COMP5012 Report

10697009

ACM Reference Format:
10697009. 2025. COMP5012 Report. In . ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Job scheduling is a fundamental problem in workforce management, particularly in domains such as healthcare, where ensuring adequate and fair staffing is both operationally critical and logistically complex. Assigning the right people to the right jobs, at the right times, while balancing organisational constraints and fairness, presents a significant computational challenge. In many real-world settings, scheduling involves not only satisfying hard constraints, such as staff needing to be qualified for specific roles, but also meeting soft goals like optimising staffing efficiency or distribution workload fairly among workers.

This study addresses a variant of the job scheduling problem where the goal is to assign workers to a fixed set of time-bounded jobs. The primary objectives are to minimise the number of workers required (to reduce staffing costs) and to promote fairness by distributing jobs evenly across workers. Two feasibility conditions are considered: first, that workers are only assigned to jobs that they are qualified to perform (a hard constraint); and second, that workers are ideally not assigned to overlapping jobs. Rather than enforcing this as a strict constraint, the presence of scheduling conflicts in penalised within the optimisation process, allowing for flexibility exploration of trade-offs between efficiency and feasibility.

These objectives are inherently conflicting. Using fewer workers may lead to concentrated workloads and increased overlap, while promoting fairness may require more staff. This makes the problem well-suited to a multi-objective genetic algorithm (MOGA) approach. In particular, evolutionary algorithms like NSGA-II are attractive because they do not require objective aggregation or strict linear formulations, and can efficiently explore trade-offs across complex, high-dimensional search spaces [1, 2]. MOGAs have been successfully applied in similar domains such as nurse scheduling, which share similar characteristics of hard and soft constraints, and competing goals [3].

2 BACKGROUND

The multi-objective job assignment problem considered in this study is combinatorial in nature and NP-hard, meaning that the time required to find an exact solution increases exponentially with problem size. This complexity motivates the use of heuristic approaches such as evolutionary algorithms, which are well-suited for approximating good solutions to large, intractable problems.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Multi-objective optimisation is especially relevant for real-world problems, particularly with constraints [1]. Unlike single-objective optimisation, where the goal is to find a single best solution, multi-objective problems yield a set of optimal trade-off solutions known as the Pareto front. Improving one objective in such problems typically comes at the cost of degrading another.

Traditional optimisation methods, such as integer programming, often require reformulating multi-objective problems into single-objective ones by assigning weights to objectives [2]. This can bias the search toward specific regions of the solution space and typically requires multiple runs with different weightings to approximate the Pareto front. In contrast, MOGAs can efficiently approximate the entire Pareto front in a single run by maintaining a population of diverse solutions.

One of the most widely used and influential MOGAs is the NSGA-II algorithm, proposed by Deb et al. [1]. NSGA-II introduced a fast non-dominated sorting approach, an elitist strategy for preserving the best solutions found so far, and a crowding-distance mechanism to ensure solution diversity. These improvements addressed key limitations of earlier MOGAs, making NSGA-II a preferred algorithm in many practical applications.

MOGAs have been successfully applied in similar domains, such as the multi-objective nurse scheduling problem (NSP), where Sharif et al. [3] used an external memory MOGA to optimise competing objectives, such as minimising staffing costs while adhering to staffing regulations. NSPs are typically NP-hard and involve a mix of soft and hard constraints, similar to the job scheduling problem addressed in this study.

In contrast, the rotating workforce scheduling problem, studied my Mörz and Musliu [4], applies a single-objective genetic algorithm to solve staff allocation while satisfying constraints like staffing levels, shift patterns, and legal regulations. This example highlights the flexibility of genetic algorithms in tackling complex scheduling domains but operates under a different problem formulation compared to MOGAs

3 METHOD

3.1 Problem Representation

The problem of assigning workers to jobs in a fair and efficient manner is represented as a set of job-worker assignments, where each individual in the population corresponds to a solution encoding the specific assignments of workers to jobs. Each solution is represented by a list of integers, with each integer representing the worker assigned the specific job:

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]$$

where x_j denotes the worker assigned to job j, and $x_i \in W$, with $W = \{w_1, w_2, \ldots, w_n\}$ being the set of all available workers.

Let $J = \{j_1, j_2, \ldots, j_m\}$ be the set of m jobs, $W = \{w_1, w_2, \ldots, w_n\}$ be the set of n workers, and $Q = \{Q_1, Q_2, \ldots, Q_m\}$ be the set of job qualifications where each $Q_j \subseteq W$ represents the subset of workers that are qualified to perform job j.

