University of Exeter
DEPARTMENT OF COMPUTER SCIENCE

AUTOMATED UNIVERSITY TIMETABLING WITH ROBUSTNESS

JAMES SAKAL

Submitted by JAMES SAKAL, to the University of Exeter as a thesis for the degree of Doctor of Philosophy in COMPUTER SCIENCE, NOVEMBER, 2023.

This thesis is available for Library use on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

I certify that all material in this thesis which is not my own work has been identified and that any material that has previously been submitted and approved for the award of a degree by this or any other University has been acknowledged.

Signed: [illegible] ....

Abstract

Generating workable timetables for universities is a hard task. For many years, the process was conducted manually, but it has benefited in recent times from greater automation. From graph theoretical and mathematical programming to metaheuristic approximation methods, a multitude of computational approaches have been tried and tested. In parallel to this, benchmarks of artificial problems have arisen to aid the development of novel algorithms. In this thesis, we focus on nature-inspired search processes as applied to the International Timetabling Competition (ITC) 2007 track 3 benchmark. Firstly, we develop an ant colony optimiser based on the MAX-MIN ant system (MMAS). Consideration is given to the ordering of lecture assignments in the construction graph. We aim to discover how permuting lectures affects overall performance and what features of the problem are important. Through a machine learning phase, a permutation predictor is then devised for unseen instances, which is assessed both with and without local search enhancements. Other aspects of the MMAS as relate to performance and efficiency are also examined, such as dynamic constraint handling and partial function evaluations. Initial results are mixed but show promise for the smaller problems, suggesting that further expansion of the methodology could be worthwhile. Secondly, a many-objective approach to the same problem is explored. Noting that treatments of the ITC2007 are typically single-objective, we instead cast individual constraint violations as distinct objectives. Inspiration is taken from NSGA-III, which is discretised and otherwise adapted for the problem at hand. Working under the assumption that decision maker preferences are not known in advance, we attempt to generate good approximations to the Pareto set, relying on hypervolume as the key performance indicator. We show that feasible solutions can be attained quickly in a constructive phase, and that targeted objectives can be optimised to zero (where possible) in a second phase. Trade-offs between the different objectives are analysed. Scalarised results are compared with published single-objective approaches and found to be competitive on some problems. More importantly, our contribution represents a method of attaining an approximation set of unique non-dominated solutions that has been lacking thus far in the literature. In the final chapter, we build upon the many-objective solver by considering symmetry and redundancy in the search space. An encoding conversion is proposed to eliminate equivalent solutions. We also investigate and visualise the plateaus in the search landscape and propose a genotype diversity mechanism to facilitate the escape from these regions. This leads to a statistically significant improvement in results. Further operators are introduced and we examine the trade-offs between computation time and increased complexity within the perturbation and selection operators. Lastly, we explore the idea of robustness of solutions, developing a method to quantify this property, before adding it to the problem as an additional objective. Our solver copes well with the extra dimension of robustness, offering promising results for future research.

Acknowledgements

First and foremost I would like to thank my supervisors, Professor Jonathan Fieldsend and Professor Edward Keedwell. They have been unflinchingly supportive over the past four years and I have gained so much from their expert insight week to week. On an academic level, they constantly challenged me, asked all the right questions, encouraged me to pursue new ideas but reigned me in when I needed focus. On a human level, I found them extremely personable and I always left our meetings with a renewed sense of direction and motivation. By chance or by design, they also complemented each other wonderfully, offering equally valuable feedback but from different angles. Overall, I could not have wished for a better pair of academics to have fulfilled their roles.

In setting me up for the PhD journey, the following deserve credit. Professor Alan Champneys at the University of Bristol, who supervised my final year undergraduate project, and Professor Peter Higgins at the University of Essex, who supervised my Masters dissertation. Both kindly provided positive references that helped me towards taking the next step. Professor Chunbo Luo and the rest of the panel also put me at ease when I was first interviewed at the University of Exeter for this project. My decision to commit fully to it was aided in no small part by the many enlightening conversations with those who had trodden the PhD path before me. In particular, Dr. Mark Yarrow, Professor Mark Barrett, Dr. Stephanie Stanton, Dr. Simon Clark and Marianne Stewart.

At the University of Exeter, I was privileged to work in an environment that was both friendly and intellectually stimulating. Various peers, colleagues and senior lecturers deserving of special mention are Dr. George De'Ath, Dr. Tinkle Chugh, Dr. Alberto Moraglio, Dr. Hugo Barbosa, Melike Dila Karatas, Dr. Abdulaziz

Almutairi, Sha Mos and Matthew Hayslep. The interactions we had — whether in reading groups, seminars, taught lectures, conferences or in social settings — widened my perspective and gave me a feeling of belonging to something much larger than just my own project. Similarly, I will remember with fondness sharing the ups and downs of PhD life with fellow candidate Liam Watts, both during and outside of our PTA sessions. Aileen MacGregor, Dr. Jamie Walker, George Humby, Ed Horncastle and Melanie Dennig were also influential colleagues in the sphere of teaching, and I thank them for allowing me to develop my skills in this area. For the opportunity to disseminate my research to a public audience, I also valued working with Dr. Clodomir Joaquim De Santana Junior, Chryssa Brown, Dr. Diogo Pacheco, Ana Maria Jaramillo and the other Pint of Science organisers and speakers. The friendships I have made transcend research alone.

I am grateful to the Engineering and Physical Sciences Research Council (EPSRC) who provided funding for my thesis, and to the Doctoral College, the Researcher Development Program, IT support, and the many anonymous reviewers whose comments helped improve my published papers.

Outside of the university setting, I must unquestionably thank my family. They provided vital emotional and practical support throughout what has been a gruelling period of study, while at the same time learning more about automated timetabling than they perhaps ever wished to. Welcoming me back home during the pandemic and setting aside a quiet study space was greatly appreciated. Without their backing I may not have been able to push through the myriad low moments and keep going. Also from outside the university, I would like to thank Carl Williams, whose friendship has meant a lot through some very trying times. Despite working in a different field, he has taken a keen interest in the technicalities of my research. His patience, encouragement and good humour throughout times of struggle will never be forgotten. I must also commend Lucy Cross, for her engagement with my Research Showcase Week poster and her valiant attempts to solve the timetabling puzzle therein.

To DT, WC, KR, LA and others who were compassionate, understanding and rational during the most difficult of times, I also offer my gratitude. Thank you for believing in me, and I hope I have repaid that faith. To my extended network of family and friends, too numerous to mention individually, I appreciate you putting up with me while I was engrossed in my PhD and otherwise unavailable.

Last but not least, I am forever indebted to the fine pubs and publicans of Exeter and East Devon — for being the ‘fourth emergency service’. Whether offering me a refuge from my buggy code, or injecting just the right amount of randomness (beer) into my thought processes, you truly oiled the wheels of the PhD adventure.

Contents

List of Figures

List of Tables

1. Introduction

For as long as humans have been engaged in cooperative, organised endeavours, timetabling has been an integral component. From offices to transportation networks, streamlined operational procedures are crucial for ensuring the smooth functioning of complex systems. Before the advent of computing, timetables were constructed manually, through a combination of intuition, trial and error, and applied mathematical knowledge. These foundational principles still have a part to play today. At the same time, modern computer hardware has opened the door to a deeper exploration of the science underpinning timetabling. A more formal description of the process of timetabling can be found in the literature:

"The allocation of given resources to specific objects being placed in space time, in such way as to satisfy as nearly as possible a set of desirable objectives, subjected to constraints." (Wren, 1995).

This definition, while helpful and concise, lends itself to a vast array of different scheduling problems, ranging from railways to rostering. The branch that comprises timetabling for schools or other academic institutions is known as educational timetabling. Some of its most commonly recognised sub-divisions are shown in Figure 1.1 (Frausto-Solis et al., 2006, Pillay, 2012).

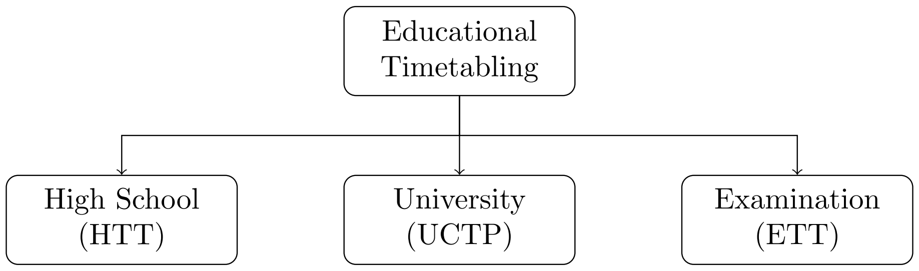


Figure 1.1: A taxonomic tree of educational timetabling problem domains.

Of these three, this thesis is concerned with the university course timetabling problem (UCTP), which was first addressed in the literature by Gotlib, 1963. The UCTP has long been understood as belonging to a class of problems known as *NP*-complete (Cooper and Kingston, 1996), meaning there exists no known polynominal-time method for solving it. As processing power has advanced, practitioners have instead developed creative and innovative search algorithms that are tailored to the problem at hand. Interest in the UCTP has grown considerably in recent decades. A survey of the field in 2019, for example, reported a 10-fold increase between 1990 and 2015 in the number of papers being published on the subject of timetabling in higher education (Oude Vrielink et al., 2019). Consequently, the diversity of both test problems and approaches has increased too.

1.1 Motivations and objectives

Solving the UCTP can be seen as both a practical and a purely academic endeavour. Artificial problem models are informed by real world requirements, while the algorithms used to solve them may be abstract in nature or borrow from theoretical findings in other domains. The work in this thesis leans more toward academic analysis. Understanding and developing the mechanisms of action of the algorithms is a thread that runs throughout. At the same time, the UCTP is not entirely removed from the wider context of being an operational problem. Consideration is given to the fact that most timetables are created as part of an annual cycle, and that this allows for relatively liberal compute times. The sizes of test problems studied are also of a realistic departmental scale, for which a complete enumeration of solutions is not practicable. Therefore trade-offs between run time and quality of solutions must be weighed up in the context of overall efficiency.

Another important motivation relates to automation. In seeking to fully automate the generation of good timetables, algorithms should be highly problem-agnostic and avoid the need for excessive parameters or tuning. And while automation implies a lack of human-in-the-loop, the presence of a final decision maker is an important feature of later chapters. Particularly in many-objective work, the algorithm should satisfy the requirements of the institution neutrally so that a human in a timetabling office can then choose based on their subjective preferences. These in turn may be influenced by factors that are difficult or impossible to encode algorithmically, such as staff retirements or building works. The evaluation of many objectives simultaneously is under-represented in the UCTP literature. The later direction of this thesis is therefore motivated by probing this gap. Another real world issue is that of the requirements sometimes changing subsequent to the drafting of a timetable, or even during the term itself. This acts as motivation for the complementary work on the so-called robustness of solutions. Once again, there is a scarcity of work in the literature exploring this feature in a many-objective setting.

In reviewing the literature, a picture is drawn of both the history of timetabling research and its current breadth. Based on this, the methodology in the technical chapters incorporates simple heuristics, metaheuristics and evolutionary techniques — all of which have shown great promise. The effective use of randomness, and how generic algorithms can be enhanced with domain-specific knowledge or operators are both subjected to empirical study. In the first chapter of technical work, a gap is identified regarding the best way to order the assignment of lectures within the context of a metaheuristic solver. An ant colony optimiser is designed in order to examine this aspect in detail. In terms of scope, the ant colony investigation is focused on the twin areas of constraint handling and the aforementioned assignment ordering, the two of which are inextricably linked. Gaining insight into the mechanisms at play 'under the hood' is therefore more important than an absolute comparison with best-known results. In this context, a machine learning pipeline is devised with the aim of predicting beneficial assignment orderings. If successful, the concept could in theory be ported to other systems in which ordering plays a role.

The thesis then develops in its treatment of the UCTP from single-objective to many-objective. While a limited number of bi- or multi-objective approaches have been proposed in the literature, a proposal for a many-objective UCTP solver that can be tested against a well known benchmark is notably lacking. Perhaps one factor contributing to this is the myriad challenges that arise when raising the number of objectives. These include:

- A lessening of discriminatory power of the dominance relation.
- Intricate inter-objective conflicts.
- Diversity preservation issues.
- Scalability to large problem instances.
- Higher time and memory complexity.
- Visualisation and analysis difficulties.

Consideration is given to all of these in the development of a many-objective solver. Other design and implementation choices such as the encoding scheme, data structures and redundancy are also scrutinised. The goals and scope in these latter chapters are to create an optimiser that is reliable across a range of problem inputs, yet not overly-complicated in design. Empirical testing is carried out at all stages of development, on a well-known benchmark set. Ultimately, while timetable quality should converge closely to the best-known single-objective results, it is acknowledged that the added complexity of the many-objective treatment makes this a difficult task. More importantly then, the optimiser should provide a good spread of solutions that represent the different trade-offs inherent in the problem. As such, appropriate metrics such as hypervolume and cardinality of the solution set are used primarily to quantify the success of the approach. Returning a well-populated trade-off front approximation becomes all the more crucial when, ultimately, a robustness objective is added too. Robustness is the property of a timetable that explains how resilient it is to unforeseen changes or disruptions to the scheduling requirements. Practical ways of defining and optimising robustness are explored.

Driven by these motivations, the following specific, measurable and achievable research objectives, as identified for each chapter, are:

Chapter 2

- To conduct an extensive literature review in order to understand how the history of timetabling research has lead to its present state. To identify how the UCTP is modelled, which strategies and ideas have been co-opted successfully for it, and what gaps exist in the research.

Chapter 3

- To build and adapt an ant colony optimiser for the UCTP, incoporating components such as dynamic constraint handling. To establish the extent to which the order of lecture assignments impacts on results and what patterns may be in evidence. To generate a training set of problems that can be tested over all lecture permutations and employ machine learning techniques to learn from the results obtained. To scale this to larger benchmark problems as a predictor of beneficial permutations. Finally, to assess the performance of the ant colony and predictor, both with and without the addition of local search.

Chapter 4

- To examine the efficacy and potential biases of constructive heuristics in initialising a population, as the first phase of a UCTP solver. To adapt a core evolutionary algorithm as the second phase of a UCTP solver, incorporating components such as selection and perturbation operators, constraint handling and efficient evaluation and archiving. Firstly, to test the system with fully relaxed hard constraints, where both hard and soft violations are objectives to minimise. Secondly, to test the system with a stricter enforcement of hard constraints. To visualise and analyse the best approximations to the Pareto front for all instances.

Chapter 5

- To define a set of distance metrics between solution designs. To generate data for different diversity preserving techniques that operate in the genotype space based on these. To reduce search space redundancy with a new encoding scheme.

To introduce additional perturbation operators and assess their complexity and performance. To examine the role tournament selection plays and assess performance based on different selection criteria. To define a robustness metric and incorporate this as a new objective. Finally, to visualise and assess the trade-offs between the quality objectives and robustness.

1.2 Contributions

In the pursuit of these aims, a number of contributions were made:

Chapter 3

- A dynamic hard constraint handling mechanism for an ant colony optimiser was proposed and calibrated using a theoretical value τ_{shelf}. This proved highly successful in delivering feasible results.

- A novel machine learning pipeline was proposed in order to learn and predict beneficial orderings for lectures in the context of an ant colony construction graph.

Chapter 4

- A novel, many-objective solver for the UCTP was developed. This led to publishing the first known results for the chosen benchmark to approximate the 4-dimensional Pareto front.

- The best-known and optimal result was equalled for one problem instance by this solver, under the same time conditions.

Chapter 5

- A novel, tree graph-based visualisation was introduced to illustrate the uniqueness of individual solutions in a population with regard to their various properties.

- A novel approach to integrating genotype crowding measures with more traditional phenotype crowding was proposed. This proved successful in promoting exploration.

- A novel *standard form* encoding was developed in order to eliminate equivalence issues found in commonly-used direct encodings for the UCTP.

- Relying on this encoding, a new hierarchy of discriminatory criteria for the tournament selection operator was proposed, which was successful in promoting diversity.

- The first results were returned for a many-objective optimiser incorporating robustness for the chosen benchmark.

- To quantify robustness, a novel metric was utilised. Despite being called sparingly and for a small fixed set of disruptions, this metric was also shown to be a consistent indicator of more generalised robustness.

1.3 Publications

During this PhD, research has been disseminated through the following publications.

