A Demonstration of an Integrated Project Management and Demo Scheduling Platform

Neel Gaglani* and Ender Özcan

School of Computer Science, University of Nottingham, Nottingham, UK
neel148g@gmail.com, ender.ozcan@nottingham.ac.uk

Abstract. This paper presents an integrated software that automate the end-to-end coordination of undergraduate projects, including panel assignment, administrative data processing, conflict detection, and demo scheduling. The system combines a modern web-based architecture with tailored heuristic and optimisation methods to address the complex resource allocation constraints typically managed manually. A key component of the platform is the demonstration timetabling engine, capable of generating feasible, balanced schedules under staff, room, and time constraints. This work illustrates the proposed system's easy-to-use interactive interface, and compares the performance of the two hyper-heuristic approaches tested on realistic problem instances. Together, these components demonstrate how heuristic optimisation and workflow automation can significantly reduce administrative overhead while improving fairness and reliability in large-scale academic scheduling.

Keywords: Timetabling · Great Deluge · Simulated Annealing · Metaheuristic · Hyper-heuristics

1 Introduction

Coordinating undergraduate final year projects involves multiple interconnected processes, including project allocation, panel assignment, file preparation, conflict detection, and the generation of a demonstration timetable. Manual spreadsheet-based methods are time-consuming and prone to human error, especially as student numbers grow. Automated timetabling is well-recognised as a highly constrained NP-hard problem [8], and educational scheduling has been extensively studied in the literature [4, 2].

This paper presents a unified software platform that automates the end-to-end workflow of the management of final year projects. The system was designed around a web-based architecture and integrates deterministic administrative tools with (meta)heuristic optimisation components. A key feature and focus of this demonstration is the automated generation of demonstration schedules; an NP-hard timetabling problem that must consider staff availability, break windows, capacity limits, and workload fairness.

* Corresponding author.

We showcase the system interface, architectural components, and search behaviour using two tuned selection hyper-heuristics [6] using the same reinforcement learning heuristic selection methods and different move acceptance methods: *Simulated Annealing* (SA) [12] and *Great Deluge* (GD) [3]. We will illustrate how the tool dynamically generates, evaluates, and updates schedules, including support for manual overrides, staff substitutions, and instantaneous analytics.

The remainder of this paper is structured as follows. Section 2 describes the overall system architecture, highlighting the modular design of the frontend, backend, and database components, and explaining how these interact to support real-time scheduling and administrative workflows. Section 3 focuses on the panel assignment while Section 4 on the demonstration scheduling challenge, detailing the constraints, objective function, and (meta)heuristic strategies used to generate feasible timetables. Finally, Section 5 concludes the paper with a summary of contributions.

2 System Architecture

The system follows a modular client–server architecture, similar to modern educational timetabling platforms, embedding component-based hyper-heuristic development support as in HyFlex [13]:

- **Frontend:** Implemented in React with Tailwind CSS, supporting interactive timetable views, availability management, and configuration of optimisation parameters.
- **Backend:** A Flask-based REST API performing data processing, panel assignment, conflict detection, and optimisation. The system follows clean separation of concerns as recommended in network software architecture design [7].
- **Database:** PostgreSQL with a relational schema adhering to 3NF principles [5], ensuring data integrity for Students, Staff, Projects, Panel Assignments, Schedules, and Availability.

The scheduling engine uses a data hydration phase to load constraint-relevant entities into memory for constant-time access. Neighbourhood operators, delta (incremental) evaluation, and reinforcement-learning–based operator selection draw on principles from selection hyper-heuristics [1, 11].

The system exposes its features through a demonstration interface, allowing users to inspect algorithm behaviour, manually adjust schedules, and visualise staff idle times and workload balances. It includes interactive timetable editing with real-time constraint validation, a staff availability editor, and reproducible multi-run optimisation with seed control. This architecture ensures that computationally expensive scheduling operations are efficient while maintaining strict usability for administrative staff.

