they temporarily worsen the solution, accept the new schedule
        - Use a Tabu List to avoid revisiting recent solutions
        - Continue search until a (near) optimal solution is reached

## Discrete improving search logic

Based on algorithm 12C in Rardin (2014)

```
Current solution = Initial sequence of orders (in order 1, 2 …, n).

Repeat:

    Repeat for pairs $(k, l)$, where $1 \le k$ and $k + 1 < l \le n$ (where n is the number of orders):

        Evaluate next 2-exchange move $(k, l)$ on the Current solution:

            Recalculate the schedule by swapping orders $k$ and $l$ in the sequence.

            For the new order sequence:

                For each order, assign it to the machine that minimizes the completion time (considering switching time and painting time).

                Update the machine states (current color and availability).

                Calculate the total time for all machines to finish.

            If move $(k, l)$ improves Current solution (e.g., reduces the total completion time):

                Current solution = Neighbor solution defined by $(k, l)$.

    Until improvement is found or all 2-exchange moves $(k, l)$ were considered.

Until all 2-exchange moves were considered and no improvement was found.

# Current solution is feasible and locally optimal with regard to 2-exchange move
```

## Tabu search logic

```
Current solution = Initial sequence of orders (random or predefined).

Tabu list = Empty list (will store recently applied moves that are temporarily
forbidden).

**Repeat**:

    **Repeat** for pairs ($k$, $l$), where 1 ≤ $k$ and $k$ + 1 < $l$ ≤ $n$ (where n is the number of
    orders):

        Evaluate the next 2-exchange move ($k$, $l$) on the Current solution:

            **If** the move is not tabu (or meets aspiration criteria — meaning
            it's a very promising move even though it's tabu):

                Recalculate the schedule by swapping orders $k$ and $l$ in
                the sequence.

                For the new order sequence:

                    Assign each order to the machine that minimizes
                    the completion time (taking into account switching
                    time and painting time).

                    Update the machine states (current color and
                    availability).

                Calculate the total time for all machines to finish.

            **If** the move ($k$, $l$) improves the Current solution:

                Update the Current solution with the neighbor solution
                defined by the ($k$, $l$) swap.

                Add the move ($k$, $l$) to the tabu list to prevent revisiting it
                for a few iterations.

                Remove oldest moves from the tabu list if it exceeds a
                predefined size (tabu tenure).

    **Until** all 2-exchange moves were considered and no improvement was found,
    or a predefined max iteration limit is reached.

**Until** all 2-exchange moves were considered, and no significant improvement was
found, or a maximum number of iterations or time limit is reached.
```

## Python implementation

We began testing our code with 3 machines and 3 orders to keep it simple and added complexity along the way.

- Orders, set of all orders O = {1, 2, 3}
- Machines, set of all machines M = {1, 2, 3}
- Functions defined:
    - Painttime(area, machine, machines)
        - Returns the time for machine M to paint an area

    - Switchtime(prevcolor, currentcolor)
        - Returns the time for any machine M to switch colors from prevcolor to currentcolor

    - Schedule_orders(orders, machines)
        - Returns a schedule S for all orders O on machines M based on when an order is completed on a machine

    - Calculate_penalty()
        - Returns the sum of the penalties for all orders O on schedule S

    - Swap_orders_optimization(orders, machines, max_iterations)
        - Returns schedule S after optimization

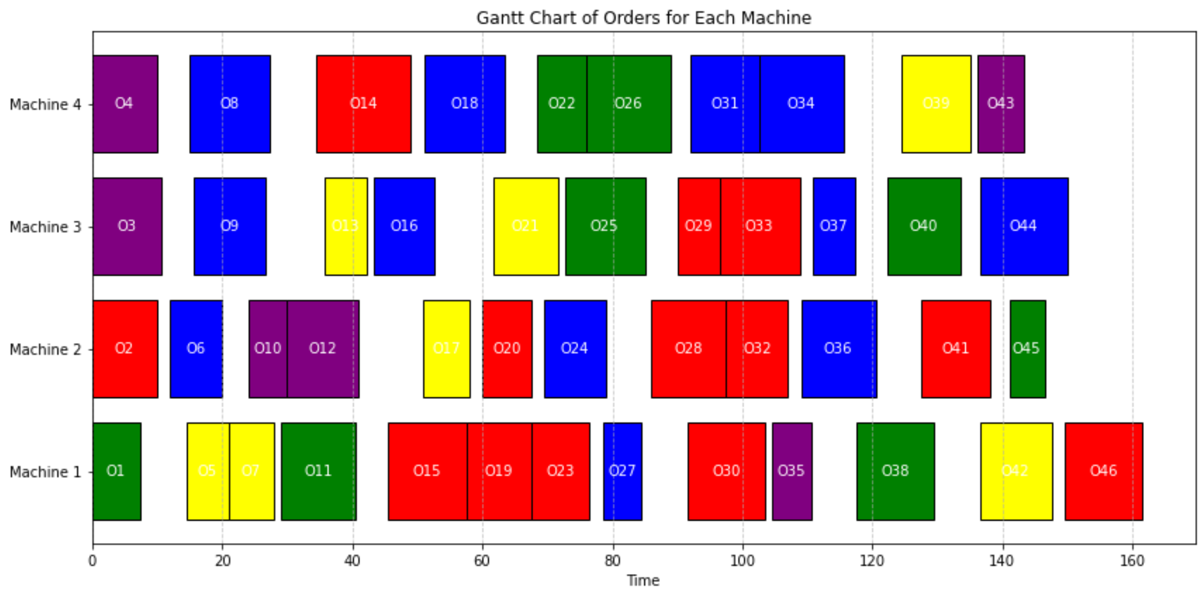
- Organisation of the code
    - There is one file containing both the implementation of the Discrete Improving Search and the Tabu Search
    - Discrete Improving Search is implemented in function swap_orders_optimizations
    - Tabu Search is implemented in function tabu_search_optimization

