Group 61

Problem description

- **Goal**: Balance the workload of hospital wards by optimising the patient admission schedule

i. Minimise operational costs

ii. Spatial and temporal workload balancing

Bi-objective optimisation

The quality of a solution is determined by admission delays, utilization of operating rooms (**i.**) and workload balancing (**ii.**)

Unlike single-objective problems, there is a set of optimal solutions, each with a different trade-off between both objectives: **Pareto front**. It's up to the decision maker to pick a solution according to specific needs

Formulation of the problem

- **Solution representation**

Array of size $2 * n\_patients$: first half for **ward assignments**, second half for **admission days**. For crossover, an array of tuples (ward, day) is used instead

- **Objective functions**

$$\min \sum_{s \in S} \sum_{d \in D} \left(W^{\mathrm{OT}} v_{sd} + W^{\mathrm{UT}} u_{sd}\right) + W^{\mathrm{WAIT}} \sum_{p \in P} \sum_{w \in W_p} \sum_{d=f_p+1}^{e_p} (d - f_p) y_{pwd}$$

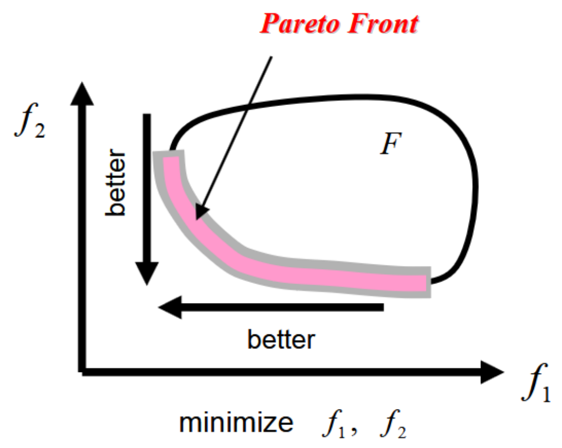
*Minimise operational costs: sum of OT utilization and patient admission delays (**and bed capacity violations**)*

$$\min z$$

$$\searrow$$

$$\max(\mathbf{X}) = \max_{d \in D}\left(\max_{w \in W} x_{wd}\right)$$

Minimise maximum workload



Formulation of the problem

- **Constraints**

i. Assigned wards and days must be feasible

ii. Bed capacity of wards must not be exceeded

This constraint was significantly difficult to repair on some datasets, so its violations were treated as part of the operational costs to minimise, in the first objective function (see previous slide). Our experiments suggested that this was effective, as most solutions we tried did not report bed capacity violations (except for certain datasets, such as *s1*)