3 Panel Assignment

Panel assignment presents its own challenge due to the requirement that each project must be allocated exactly three unique panel markers while ensuring workload fairness, feasibility, and adherence to academic constraints.

Problem Definition. Let P be the set of all projects, S the set of supervisors and M the set of panel members. For each project i, the solution assigns a set of three unique panel members $\{PM_{i,1}, PM_{i,2}, PM_{i,3}\} \subseteq M$. Each project also has a pre-determined main supervisor, denoted by S_i.

Hard Constraint. All three panel members must be unique, and the supervisor cannot be assigned as a panel member. Formally, the four assigned academic roles must be pairwise distinct:

$$|\{S_i, PM_{i,1}, PM_{i,2}, PM_{i,3}\}| = 4 \qquad \forall i \in P.$$

Soft Constraint 1 (Total Workload Balance). The total dissertation marking load W_j^{total} for staff member j should be distributed as evenly as possible. The target workload α represents the total number of panel seats ($3|P|$) divided by the number of available panel members ($|M|$), allowing a maximum deviation of 1:

$$\lfloor \alpha \rfloor \;\leq\; W_j^{\text{total}} \;\leq\; \lfloor \alpha \rfloor + 1, \qquad \text{where } \alpha = \frac{3|P|}{|M|}.$$

Soft Constraint 2 (Demo-Marker Balance). The demo workload of each panel member W_j^{demo} should be balanced, targeting an average value β. Using our approach, where the first panel member assigned to mark a project dissertation is also required to attend its demo, we ensure that workload balance is fully maintained across both dissertation marking and demo attendance for all panel members.

$$\lfloor \beta \rfloor \;\leq\; W_j^{\text{demo}} \;\leq\; \lfloor \beta \rfloor + 1, \qquad \text{where } \beta = \frac{|P|}{|M|}.$$

The primary issues arise from overlapping staff roles, many supervisors are also panel markers, which creates frequent conflicts where a supervisor may accidentally be assigned to mark their own student. Additional complications include uneven workload distribution across staff, the need to balance demo-marker assignments, and the high likelihood of conflict propagation when attempting naïve manual adjustments. To address this, the system implements a pattern-based construction heuristic, which first generates three permutations of available panel markers and cyclically maps these patterns onto the project list. This guarantees near-uniform workload distribution before any conflict resolution occurs. A subsequent greedy repair stage identifies supervisor conflicts and resolves them by performing constrained swaps that preserve uniqueness conditions and workload boundaries. This hybrid approach enables the system to generate valid, balanced panel assignments in deterministic time, eliminating the need for manual shuffling and preventing downstream scheduling conflicts.

4 Demonstration Scheduling

Demonstration scheduling is formulated as a multi-constraint optimisation problem comparable to classical university timetabling challenges. Each project requires assignment of a demo marker, supervisor, computer station, and time slot while satisfying the following strict hard constraints:

- No staff member (supervisor or marker) can be in two places simultaneously.
- Demonstrations occur across two fixed days (e.g., from 10:00–16:00) with a mandatory lunch break (e.g., from 12:00–13:00).
- Up to a predefined maximum number of parallel demonstrations/sessions may occur.
- Staff availability constraints must be honoured.

Soft constraints aim to improve fairness and convenience:

- Minimising idle gaps between sessions.
- Consolidating small workloads into a single day.
- Balancing large workloads evenly across both days.

We have developed a hyper-heuristic approach to tackle this problem. The design of the algorithmic components are presented in the following subsections.