## Tests and experiments

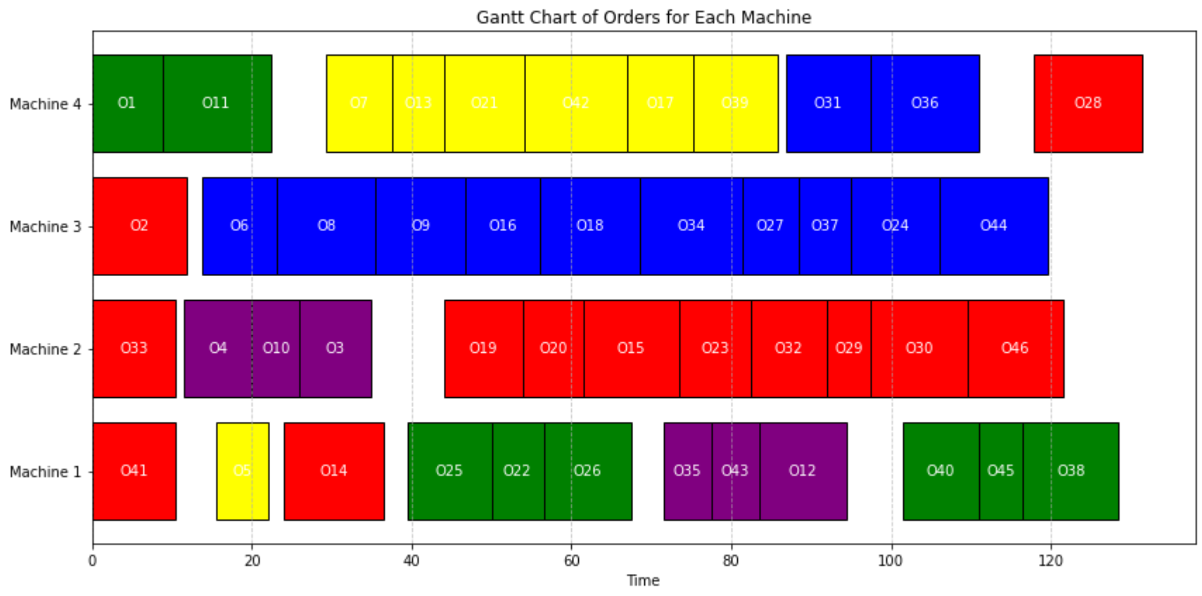
- Started with a simplified version with only one machine and no deadlines and penalties
- Started with a ChatGPT snippet for scheduling the orders
    - Double-checked implementation of switchtime: NOK
        - Didn't use the right syntax for variable type
    - Double-checked implementation of painttime: NOK
        - Didn't use the right syntax for variable type
- Implemented function to calculate the penalties of all orders
- Added orders to test function calculate_penalty()
- Implemented function to draw the schedule
- Implemented function to swap orders
    - Checked for accurate swapping: NOK
        - Was not swapping the orders selected correctly
    - Checked the updated schedule: OK
- Implemented function to do Tabu Search
    - Checked the tabu list: NOK
        - Kept adding to the list, not removing
    - Checked iteration list: NOK
        - Was not counting iterations in the desired way
    - Checked if there were improvements found: OK
- Added a plot to compare the used methods
- Added a print statement to view the penalty
- Tested tabu search with 10, 20 and 100 iterations
    - 100 iterations took too long to compute