A solution is considered valid only if each assigned worker is qualified for the job that they are assigned to. That is, for all $j \in J$:

$$x_j \in Q_j$$

This ensures that only feasible solutions - where all job assignments respect the qualification constrainst - are evaluated.

In the implementation, individuals are initialised such that every job is assigned a worker that is qualified for that job. The job IDs are determined by their index position in a list of tuples representing their scheduled time slots, given as `(start, end)`. To discourage scheduling conflicts, a penalty is applied to the objective values when any worker is assigned overlapping jobs. Whle qualification is treated as a hard constraint (invalid solutions not allowed), overlapping assignments are allowed during evolution but heavily penalised to steer the algorithm toward conflict-free solutions.

This representation was chosen for its simplicity and suitability for manipulation using genetic operators such as mutation and crossover.

3.2 Objective Functions

Two objectives were defined based on practical staffing goals. The first objective is to minimise the number of workers, and therefore minimise the amount of money being paid on staffing costs. This is captured by:

$$\min f_1(\mathbf{x}) = \left|\{x_j \mid j \in J\}\right|$$

The second objective aims to promote fairness by evenly distributing the jobs among workers. This helps to reduce worker burnout and increase job satisfaction. Let w_k be the number of jobs assigned to worker $k \in W$, and μ be the mean number of jobs per worker:

$$\mu = \frac{1}{|W|} \sum_{k \in W} w_k$$

Then, the workload variance is:

$$\text{Var} = \frac{1}{|W|} \sum_{k \in W} (w_k - \mu)^2$$

The second objective then becomes:

$$\max f_2(\mathbf{x}) = -\text{Var}$$

Using a negative value for the variance was chosen so that this objective was obviously interpretable as maximising the fairness of the job distribution.

3.3 Overlap Penalty

To account for scheduling feasibility, a penalty is applied when a worker is assigned overlapping jobs. For each worker w_k, let $A_k = \{j \in J \mid x_j = w_k\}$ be the set of jobs assigned to them. Each job j has an associated time interval (s_j, e_j). An overlap occurs if any pair $(j_a, j_b) \in A_k$ satisfies:

$$[s_{j_a}, e_{j_b}) \cap [s_{j_b}, e_{j_b}) \neq \emptyset$$

The total number of overlapping pairs across all workers is denoted overlap_count($\mathbf{x}$), and a penalty is applied proportionally:

$$\text{penalty}(\mathbf{x} = \lambda \cdot \text{overlap\_count}(\mathbf{x})$$

This penalty is added only to the first objective function during evaluation to discourage, but not prohibit, overlaps.

Initially, overlapping job assignments were treated as a hard constraint; any solution containing overlap was considered infeasible and discarded. However, this lef to a limited search space and poor diversity in early generations. To allow the algorithm to explore a broader range of potentially high-quality solutions, overlaps were instead treated as a soft constraint and penalised within the objective function.

The overall multi-objective problem can be summarised as:

$$\begin{aligned}
&\text{minimise} && f_1(\mathbf{x}) = \left|\{x_j \mid j \in J\}\right| + \lambda \cdot \text{overlap_count}(\mathbf{x}) \\
&\text{minimise} && f_2(\mathbf{x}) = \text{Var}(w) = \frac{1}{|W|} \sum_{k \in W} (w_j - \mu)^2 \\
&\text{subject to} && x_j \in Q_j \quad \forall j \in J
\end{aligned}$$

3.4 MOGA Structure

In this study, the multi-objective optimisation problem is solved using the NSGA-II (Non-dominated Sorting Genetic Algorithm II), a well established algorithm for handling problems with multiple conflicting objectives. NSGA-II is designed to maintain a population of solutions and evolve it over time through the application of genetic operators such as crossover, mutation, and selection. The algorithm aims to find a set of Pareto-optimal solutions, where no solution in the set is strictly better than any other with respect to all objectives.

NSGA-II operates in a generational loop, where at each generation, the following steps are performed:

1. **Selection**: Individuals are selected based on their rank in the non-dominated sorting process and their crowding distance, ensuring diversity in the population.
2. **Crossover**: Pairs of individuals are combined to generate offspring, allowing the algorithm to explore the solution space.
3. **Mutation**: Individuals undergo random changes to introduce variability and prevent premature convergence.
4. **Non-dominated Sorting**: The population is ranked based on non-dominance, and the Pareto front is determined.
5. **Archive Update**: A population archive stores the best solutions found across generations, ensuring diversity is maintained.