4.1 Common Algorithmic Components

Solution Representation. The solution is represented as an array of project events:

$$\text{Events}_i = \{\text{DemoMarker}_i,\ \text{Day}_i,\ \text{TimeSlot}_i,\ \text{Computer}_i\}.$$

Objective Function. The objective (cost) function evaluates the quality $f(s)$ of a solution using a weighted sum of number of hard-constraint violations (H) and total penalties incurred due to all soft-constraint violations. Hard constraints, weighted by w_H, are prioritised significantly over soft constraints.

$$f(s) = (w_H \times H) + (\text{Penalty}_{\text{gap}} + \text{Penalty}_{\text{small}} + \text{Penalty}_{\text{heavy}}).$$

Three staff-centric targeted criteria are incorporated into the objective function to maximise staff convenience and improve the quality of generated demonstration timetables.

- *Gap Penalty.* To reduce idle time (gap_j) between consecutive demonstrations (in minutes) for any staff member j, gaps are penalised quadratically. Longer waiting periods incur disproportionately higher penalties:

$$\text{Penalty}_{\text{gap}} = (\sum_{\forall j} (\text{gap}_j)^2) \times w_{\text{gap}}.$$

– *Small Workload Consolidation.* For (c) staff members whose total number of assigned sessions is below a configurable threshold (initially set to 4), a fixed penalty of 2000 is applied if their sessions are split across both days. This encourages consolidating small workloads into a single day.

$$\text{Penalty}_{\text{small}} = c \times w_{\text{small}}.$$

– *Heavy Workload Balancing.* For staff members whose workloads exceed the threshold, the algorithm seeks to balance sessions across both days. A cubic penalty is imposed on imbalances, ensuring that heavier workloads are distributed equitably:

$$\text{Penalty}_{\text{heavy}} = (\sum_{\forall j}(|D_{j,1} - D_{j,2}|^3) \times w_{\text{heavy}}.$$

In our experiments, we have set $w_{\text{gap}} = 20$, $w_{\text{small}} = 2000$ and $w_{\text{heavy}} = 500$.

Initialisation. A construction heuristic has been employed to create the initial solution. This heuristic firstly groups projects by their assigned demo marker and sorts them by their load size. The algorithm then attempts to place these groups in consecutive blocks, splitting larger groups evenly across the two available demonstration days. Unplaced projects are then greedily assigned to any valid computers and timeslots available.

The search landscape is highly constrained and often dense, particularly when staff supervise many projects. The system implements two selection hyper-heuristics sharing a common objective function, heuristic selection method, neighbourhood operators, and delta (incremental) evaluation engine, but different move acceptance methods as outlined below.

4.2 Move Acceptance

Simulated Annealing (SA). Based on the classical formulation of Kirkpatrick et al. [12], SA uses geometric cooling (with $\alpha = 0.998$ and minimum temperature of 1.0). A dynamic reheating mechanism was integrated to combat the risk of "deep" local optima. If the algorithm failed to find a new global best solution for a continuous span of the maximum number of evaluations, the temperature would temporarily be multiplied by a boost factor. This would allow the heuristic to accept worse moves temporarily so that it can climb out of the local minimum trap before resuming its standard geometric cooling again.

Great Deluge (GD). Originally proposed as an alternative monotonic descent method [3], GD accepts solutions beneath a decaying water level. GD often performs better in compact search spaces with fewer deep local minima. The initial water level L_{initial} is set by evaluating the cost of the initially generated solution and applying a 5% buffer. This provides a brief exploratory phase before the acceptance boundary becomes restrictive:

$$L_{\text{initial}} = f(s_{\text{initial}}) \times 1.05.$$

The water level is then reduced linearly using a constant decay rate ΔL. To ensure that the algorithm transitions into a strict hill-climbing exploitation phase during the final portion of its execution, the decay rate is defined such that the water level reaches zero precisely at 90% of the total evaluation budget. Hence $\Delta L = L_{initial}/(\text{maximumNumberOfEvals} \times 0.9)$.

A hyper-heuristic terminates when the maximum number of evaluations is exceeded.

4.3 Low Level Heuristics

A selection hyper-heuristic controlling a set of perturbative operators typically consists of two key components: heuristic selection and move acceptance methods. In this work, we have combined an online reinforcement-learning heuristic selection mechanism, similar to multi-armed bandit strategies studied in hyper-heuristics [10], with Simulated Annealing and Great Deluge move acceptance methods as described above.