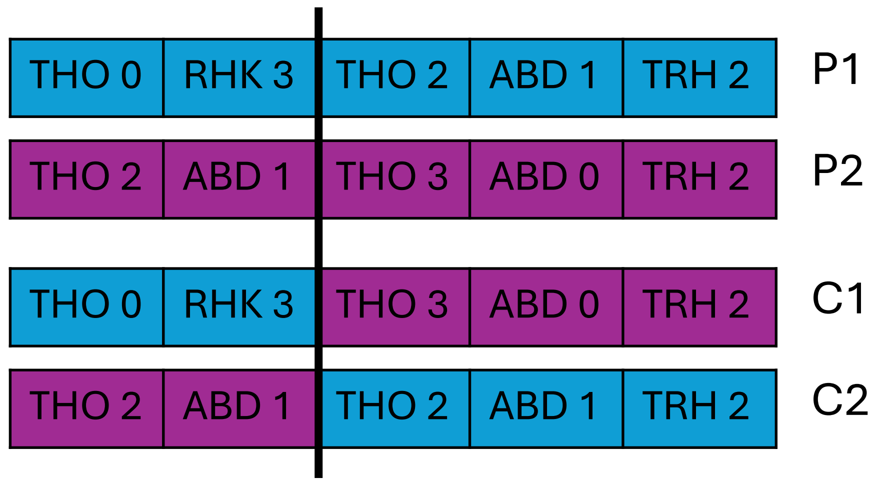
NSGA-II

- **Basic idea**

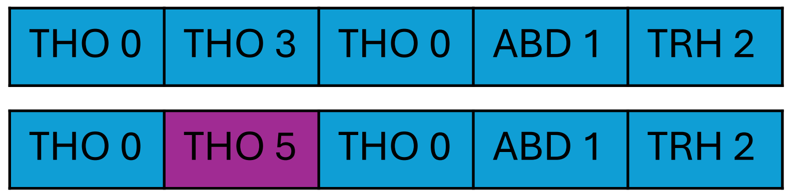
1. Non-dominated sorting: different levels of Pareto fronts
2. Crowding distance: favour solutions in less crowded regions
3. Binary tournament selection: individual with better rank (front) or, if equal, larger crowding distance, is chosen
4. Crossover and mutation
5. Combine parent and offspring (same size N) populations (2N)
6. Non-dominated sorting on 2N population, select best N (use crowding distance to select exactly N)
7. Repeat until fixed number of generations

For the implementation of this algorithm, we used the *pymoo* package

Crossover – Random point crossover



Mutation



Change random patient's date to another arbitrary day within time window

We also used a **repair operator**, to turn unfeasible solutions into feasible solutions. For every patient in the solutions, this operator starts by checking if the ward assignment is valid: if not, it reassigns the patient to a random valid ward. If the day is not valid either, then it picks a random day within the patient's time window.

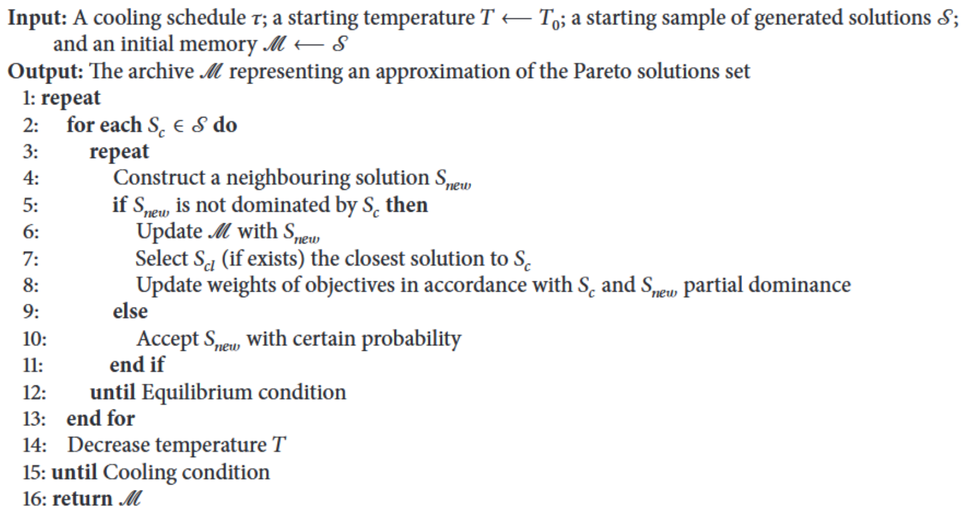
The user can change the following parameters of the algorithm:

1. Maximum number of generations (default=100)
2. Population size (default=100)
3. Mutation probability (default=0.1)

The algorithm stops when either the maximum number of generations has been reached or when the internal conditions of *pymoo* are met (which relate to the convergence of the Pareto front)

Pareto Simulated Annealing (PSA)

- **Basic idea**



```
Input: A cooling schedule $\tau$; a starting temperature $T \longleftarrow T_0$; a starting sample of generated solutions $\mathcal{S}$;
    and an initial memory $\mathcal{M} \longleftarrow \mathcal{S}$
Output: The archive $\mathcal{M}$ representing an approximation of the Pareto solutions set
1: repeat
2:    for each $S_c \in \mathcal{S}$ do
3:       repeat
4:          Construct a neighbouring solution $S_{new}$
5:          if $S_{new}$ is not dominated by $S_c$ then
6:             Update $\mathcal{M}$ with $S_{new}$
7:             Select $S_{cl}$ (if exists) the closest solution to $S_c$
8:             Update weights of objectives in accordance with $S_c$ and $S_{new}$ partial dominance
9:          else
10:            Accept $S_{new}$ with certain probability
11:         end if
12:      until Equilibrium condition
13:   end for
14:   Decrease temperature $T$
15: until Cooling condition
16: return $\mathcal{M}$
```