The choice of NSGA-II is motivated by its ability to efficiently handle problems with conflicting objectives and its effectiveness in maintaining diversity within the population throughout the search process.

3.4.1 *Operators.*

- **Selection:** `tools.selNSGA2` from the DEAP library.
- **Crossover:** Two-point crossover.
- **Mutation:** Custom mutation that maintains constraint validity (i.e, only assigns qualified workers to jobs).

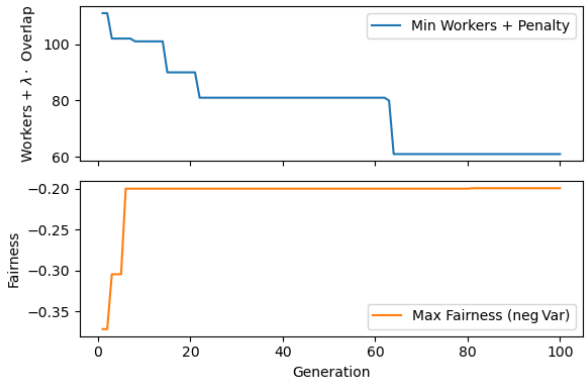

Figure 1: Convergence behaviour when population = 200, mutation probability = 0.3, λ = 10, and generations = 100.

3.4.2 Archive and Diversity Preservation. To preserve diversity and avoid premature convergence, an archive of non-dominated solutions is maintained throughout the evolutionary process. At each generation, the current population is merged with the archive, and non-dominated sorting is applied to this combined pool. This strategy ensures that not only newly generated individuals, but also high-quality solutions from previous generations, are retained.

The archive allows the algorithm to maintain multiple trade-offs between objectives, rather than prematurely focusing on a single area of the search space. By consistently keeping the best diverse solutions found so far, the algorithm is better guided toward a well-distributed approximation of the Pareto front. NSGA-II also inherently supports elitism and archive maintenance through its selection strategy, ensuring that the best solutions are not lost over time.

4 EXPERIMENTAL SETUP

The algorithm was implemented using the DEAP 1.4.2 library [5] and Python 3.11.8. Experiments were conducted on a Macbook Pro with an M1 chip and 16GB of RAM. The full implementation and all assosciated code can be found in the **Github Repository** [6].

The input dataset consisted of 40 separate jobs, with each job being associated with a specified start and end time, and also a set of qualified workers.

To evaluate the performance and robustness of the NSGA-II optimiser, a comprehensive grid search was conducted across key hyperparameters. The following values were explored:

- **Population size:** 50, 100, 200
- **Mutation probability:** 0.1, 0.2, 0.3
- **Penalty weight (λ) for overlapping job assignments:** 5, 10, 20
- **Number of generations:** 50, 100, 200

Each configuration was evaluated in a single run of the algorithm. At the conclusion of each run, the final non-dominated archive was inspected. To enable consistent comparison across configurations, the individual with the highest fairness score was extracted from the archive. From this solution, the true number of distinct workers (excluding penalties) and the fairness score (defined as the negative variance of job distribution) were recorded. This allowed for comparison of fairness and workforce size across parameter settings. The results of all runs were logged to a summary CSV file for comparative analysis, which is available on the project repository.

Due to the large number of configurations, visualisations were generated only for a selected subset:

- Population size = 50, Mutation probability = 0.2, λ = 10, Generations = 100
- Population size = 100, Mutation probability = 0.2, λ = 10, Generations = 100
- Population size = 200, Mutation probability = 0.3, λ = 10, Generations = 100

This subset was selected to represent low, medium, and high parameter values. Based on qualitative inspection of convergence plots and Pareto front evolution, the configuration with population = 100, mutation probability = 0.2, and generations = 100 was selected for final testing. This configuration consistently demonstrated stable convergence, diverse non-dominated solutions, and good objective values. By contrast, Figure 1 shows the convergence behaviour when the population = 200, and the mutation probability = 0.3 (with all other parameters held constant) As shown, the optimiser converges prematurely under those settings, evidenced by an early plateau in both objectives, so this configuration was not chosen.

Subsequent experiments focused on tuning the penalty weight λ using this configuration. The overlap penalty was implemented as a soft constraint in the first objective, allowing the optimiser to explore trade-off between fairness and scheduling feasibility.