Four different perturbative neighbourhood operators are implemented as low-level heuristics and the selected operator is applied to the incumbent solution at each iteration of the hyper-heuristic.

- **LLH1: Magnetic Move.** Selects a random project and attempts to move it to a timeslot immediately preceding or following another project involving the same staff member.
- **LLH2: Repair Gap.** Identifies the staff member with the current largest idle-time gap and applies a magnetic move to reduce that gap.
- **LLH3: Consolidate Day.** Locates a staff member whose total workload falls below the configurable threshold and attempts to migrate all of their assigned projects into a single day.
- **LLH4: Repair Heavy Balance.** Detects a staff member with an overloaded and unbalanced schedule (above the threshold) and shifts one of their projects from the heavier day to the lighter day.

4.4 Heuristic Selection Method

The designed hyper-heuristics employ an online reinforcement learning mechanism to dynamically learn which operators are most effective at different stages of the search. To keep the computational overhead low, the search process is divided into discrete learning segments, or *epochs*, each consisting of 50 iterations. A segment size of 50 offers a suitable balance: it provides enough samples to measure operator performance while still reacting quickly to changes in the fitness landscape. During each epoch, the algorithm records the number of invocations of each operator and assigns a stratified reward based on the quality of the resulting move. At the end of each 50-iteration epoch, the average performance p_{avg} of each operator is computed by dividing its accumulated reward by

its number of applications. The selection probability weight for the next epoch is then updated using an exponential moving average:

$$w_{\text{new}} = (w_{\text{old}} \times (1 - \gamma)) + (p_{\text{avg}} \times \gamma),$$

where γ is the reaction factor, set to 0.5. This balance between historical memory (w_{old}) and immediate feedback (p_{avg}) allows the algorithm to shift operator preference as the nature of the timetabling bottleneck evolves. To avoid premature convergence and prevent any single operator from dominating the search, a hard lower bound of 0.05 is imposed on all weights. This ensures continued exploration across the solution space.

4.5 Performance Comparison of Hyper-heuristics Using Simulated Annealing and Great Deluge

To compare the performance of hyper-heuristics, random realistic problem instances with varying characteristics in terms of size and conflict density were generated based on past datasets[1]. The problem instances range from 100 to 180 projects and represent different staff–project interlock patterns. The characteristics of each problem instance from the demo scheduling aspect are provided in Table 1.

Table 1. Structural characteristics of the five randomly generated datasets (problem instances).

ID	$\lvert S\rvert$	$\lvert M\rvert$	$\lvert S \cap M\rvert$	$\lvert P\rvert$	**Density**
1	60	40	20	180	21.6%
2	50	30	24	125	31.4%
3	42	20	6	160	37.3%
4	30	20	8	100	41.6%
5	49	35	21	150	42.8%

Each hyper-heuristic algorithm with a different move acceptance method was run ten times using the same computational budget of 7,000 maximum evaluations. Across all problem instances, the average computation time per run was 19.71 seconds for Simulated Annealing and 20.56 seconds for Great Deluge, with individual instance averages ranging from 14.11 to 24.89 seconds. This rapid execution further validates the system's suitability for practical, real-time administrative usage.

Table 2 summarises the mean penalty cost (lower is better), based on the shared multi-component objective function, and Figure 1 provides an overview of box plot illustrations for the SA and GD-based hyper-heuristics, run on each instance.

[1] link to the github repository: https://github.com/neelg148/TestingDatasets

Table 2. Performance comparison of SA and GD-based hyper-heuristics for each of the five datasets based on mean cost and best cost, obtained over the ten runs.