The preliminary findings in Chapter 3 were published as:

J. Sakal, J. E. Fieldsend and E. Keedwell (2021). ‘Learning Assignment Order in an Ant Colony Optimiser for the University Course Timetabling Problem.’ In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 77–78. DOI: 10.1145/3449726.3459534.

In the context of conference proceedings, material presented in Chapter 4 was published as:

J. Sakal, J. E. Fieldsend and E. Keedwell (2022). ‘Towards a Many-Objective Optimiser for University Course Timetabling’ In: *Proceedings of the 12th International Conference on Artificial Evolution*, pp. 171-184.

In the context of the *Lecture Notes in Computer Science* series, material presented in Chapter 4 was published as:

J. Sakal, J. E. Fieldsend and E. Keedwell (2023). ‘Towards a Many-Objective Optimiser for University Course Timetabling.’ In: *Lecture Notes in Computer Science*, Volume 14091, pp. 133-144. DOI: 10.1007/978-3-031-42616-2_10.

Material presented in Chapter 5 was published as:

J. Sakal, J. E. Fieldsend and E. Keedwell (2023). ‘Genotype Diversity Measures for Escaping Plateau Regions in University Course Timetabling’ In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 2090-2098. DOI: 10.1145/3583133.3596334.

1.4 Thesis outline

The remainder of this thesis is arranged as a series of chapters. Chapter 2 provides background in the form of a literature review. Focus is placed on the UCTP and a particular benchmark that is used in later work. However, other literature peripheral to this is called upon when context requires it. Thematically, the chapter first introduces the UCTP, before elucidating its formulations, benchmarks, and offering a breakdown of different approaches to solving it. Chapter 3 features technical work on the ant colony solver, preceded by a short section on the background specific to this metaheuristic. Chapter 4 introduces the work on a many-objective solver, with its various components expounded upon in the methodology section. Experiments with relaxation and strict enforcement, graphical plots, and conclusions follow. The solver is developed further in Chapter 5, which details investigations into diversity, operators and finally robustness. Chapter 6 provides a summary and conclusions to the thesis, before a full list of references is given in the bibliography.

2. Background

This aim of this chapter is to furnish the reader with the necessary background to support the technical work in later chapters. Section 2.1 introduces the different types of university course timetabling problem (UCTP) and how they are formalised. Section 2.2 provides a chronology of benchmark problem sets that have been developed and adopted by practitioners. Section 2.3 gives an overview of some of the various popular technical approaches that have been used to solve the UCTP. Some state of the art results are tabulated in Section 2.4, before Section 2.5 draws together the various themes and offers a summary.

2.1 University course timetabling problem

In its broadest sense, the university course timetabling problem (UCTP) is a class of problems in which an assignment and schedule of lectures is sought for a particular university in order to create a working timetable (Rossi-Doria et al., 2003). A satisfactory solution is one that is both feasible with regard to certain physical constraints and desirable from an operational point of view. Figure 2.1 extends the taxonomy tree from Figure 1.1 in order to show the further sub-divisions of the UCTP domain.

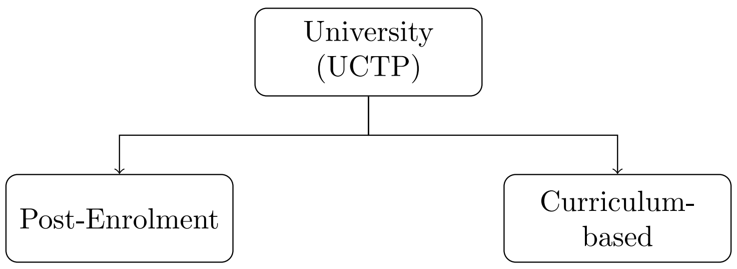


Figure 2.1: A taxonomic tree of UCTP problem types.

While post-enrolment and curriculum-based timetabling are superficially similar problems, they have some crucial differences. In post-enrolment, a timetable is constructed with prior knowledge of which lectures students have enrolled in. Conflicts arise when lectures with students in common are scheduled in the same period. Also, the suitability of a room for certain lectures is determined by a combination of its seating capacity and its features. In addition, precedence relations are imposed, meaning that certain lectures must be scheduled before or after others. All of these constraint requirements are defined on a lecture-by-lecture basis and there is no higher structural organisation of lectures. In contrast to this, all lectures in curriculum-based timetabling are grouped into courses, which in turn are members of a curriculum or curricula. Student commonality is therefore implicit, and lectures cannot be scheduled in the same period if their courses belong to one or more of the same curricula. It was noted by Ceschia et al., 2023 that students in post-enrolment and curricula in curriculum-based timetabling are, in effect, equivalent entities. What fundamentally differentiates curriculum-based timetabling though is that properties are always defined at the level of a course rather than an individual event.

2.1.1 Constraint modelling

The UCTP is modelled by discrete variables and can be considered as a combinatorial optimisation problem. A feasible solution to a problem instance is one that assigns all lectures to a timeslot and a room without violating any ‘hard constraints’. Typically, hard constraints ensure that rooms are not double-booked and that lecturers and students do not have clashes that would make their commitments impossible to fulfil.

From a human operational perspective, not all of the timetable solutions within this feasible space are equally attractive, however. To discern between different feasible solutions, it is common also to use ‘soft constraints’ — constraints for which is it merely preferable, rather than necessary, to satisfy. Examples from the literature include the number of lectures a student has in a day or the physical distance between buildings holding consecutive lectures (Müller et al., 2018). A cost can be defined as some function of the number and magnitude of violations of these soft constraints, typically given as a positive integer value. Since a lower cost signifies a more desirable solution, the optimisation process then becomes a minimisation problem on the value given by this function. A timetable with no violations of either kind is known as a no-cost or ‘perfect’ solution. If such a solution exists, it is by definition the best (or joint best) achievable timetable for a particular problem. In practice though, the optimal solution is rarely a perfect one.

2.2 Formulations and benchmarks

Problem formulations and data sets used to test approaches to the UCTP fall into one of two categories: Real world and artificially generated (Ceschia et al., 2023). In a real world scenario, the timetabling needs of a particular university may be idiosyncratic or unique to itself. Any technique used to derive a useful solution may therefore not be generally applicable to other universities (Ariyazand et al., 2022). However, such real world data sets do have the advantage of capturing nuanced and challenging aspects of timetabling that may lead to novel and important developments in research. In addition to providing practical solutions for particular institutions, they have also informed the design of commercial software solvers. Artificial problems, on the other hand, often dispense with the higher complexities of real world scenarios. These problem sets are appealing to researchers because of their simplicity and generality. They serve as a useful test-bed on which to run new algorithms, collaborate and compare like-for-like results. Through repeated use over time, a number of these have become established and well-known benchmarks. The following section documents some important problem sets from the literature, from both categories.

2.2.1 Metaheuristics Network 2000-2004

In a survey of the state of the art, Burke et al., 1997 lamented both the lack of a standardised description of the UCTP and a benchmark for cross-comparing timetabling algorithms. It was in the context of this research landscape that the Metaheuristics Network was formed. A European Commission project involving collaboration between five European institutes (Rossi-Doria et al., 2006), its goal was to empirically compare the performance of different metaheuristics on combinatorial optimisation problems, including university timetabling. From the work of Rossi-Doria et al., 2003, a standardised UCTP formulation was devised, which would enable results from different researchers to be compared more evenly (Alhuwaishel and Hosny, 2011). An artificial instance generator, referenced in Lewis and Paechter, 2005, was created by Ben Paechter to reflect aspects of Napier University's timetabling. It worked on eight input parameters (events, rooms, features, features per room, percentage of features used, students, max events per student and max students per event) and a random seed. The generator initially yielded a benchmark set of twelve UCTP problem instances, designated as "easy", "medium" and "hard" (later adapted in terminology to "small", "medium" and "large") according to their complexity and size. This set contains five, five and two of each type respectively and all have at least one perfect solution. It appears in the literature variously as the Paechter benchmark, the Socha et al. (2002) set, Socha benchmark or Rossi-Doria et al. (2003) set.

2.2.2 The Sixty Instances.

In Lewis and Paechter, 2005, sixty further instances were generated by the Paechter generator. Again, these were subdivided into three sets of "small", "medium" and "large", with 20 instances in each category. These have been made freely available for testing, and continue to be popular (Song et al., 2018) with the instances of highest complexity being of interest to pure feasibility solvers (see also Section 2.3.2.4).

2.2.3 International Timetabling Competition 2002

The work of the Metaheuristic Network resulted in the establishment of the International Timetabling Competition (ITC) in 2002. For this, standardised data formats (with the file extension `.tim`) and the rules formalised by Ben Paechter were utilised. The first set, in the post-enrolment format, comprised 20 instances. These were released to the competitors at staggered intervals in the run-up to the competition and are now maintained online. They continue to be a popular benchmark, referenced for example in Badoni et al., 2014, in which the performance of two algorithms was tested.

2.2.4 International Timetabling Competition 2007

Post-enrolment timetabling featured again, in a slightly modified form, as track 2 of the second ITC edition in 2007. Of greater interest in this thesis though is the track 3 formulation, curriculum-based (Gaspero et al., 2007). A set of 21 instances, nominally `comp01` to `comp21`, were modelled on the real world timetabling problem of the University of Udine. For the sake of generality, a range of requirements from different faculties were included. The magnitude of the instances, which have lecture counts in the hundreds, is mostly on a departmental scale. The model was built around a number of entities and variables, for which we use the following notation:

- **Days.** A day with index i is denoted d_i. The number of days in a week that are available for teaching is fixed by the problem instance. The set of days is $\mathcal{D}$.

- **Timeslots.** Each day is divided into an equal and fixed-sized set, t, of timeslots.

- **Periods.** A period, p_i, is a day $\times$ timeslot. The set of periods is $\mathcal{P}$.

- **Rooms.** A room, $r_i \in \mathcal{R}$, is defined by its seating capacity, $cap(r_i)$ which ought not to be exceeded. In all other respects, any room is suitable to host any lecture. Adopting the terminology used in Lewis, 2006, a room/period pair is referred to as a *place*, and the action of assigning a lecture to a particular room/period as *placing*.

- **Lectures.** Lectures, l_i, are events that must be assigned to a suitable place. The set of all lectures is $\mathcal{L}$.

- **Courses.** Every lecture belongs to exactly one course, C_i, while a course can comprise any number of lectures. Each course C_i has a fixed number of enrolled students, $stud(C_i)$ as well as several other properties. The set of all courses is $\mathcal{C}$.

- **Teachers.** Exactly one teacher is pre-assigned to each course, while teachers can teach multiple courses. The unit set consisting of the teacher of a course C_i is denoted $teach(C_i)$ and the set of all teachers is $\mathcal{T}$.

- **Curricula.** A curriculum, u_i, is a set of courses. A course may belong to multiple curricula but must belong to at least one. Any two courses in the same curriculum implicitly have students in common and the set of all curricula is $\mathcal{U}$.

Figure 2.2 offers a visualisation of the relationship between curricula u_1, u_2, courses $C_1, \ldots, C_4$ and lectures $l_1, \ldots, l_{16}$ for a published toy example from the ITC2007 track 3. Table 2.1 meanwhile gives the characteristics of the `comp*` instances.

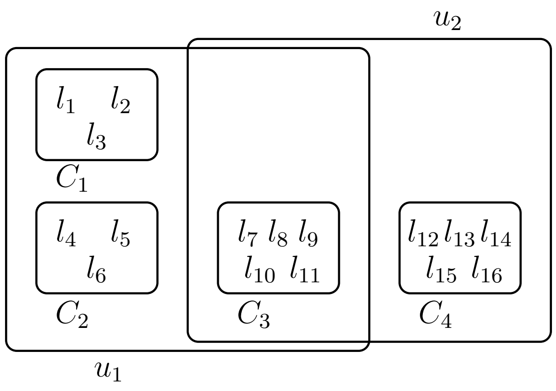

u_2
l_1 l_2 l_3
C_1
l_4 l_5 l_6
C_2
l_7 l_8 l_9 l_{10} l_{11}
C_3
l_{12} l_{13} l_{14} l_{15} l_{16}
C_4
u_1

Figure 2.2: A Venn diagram showing the set relations of curricula u_1, u_2, courses $C_1 \ldots C_4$ and lectures $l_1 \ldots l_{16}$ for an ITC2007 track 3 toy example. Lecture l_{11}, for instance, belongs to course C_3, which in turn belongs to both curricula u_1 and u_2.

The formulation also prescribed a set of hard and soft constraints, which are denoted **H** or **S** and named and numbered as below. The penalty points incurred for violations of the soft constraints are also given.

- **H1: AllLectures.** All lectures must be assigned to a distinct place.

- **H2: RoomOccupancy.** A room can host a maximum of one lecture per period.

- **H3: CurriculumConflicts.** Lectures of courses belonging to a common curriculum cannot be scheduled in the same period.

- **H4: UnavailablePeriods.** One or more periods may be pre-defined as unavailable for a particular course C_i. As such, no lecture belonging to that

Table 2.1: Characteristics of the original ITC2007 track 3 problem instances, ordered by number, and headed using the notation introduced in this section. From left to right, the table shows name, number of courses, lectures, curricula, teachers, days per week, timeslots (or periods per day), rooms and unavailable periods.

Name	$\|\mathcal{C}\|$	$\|\mathcal{L}\|$	$\|\mathcal{U}\|$	$\|\mathcal{T}\|$	$\|\mathcal{D}\|$	$\|t\|$	$\|\mathcal{R}\|$	$\|\mathcal{N}\|$
`comp01`	30	160	14	24	5	6	6	53
`comp02`	82	283	70	71	5	5	16	513
`comp03`	72	251	68	61	5	5	16	382
`comp04`	79	286	57	70	5	5	18	396
`comp05`	54	152	139	47	6	6	9	771
`comp06`	108	361	70	87	5	5	18	632
`comp07`	131	434	77	99	5	5	20	667
`comp08`	86	324	61	76	5	5	18	478
`comp09`	76	279	75	68	5	5	18	405
`comp10`	115	370	67	88	5	5	18	694
`comp11`	30	162	13	24	5	9	5	94
`comp12`	88	218	150	74	6	6	11	1368
`comp13`	82	308	66	77	5	5	19	468
`comp14`	85	275	60	68	5	5	17	486
`comp15`	72	251	68	61	5	5	16	382
`comp16`	108	366	71	89	5	5	20	518
`comp17`	99	339	70	80	5	5	17	548
`comp18`	47	138	52	47	6	6	9	594
`comp19`	74	277	66	66	5	5	16	475
`comp20`	121	390	78	95	5	5	19	691
`comp21`	94	327	78	76	5	5	18	463

course can take place in such a period. This set of periods is denoted as $unav(C_i)$, while the union of these sets of over all the courses is denoted $\mathcal{N}$.

- **H5: TeacherConflicts.** Lectures of courses sharing a common teacher cannot be scheduled in the same period.

- **S1: RoomCapacity.** The number of students should be less than or equal to the capacity of the room holding the lecture. Each student above capacity counts as one point of penalty.

- **S2: MinimumWorkingDays.** The scheduling of the lectures of a course C_i should be spread over a minimum number of distinct days, denoted $mwd(C_i)$. Each day below this minimum counts as five points of penalty.

- **S3: CurriculumCompactness.** Lectures should be adjacent in timeslot to some other lecture from the same curriculum. Any lecture that is 'isolated' in this respect counts as two points of penalty. The last timeslot in a day does not count as adjacent to the first timeslot in the next day.

- **S4: RoomStability.** Within each course, lectures should be held in the same room. Every distinct room beyond the first used for a course counts as one point of penalty.

The soft constraint penalty scheme outlined above provides a method by which to express the overall quality of a solution. For feasible solutions, the cost is simply the sum of the penalties incurred for *soft constraint violations*, a value referred to by SCV. A high SCV reflects a low quality and vice versa. In certain circumstances, it may also be useful to ascribe a cost to infeasible solutions. The archetypal use case for this is when an optimisation process is allowed to traverse the infeasible space in search of disconnected regions of feasibility. In Mayer et al., 2012, the *distance to feasibility* (DTF) was defined as the sum total of students taking lectures that have failed to be scheduled. DTF — described by Schaerf, 1999 as a nominal measure of the magnitude of a solution's infeasibility — can be conceptualised and expressed in other ways, such as the number of courses unassigned, or a sum total of all hard violations. A value representing DTF can then be returned as a summary statistic, or amalgamated with the SCV, at the practitioner's discretion.