5 RESULTS

Using the selected configuration (population = 100, mutation probability = 0.2, generations = 100), the penalty parameter λ was varied to assess its effect on solution quality. Table 1 reports the outcomes for the individual with the highest fairness score in each run:

λ	**Workers**	**Fairness (-Var)**
5	22	-0.149
10	18	-0.173

Table 1: Comparison of solution quality under different penalty weights.

The λ = 10 setting led to fewer workers and better fairness, suggesting that a stronger penalty for overlapping jobs improved overall solution quality by guiding the optimiser toward more feasible and equitable schedules. Consequently, λ = 10 was selected as the final parameter setting.

The algorithm successfully produced a diverse set of trade-off solutions between minimising the number of workers and maximising the fairness of job distribution. Figure **??** shows the final Pareto front after 100 generations. As expected, solutions that used fewer workers had a less fair job distribution than those that used more workers.

Figure 2 shows the best value of objective 1 (number of workers) across generations. Even after 80 generations, the optimiser continued to make improvements. The algorithm was also run for 200

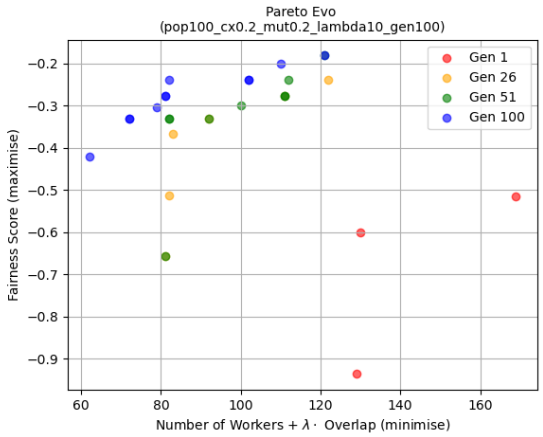

Figure 4: Pareto front evolution across selected generations.

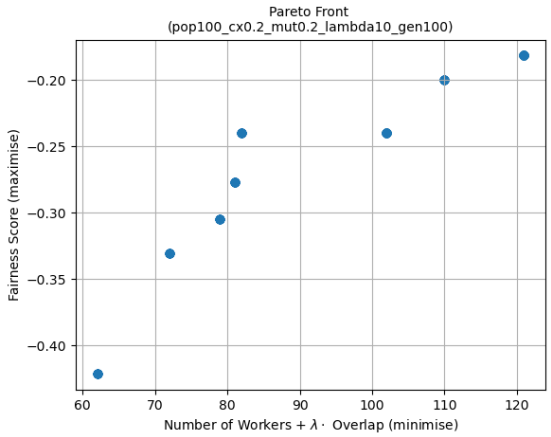

Figure 5: Final Pareto front of non-dominated solutions.

generations, but no further improvement in either objective were observed. Based on this, 100 generations was selected as a suitable cut-off point for this problem.

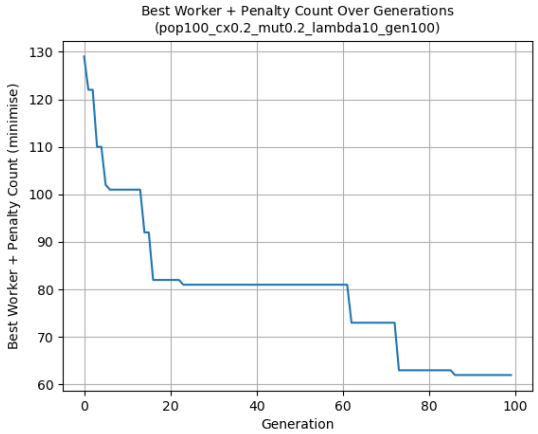

Figure 2: Best number of workers across generations.

Figure 3 shows the best fairness score (objective 2) across generations. Unlike the first objective, improvements in fairness plateaued just after 40 generations. However, due to the continued improvement of the first objective beyond this point, the full 100 generations were retained.

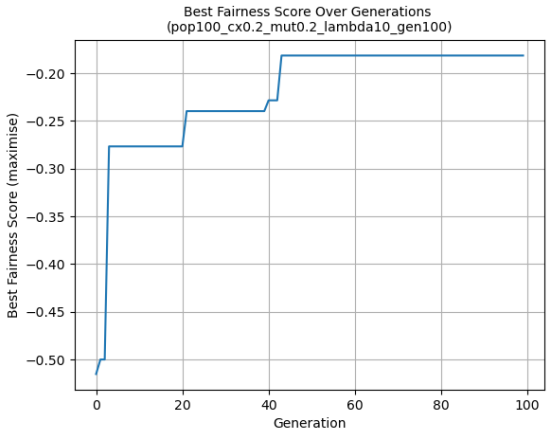

Figure 3: Best fairness score across generations.