ID	SA Mean Cost	GD Mean Cost	SA Best Cost	GD Best Cost
1	**1,593,100**	1,616,300	**1,022,500**	1,228,000
2	1,080,850	**711,650**	**472,500**	542,500
3	**2,139,950**	2,827,800	**1,800,000**	2,315,500
4	1,032,550	**463,950**	693,000	**225,500**
5	1,708,750	**1,597,400**	1,219,000	**1,140,000**

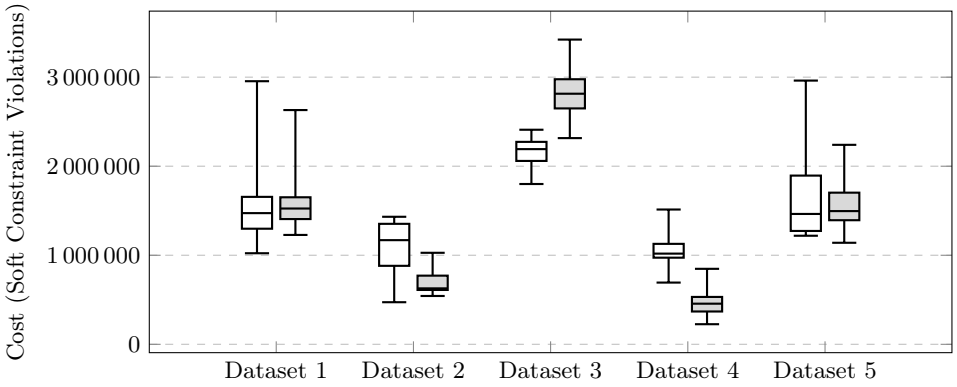

Fig. 1. Box-plot illustrations comparing the distribution of final penalty costs from 10 runs across five problem datasets. **White boxes** represent Simulated Annealing (SA) and **grey boxes** represent Great Deluge (GD).

This study illustrates the following observations, supported by prior findings on the behaviour of local search heuristics [9].

- **All hard constraints are satisfied**, as mean costs in Table 2 indicate mean soft constraint violations, averaged over the 10 runs for each instance.
- **SA-based hyper-heuristic obtains the best solution in 10 runs for 3 problem instances** (Datasets 1, 2, 3), performing slightly better than the GD-based hyper-heuristic as illustrated in Figure 1.
- **GD-based hyper-heuristic excels on smaller, compact search spaces** (Datasets 2 and 4), providing statistically significant improvements due to its deterministic descent based on their average performances.
- **SA-based hyper-heuristic outperforms GD-based hyper-heuristic on dense, high-interaction datasets** (Dataset 3), due to its ability to escape deep local minima via probabilistic acceptance and reheating based on their average performances.
- **Neither algorithm universally dominates**, reinforcing the system's design decision to include both methods and allow the user to choose based on dataset characteristics based on average performance comparison.

5 Conclusion

This paper presents an integrated software platform [2] that automates the full project coordination workflow, with emphasis on solving the complex demonstration scheduling problem using two high-performing hyper-heuristics. Through an interactive interface (see Figure 2), users can explore the scheduling engine, compare algorithmic behaviours, and observe real-time optimisation outcomes. The system demonstrates how heuristic optimisation, careful data engineering, and user-centered design can be combined to improve high-stakes academic administrative processes.