An augmentation of this set of soft constraints was proposed by Bonutti et al., 2012. The purpose was both to increase the model complexity and to address some perceived inadequacies of the original definitions. For example, the intention behind **CurriculumCompactness** was to discourage lengthy gaps in a student's day, by penalising any temporally isolated lectures belonging to some curriculum. Consider a 10-timeslot day. The placing of two common lectures in timeslots 1 and 10 would, in isolation, trigger a penalty. If four lectures were instead scheduled in timeslots 1, 2, 9 and 10, no such penalty would be incurred — despite the existence of a time gap of comparable size. In Bonutti's alternative definition of **CurriculumCompactness**, named **Windows**, unwanted time gaps incur the same number of violations as their length in periods. A full list of suggested constraints is given in Figure 2.2, as well as five proposed configurations. Despite the merits of these, the original configuration,

Table 2.2: An extension of the ITC2007 track 3 formulation, from Bonutti et al., 2012. Five configurations were suggested, where a configuration is the designation of a specified subset of constraints as hard (denoted by **H**) and another subset as soft (denoted by an integer). For soft constraints, the integer gives the value of the penalty incurred for one violation. The prefix label 'UD' in the headings is borrowed from the author, and refers to the University of Udine, whose timetabling the problem was based on.

Constraint	UD1	UD2	UD3	UD4	UD5
AllLectures	**H**	**H**	**H**	**H**	**H**
RoomOccupancy	**H**	**H**	**H**	**H**	**H**
CurriculumConflicts	**H**	**H**	**H**	**H**	**H**
UnavailablePeriods	**H**	**H**	**H**	**H**	**H**
TeacherConflicts	**H**	**H**	**H**	**H**	**H**
RoomCapacity	1	1	1	1	1
MinimumWorkingDays	5	5	-	1	5
CurriculumCompactness	1	2	-	-	1
Windows	-	-	4	1	2
RoomStability	-	1	-	-	-
StudentMinMaxLoad	-	-	2	1	2
TravelDistance	-	-	-	-	2
RoomSuitability	-	-	3	**H**	-
DoubleLectures	-	-	-	1	-

UD2, has remained the most enduring in the literature and is adopted in our work also.

Supplementary problem instance sets have since been created for the track 3 formulation. These are: `DDS*` (7 instances), `test*` (4 instances) (Bonutti et al., 2012), `erlangen*` (6 instances), `EA*` (based on EasyAcademy timetables, originally 12 instances in 2014, to which 11 more were added in 2021) and `Udine*` (9 instances). Of the problems within these sets, `erlangen*` are by some margin the largest, with course counts ranging from 705 to 850 and lecture counts from 788 to 930.

2.2.5 International Timetabling Competition 2011

While the 2011 competition focused on high school timetabling rather than university, it did provide some interesting cross-domain innovations. The 15 constraint types, ranging from lecture spread to student idle times, could be designated as either soft or hard, enabling complete modular control over problem design.

2.2.6 International Timetabling Competition 2019

The facility to mix and match constraint types was carried forward to the 2019 competition. By that year, another trend had become apparent — formulations were moving progressively away from artificially generated sets and towards real world. As confirmed by the organisers at the time, *"We already have an agreement with ten institutions including Purdue University in the USA, Masaryk University in the Czech Republic, AGH University of Science and Technology in Poland and Istanbul Kultur University in Turkey that we can use their data."* (Müller et al., 2018)

The new formulation introduced greater flexibility than the ITC2007 tracks. The inclusion of several novel features also helped to bring theory closer to practise in terms of simulating the complexities of a real world problem. Outlined below are some of the major differences between ITC2007 and ITC2019:

- **Student sectioning.** Previously, student clashes were implied by common curriculum membership of courses. Such violations are explicit in ITC2019, as every student is defined as an entity with individual enrollment requirements.
- **Constraints.** A wider pool of constraints, referred to as 'distributions', was introduced. The majority of these can be applied as either hard or soft, while a smaller number are invariant. Some constraints were defined as pairwise, while others may apply to multiple classes.
- **Teachers.** Teachers are absent from the newer formulation. Instead, they can be modelled either implicitly using the available constraints, or explicitly in the guise of a 'student'.
- **Courses.** Courses are subdivided into 'configs' and then 'subparts'. This allows modelling of different versions of the same course/module, for example undergraduate vs. masters.
- **Classes.** Equivalent to 'lectures' in ITC2007, classes exist within subparts. A more complex hierarchical structure is made possible. Some pairs of classes may have parent-child relations. Any student sectioned to a 'child' class is also

obligated to attend the 'parent' class. This is a convenient way to model a lecture-lab session dependency, for example.

- **Granularity.** Time is discretised into small slots of 5 minutes, meaning that classes can have staggered start times and potentially partial overlaps too.
- **Room location.** Travel distances are defined between rooms. This is to encourage consideration of the physical geography of the campus.
- **Weekly variation.** The ITC2019 allows for different weekly schedules over the course of a semester, rather than a static timetable that is repeated every week.
- **Format.** Due to their tree-like structure, ITC2019 problems are supplied in XML format rather than as plaintext `.ctt` files.

The following hard constraints are embedded within the scoring system:

- **H1:** All students must be sectioned into one class of every subpart of exactly one configuration.
- **H2:** Any parent-child class relations must be respected in the above.
- **H3:** Class limits on number of enrolled students must not be exceeded.
- **H4:** Rooms cannot be used during any pre-defined 'unavailable' time.
- **H5:** Rooms cannot host more than one class at any time.
- **H6:** If a room assignment is demanded for a class, this room must come from the domain of that class. All rooms in such a domain are necessarily suitable for hosting the relevant class, both in their implied facilities and stated capacity.
- **H7:** Classes must be assigned a time from their respective domains.

Note that **H4** and **H5** correspond directly to ITC2007's **H4** and **H2** respectively. **H3** is a stronger variation on the idea of student capacity as seen previously in ITC2007's **S1**. The remainder are prescriptive constraints relating to room and time suitability, precedence and student sectioning — aspects that were either undefined, non-explicit or absent in ITC2007.

The fixed soft constraints in ITC2019 are:

- **S1:** Student conflicts. A conflict occurs when a pair of classes that a student is assigned to has any kind of time overlap. This counts regardless of the total length and/or whether the overlaps are contiguous or not.

- **S2:** Time penalty. Penalty values of zero or greater are associated with each available time in the time domain of a class.

- **S3:** Room penalty. An equivalent penalty is associated with each room in the room domain of a class.

A further 19 flexible constraints (**F1 - F19**), referred to as the aforementioned 'distributions', can be treated as either hard or soft, or omitted entirely. Without reproducing the full definitions here, these are: **SameStart**, **SameTime**, **DifferentTime**, **SameDays**, **DifferentDays**, **SameWeeks**, **DifferentWeeks**, **Overlap**, **NotOverlap**, **SameRoom**, **DifferentRoom**, **SameAttendees**, **Precedence**, **WorkDay(**S**)**, **MinGap(**G**)**, **MaxDays(**D**)**, **MaxDayLoad(**S**)**, **MaxBreaks(**R,S**)** and **MaxBlock(**M,S**)**. Semantically, there is provision for both affirmative (e.g. certain classes *should* start at the same time) and prohibitory (e.g. certain classes should *not* start at the same time) requirements around entities such as days, weeks and rooms. There is also scope to impose a precedence relation between the timing of a first class and subsequent classes. Lastly, the behaviours of some constraints are reliant upon arguments. For example, the number of breaks between classes exceeding S time slots can be limited by **MaxBreaks** to a maximum of R per day. The cost of a solution is a weighted sum of the individual violation sums for student (**S1**), time (**S2**), room (**S3**) plus any 'distributions' (**F1-F19**) that have been applied as soft. The four weights applied to these violation counts are problem-specific and supplied in the XML attribute 'optimization'.

2.2.6.1 Problem interrogation

Some initial insights into the ITC2019 model can be gained by interrogation of the problem data by way of the following: After pre-processing, instances were read into a struct array. Variables 'days' and 'weeks' were encoded as bitstrings, while all other numeric values were stored as 16-bit unsigned integers. Native IDs of entities such as rooms, courses, configs, subparts, classes and students, which may be character strings or numeric, were mapped to sequential integer IDs. The variable counts could then be extracted as well as a directed graph of class parent relations. Figure 2.3 illustrates the parent-child relations for an example problem `agh-ggis-spr17`.

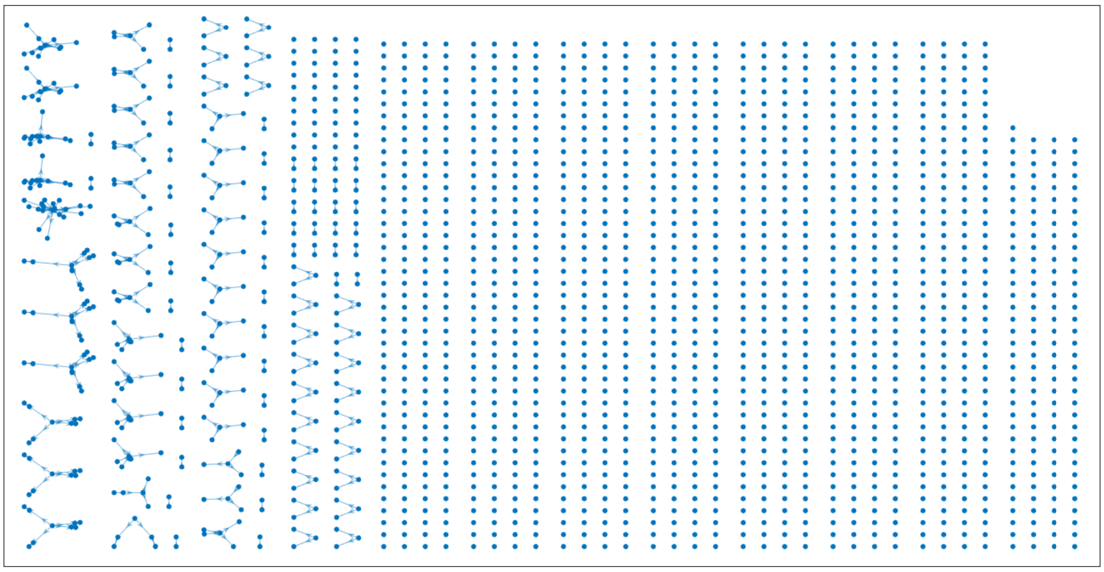


Figure 2.3: A highly disconnected graph showing parent-child relations between classes in ITC2019 problem instance `agh-ggis-spr17`. Classes have a maximum of one parent, and any student sectioned to a 'child' class must also attend its 'parent' class, where it exists. Each node represents one of the 1,852 classes, while directed edges show relations. Prominent features include star graph components and a predominance of classes with no relation at all. The layout of the plot is such that larger components are forced to the left of the window, but otherwise there is no inherent meaning behind the node positions as shown.

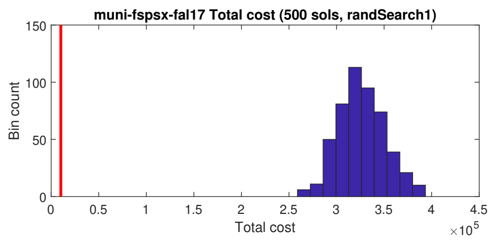
A simple form of pure random search can be executed in the solution space which, by the nature of its sampling algorithm, automatically enforces the subset {**H1**, **H6**, **H7**} of hard constraints. This sample space can be further narrowed by employing a more sophisticated approach for student sectioning, informed by the digraph. Algorithm 1 gives the pseudo-code. By way of rejection sampling, the algorithm considers parent-child relations as well as selecting exactly one class per

Algorithm 1: `randSearch1`: An unbiased rejection sampler for student sectioning

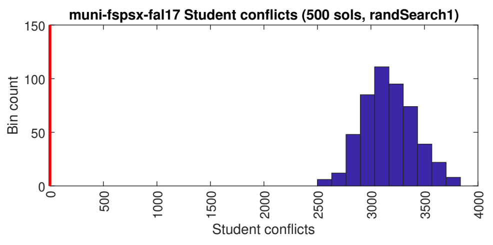
```
1  Inputs: Student $s$, randomly chosen config of a required course
2  Return: Sectioned classes for student $s$
3  if numSubparts = 1 then
4      Choose random class for the subpart and section (assign) student $s$ to it
5  else
                                                        // Multiple subparts
6      Initialise C, column vector of zeros, length numSubparts.
7      while C $\neq$ vector of ones do
8          if $\exists$ $x$ such that C($x$) $> 1$ then
9              Reset C to the zero vector
10         else
11         Find the zero element in C with the greatest index, $i$
12         Choose a random class for subpart $i$
13         C($i$) = C($i$) + 1
14         while Current class has a parent in subpart $j$ & max(C) $\leq 1$ do
                                            // Recursively find chain of parents
15             if No class chosen yet for subpart $j$ then
16                 Choose parent class
17                 Current class = parent class
18                 C($j$) = C($j$) + 1
19             else if Parent class already chosen for subpart $j$ then
20                 Current class = parent class
21             else
                       // Some other class already chosen for subpart $j$
                                                    // Reject this sample
22                 C($j$) = C($j$) + 1
23     Section student $s$ to chosen classes
```

subpart of a randomly chosen config. This guarantees that **H2** is also respected, and the distance to feasibility is consequently reduced.

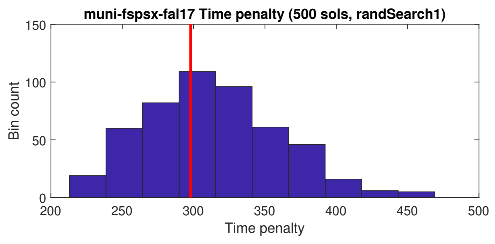
A set of 500 solutions were sampled using this method, for the instance `muni-fspsx-fal17`. Solutions were written back to XML documents before being converted into the recognised format using XSL style sheets. These were evaluated by making programmatic API calls to the ITC website validator. The embedded code is currently protected and we are unaware of any open source function evaluator for this problem. Histograms for the four aggregated objective values, as well as the weighted sum total cost, are shown in Figure 2.4. All sampled solutions have a positive overall *distance to feasibility*. For context, the vertical red line shows the individual values that make up the best known feasible (single-objective) solution.



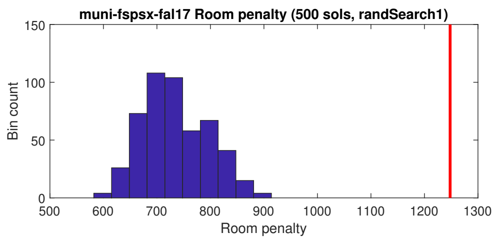
(a) Single-objective total



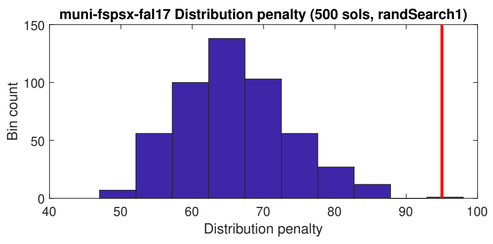
(b) First



(c) Second



(d) Third



(e) Fourth

Figure 2.4: Distributions of values for four objectives over 500 solutions generated by the sampling method respecting **H1**, **H2**, **H6**, **H7** for problem `muni-fspsx-fal17`. The red line indicates the value for the best known (single-objective) feasible solution.

Some very basic intuitions about possible trade-offs between different constraints can be gleamed from the positions of the red lines. Sample values for student conflicts in Figure 2.4(b) are all worse than the best known result, while those for room penalty in Figure 2.4(d) conversely are better. Developing robust techniques for the sampling of solutions, aided by an understanding of which hard constraints are being respected and what biases may be present, forms an integral part of the technical work in later chapters.

2.2.7 Other real world instances

It is evident from the above that the ITC2019 represented a large step forward in bridging the gap between timetabling theory and practise. While some aspects of real world problems remain unaccounted for in this model, these are mostly esoteric. Perhaps the least so, though, is the concept of priority or preference on the part of students and/or teachers. Some real world institutions allow students to make reservations, giving them priority to be assigned to a particular class. The first version of the UniTime project (UniTime, 2023) was designed around the autumn (fall) and spring 2007 Purdue University data sets, with data broken down by department. An example of expressed preferences in UniTime may be those of a teacher, for whom certain time slots, buildings or rooms are more appealing than others. Seven nominal levels of preference were made available: *Required*, *strongly preferred*, *preferred*, *neutral*, *discouraged*, *strongly discouraged*, *prohibited*. This opened up more of a continuum across constraint types, spanning hard (prohibited) to soft (the rest). In the original Purdue timetabling problem data set, there were approximately 750 classes, 29,000 students and 41 large lecture rooms (Vermirovsky, 2003). All data are available in anonymised form on the UniTime website.