To analyse the trade-offs discovered by the optimiser, the Pareto front was visualised at several generations (Figure 4). The solutions move toward the top-left corner, representing better trade-offs between fewer workers and a fairness closer to 0.

Finally, the complete set of non-dominated solutions from the final archive is shown in Figure 5. Although the Pareto front is small (four points), it shows meaningful trade-offs. For example, one solution uses fewer workers but has a lower fairness score, while another distributes jobs more evenly at the cost of employing more workers.

The archive maintained by NSGA-II ensured than non-dominated solutions from previous generations are not disregarded, ensuring that the final Pareto front is adequately diverse.

Each solution in the final Pareto front was inspected to verify validity, shown in Table 2. All solutions incurred zero qualification errors, confirming that every assigned worker was properly qualified. Overlap counts-i.e the number of pairwise time clashes-ranged from 4 to 11. The fact that even the least fair solutios had only a handful of overlaps demonstrates that the chosen penalty weight ($\lambda = 10$) effectively enforces feasibility while preserving a diverse set of trade-offs. In practice, this balance ensures that scheduling solutions remain both equitable and executable.

6 CONCLUSIONS

This study explored the use of a MOGA, specifically NSGA-II, for solving a constrained job assignment problem with competing objectives: minimising the number of workers and maximising fairness in workload distribution. By encoding qualification requirements as a hard constraint and penalising overlapping assignments

Sol.	Workers	Fairness	Qual. Violations	Overlaps
1	21	-0.181	0	11
2	22	-0.240	0	9
3	22	-0.240	0	8
4	20	-0.200	0	9
5	21	-0.278	0	6
6	21	-0.278	0	6
7	22	-0.240	0	6
8	22	-0.331	0	5
9	22	-0.331	0	5
10	19	-0.305	0	6
11	22	-0.421	0	4

Table 2: Table of trade-offs and constraint violations for the final Pareto front.

within the fitness function, the evolutionary process successfully navigated a highly constrained solution space. The inclusion of the overlap penalty in particular allowed the model to flexibly balance feasibility with optimisation, rather than discarding otherwise high-qualite solutions due to minor violations.

NSGA-II proved to be a flexible and effective optimisation approach, capable of balancing conflicting goals without requiring a predefined aggregation of objectives. The final population of solutions offered a clear Pareto front, allowing for informed decision-making based on organisational priorities.

However, there are several areas for improvement. The current implementation allows overlapping assignments with a penalty, but in many real-world contexts, it is physically impossible for a worker to perform two jobs at the same time. In such cases, overlapping jobs should arguably be treated as a hard constraint rather than a soft one, or the penalty should be scaled more aggressively to reflect the severity of the violation. Additionally, better calibration of this penalty could lead to more realistic and useful solutions.

Future work could involve the inclusion of more complex and realistic scheduling constraints, such as mandatory rest days, shift patter restrictions, or preferences for two-day weekends,. Comparing the performance of NSGA-II to traditional optimisation methods, such as greedy algorithms or integer linear programming, would also provide useful benchmarks. Finally, applying the algorithm across a wider range of dataset sizes and configurations would help evaluate its scalability and robustness in real-world scenarios.

REFERENCES

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6, 2, 182–197. DOI: 10.1109/4235.996017.

[2] R Timothy Marler and Jasbir S Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26, 369–395. DOI: 10.1007/s00158-003-0368-6.

[3] Omid Sharif, Ahmet Ünveren, and Adnan Acan. 2009. Evolutionary multiobjective optimization for nurse scheduling problem. In *2009 Fifth International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control*, 1–4. DOI: 10.1109/ICSCCW.2009.5379458.

[4] M. Morz and N. Musliu. 2004. Genetic algorithm for rotating workforce scheduling problem. In *Second IEEE International Conference on Computational Cybernetics, 2004. ICCC 2004.* 121–126. DOI: 10.1109/ICCCYB.2004.1437685.

[5] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. Deap: evolutionary algorithms made easy. *J. Mach. Learn. Res.*, 13, 1, (July 2012), 2171–2175.

[6] Ava Keeling. 2025. Job scheduling optimiser. Accessed: 2025-04-22. (2025). https://github.com/avakeeling199/job_scheduling_optimiser.