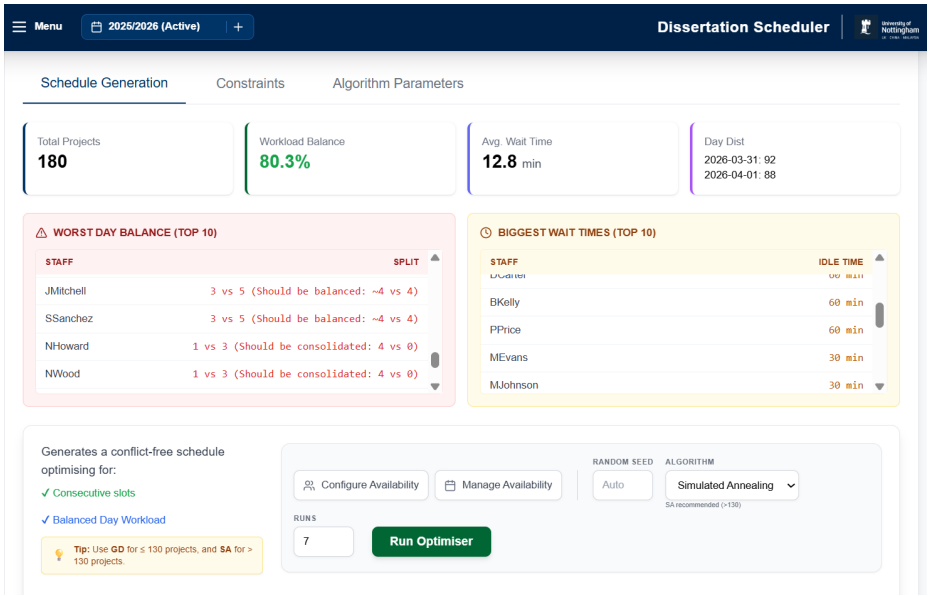

Fig. 2. The interactive user interface of the demonstration scheduling platform.

The system's usability and core functionality were thoroughly tested and validated through User Acceptance Testing, and the software is scheduled for full real-world deployment in the upcoming academic year. It will be installed locally on the module convenor's hardware to manage the live dissertation cohort.

During the design phase, exact optimisation techniques such as Integer Linear Programming (ILP) were also considered. However, these approaches were ultimately deemed impractical due to their susceptibility to combinatorial explosion when applied to the full-scale NP-hard scheduling problem. Consequently, their exploration has been deferred as a direction for future work. A further avenue for

[2] Source code of the implementation: https://github.com/neelg148/Dissertation-Management-Tool

future work is to investigate the application of algorithm selection techniques [14] to identify the most suitable algorithm based on instance-specific features of the demo scheduling problem.

References

1. Burke, E., Kendall, G., Soubeiga, E.: A survey of hyper-heuristics. In: *Handbook of Metaheuristics* (2003)
2. Burke, E., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling. *Artificial Intelligence Review* **9**, 451–457 (2004)
3. Bykov, Y.: The great deluge algorithm and its application to the course timetabling problem. *PATAT* (2003)
4. Ceschia, S., Gaspero, L., Schaerf, A.: Educational timetabling: Problems, benchmarks, and state-of-the-art results. *Artificial Intelligence Review* **(308)**1 (2023)
5. Connolly, T., Begg, C.: *Database Systems: A Practical Approach*. Pearson (2014)
6. Drake, J.H., Kheiri, A., Özcan, E., Burke, E.K.: Recent advances in selection hyper-heuristics. Eur. J. Oper. Res. **285**(2), 405–428 (2020).
7. Fielding, R.: Architectural styles and the design of network-based software architectures. PhD Thesis, UC Irvine (2000)
8. Garey, M., Johnson, D.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman (1979)
9. Jackson, W., Özcan, E., John, R.: Move Acceptance in Local Search Metaheuristics for Cross-domain Search. *Expert Systems with Applications* **109** 131–151,(2018)
10. Jackson, W., Özcan, E., John, R.: Tuning simulated annealing for cross-domain search. *Proc. of IEEE Congress on Evolutionary Computation (CEC)*, 1055–1062 (2017)
11. Kheiri, A., Özcan, E.: High school timetabling worldwide using selection hyper-heuristics. *Expert Systems with Applications* **42**(13), 5463–5471 (2015)
12. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated annealing. *Science* (1983)
13. Ochoa, G., Hyde, M., Curtois, T., Vörnkorg, J., Burke, E.K.: HyFlex: A benchmark framework for cross-domain heuristic search. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012, LNCS **7245**, 136–147. Springer, Berlin, Heidelberg (2012)
14. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976).