Other early uses of real world timetabling problems can be found in Abdullah et al., 2007, Avella and Vasil'ev, 2005, Daskalaki et al., 2004, Dimopoulou and Miliotis, 2004 and Santiago-Mozos et al., 2005. More recently, Maya et al., 2016 utilised three data sets from different Mexican universities in Zitacuaro, Valle de Morelia and Tuxtla Gutierrez, while Babaei et al., 2019 used a hybrid fuzzy and clustering algorithm to satisfy the multi-departmental demands of the Islamic Azad University of Ahar branch. Cross-pollination between the real and artificial is possible too. A large, real world data set from College of Arts and Sciences, Universiti Utara Malaysia known as `UUMCAS_A131` (247 courses, 850 lectures, 32 rooms, 350 teachers, and 20,000 students) had such synergy with the ITC2007 that it was proposed as a test problem under that formulation (Wahid and Hussin, 2017).

In this section, some background has been presented on the different UCTP models that have been used by researchers. Over time, a diversity of approaches have been explored to find optimal or satisfactory solutions to such problems, both artificial and real world. In the next section, a fuller review is offered of some of the most prominent techniques.

2.3 Approaches to solving the UCTP

Due to the large size and complexity of real-world UCTP instances, many traditional optimisation techniques are impractical (Garg, 2009). An exact algorithm, guaranteeing optimality, has proven elusive for all but unrealistically small, artificial cases (Schaerf, 1999). Much research has therefore been carried out into heuristic and approximation techniques, as well as hybridisation with traditional methods. The fundamental idea is to find solutions that are both good enough in practise and achievable with finite computing resources. As well as weighing up these unavoidable trade-offs, researchers have also grappled with some interesting philosophical considerations, for example how to handle the different constraint types. Eiselt and Laporte, 1987 proposed separating the hard and soft constraints in order to solve for the former first then the latter. Subsequently, two-phase metaheuristic approaches have gained in popularity and continue to show promise for the UCTP (Rossi-Doria et al., 2003, Lewis and Paechter, 2005). One-phase metaheuristics, as their name suggests, seek to solve for both types of constraints concurrently. Another design choice concerns infeasible regions of the solution space. Hertz, 1991 deliberately included these in order to maintain connectedness across the entire search space, while others have preferred to restrict the search to the feasible-only space. Some of the important comprehensive surveys referenced when documenting previous approaches include Schaerf, 1999, Lewis, 2008b, Babaei et al., 2014, Pandey and Sharma, 2016, Chen et al., 2021 and Ceschia et al., 2023. More targeted surveys such as Pillay, 2016 and Ilyas and Iqbal, 2015 are referenced for hyper-heuristic and hybrid methods respectively. Broadly speaking, the same taxonomy is used as in Chen et al., 2021, with some rearrangement and expansion. While acknowledging that the categories are not mutually exclusive, a non-exhaustive overview is presented under the following headings: Operational research techniques (Section 2.3.1), single solution based metaheuristics (Section 2.3.2), population based metaheuristics (Section 2.3.3), multi-agent systems (Section 2.3.4), novel intelligent methods (Section 2.3.5) and multi/many-objective approaches (Section 2.3.6). The presentation of these themes roughly corresponds to a chronology of popular usage, such that the reader can trace the development of ideas through this chapter.

2.3.1 Operational Research techniques

Operational research techniques have a long history of use for timetabling and other scheduling problems. They have the advantage of being relatively intuitive to model and implement. However, due to scaling issues, they are often inefficient for the UTCP when used in strict isolation.

2.3.1.1 Reduction to Graph Colouring

Early timetable models were formulated as graph colouring problems — with courses as nodes, conflicts/constraints as edges, and periods as colours. In this pared back representation, the chromatic number $\chi(G)$ implies the minimum number of periods required for a feasible course-teacher schedule. Figure 2.5 shows an example solution to a five course problem with pairwise conflicts {1,2}, {2,3}, {3,4}, {4,5}, {5,1}.

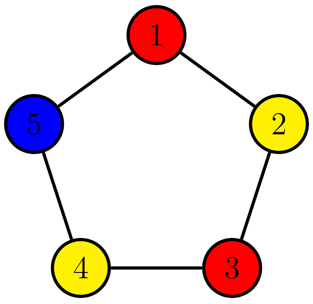


Figure 2.5: A solution to a simplified timetabling problem modelled as a graph, in which different colours represent distinct required periods.

Welsh and Powell, 1967 were two of the first authors to point out the structural similarities between graph models and timetabling, as well as offering an improved upper bound for $\chi(G)$ and an algorithm to obtain a valid colouring. This early model had inherent limitations however, such as being uncapacitated (paying no heed to rooms or their capacities) and not allowing for additional real-world constraints. The second of these issues was addressed by Neufeld and Tartar, 1974, who introduced the possibility of certain preconditions on course assignments. By imposing restrictions on the colouring of particular nodes, unavailability constraints could be accounted for. Meanwhile, preassigned meetings were enforced by limiting particular nodes to a single colour. The authors offered a formal mathematical proof that the existence of a $|\mathcal{P}|$-colouring (where $|\mathcal{P}|$ is the number of discrete periods) is a necessary and sufficient condition for the existence of a feasible solution. A number of different threads of timetabling operational research were later brought together by de Werra, 1985, who discussed algorithmic approaches for valid graph colourings. An established method at that time was *degree saturation* (Brélaz, 1979), which proceeds as follows: Initially the node with the greatest number of neighbours is selected, and coloured. At each subsequent step, the uncoloured node with the greatest number of colours in its immediate neighbourhood is selected. The smallest colour (based on a lexicographic ordering) that has not been assigned to any of its neighbours is then assigned for this node. The process is iterated until completion or impasse. While *degree saturation* provides an infallible method for the optimal colouring of bipartite, cycle and wheel graphs, it is considered a heuristic method when used on the other types of graphs more likely to be encountered in timetabling.

Bipartite graphs are responsive to the *degree saturation* method because of their structure, in which edges only connect nodes from different disjoint sets, of which there are two. These convenient properties were later exploited by Badoni et al., 2014, who tackled an uncapacitated school timetabling problem with a novel, two-phase approach. In phase one, a bipartite multigraph (whose parts are lectures and teachers) was used to derive a daily requirement matrix from a given weekly requirement matrix. In the second phase, an edge colouring was generated for a second bipartite graph (lectures and periods) using the *highest degree first* heuristic. Thus a connection was made between the three entities (lectures, teachers and periods) such that all constraints were satisfied. It must be noted that once again, the model was a simplified and artificial one, without courses, rooms or other dependencies, and small-scale instances were used for testing.

2.3.1.2 Direct constructive heuristics

Degree saturation and *highest degree first* are both examples of direct constructive heuristics. These are used to build solutions incrementally, adding lectures to an empty timetable one by one, according to a set of logical rules. Constructive heuristics formed the basis of the 1980's SCHOLA timetabling software (Junginger, 1986). Using this strategy, lectures are generally sorted according to some measure of how constrained they are. The rationale for this is an intuitive one — lectures that are harder to place should be handled with greater priority. The sorting process can either be *static* (executed once and then fixed) or *dynamic* (recalculated at each decision point over the lectures yet to be assigned). Another example of the dynamic type is *colour degree*, in which priority is determined by the number of conflicts that each lecture has with those already assigned. Constructive heuristics are by nature greedy, making locally optimal moves at the expense of global consequences. Enhancements such as swapping rules or backtracking can be added to extricate such heuristics from unproductive construction paths. Even so, while construction by heuristic may produce feasible timetables, it cannot be relied on to generate optimal ones. Petrovic and Burke, 2004 cautioned against using such solutions as anything other than a starting point for further optimisation or as a baseline for comparison.

2.3.1.3 Network flow

Graph theory is once again invoked in the network flow approach. In this paradigm, the UCTP is formulated as one or more flow networks (Dyer and Mulvey, 1976, Mulvey, 1982, Chahal and Werra, 1989). A graph is generated in which edges have numerical flow capacities, limited to 0 or 1. In Dinkel et al., 1989, the vertices were layered at intermediate levels between a sink and source, representing departments, teacher/course and room/timeslot combinations respectively. Edges were omitted wherever co-assignments of particular variables were impossible, while the unimodularity property ensured integrality of solutions. While the resultant maximum flow problem can be solved in polynomial time, the authors noted that their approach occasionally required human intervention. This is because constraints such as **H5** —

the prohibition on assigning a single teacher to multiple lectures simultaneously — could not automatically be guaranteed.

These shortcomings were highlighted further in the more recent work of Kampke et al., 2019, which modelled the ITC2007 track 3 problem using ostensibly the same graph architecture as Dinkel et al., 1989. Hard constraints **H2** and **H3** were guaranteed, while the remainder were not. Due to a lack of backtracking, it was possible to reach a situation in which no feasible places remained for the unassigned lectures. This induced a no-win scenario in which the only options were to return an incomplete timetable (violating **H1**) or introduce conflicts (violating **H4** or **H5**).

Soft constraints were not considered explicitly, but modifications to the base solving algorithm indirectly assisted **S2** (room stability) and **S3** (minimum working days). A solution was represented by a valid flow through the network, which was found using the Ford-Fulkerson method with breadth first search (Ford and Fulkerson, 1962). The added refinements in Kampke et al., 2019 were two-fold:

1. For each course $C \in \mathcal{C}$, nodes in the room/period layer were visited in ascending order of room capacity, beginning with the smallest room whose capacity exceeded $stud(C)$. When/if these were exhausted, the search continued with the undersized rooms, this time in descending order.

2. Feasibility checking was implemented such that infeasible placements were avoided and the better alternative was always preferred.

The first of these modifications is a greedy heuristic from the same family as those in Section 2.3.1.2, demonstrating again how deceptively simple rules can aid certain objectives. The authors stated that the two constraints served by this heuristic were almost never violated. Returning complete or feasible solutions, however, proved more difficult, with the overall approach failing consistently on 12 of the 21 instances. In 8 of the others (`comps` `01`, `04`, `06`, `08`, `09`, `13`, `14` and `18`), feasible solutions were found only in a fraction of the trials (55%, 22%, 3%, 3%, 3%, 37%, 7% and 84% respectively), while `comp11` was the sole problem to achieve 100% on this metric. It was only through supplementary stages that the initial returns could be moulded into high quality timetables. Partial solutions from the network flow phase were completed by a constructive algorithm IS_{CB-CTT} (Kampke et al., 2015), before further optimisation took place through a simulated annealing process using a greedy randomised adaptive search procedure (GRASP).

The limitations of network flow as a standalone modelling approach are evident. A related field, offering a richer modelling environment in which to overcome these, is mathematical programming. In the next section, some important integer and mixed integer based approaches are relayed.

2.3.1.4 Mathematical Programming

Mathematical programming has been used to address timetabling as an assignment problem (or a collection of sub-problems), often using binary variables for direct representation. One of the earliest examples is Lawrie, 1969, who used a branch-and-bound procedure with Gomory cuts to find a feasible timetable for a high school problem. Other early proponents include: Breslaw, 1976, Shin and Sullivan, 1977, McClure and Wells, 1984, Ferland and Roy, 1985, and Tripathy, 1992. Tripathy, 1984 and Carter, 1989 employed Lagrangian relaxation to solve instances of a course timetabling problem with up to 287 lectures. While this may be comparable in size to the ITC2007 benchmark, these approaches only considered straight-forward conflicts of the type depicted in Figure 2.5. More complex interactive effects were disregarded as these would have increased both modelling difficulty and solver run time. Indeed, in Carter, 1989, the assignment variable was indexed only by course and room.

Burke et al., 2010 went further in attempting to include all hard and soft constraints from the ITC2007 track 3 benchmark in an integer linear programming (ILP) model, named *Monolithic*. Five sets of decision variables were defined as follows:

$x_{Cpr} \in \mathbb{B}$. Lectures of course C should be placed in period p, room r if and only if this variable is set to 1.

$v_{Cd} \in \mathbb{B}$. At least one lecture of course C is scheduled on day d if and only if this variable is set to 1.

$z_{pu} \in \mathbb{B}$. A lecture in curriculum $u \in \mathcal{U}$ is 'isolated' in period p if and only if this variable is set to 1.

$y_{Cr} \in \mathbb{B}$. Some lecture of course C is scheduled in room r if and only if this variable is set to 1.

$i_C \in \mathbb{Z}$. This value (bounded by zero and the number of days per week) is the number of days that the lectures of course C fall short of the recommended *minimum working days*.

As instance properties in this benchmark are defined by course rather than by lecture, the graph theoretic interpretation of the problem became supernodal. That is to say, each course node (akin to those in example Figure 2.5) became a clique of nodes representing its constituent lectures, and inter-clique edges proliferated accordingly. The core binary decision variables, x_{Cpr}, therefore remained indexed by course rather than lecture, while the values of the other (dependent) variables were inferred during the solving process. Two equations were used to enforce **H1** (all lectures must be assigned) and **H4** (no unavailable periods can be used) while the remaining hard constraints were guaranteed by a set of inequalities. The soft constraints, which were formulated as six additional inequalities, presented a more challenging aspect in their design. The **S3** (curriculum compactness) constraint, for example, was expressed by:

$$\sum_{C \in u} \sum_{r \in \mathcal{R}} (x_{Cpr} - x_{C,p-1,r} - x_{C,p+1,r}) \leq z_{pu} \quad u \in \mathcal{U}, p \in \mathcal{P} \tag{2.1}$$

With the added complication that if p coincided with the first (or last) timeslot of a day, $p-1$ (or $p+1$ respectively) ceased to exist for the purposes of this inequality and had to be taken as zero. This follows from the definition of **S3**.

The objective function, which returned the solution cost to be minimised, was similarly unwieldy:

$$
\begin{aligned}
w^{S1}\sum_{r\in\mathcal{R}}\sum_{p\in\mathcal{P}}\sum_{\substack{C\in\mathcal{C}:\\ stud(C)>c(r)}} x_{Cpr}(stud(C)-c(r)) + w^{S2}\sum_{c\in\mathcal{C}} i_C \\
+w^{S3}\sum_{p\in\mathcal{P}}\sum_{u\in\mathcal{U}} z_{pu} + w^{S4}\sum_{c\in\mathcal{C}}\left(\sum_{r\in\mathcal{R}} y_{Cr}-1\right)
\end{aligned}
\tag{2.2}
$$

Where w denotes the penalty weights for each soft constraint. This ILP was solved using ILOG CPLEX 11 Dual Simplex LP Solver, and theoretically finds an optimal timetable, given enough time. Bettinelli et al., 2015 remarked that the ILP could run for days without return though and *Monolithic* was thus only suitable for modest sized or trivial problems. Only three problems (`comp01`, `comp05` and `comp11`) from the 14 released at the time were solved within 40 CPU units, where 1 CPU unit $\approx$ 780 seconds. *Monolithic* nonetheless proved useful in identifying lower bounds for the benchmark, and inspired several developments. By discounting certain violations by setting their weights to zero, two variants derived by the same authors (Burke et al., 2010) were capable of obtaining better lower bounds on a majority of instances, at an order of magnitude speed-up. Other proposed simplifications included aggregating equivalent rooms in order to reduce the quantity of variables which, as pointed out in Cacchiani et al., 2013, could be exponential in number. Embedding the ILP within a heuristic framework so as to locate an area of promising solutions before 'diving in' was another improvement suggested by Burke et al., 2010. In later work (Burke et al., 2012), cuts were suggested to narrow the search bounds — the simplest example of this being the tightening of the upper bound of $i_C \in \mathbb{Z}$ to $mwd(C)$.

In Cacchiani et al., 2013, which also examined the ITC2007 track 3 benchmark, a strategy was proposed in which the main problem was partitioned into more manageable sub-problems. The fact that constraints **S1** and **S4** relate to the assignment of lectures to rooms, while **S3** and **S2** relate to time, suggested a natural decomposition. The authors found that a fully descriptive ILP gave competitive results when the sub-problems were small. In larger cases, a linear relaxation was imposed and a column generation procedure used to deliver a similar quality of results.