- **Parameters**

1. Size of initial sample (default=10)
2. Initial temperature (default=100)
3. Max outer iterations (default=100)
4. Inner iterations (default=10)
5. Cooling rate (default=0.75)

- Neighbouring solution S_{new}

Depending on probability, either smaller of larger changes can be made to solution

- Closest solution S_{cl}

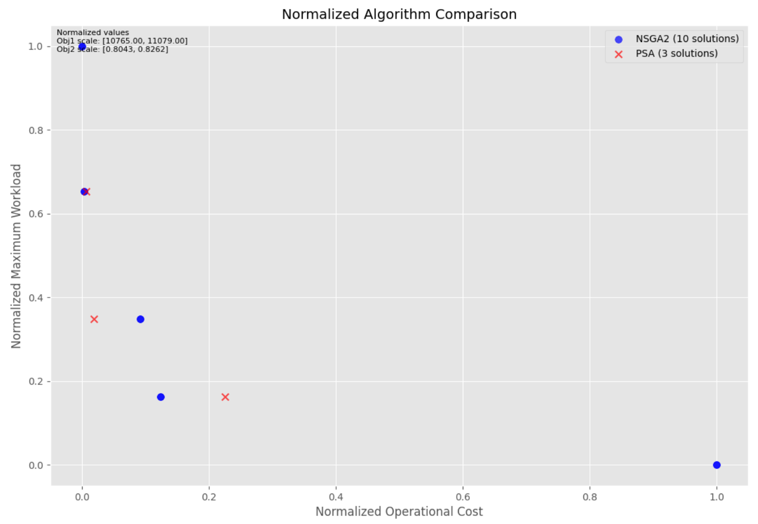
Solution in the archive with smallest Euclidean distance in normalised objective space

- Equilibrium condition

Number of inner iterations (for same temperature) is reached

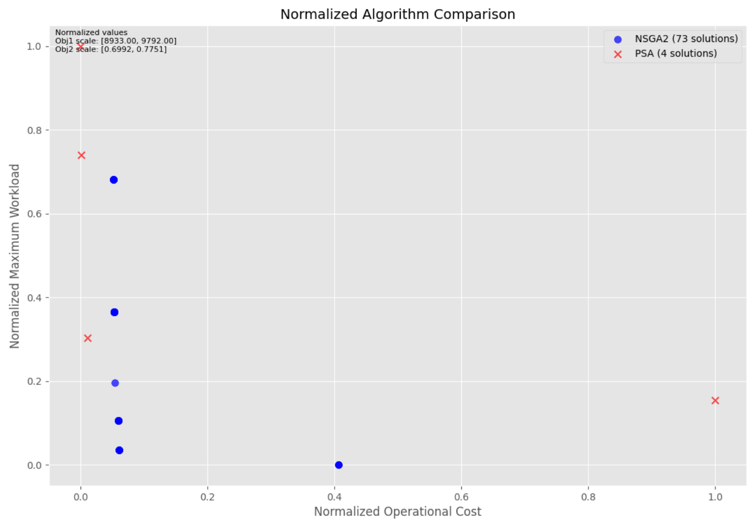
- Cooling condition

Very small temperature ($1 * 10^{-6}$) or maximum number of outer iterations reached. Iterations at near-zero temperature can help converging to local optima