## Visualizations

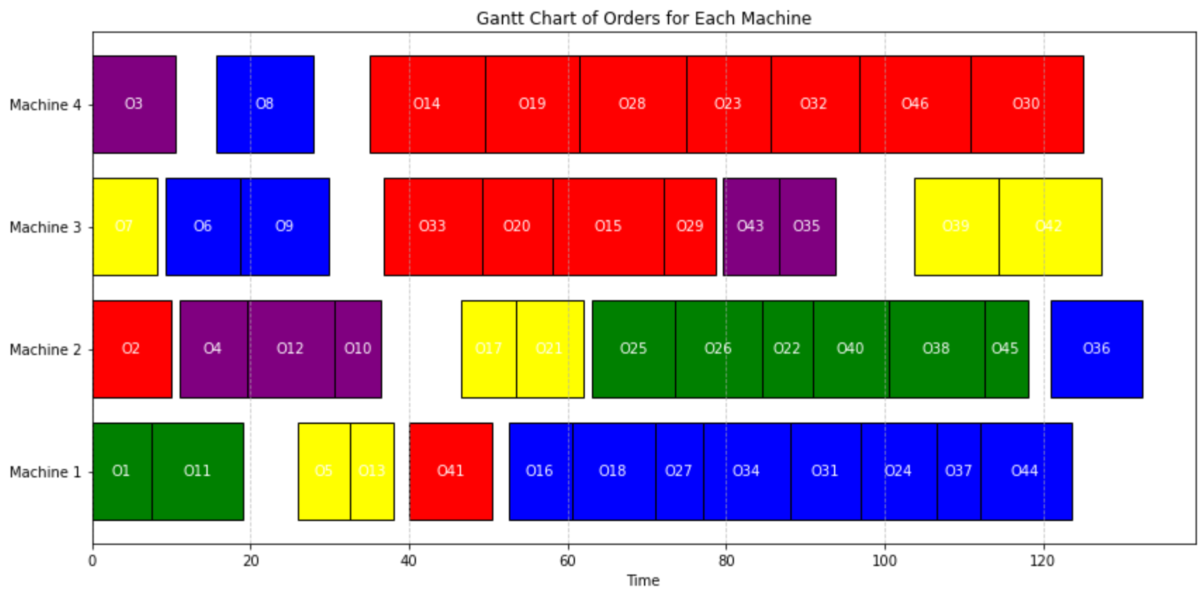
### Start planning - 7527.26 penalty



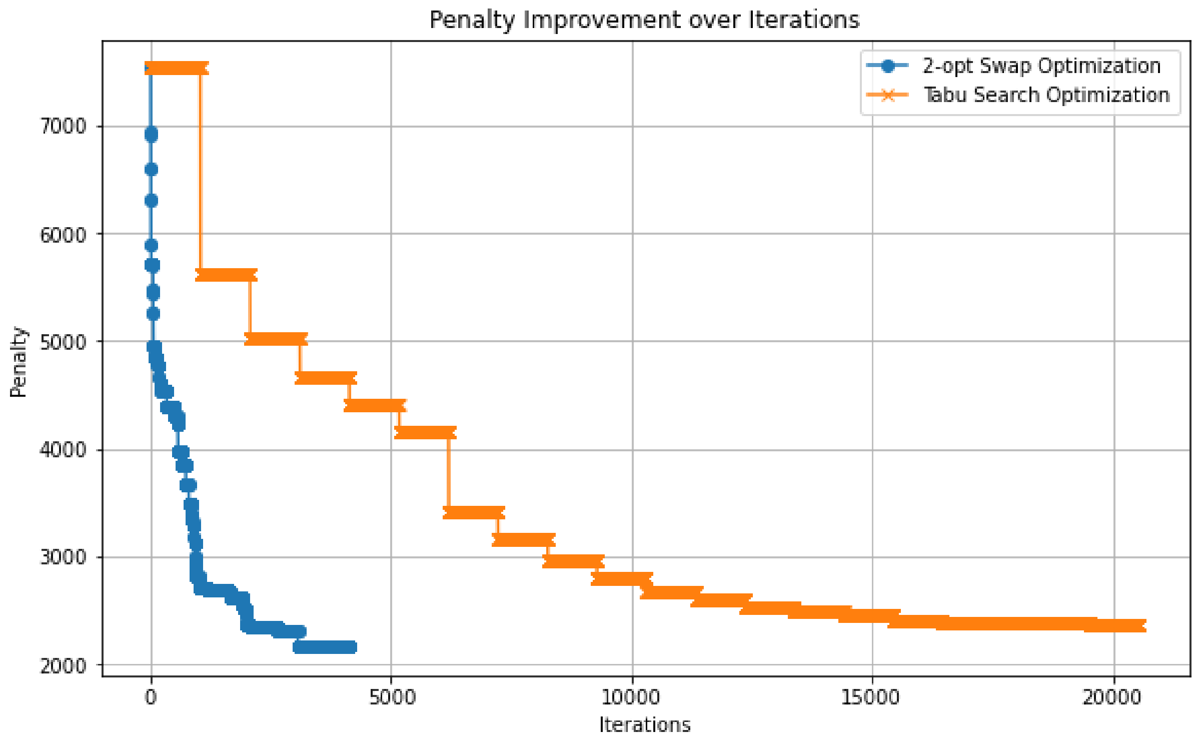
### 2-Opt swap planning – 2166.97 penalty



## Tabu Search planning – 2308.91 penalty



## Comparison graph



## Realized depth in OR

- Implemented Constructive heuristic
- Implemented 2-opt Improving Search
- Implemented Tabu Search
- Compared performance between all implementations based on lowest penalty