Most notably, lower bounds were improved for many instances, while some known upper bounds were mathematically proved to be optimal. Splitting the problem constraint entities along spatial vs. temporal lines echoed the approach of Lach and Lübbecke, 2012.

A different form of problem partitioning was also exploited as a first step in the divide-and-conquer approach of Hao and Benlic, 2011. This decomposition was based on minimising the number of relaxed **S3** constraints linking pairs of courses in distinct parts. It was created using a tabu search and refined by a perturbation phase, the two of which were called cyclically until some stopping criterion was met. In subsequent steps, lower bounds were obtained on the sub-problems by generic ILP solvers, before being summed to give a solution for the main problem.

This effective melding of mathematical and metaheuristic techniques also proved successful in Lindahl et al., 2018. In the authors' proposed matheuristic, termed a *fix-and-optimise* approach, a mixed integer program solver explored a large neighbourhood in which a subset of the variables were fixed. This was inspired by the 'corridor' method (Sniedovich and Voß, 2006) of solving smaller sub-problems by exact methods. The technique currently holds the best known result for one instance in the ITC 2007 track 3 benchmark and was highly competitive across the remainder.

On the whole, while standalone integer or mixed integer programs can exactly describe a problem, in practise they are most useful for proving bounds on the optimum. Constraint satisfaction programming, discussed in the next section, focuses not on an explicit objective function, but instead on finding consistent assignments that meet the specified problem constraints.

2.3.1.5 Constraint Satisfaction Programming

In a constraint satisfaction programming (CSP) formulation, conditions are placed on variables, thereby constraining their values to some finite feasible domain. An assignment of values to all variables represents a solution in which every constraint is satisfied. Yoshikawa et al., 1996 described a constraint relaxation problem solver (COASTOOL) applied to a high school timetabling problem, while Deris et al., 1999 used CSP in combination with a genetic algorithm to address the UCTP, before furthering the research in Deris et al., 2000. Other notable contributions include Zhang and Lau, 2005. One of the most prominent examples though can be found in the UniTime software as mentioned in Section 2.2.7. This is an open source solver for UCTPs. Conceived in 2001 at Purdue University, its first phase allows the modelling of a problem instance by constraint programming primitives (constraints, variables and values). Operating an iterative forward search algorithm, UniTime differentiates itself from traditional local search methods by including incomplete (partially assigned yet internally feasible) solutions within its search space. Some evident benefits of this are:

1. A heuristic-guided local search that includes partial solutions is generally more efficient, with respect to response time, than a systematic one that only allows fully formed solutions.

2. The system can stop, start or continue from any given feasible solution, no matter the level of its incompleteness.

3. An otherwise feasible timetable with missing lectures is more meaningful and interpretable than a fully assigned timetable with multiple hard constraint violations.

4. Built-in backtracking means the system does not suffer from the so-called 'early mistake problem'. Any decision suspected to lead to a dead-end in a partial solution can be undone [1].

In this constructive phase, cycling is prevented by the use of conflict-based statistics (CBS) (Müller et al., 2004). CBS is a data structure that records previously encountered conflicts between assignment variables, along with their frequencies. Conceptually similar to the tabu list (discussed in a later section), CBS helps steer the search away from potentially detrimental regions. Unlike the aforementioned approach of Kampke et al., 2019, UniTime waits for a complete solution to be found

[1] The 'early mistake problem' is a crucial element of investigation in the work on ACOs in Chapter 3, in which different form of remediation is proposed.

before entering its second phase. This is a minor distinction, however. The more important commonality is the emergence of a multi-phase system in which rule-based construction precedes optimisation. For UniTime, optimisation is achieved through a recursive chain of metaheuristics: Hill climbing, great deluge and simulated annealing.

UniTime was a finalist in all three tracks of the ITC2007 and the winner of two. Although finer algorithmic details such as neighbourhoods and parameter settings were redefined for each problem domain, the underlying principles were shown to be encouragingly robust.

2.3.1.6 Logic Programming

While CSP involves finding solutions that satisfy a set of constraints, logic programming provides a framework for expressing and solving such problems through logical relationships, predicates and the leveraging of powerful inference mechanisms. One well-known framework is Answer Set Programming (ASP) — an approach based on a declarative logic paradigm (Marek and Truszczynski, 1999, Niemelä, 1999). It is only in recent years that ASP has been applied to the UCTP, however.

A simple ASP logic program, P, is made up of *rules*, *facts* and *constraints*. *Rules* are of the mathematical form:

$$h_1 \vee \cdots \vee h_k \impliedby a_1 \wedge \cdots \wedge a_n \wedge \boldsymbol{not}\, b_1 \wedge \cdots \wedge \boldsymbol{not}\, b_m. \tag{2.3}$$

Where letters symbolise classical first-order logic atoms, which may be predicates on one or more variables. *Facts* and *constraints* are defined by expressions that are empty on the right or left of the implication sign, respectively. A typical ASP workflow consists of three stages:

1. Modelling. In which the problem is formalised and declared for the parser.

2. Grounding. In which variables are eliminated.

3. Solving. In which the 'stable models', or eponymous 'answer sets', are generated.

Answer sets are equivalent to the set of solutions across the defined feasible search space. Programs may have any number of answer sets, with the case of none implying no feasible solution.

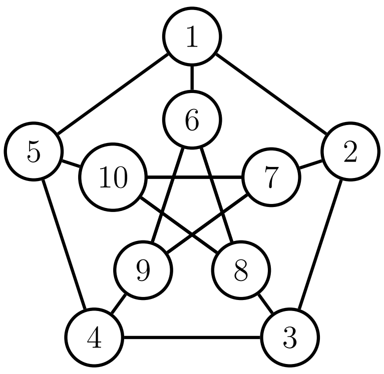


Figure 2.6: The Petersen graph.

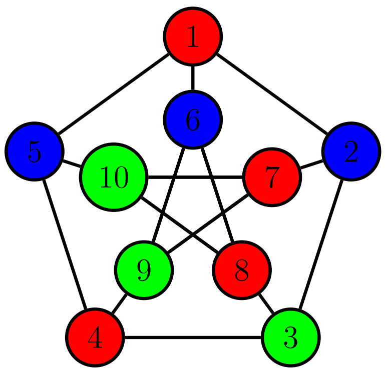


Figure 2.7: A 3-vertex-colouring.

In a similar vein to Figure 2.5, Figure 2.6 shows a simplified timetabling example based on the Petersen graph, which has 10 nodes and 15 pairwise constraints. To find a 3-vertex colouring — one solution of which is shown in Fig. 2.7 — the program in Listing 2.1 could be used. Lines 1-7 define the graph by way of node and edge predicates, while line 9 accounts for the three colours. Line 11 specifies that exactly one colour should be applied to each node and line 13 is a constraint stating that adjacent nodes must have different colours.

```
 1 node(1..10).
 2
 3 edge(1,2). edge(1,6). edge(1,5). edge(2,1). edge(2,3). edge(2,7).
 4 edge(3,2). edge(3,4). edge(3,8). edge(4,3). edge(4,5). edge(4,9).
 5 edge(5,1). edge(5,4). edge(5,10). edge(6,1). edge(6,8). edge(6,9).
 6 edge(7,2). edge(7,9). edge(7,10). edge(8,3). edge(8,6). edge(8,10).
 7 edge(9,4). edge(9,6). edge(9,7). edge(10,5). edge(10,7). edge(10,8).
 8
 9 col(r). col(b). col(g)
10
11 1 {colour(X,C) : col(C)} 1 :- node(X).
12
13 :- edge(X,Y), colour(X,C), colour(Y,C).
```

Listing 2.1: A reductive timetabling problem expressed by Answer Set Programming.

The expressive power of ASP has been growing as further semantic constructions, many vital for modelling the UCTP, are supported. These include *conditional literals*, *cardinality constraints*, *aggregates*, *choice rules*, *weights* and *arithmetic op-*

erators, as well as *blanks* such as in the predicate `penalty(_,_,P)` which are used to sum across all values of the first two elements of the tuple. Popular off-the-shelf solver Clingo features optimisation commands `#maximise` and `#minimise`, which could be employed with the example above like so: `#minimise[penalty(_,_,P) = P]` to find a UCTP solution with the lowest soft constraint violation cost.

Banbara et al., 2013 describe an ASP program which was tested against 57 problem instances in the ITC2007 track 3 formulation, using each of the UD1-UD5 configurations. On the $57 \times 5 = 286$ problems from the sets named in Section 2.2.4, the previously best known bounds were matched or bettered for 175 problems, while optimality was proved for 46.

Some benefits of logic programming are readily apparent. ASP enables a compact formulation that is human-readable. There is high 'elaboration tolerance' owing to the way rules, facts and constraints are independently declared — meaning that activating, deactivating or switching UCTP constraints is easy. The on-going and rapid development of grounders, solvers and monolithic hybrids has also shown the potential for extensibility, as in Clingo's priority level multi-objective optimiser (Banbara et al., 2019). Furthermore, logic programming obviates the need for the parameter tuning required by some metaheuristics. Yet there are still design choices to be taken. Clingo offers users a choice of search strategies, such as backtracking (Ward and Schlipf, 2010) and conversion to a Boolean satisfiability (SAT) problem. Different configurations of these strategies were tested in Banbara et al., 2013 with mixed results.

Perhaps the biggest issue concerns the timeouts that result from excessively large ground programs. In Banbara et al., 2013, problem instance `EA03` (consisting of 145 courses, 65 rooms in 9 buildings, 65 curricula, 3,207 unavailability constraints, and 1,350 room constraints) was unsolvable under formulation UD5, due to a combinatorial explosion of clauses. One particular soft constraint, **TravelDistance**, caused the ground program to blow up to 7.9GB in size (versus 70MB with **TravelDistance** omitted) while `EA07` caused similar issues. Likewise, in Banbara et al., 2019, `UUMCAS_A131` on UD5 exceeded the available memory limit of 20GB and a large instance of `erlangen*` could not be grounded in a day. Such issues speak to the impracticality of pure exact approaches for large-scale UCTP problems.

The following two sections move the discussion on to the field of metaheuristics, beginning with single solution based strategies.

2.3.2 Single-solution-based metaheuristics

Single-solution-based metaheuristics are approximation approaches, in which a single solution (a timetable, in this context) is iteratively refined by low-level heuristic operators until some termination criteria is met (Bianchi et al., 2009). Through the informed design and paramaterisation of both the operators and acceptance criteria, a path is navigated through the search space and the algorithm converges on solutions of high, if not optimal, quality. A key feature of many metaheuristics is their stochastic element, which enables disparate regions of the search space to be accessed through random (or partly random) exploration. Regions of the landscape thought to contain promising solutions can then be exploited further.

2.3.2.1 Local search

Local search is based on exploring nearby solutions as defined by some neighbourhood structure. The simplest form is random search. Given a starting timetable, a new solution is sampled from its neighbourhood, evaluated, and accepted if and only if it lowers the solution cost. Hill climbing is similar, except that the neighbourhood is assessed exhaustively and the most improving solution is chosen. A known drawback of local search is its tendency to get stuck in local optima, in which better solutions exist elsewhere but are not reachable through the immediate neighbourhood. Local search is therefore more commonly used in conjunction with other techniques in order to fine-tune solutions. Examples include Joudaki et al., 2011, which incorporated local search in the form of a memetic algorithm, and Yang and Jat, 2011 and Shahvali et al., 2011, both of which proposed enhanced genetic algorithms with local search capabilities.

2.3.2.2 Tabu search

In order to avoid repetition or cycling during such a search, a list of prohibited moves from the recently-visited solution space can be maintained. This is known as tabu search. A critical feature of tabu search for the UCTP is the choice of neighbourhood, which is used to determine the candidate list of next possible moves. In the multi-phase tabu approach of Alvarez-Valdes et al., 2002, two neighbourhoods of interest were studied[2]. A solution x was a neighbour of x' if and only if:

1. *Simple move*: x can be reached from x' by the reassignment of exactly one lecture in x' to a new period.

2. *Swap*: x can be reached from x' by swapping the places of exactly two lectures in x'.

The authors found that the type of neighbourhood was the most significant factor affecting performance, ahead of other design choices such as the size of the tabu list or candidate list. Of the neighbourhoods tested, *simple move* gave the poorest quality results, which was explained by its move set being limited to only temporal (period) moves and not spatial (rooms). The search landscape it induced suffered from disconnectedness, meaning *simple move* can be thought of primarily as an intensifying move. This result highlighted the importance of offsetting intensification with sufficient diversification.

Aladag et al., 2009 attempted to improve the balance between the two by proposing additional neighbourhoods, *mixed_1* and *mixed_2*, that combined elements of the original *simple move* and *swap*:

1. *mixed_1*: If, after a predetermined number of applications of *simple move*, an improved local optimum has not been found, the search relocates to the best *swap* neighbour of the current best local optimum.

[2] A third neighbourhood, *multiswap*, was also studied, although this was problem-specific to the real world Spanish university formulation under consideration, in which lectures of variable duration were allowed. This feature was similarly explored in a previous tabu approach of Hertz, 1992.

2. *mixed_2*: *Simple move* and *swap* are treated as a union rather than individually, thereby expanding the pool from which the candidate list is formed. The authors noted that evaluating entire neighbourhoods was expensive. Therefore, only one random move by each lecture was carried out. By implication, the final candidate list was the same length as the number of lectures.

The best results were achieved by *mixed_1* and *simple move*, both of which were significantly superior to the other neighbourhoods as well as a random baseline. The good performance of *simple move* appeared to contradict the findings of Alvarez-Valdes et al., 2002. However, the authors' inclusion of a local optima escape mechanism showed that, with this addition, a simple intensifying neighbourhood can deliver good results over a long term search. The plots of objective cost vs iteration number showed a series of spikes where the search jumped from one local optimum in order to descend a different basin of attraction. *mixed_1* conferred a similar type of benefit in the sense that its fall-back operator, *swap*, was more diversifying.

The relatively poor performance of *mixed_2* raises an interesting question about the structuring of composite neighbourhoods. A promising alternative to the set union is a token ring approach. This was defined in di Gaspero and Schaerf, 2003 as follows: Given a pool of neighbourhoods $N_1 \dots N_q$, The input solution is operated on by N_i in the predetermined sequence $i = 1 \dots q$, with each N_i working on the output produced by N_{i-1}. $i = q$ is followed by $i = 1$ so that the sequence is circular. The global best is cached and the cycle is broken when a fixed number of non-improving rounds have been completed. In di Gaspero and Schaerf, 2003, the notation $N_1 \rhd \cdots \rhd N_q$ was suggested. An empirical comparison between the set union and token ring configurations was carried out by Lü et al., 2011, on a pool $\{N_1, N_2\}$. N_1 was *simple move*. N_2 was an 'advanced' neighbourhood — *KempeSwap* — adopted from the adaptive tabu search method of Lü and Hao, 2010. The move was defined by the interchange of two Kempe chains. In the context of a timetable as a graph of lectures and their conflicts (as per Figure 2.5), a Kempe chain is a connected component of nodes in the subgraph that is induced by nodes belonging to two periods (or colours). The results in Lü et al., 2011 suggest that a token ring approach is superior to a set union, and that the order of operators at the start of the ring cycle was immaterial.

Other structures and elementary neighbourhood operators have been proposed for tabu search that are more aggressive, more nuanced and/or more targeted towards improving a particular objective. Awad et al., 2022 described four such neighbourhoods, two of which not only enforced feasibility but also mandated the lowest-cost move. Exploration was encouraged by stochastic selection. While results did not improve upon the best known for the chosen benchmark (11 problems from the Socha et al. 2002 set), the algorithm performed competitively against other metaheuristics. Aspiration criteria were declared in order to provide exceptions to the strict prohibitions of the tabu list, and the list itself was dynamically sized. Awad et al., 2022 remarked on the difficulty of fine tuning the parameters that control these aspects of a tabu approach.

Besides parameter tuning, another routinely confounding issue, identified in Lü and Hao, 2010, is that of computational complexity. While the authors' aforementioned Kempe chain move was useful in affecting large, feasibility-preserving perturbations, handling Kempe chains (or other complex topologies) can be expensive, meaning the move could only be used sparingly. The authors mitigated this with a 'reduction' technique, which estimated the goodness, using the period-based sub-cost, to decide whether a full execution of *KempeSwap* was worth calling or not. This is one example of an efficiency saving that can help make complex operators more attractive, and will be explored further in later chapters.