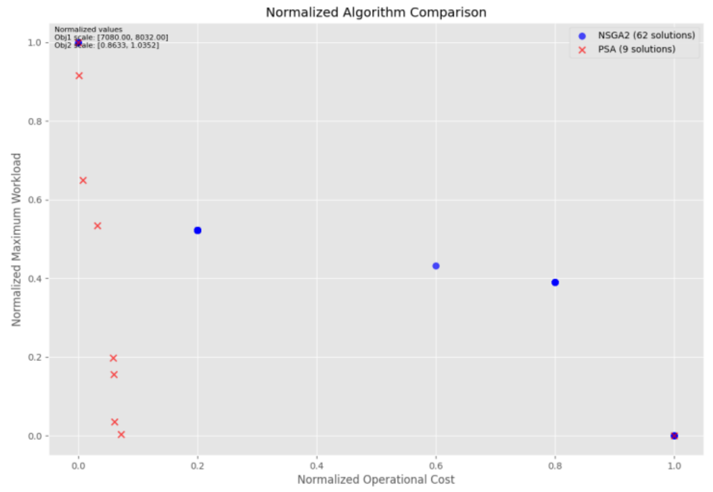


NSGA-II (53.96s) and PSA (47.51s) on s0m0 dataset, default parameters

With our chosen default parameters, the PSA algorithm is slightly more efficient than NSGA-II

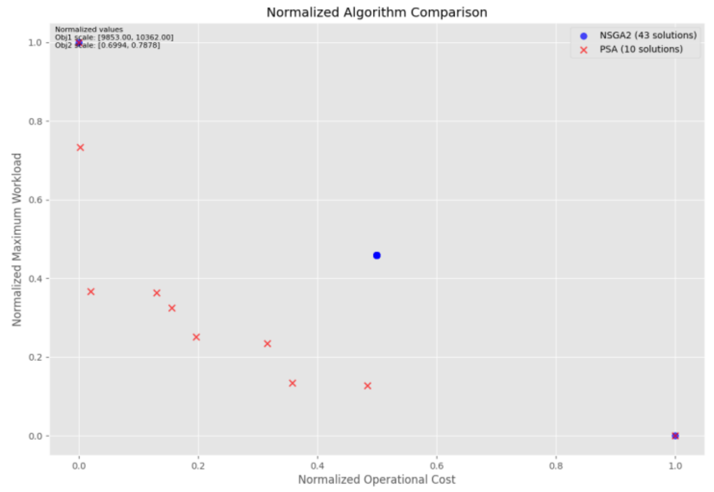


NSGA-II (17.28s) and PSA (16.05s) on s16m3 dataset, default parameters



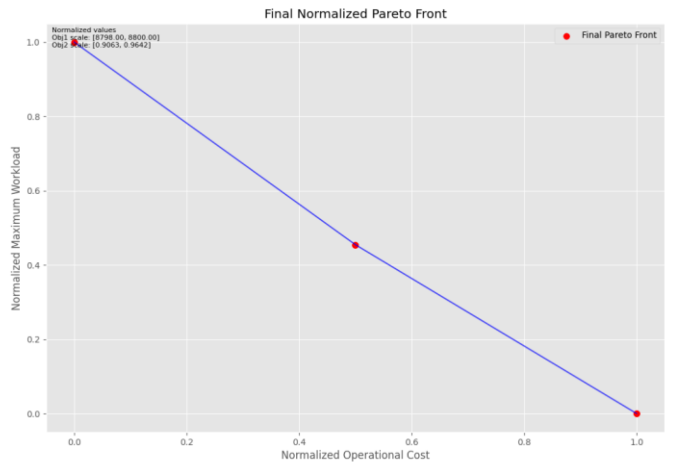
NSGA-II (39.05s) and PSA (36.46s) on s242m2 dataset, default parameters

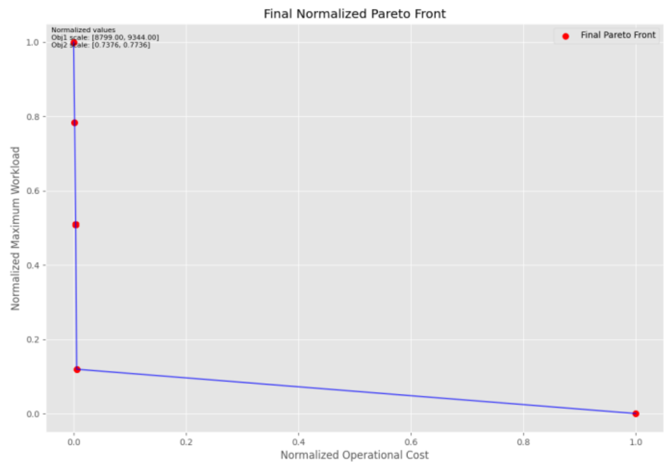
Here, the PSA algorithm results in a more diverse and optimal Pareto front