2.3.2.3 Iterated Local Search

Improving a solution using local search operators, before applying larger perturbations to escape local optima traps is known as iterated local search (ILS). In fact, in Lü and Hao, 2010, an ILS scheme was adaptively combined with the tabu search. Their findings empirically demonstrated that the judicious integration of metaheuristics, and indeed the balancing of large and small moves, can produce a more powerful optimiser compared to the use of a single technique in isolation.

Parameterisation of η (the ILS perturbation strength) was noted as being crucial. When set too high, each iterative jump behaves as a random restart. Too low a value, meanwhile, will not provide an escape from the current local optimum. A related parameter, $q > \eta$, was also used. This represented a number of the most highly-penalised lectures, η of which were then selected probabilistically and reassigned by a sequence of η randomly chosen (but feasible) moves. In this way, parameters q and η strongly influenced diversification levels. Using the original stopping conditions of the ITC2007 track 3 benchmark, this hybrid system improved or matched the best known results at the time over all 21 `comp*` instances.

2.3.2.4 Simulated Annealing

Pure ILS manages the exploration/exploitation balance in timetabling optimisation by alternating between incremental and larger moves. Another differing but promising approach is simulated annealing (SA), which has delivered strong results on both artificial and real world UCTP problems (Akbulut, 2024). Inspired by the annealing process in metallurgy — in which a heated metal is slowly cooled in order to reduce defects and achieve a more ordered, stable and desirable crystalline structure — SA permits the occasional inferior solution based on a probabilistic acceptance criteria.

Aycan and Ayav, 2008 offered a proof of concept for SA on a real world problem from the Izmir Institute of Technology with nine hard constraints. The perturbation of a solution, x, led to a new solution x^{new}, which was accepted if and only if:

$$((\delta \leq 0) \text{ or } e^{-\delta/v} < rand[0,1]) \tag{2.4}$$

Where $\delta = eval(x') - eval(x)$, and v is a 'temperature', which decreased over time according to a geometric cooling schedule. With only this standard SA design, the authors were able to reduce the departmental timetable cost from 5,011,800 (for a manually prepared version) to just 3,600.

Across recent SA approaches, a few commonalities stand out. Firstly, the multi-phase model is prominent. Three distinct stages were used by Lewis, 2008a in solving the post-enrolment problem. At each stage, constraints satisfied in the previous stage could not be violated. A different three-stage strategy — closer resembling that of Lü and Hao, 2010 — was adopted by Song et al., 2018. Initialisation by heuristic was followed by intensification by SA (using a constructive heuristic-inspired operator) and diversification by perturbation. The latter two phases operated in the iterative style of ILS and resulted in feasible solutions for 58 of the 'Sixty Instances' problems, including for three of the 'largest' instances that had, up to that point, proved elusive (Lewis and Paechter, 2006).

Some authors prefer to view the intertwining of intensification and diversification as a single 'improvement' phase, as distinct from the constructive stage, as described in the two-phase SA algorithms of Goh et al., 2019 and Goh et al., 2020. Another recurrent theme, in relation to small neighbourhoods, is the predominance of the single lecture move and swap operators (or subtle variations thereof), as seen in Ceschia et al., 2012 and others. Many novelties have arisen too, some specific to SA and others with general applicability. In Geiger, 2012, the probabilistic element of SA was substituted for a deterministic 'threshold accepting' test, which built upon ideas by Kirkpatrick et al., 1983 and Dueck and Scheuer, 1990. Solutions that worsened the objective were permitted up to a given threshold, while an element of randomness was retained within the constructive perturbation operator. Geiger, 2012 achieved fourth place in the ITC2007 track 3 competition.

In Tarawneh et al., 2013, an SA with memory was proposed, in which previously rejected solutions could be recalled to aid in local optima escape at low temperatures. The two popular neighbourhoods were also joined by a third, wherein the highest-cost timeslot (as summed over all days of the week) was swapped *en masse* with a new, random one. The approach was competitive with contemporaneous ITC2007 results.

Perhaps one of the most interesting novelties in SA optimisers is the integration of machine learning to improve contextual awareness. Bellio et al., 2016 derived a linear regression model between (easily extracted) features of the instance and the search parameters. The training data was produced by the generator of Lopes and Smith-Miles, 2010 with heavy *post hoc* manual filtering. Goh et al., 2019, meanwhile, applied reinforcement learning techniques to the task of obtaining suitable neighbourhood structures. Six new best results were found for the ITC2007 track 2. Finally, in Goh et al., 2020, which introduced periodic 'reheating' of the temperature to encourage exploration, two preliminary runs were undertaken. The information learned from these was leveraged in order to improve the efficiency of the subsequent runs. Helped by these savings, the approach delivered seven new mean best and three new best results across a set of standard benchmarks.

2.3.2.5 Variable Neighbourhood Search

Variable Neighbourhood Search (VNS) was born out of a recognition that a local minimum under one neighbourhood structure may not be a local minimum under another (Mladenović and Hansen, 1997). Consisting of two main phases — shaking and local search — this metaheuristic, which can be deterministic or stochastic, is centred around a pool of neighbourhoods. Early work on VNS for the UCTP by Abdullah et al., 2005 led to the development of a 'randomized iterative improvement with a composite neighboring algorithm' (RIICN) in Abdullah et al., 2007. More recently, Vianna et al., 2020 proposed a hybrid of VNS and tabu search for a real world example from Federal Fluminense University. Each iteration started with a solution, x, and a neighbourhood, k. A random neighbouring solution, x^{new}, was chosen and then improved through local search. If this refinement was better than x then it was accepted, otherwise k was shuffled to the next neighbourhood in the pool. The authors reported that, consistent with numerous other metaheuristic studies previously discussed, the hybrid outperforms either of its individual components. As a whole, the system was deemed both good enough and flexible enough to be put to real world use.

Almeida et al., 2023 proposed a hybrid VNS that included an adaptive large neighbourhood search and guided local search. A novel instance decomposition technique was used, whereby the problem was broken down by curricula. At a certain threshold of non-improvement, difficult constraint groups were re-weighted and the search re-focused on improving for those, potentially at the expense of others —

thereby allowing moves that worsened the overall quality. The extended approach found high quality solutions faster than the native VNS.

2.3.3 Population-based metaheuristics

An alternative to the single-solution-based metaheuristic is one with a population of solutions. This offers a number of advantages. A population search is often more robust and less sensitive to initial conditions. Wide areas of the search space can be explored simultaneously, leading to the discovery of the best timetable. The drawbacks include higher computational load, slower convergence speed and often additional parameters that require tuning. In this section, the most relevant and important methods are reviewed, including swarm intelligence.

2.3.3.1 Genetic Algorithm

A subclass of evolutionary algorithms (EAs), genetic algorithms (GAs) are iterative procedures inspired by the biological process of natural selection. GAs have been known to perform well on non-convex fitness landscapes, both single and multi-objective problems, and also lend themselves well to parallelisation. This makes them well-suited for *NP*-hard, combinatorial optimisation tasks. Algorithm 2 shows a simple GA, as successfully adapted with additional local search for the timetabling problem in Psarra and Apostolou, 2023.

Algorithm 2: A simple genetic algorithm

```
1 Input: Operational parameters
2 Output: Population at termination
3 Initialise population of solutions
4 Evaluate population
5 while Termination criteria not met do
6     Select solutions for next generation
7     Perform crossover
8     Perform mutation
9     Evaluate population
```

Various flavours of GA have been conceived for the UCTP, which differ in one or more of the fundamental components discussed below.

Representation Within the optimiser, solution timetables must be encoded in a consistent way. In GAs, this is a phenotype-to-genotype mapping, in which each solution is represented as a chromosome. For theoretical as well as practical reasons, it is desirable that the entire space of feasible solutions can be represented and that there is little or no redundancy in the genotype space. Thus, the mapping is typically bijective. Its domain may encompass part or all of the infeasible solution space too, in the case where an algorithm permits individuals to evolve through infeasibility in order to reach a better final solution.

An intelligent choice of representation is a convenient mechanism by which to implicitly enforce hard constraints. Abouelhamayed et al., 2017 recounted a popular encoding for certain UCTP formulations (seen in Akkan and Gülcü, 2018 amongst others) which consisted of a 2-D array of rooms vs. weekly periods with the gene values in cells denoting the ID of an assigned lecture. However, it was noted by the authors that clumsy design can also narrow the represented search space by inadvertently enforcing hard constraints that are not present in the problem description. An example given was a lecture that alternates every two weeks, which the aforementioned array had no means of encoding. The proposed alternative to cater for this requirement was a 1-D array. Two identically sized vectors, with indices corresponding to the set of unique lecture IDs, were concatenated. The value for an index in the first half of the resultant array encoded the period to which that particular lecture had been assigned, while a value for the same index in the second half encoded the associated room.

Chinnasri et al., 2012 also employed a flattened 1-D array, but constructed of a sequence of indices corresponding to the periods, repeated $|\mathcal{R}|$ times. Into these positions were placed tuples $\langle$Subject, Lecturer, Student, Type of subject, ID$\rangle$ which were drawn from a fixed pool. In Sutar and Bichkar, 2016, a conceptually similar yet complementary approach is found. Here, the period indices were repeated over the number of classes rather than rooms, and the value inserted into the chromosome was instead a $\langle$room, teacher$\rangle$ tuple. Such tuples were often indivisible 'blocks', which proved useful for preserving *a priori* (known or established) associations from one generation to the next. In contrast, some practitioners prefer a 2-D matrix to enable shuffling of variable values that may not be fixed in blocks. In Asiyaban and Mousavinasab, 2012, values encoding the teacher, period and room were stored in a column of length 3 whose matrix row index corresponded to the associated course event. For UCTP formulations with higher variable dimensionality, 3-D matrices have also appeared in the literature (Sigl et al., 2012).

As the UCTP is, at its heart, a permutation problem, encoding designs commonly feature integer rather than binary values, as they complement better the type of operators used. In addition, integer values eschew problems such as the 'Hamming cliff' that can cause issues when manipulating bit-strings (Wang et al., 2018). It is worth noting that all of the representation styles discussed above are forms of direct encoding, as these predominate in the literature. Some examples of indirect encodings for GAs on the UCTP can be found in Paechter et al., 1998 and Chaudhuri and De, 2010.

Operators GAs are built on a set of operators, usually selection, crossover and mutation, each of which performs a different role in driving the search. Relying on traditional operators alone (for example single-point crossover) may not be appropriate for the UCTP and result in offspring that are consistently less fit than their parents or, in the worst case, invalid. Chinnasri et al., 2012 noted that careful design, application and parameterisation of operators are therefore critical for GA performance in this context. The authors provided a comparison of three different crossover operators — partially matched (PMX), order (OX) and cycle (CX).

Figure 2.8 shows a worked example of PMX, which begins with the uniform random selection of two crossover points. These delimit the matching area common to both parents, which is shaded in grey in the top block. The intermediate stage involves a position-by-position swap of alleles within the matching area, which can lead to unwanted duplicate values in outer positions (shaded again in grey, in the middle block). To resolve this issue, these values are then switched with their counterparts outside of the matching area, as shown in the bottom block. This completes offspring A and B.

Parent A	9	4	1	2	6	5	8	3	10	7
Parent B	6	1	7	8	5	10	2	9	4	3

Intermediate A	9	4	7	8	5	5	8	3	10	7
Intermediate B	6	1	1	2	6	10	2	9	4	3

Offspring A	9	4	7	8	5	6	2	3	10	1
Offspring B	5	7	1	2	6	10	8	9	4	3

Figure 2.8: An illustration of the partially matched crossover (PMX) operator. Within a randomly selected matching area, alleles 1, 2 and 6 in Parent A are swapped with 7, 8 and 5 in Parent B respectively. To eradicate duplicates, 5, 8 and 7 in Intermediate A are swapped with 6, 1 and 2 in Intermediate B to produce the final offspring.

Parent A	9	4	1	2	6	5	8	3	10	7
Parent B	6	1	7	8	5	10	2	9	4	3

Intermediate B 1	H	H	7	8	5	10	H	9	4	3

Intermediate B 2	8	5	H	H	H	10	9	4	3	7

Offspring B	8	5	1	2	6	10	9	4	3	7

Figure 2.9: An illustration of the order crossover (OX) operator generating Offspring B. Alleles 1, 2 and 6 (taken from the matching area in Parent A) are replaced in Parent B by 'holes', H. A leftwards sliding mechanism repositions the 'holes' to the matching area, where they are filled by 1, 2 and 6.

Order crossover, shown in Figure 2.9, begins in the same fashion as PMX. However, in the first intermediate step the alleles within the matching area of Parent A are removed wherever they occur in Parent B, leaving 'holes' in the genome denoted by H. A sliding motion is applied to the remaining values — which in the given example is leftwards from the second crossover point. The holes are thus maneuvered into the matching area, where they can then be filled by the equivalently positioned gene values from Parent A. The second offspring is generated by swapping the roles of the parents.

Through their respective mechanisms, PMX has a tendency to respect absolute positions of alleles, whereas OX favours relative positions.

Figure 2.10 illustrates cycle crossover. As its name implies, CX initiates by looking for a cycle — from a randomly chosen gene in Parent A to itself. Permitted moves are from gene n in Parent A to gene n in Parent B, and from allele m in Parent B to allele m in Parent A. In this example, the starting gene is gene 1, and a

Parent A	9	4	1	2	6	5	8	3	10	7
	i ↓	v ↓	vi ↓		ii ↓	iii ↓		viii ↓	iv ↓	vii ↓
Parent B	6	1	7	8	5	10	2	9	4	3

Cycle	9→	6→	5→	10→	4→	1→	7→	3→	9

Intermediate A	9	4	1	H	6	5	H	3	10	7

Offspring A	9	4	1	8	6	5	2	3	10	7

Figure 2.10: An illustration of the cycle crossover (CX) operator generating Offspring A. Starting with gene 1 in Parent A, a cycle through the alleles of both parents is found. The steps of the cycle are denoted by the numerals i to viii. All alleles that are not part of this cycle are replaced in Parent A by 'holes', H, which are then filled by 8 and 2 from Parent B.

cycle of length 8 is found. All alleles in this cycle are preserved in position, while the remaining genes are replaced with 'holes', as seen in Intermediate A. Finally 'holes' are filled by the corresponding gene values from Parent B. Parent roles are switched to generate Offspring B.

The authors ran a grid search over mutation and crossover rate for a problem based on a department of Rangsit University in Thailand. In conclusion, OX was deemed to be the most successful operator in obtaining feasible solutions, while CX surpassed it in finding perfect (zero violation) solutions, on average. However, the discrepancy in results was not marked, and success was highly dependent on the aforementioned parameters in addition to the number of generations used.

In Yu and Sung, 2002, a modified PMX, known as sector-based PMX, was conceived, in which the 2-D matrix encoding was divided into 'sectors' according to variable type and size. Randomly generated blocks then defined the active crossover regions. Akkan and Gülcü, 2018 varied this approach by using a a period-based PMX. In both of these cases (as well as Kohshori et al., 2012, Asiyaban and Mousavinasab, 2012, Assi et al., 2018 and others), the potential for crossover to induce infeasibility was acknowledged, and different remedies were suggested. Yu and Sung, 2002 used a 'check and repair' routine. Noting that five of the hard constraints were automatically satisfied by virtue of the representation, the remaining two were handled by reassigning conflicting lectures while considering the effects on all constraints. The repair here was deterministic but others, such as Sutar and

Bichkar, 2016, have included a probabilistic element. Akkan and Gülcü, 2018 on the other hand, posited the existence of a trade-off between the flexibility of an operator and the subsequent need for chromosome repair. The authors also found that the quality of offspring mended by a complex repair procedure was not significantly better than that of its parents. This, and the time-intensive nature of explicit repair, led the authors to dispense with the repair function entirely. An efficient feasibility checker was embedded within the PMX operator instead.

Along with sector-based PMX, other crossover operators tailored to the UCTP — namely day, students, conflict-based — were tested by Lewis and Paechter, 2002. A key idea motivating all of these operators was that certain sub-timetables, or 'blocks' within the main timetable, may be worth preserving due to their positive net contribution to the wider solution quality. Conflict-based crossover, in which blocks of assigned lectures with high student commonality were prioritised for preservation, was reported to have the highest relative success.

Like crossover, standard mutation operators have been adapted in various ways for the UCTP. Abouelhamayed et al., 2017 described three that, used in combination, affected different behaviours in the GA:

1. The chosen chromosome is replaced, wholesale, with a randomly generated one, in order to promote exploration of the space.
2. The chromosome is replaced with the currently highest rated solution. As a countervailing measure to the first operator, this nudges the GA towards making more incremental changes.
3. All genes are fixed, except one whose value is varied across all of its possible other values. With each change, the fitness is tested. The procedure loops through all genes before returning the mutation with the largest fitness improvement.

Numbers 1 and 2 begin by discarding the chromosome, while the third works directly on it as a form of local search more akin to the moves discussed throughout Section 2.3.2. Mutation as local search is indeed a common theme in the literature. While traditional mutation is entirely random, Sutar and Bichkar, 2016 stated that room / teacher clashes can easily be introduced without careful restriction. In the 'informed genetic algorithm' of Suyanto, 2010, two aspects were considered in order to overcome this. Firstly, which lecture/s were mutated, and secondly, to where. Conflict-causing lectures were prioritised in a first stage, and only feasible places were made available for their reassignment. In a second stage, the mutator was configured to be sensitive to its effect on soft constraints as well, meaning that it played a active role in 'guiding' the search towards higher quality individuals. Forms of guided mutation were also used successfully in Yousef et al., 2017, Assi et al., 2018 and Yusoff and Roslan, 2019, with the latter adopting a hill climbing approach wherein only fitness-improving mutations were accepted. Understanding the tension between overly-prescriptive perturbation operators and the need for randomised exploration remains an important and open area of research.

Selection This tension is also present in the selection operator, which determines the solutions that are put forward for breeding. A regime that is too rigidly elitist can cause premature convergence to sub-optimal solutions by compromising the population diversity. On the other hand, if selection pressure is too low then little improvement is made from one generation to the next.

Yousef et al., 2017 listed some of the most popular selection techniques — stochastic uniform sampling, roulette wheel, tournament, and stochastic uniform selection — before arguing in favour of 'gender selection'. In this approach, the population is sorted according to fitness, with the top and bottom halves being labelled as female and male respectively. A default tournament size of four is run to pair male individuals with female. These pairs are then bred, ensuring that each couple includes one parent with high fitness and another with relatively low fitness, thus promoting diversity.

Other selection operators in other GAs mentioned include standard roulette wheel (Assi et al., 2018), linear scaling plus roulette wheel (Sutar and Bichkar, 2016) and tournament with elitism (Yusoff and Roslan, 2019). It is important to remember that the effectiveness of a selection operator is dependent on the interplay between problem type, instance characteristics and parameter settings. While the key concerns — diversity preservation, sufficient pressure for timely convergence, and efficiency — are common across UCTP formulations, no one size operator fits all.

Approaches to fitness evaluation Broadly, fitness tends to be defined as the sum of *soft constraint violations* (SCV), with the following adaptations also appearing in the literature:

1. Where the GA is restricted to feasible solutions either by representation or repair, the SCV is used. However, its weights w_i may be user-defined or, in the case of Yusoff and Roslan, 2019, determined by a survey asking timetable end users to rank various soft constraints in order of desirability. As soft constraints are themselves based on qualitative human preferences, some researchers incorporate fuzzy logic in order to deal with uncertainties in definition and inter-constraint conflicts. Kohshori et al., 2012, like Asiyaban and Mousavinasab, 2012, defined a fuzzy set membership function μ_i bounded between 0 and 1. The fitness was then the sum of the products of μ and w over all soft constraint violations. In Al-Ashhab and Abdulrahman, 2018, functionality resembling that of UniTime software is seen. Any entity such as a day, period or room can be given an explicit priority penalty.

2. Where the GA is permitted to enter infeasible regions, the violation of a hard constraint can be assigned an arbitrarily large penalty, which is added to the SCV. The idea ensures that, while the GA can traverse a wider search space between disconnected regions, infeasible solutions are ultimately evolved out of the population by virtue of their poor fitness. In Abouelhamayed et al., 2017, the minimum penalty for a hard violation was guaranteed to be greater than the maximum possible sum of soft constraint violations, meaning that feasible solutions were always superior.

Operationally, GA is a powerful framework for the UCTP, particularly so when hybridised with local search methods (Ilyas and Iqbal, 2015). Augmenting GAs (or evolutionary algorithms more generally) with individual learning procedures gave birth to a category known as the memetic algorithm (MA). Jat and Shengxiang, 2008 and Joudaki et al., 2011 applied MAs to the original Paechter benchmark. The former found that including two types of local search operator vastly improved performance when compared to just one. The latter then equalled or bettered results for 9 out of 11 problems (as compared to Jat and Shengxiang, 2008 and 6 other competitors) by embedding a simulated annealing process. GAs have also been used not as direct timetable solvers but as a distinct phase in a multi-phase algorithm. Xiang et al., 2024 used a GA to cluster courses into sets by leveraging graph coloring techniques, before a second tabu phase was used to then optimise the solutions.

From a software engineering perspective, there are many options for efficient parallelisation of both GAs and MAs across CPU cores or stream processors in the GPU. In the GPU-based GA of Yousef et al., 2017 for example, the fitness function evaluation was parallelised, due to its relative high complexity, the independence between individual solutions, and the large data parallelism required.

Other design aspects of timetabling GAs, such as initialisation, operator rates and crowding measures will be discussed further in Chapter 4.

2.3.3.2 Ant Colony Optimisation

First described by Dorigo, 1992, ant colony optimisation (ACO) is inspired by the collaborative behaviour of ants in seeking out the most efficient path between colony and food source. If an instance of the UCTP can be represented by a graph of nodes and edges, then a tour through this graph can encode a solution. This is referred to as the *construction graph.* Just like representation in a GA, choices about its design can impact greatly upon the metaheuristic performance. When virtual ants complete such tours probabilistically, an amount of artificial pheromone is deposited on the tour edges in proportion to the goodness of the solution found. At the same time, pheromone values are universally reduced by ‘evaporation’ after each iteration. Edges with boosted pheromone levels naturally become more attractive to subsequent virtual ants, and this positive feedback loop drives convergence to a good solution. The information encoded by pheromones is sometimes referred to as stigmergic information.

One of the first applications of ACO to the UCTP is found in Rossi-Doria et al., 2003. In this study, ACO was one of five algorithms compared by researchers from the Metaheuristic Network project. The authors reported that an ACO variant known as ant colony system (ACS) performed competitively on small benchmark problem instances, but was outperformed by tabu search, genetic algorithm, simulated annealing and iterated local search on medium sized problems. Another of the most consistently cited and successful variants of ACO besides ACS is the MAX-MIN ant system (MMAS) (Stutzle and Hoos, 1999), which was applied to the UCTP by Socha, 2002 as an unofficial entry to the first ITC. Pertinent points from these approaches and others will be reviewed in greater depth as background material in Chapter 3.

2.3.3.3 Particle Swarm Optimisation

Another nature-inspired approach that takes advantage of emergent population behaviour is particle swarm optimisation (PSO). Particles, or individuals, each maintain a vector position and a velocity. The former represents a solution in n-dimensional decision space, while the latter designates the direction for further exploration. Through the coordinated movements of its members, the population mimics the social dynamics of a swarm of insects or a flock of birds. PSO exhibits good resilience to local optima while converging towards solutions of high fitness.

The hybrid PSO of Chen and Shih, 2013 — tested on a real world UCTP — reported an interesting finding. A swap operator (described as an 'interchange') was inserted as a local search enhancement to the basic PSO. This was found to be a strong contributor to performance, irrespective of other factors such as constriction factor, learning factors, inertia and parameter values.

Later, Imran Hossain et al., 2019 pointed out that PSO was originally conceived for continuous optimisation problems, in which mathematical operators could be applied to floating point representations. Previous work (including Shiau, 2011 and the aforementioned Chen and Shih, 2013) was cited in which discrete timetabling variables had been transformed into this domain. Instead of changing their Khulna University problem to fit the PSO framework, however, Imran Hossain et al., 2019 took the converse approach, modifying the algorithm. A sequence of swaps of differing types was adapted for velocity, while selective search and forceful swap with repair were deployed for constraint handling. Over the sequence of swap types, if an intermediate solution bettered the final one, then the former was used. In terms of parameterisation, the population size was also deemed to be of high importance, with the steepest improvement seen over the range 0-150 and lesser gains thereafter.

2.3.3.4 Other nature-inspired algorithms

PSO is one metaheuristic that emulates phenomena from the natural world but, as mentioned in Chen et al., 2021, several other types have been applied to the UCTP. Amongst this plethora of metaphor-based approaches, Sörensen, 2015 argues that true novelty or distinction is hard to discern. Nonetheless, some of these efforts can at least corroborate past findings, or at best offer domain-specific components or adaptations to inspire future work. Three examples follow:

In Turabieh et al., 2010, another swarm-based approach was proposed ('intelligent fish'), in which the search space was dynamically partitioned into regions that were either empty, crowded or non-crowded. In empty areas, steepest descent was used. In crowded areas, the great deluge algorithm (Dueck, 1993) was used, with an estimated quality derived from a Nelder-Mead simplex algorithm (Nelder and Mead, 1965). In non-crowded areas, great deluge was once again used, but with an estimated quality defined by either the best current solution or the central point of the non-crowded space. Traditional single lecture move and swap operators were used to perturb solutions.

In Sabar et al., 2012, a comparison was made between the original honey bee mating algorithm (HBMO) (Abbass, 2001b, Abbass, 2001a) and a version modified for educational timetabling (HBMO-ETP). In a honey bee approach, queen drone bees are analogous to the best-found solution, broods to incumbent solutions, worker bees to new trial solutions, and mating to the crossover operator as found in GAs. A form of elitism pervades the algorithm, as the genetic material carried by the queen is invoked in every iteration. Premature convergence was prevented in HBMO-ETP by forbidding the repeat use of genetic material from any one drone.

Abuhamdah et al., 2014 developed a population based local search (PS-LS) approach based on gravitational emulation local search (GELS) Webster, 2004. Single-direction and all-direction force were utilised for searching, while MPCA-ARDA (multi-neighbourhood particle collision algorithm and adaptive randomized descent algorithm) were embedded. This was in order to off-set the exploitative shortcomings of the core population based method.

The three approaches above were all tested on the Socha dataset, enabling a direct comparison. Hybridised variants consistently outperformed base algorithms in the case of both HBMO and PB-LS, while the greatest number of best results overall was achieved by PB-LS.

2.3.4 Multi-agent systems

The population-based methodologies discussed above involve sets of individual solutions that are largely passive and uncommunicative. A distributed multi-agent framework has also been the subject of UCTP research. In such a system, intelligent agents may, for example, collaborate and exchange information through a shared language, and negotiate for resources in order to resolve conflicts and force a solution. Drawing on observations of real world timetabling practises at the University of Maribor, Strnad and Guid, 2007 argued that such an approach has a concrete theoretical foundation. A multi-agent simulation can handle multiple allocation requests made by stake-holders in the problem such as teachers, students or even abstractions such as courses. In a loose thematic parallel to the honey bee metaheuristic, these different entities are reflected by agents who are specialised in particular tasks. Srinivasan et al., 2011 suggested that some characteristics should be common to all agents, however: 1. Autonomy, 2. Intelligence, 3. Reaction, 4. Pro-action, 5. Learning, 6. Mobility, 7. Cooperation. As indicated by point 1, there is no central control, rather agents interact asynchronously according to queuing, round robin or interleaved queuing rules. A detailed survey of multi-agent timetabling approaches prior to 2014 can be found in Babaei and Hadidi, 2014, while Nouri and Driss, 2016 Houhamdi et al., 2019 are more recent contributions to the literature.

2.3.5 Novel intelligent methods

So far in this chapter, a progression of techniques have been traced that are, to various extents, well-established. These span classical mathematical forms through to metaheuristic search. While accepting there is no precise demarcation to the term; 'novel intelligent methods' refers to new ideas, adaptive combinations of traditional ones, or the inclusion of an element of learned behaviour or artificial intelligence. Some of the approaches falling under this umbrella are outlined below.

2.3.5.1 Hybrid algorithms/heuristics

Many novel approaches seek to compensate for the weakness of a certain heuristic by combining it with others, often executing base heuristics in a phased or layered approach (Kostuch, 2005). Some examples of this have already been touched upon and a fuller review can be found in Ilyas and Iqbal, 2015.

A good example of a hybrid solver using an as yet unmentioned technique, case-based reasoning (CBR), at its core, can be found in Grech and Main, 2005. CBR relies on *a priori* domain knowledge. It works on the principle that previously discovered solutions to similar problems can inform a new solution to the problem at hand. Large amounts of memory are requisitioned to store old solutions in a 'case base'. From there, four operational steps — retrieve, reuse, revise, retain — are deployed in sequence. The novelty offered by the authors was the incorporation of a GA into the 'revise' stage. The GA was able to learn and apply beneficial mutations in order to improve the health of the retrieved solutions. These were then retained as part of the CBR life cycle.

Two other hybrid approaches are of particular interest to this thesis as they produced the vast majority of the best known results for the ITC track 3 benchmark (reproduced in Table 2.3). Abdullah and Turabieh, 2012 employed a hybrid genetic algorithm with tabu search, whose movement through the search space was determined by a sequence of large neighbourhood operators. The algorithm leveraged information from its own search history to bias the choice of operators in favour of those with a high percentage of success. Kiefer et al., 2017, meanwhile, embedded an adaptive large neighbourhood search within a simulated annealing framework. Several destroy-repair operators were used. Both of these highly successful approaches used large neighbourhood structures and exhibited dynamic, adaptive behaviour, exploiting past knowledge to inform future decisions. The cumulative power of integrated metaheuristics is also a common feature and key strength.

A note of caution though was sounded in Feutrier et al., 2023, in which a hybrid local search method (comprised of sequential hill climbing, great deluge and simulated annealing) was broken down into its constituent components. Ablation testing and irace were used to determine the best ratios of deployment. The authors found that in the optimal configurations, some components had been deactivated completely. Great deluge alone was enough, on occasion, to surpass the original hybrid in performance. These results emphasised the need for sensitive tuning in hybridised solvers.

2.3.5.2 Hyper-heuristics

Hyper-heuristics depart from the convention of searching the solution space. Instead, they operate at a higher level of abstraction, searching the heuristic space in order to select the best low-level heuristics, or combination thereof, to suit a particular problem. In practise, this is achieved by adapting and/or combining promising components of known heuristics, or generating new ones. CBR can be employed as a point of reference and a measure of the success of previously found solutions.

In Pillay, 2016, four categories of hyper-heuristic were defined: 1. Selection constructive, 2. Selection peturbative, 3. Generation constructive, 4. Generation peturbative. In all cases, the first word describes whether the low-level heuristics are chosen or created, while the second word describes the nature of their action. 'Constructive' implies the type of heuristics in Section 2.3.1.2 such as *saturation degree*, while 'peturbative' refers to mutation operators.

Ahmed et al., 2015 designed nine low-level heuristics for a high school problem, seven of which were mutational and the remainder constraint-specific hill climbers. At each iteration, a formula was used to rate the heuristics according to three factors $\{f_1, f_2, f_3\}$. f_1 measured the previous performance, both in terms of time efficiency and impact on objective cost. f_2 related to the pairwise dependencies between heuristics, while f_3 recalled the time passed since the last invocation. The authors reported that this adaptive framework performed better than either a fully deterministic or probabilistic equivalent.

Aside from this high school solver, Pillay, 2016 noted that hyper-heuristics have predominantly been tested on the examination track of educational timetabling. However, an application to the ITC2007 track 3 can be found in Habashi and Yousef, 2018. An add-delete list of variable length was compiled at each step and lectures assigned, fixed or reassigned accordingly. The approach generated superior initial solutions when compared to three nearest competitors.

The most promising aspect of hyper-heuristics is their ability to generalise better over unseen problem instances than a prescribed heuristic or metaheuristic can (Obit, 2010). However, careful operator design and parameterisation is still crucial to performance.

2.3.6 Multi/many-objective approaches

In all of the methodologies discussed so far, the cost of a solution timetable has been quantified by a single value — typically a weighted sum of constraint violations. Under this scalarised approach, optimisers are by definition single-objective. Datta et al., 2007, however, criticised this use of weights, and hence the scalar objective, as arbitrary. Selecting numeric weights for incommensurable objectives, such as the desires of teachers vs. students, is inherently subjective. Moreover, constraints can conflict with one another, either directly or obliquely. A scalar cost cannot capture this intricate interplay, and thus obscures the more granular qualities of a timetable. In this context, single-objective solvers necessarily suffer from lossy information.