NSGA-II (51.92s) and PSA (50.74s) on s2m3 dataset, default parameters

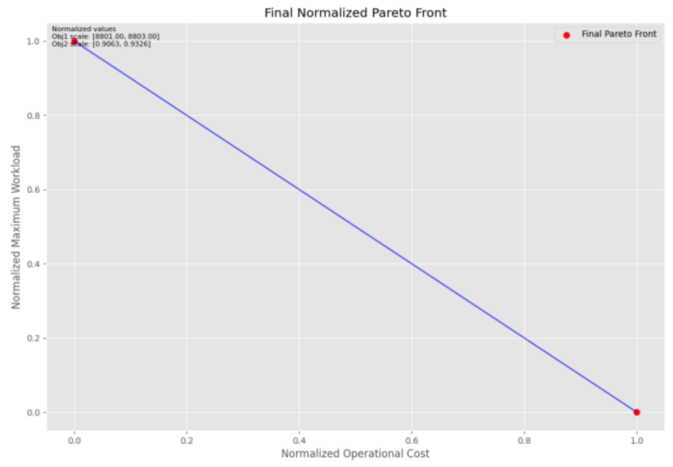
NSGA-II on same dataset, 0 and 3 minor specialisations per ward

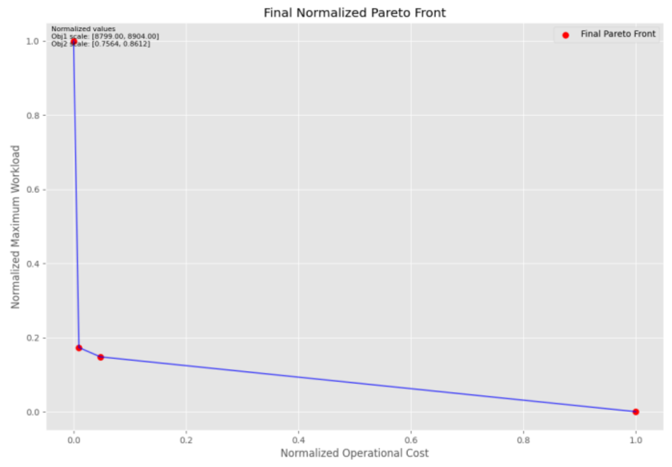




Increasing the number of minor specialisations per ward results in better Pareto fronts, where we can significantly improve an objective without a noticeable impact on the other objective

PSA on same dataset, 0 and 3 minor specialisations per ward





The decision maker should experiment with the algorithms' parameters: sometimes, a change can result in better solutions, but not always. In the case of NSGA-II, it's important to increase the number of generations if the Pareto front does not stabilise in 100 generations

Conclusions

While the results were varied and largely depend on the chosen parameters, the PSA algorithm seemed to fare better overall, both in terms of diversity and optimality of the solutions. However, PSA usually resulted in fewer solutions (due to a small initial sample), and increasing the number of initial solutions degraded performance

Experimentation is key when dealing with metaheuristics

Constraints, particularly if very restrictive, can negatively impact results, as repairing solutions interferes with the algorithm's optimisation process. This likely influenced the results of NSGA-II

Therefore, PSA may potentially be a more powerful algorithm for solving multi-objective optimisation problems with restrictive constraints

References

- https://www.sciencedirect.com/science/article/pii/S2211692323000139
- https://data.mendeley.com/datasets/3mv4rtxtfs/1
- https://ieeexplore.ieee.org/document/996017
- https://onlinelibrary.wiley.com/doi/full/10.1155/2019/8134674
- https://ieeexplore.ieee.org/document/1599245
- https://intleo.csu.edu.cn/codes/A-Comparative-Study-of-CHTs-in-CMOPs.pdf
- https://pymoo.org/index.html