Multi-objective optimisation addresses this by presenting solution cost as a vector comprising a number of individual objective costs, or ‘scores’. These can either be simple constraint violation sums, or derived values representing more intangible properties, such as the conflicting teacher and student satisfaction scores in Ariyazand et al., 2022. The Pareto dominance relation can then be used to define a partial ordering on these vectors. Rather than returning a single best timetable, in the multi-objective case, a set of solutions is generated that are mutually non-dominating. These sketch out an approximation to the true Pareto front and help illustrate the intrinsic trade-offs between different objectives. A more detailed background on the multi-objective treatment of the UCTP, including its relative merits, demerits, and a review of previous approaches, is included as part of Chapter 4.

2.4 State of the art

The algorithms developed later in this thesis are based around the ITC2007 track 3 formulation, and so it is to these benchmark results that the most attention is paid when considering the progression of the state of the art. The majority of authors have followed a single-objective treatment, as prescribed by the competition rules. The five original finalists were Müller, 2009, Lu et al., Atsuta et al., 2008 Geiger, 2008 and Clark et al., 2008, from which the multi-phase constraint-based solver of Müller, 2009 was declared the winner. The ‘benchmark analysis’ section in Gozali and Fujimura, 2020 features an incomplete, but more recent, list of ITC2007 results, while Ceschia et al., 2023 includes the most up-to-date state of the art results. The majority of the overall best known results under the original timeout are shared between Abdullah and Turabieh, 2012 and Kiefer et al., 2017 (whose methods are described in Section 2.3.5.1). Lindahl et al., 2018 (discussed in Section 2.3.1.4) retains the best known result for `comp09`. For ease of reference all of these cost values are reproduced in Table 2.3, alongside the tightest currently known lower and upper bounds on the optima. The bounds have been contributed by various authors either using mathematical techniques or search methods with large iteration budgets.

Table 2.3: Best known results for the ITC2007 track 3 benchmark under competition rules (from Abdullah and Turabieh, 2012, Kiefer et al., 2017 and Lindahl et al., 2018 respectively) as well as best known lower and upper bounds. The best average results are shaded.

	Abdullah and Turabieh, 2012		Kiefer et al., 2017		Lindahl et al., 2018	Bounds	
Instance	Avg	Best	Avg	Best	Avg	LB	UB
`comp01`	5.00	5	5.0	5	12.0	5	5
`comp02`	36.36	26	41.5	34	49.5	24	24
`comp03`	74.36	70	71.7	68	74.5	58	62
`comp04`	38.45	35	35.1	35	38.5	35	35
`comp05`	314.45	295	305.2	294	373.5	247	284
`comp06`	45.27	30	47.8	41	58.3	27	27
`comp07`	12.00	7	14.5	10	35.0	6	6
`comp08`	40.82	37	41.0	39	49.7	37	37
`comp09`	108.36	102	102.8	100	100.5	96	96
`comp10`	8.36	5	14.3	7	25.7	4	4
`comp11`	0.00	0	0.0	0	6.5	0	0
`comp12`	320.27	315	319.4	306	360.7	248	294
`comp13`	64.27	59	60.7	59	69.0	59	59
`comp14`	64.36	61	54.1	51	56.9	51	51
`comp15`	72.73	69	72.1	66	74.5	58	62
`comp16`	23.73	18	33.8	26	37.1	18	18
`comp17`	76.36	60	75.7	67	86.1	56	56
`comp18`	75.64	69	66.9	64	72.9	52	61
`comp19`	66.82	57	62.6	59	64.8	57	57
`comp20`	13.45	7	27.2	19	34.3	4	4
`comp21`	100.73	86	97.0	93	103.8	74	86

To consolidate results from disparate sources, a centralised repository has recently been established (Ceschia et al., 2023). It is known as OptHub. Instance and solution data from a wide range of scheduling and timetabling problems is accessible at https://opthub.uniud.it/.

2.5 Summary

In the final section of this chapter, a summary is offered of the reviewed literature and conclusions are drawn.

2.5.1 Formulations

A distinction has been drawn between artificial UCTP benchmarks that are widely used, and individual problems that may have been studied by just one researcher.

As real world problems can be quirky or even unique, the goal of standardisation is to present challenging problems that preserve generality without over-simplification.

Ceschia et al., 2023 surveyed six such educational timetabling problem benchmarks. The authors noted with interest that the chronology corresponded with growing levels of complexity. The recent ITC2019, for example, incorporated elements of earlier post-enrolment and curriculum-based models, making it a rich, customisable dataset. Despite this undeniable trend towards increased intricacy and nuance, the authors make the case that earlier benchmarks have not yet exhausted their purpose. Many instances in ITC2007 have been solved to optimality, but these problems remain a useful test bed for landscape analysis, computational efficiency and theoretical investigation. Not least of all, the volume of past work and published results (which presently eclipses that for ITC2019) enables robust comparisons to be drawn for novel algorithms. A potential downside is the over-fitting of an algorithm to a particular benchmark, restricting its usefulness. However, concepts from successful approaches (such as the style of metaheuristic, or the profile of a cooling schedule) can provide valuable insights into how to tackle a wider range of bespoke, real world problems, thus building a bridge between theory and practise. Another benefit of standardisation (and the international competitions) is to provide a focal point for the research community and encourage collaboration and cross-pollination of ideas. Some approaches have also crossed domain boundaries within the timetabling taxonomy in Figure 1.1. GAs, applied to the examination problem by Colorni et al., 1992, are now popular for course timetabling, for example.

We acknowledge enduring issues around both constraint modelling and timetable quality measures. A good illustration of the (perhaps) unintended consequences of strict constraint definitions is the **CurriculumCompactness** vs. **Windows** distinction from Section 2.2.4. While hard constraints are mostly straight-forward to formalise, soft constraints such as these can be more intricate and difficult to define appropriately. In some cases, dependencies or other inter-variable relationships may be required. A subjective aspect may exist, which is also true of timetable quality measures/costs. Ranking the relative importance or attributing weights to different objectives is not an exact science. Solvers without hard-coded weights can offer some relief here, as well as the multi-objective approach, which does not rely on a scalarised objective at all. The latter is an evident gap in the field as the vast majority of UCTP research has taken a single-objective stance (Ceschia et al., 2023). Another avenue ripe for exploration is the inclusion of more 'intangible' objectives. Limited examples of these can be found, such as Mühlenthaler and Wanka, 2016 and Akkan and Gülcü, 2018, who devised metrics for 'fairness' and 'robustness' respectively, but most researchers persist with the prescribed objectives. Additional holistic objectives such as these lend themselves well to a bi- or multi-objective treatment, in which an approximation to the Pareto front can then reveal the extant trade-offs.

2.5.2 Solvers: A thematic summary

Across the myriad approaches applied to the UCTP, many thematic contrasts are in evidence. The pursuit of optimality — the exact solution — has fallen out of favour, with approximation techniques now more prevalent. While metaheuristics cannot guarantee optimality like a pure mathematical model can, the latter has proven impractical for navigating the immense search spaces induced by departmental-sized problems. A potential trade-off in quality is thus considered acceptable. Chen et al., 2021 remarked that techniques from operational research may be effective for generating feasible solutions, but remain cumbersome for further optimisation. The legacy of this is reflected in the popularity of solvers with multiple phases. The sequencing strategy can vary, with some authors assigning periods first and rooms second. Others address the hard constraints first before optimising over the soft constraints — a method that has proved highly successful.

Concentrating on metaheuristics for university timetabling, a meta-study by Bashab et al., 2020 gave an insight into the frequency of use of various types. A sample of 131 papers published between 2009-2020 (a period of increasing interest in the field) is broken down in Figure 2.11. Nature-inspired methods are prominent, while genetic algorithms and hybrids, between them, make up nearly half of the total. In terms of hybrids, we have discussed the compositing of exact and inexact methods,

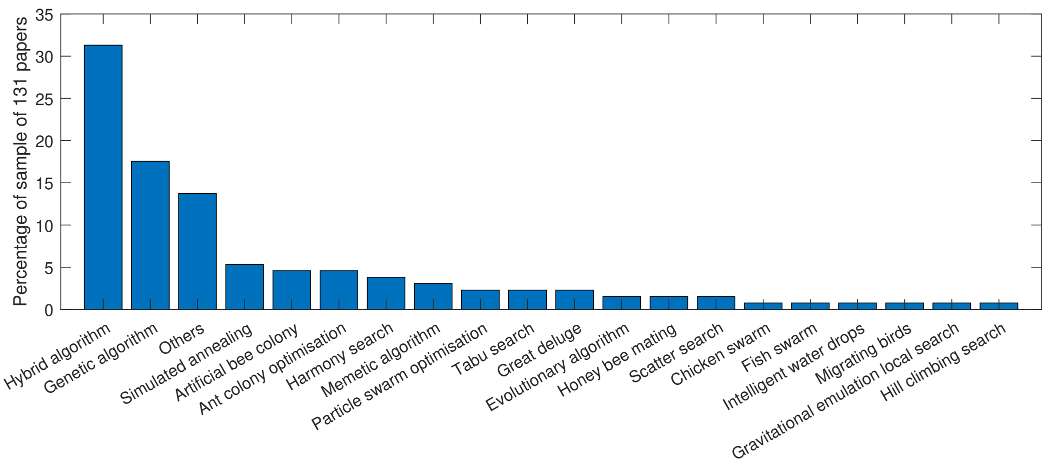


Figure 2.11: A breakdown of the metaheuristics employed in a sample of 131 academic papers published between 2009 and 2020 for examination, course timetabling or both. Adapted from Bashab et al., 2020.

as well as the mixing of metaheuristics that complement each others' strengths. One conclusion is that a carefully designed hybrid can out-perform its constituent components, resulting in an extraordinarily powerful search.

The topology of this search space is partly dependent on the encoding scheme. This review focused on direct style encodings, whose popularity is explainable by a number of factors. Their interpretation is highly intuitive (for example representing a day of the week by an integer from 0 to 4, or 1 to 5). It is easy to 'bake in' the satisfaction of certain hard constraints using appropriate variable ranges. Similarly, the design of genetic and other operators is much simplified. For these reasons, a direct encoding is adopted in the technical work that follows. It is recognised however that indirect encodings can offer greater expressiveness with regard to complex constraints, acting as timetable 'builders', much like the constructive heuristic rule sets discussed previously.

In terms of constraint handling, a range of perspectives have emerged on the question of strict enforcement vs. full relaxation. Allowing movement through the infeasible space, subject to harsh penalties, can be a useful way to reconcile the two. Aiming for strict enforcement can be problematic due to the nature of the operators and the desire to avoid expensive repair mechanisms. Alvarez-Valdes et al., 2002 pointed out the difficulties in navigating the search space using overly simple moves.

Assignments that may be good in one respect for a lecture may also be impossible due to the clashes induced between rooms or other entities.

The scope and size of perturbation moves is equally important. The success of adaptive algorithms that can call on a variety of neighbourhood structures suggests that both large and small moves have a role to play. The work of Yusoff and Roslan, 2019 also indicates that a guided mutator can aid convergence better than pure random mutation can. There is a danger, however, in being overly prescriptive and stifling exploration. Achieving the right balance also requires the prudent setting of budgets and other relevant parameters.

Optimisers can indeed be sensitive to individual parameter values or combinations thereof. Systems with fewer settings to tune, or those that have adaptive hyper-parameter tuning, are therefore naturally attractive. A trend is also evident towards problem agnosticism in solver design. Instances in the ITC2007 benchmark vary from large and heavily constrained (`comp05`) to moderately easy to solve (`comp11`). A versatile and reliable optimiser ought to perform consistently across this diversity of problems without the need for explicit re-tuning.

It has also been seen how both stochastic and deterministic processes can coexist or be nested within components of an optimiser. Both are present within certain constructive heuristics, where the ordering (and hence choice) of lecture is deterministic at each step, while the assignment period itself is left to chance. Evolutionary algorithms, by their nature, incorporate randomness. Consideration must be given then to the variance of the performance measure across multiple repetitions of such an algorithm. Low deviation from the mean is preferred as this allows stronger claims to be made about the expected performance.

It is implied by various authors (Bashab et al., 2020, Wahid and Hussin, 2017, Abdelhalim and El Khayat, 2016) that run time is not of paramount concern in the field of timetabling. While automation-enhanced efficiency is laudable, universities often prepare a timetable weeks or months in advance of a new term. In many cases, running an optimiser for days (or longer) is unlikely to cause practical issues. A notable exception to this is when the full requirements of the timetable are not known or may be subject to repeated changes over time, which is addressed in the later work on robustness. For the fair comparison of algorithms, the ITC2007 rules mandate the use of a single CPU core. In a real world use case though, independent runs of stochastic algorithms can be parallelised across multiple CPU cores and the best result returned. Further, the use of GPU-based programming has been mentioned as an avenue for future inquiry. Evolutionary approaches are tailor-made for parallelisation (across fitness functions or populations, for example) and could benefit from the high core count and multiple thread execution of a GPU.

It is probably no coincidence then that population-based approaches have often been preferred over single-solution / trajectory-based alternatives. If a slower convergence rate and extra computational burden are deemed acceptable, then population-based solvers can offer advantages. Global exploration is particularly important in complex landscapes, and populations are generally more resilient with regard to local optima traps. Maintaining a population also means a diversity of possible solutions can be supplied to a decision maker without advance knowledge of his or her preferences.

Finally, it is recognised that some published procedures or commercial solvers for timetabling incorporate (or require) a manual override for adjustments. These are generally referred to as interactive or semi-automatic solvers. In real-world applications, human intervention may prove useful in managing a timetable in the face of unforeseen changes that occur during a semester. Understanding what makes a timetable robust to such changes in the first place though is an interesting area of open research and could potentially increase the functionality of a fully automatic solver. The work presented in Chapters 4 and 5, in which robustness is ultimately incorporated into a many-objective solver, is premised on this very scenario.

In dissecting the UCTP trends in the literature, it is apparent that some methodologies show greater promise than others over a wider range of formulations. No single approach or algorithm is consistently superior, however.

In the chapter that follows, a review is provided of a popular metaheuristic — ACO, which ranked highly in Figure 2.11 — and an implementation known as

MAX-MIN ant system is developed. In particular, research is carried out into how performance is affected by the order of course/lecture assignment and whether beneficial permutations can be predicted through learning. Chapter 4 then presents and develops a many-objective approach, in which individual soft constraint violations are treated as individual objectives. This work is built upon in the subsequent Chapter 5, in which ideas of diversity in the genotype space, additional perturbation operators and, finally, a robustness objective are investigated. Chapter 6 summarises the thesis and provides directions for future work.

3. A Study on Course Ordering in an Ant Colony Optimiser

3.1 Introduction

A number of studies have employed ant colony optimisation (ACO) to solve the UCTP, with some adaptations producing promising results. Particular focus is given in this chapter to two key and linked aspects of ACO design: constraint handling, and the order in which courses and lectures are assigned by the virtual ants. The latter is an unavoidable and non-trivial design choice in ACOs, and yet there is uncertainty regarding the best approach. Various ideas such as random ordering or the use of simple heuristics based on graph colouring have been proposed in the literature. However, the relationship between lecture ordering and quality of constructed timetables remains unclear. A common underlying aim is to try to assign 'harder' (or more tightly constrained) lectures earlier — when a wider choice of feasible placement options remains available.

The work in this chapter seeks to investigate the effect of course lecture ordering empirically through the development and implementation of a novel MAX-MIN ant system (MMAS). A basic model is first used as a proof of concept, before a refined model is developed. In each case, the MMAS is run over all possible course permutations of a set of small training problems, to examine how this re-ordering affects its ability to locate the optimal timetable. A machine learning pipeline is then devised with the aim of predicting beneficial permutations for unseen larger problems from the ITC2007 track 3 benchmark. Specifically, a regression model is trained using human-interpretable features such as 'number of lectures' and 'number of unavailable periods'. The purpose of this deliberately simplified approach is to assess whether permutation predictions can be made quickly on these unprocessed summary statistics alone. The target value for the supervised learning is a measure of the success of a particular permutation in locating the optimum over multiple repetitions of the MMAS. By leveraging this information and using a mapping, a course Permutation Predictor is developed for much larger problems. This learning and scaling approach is then trialled on subsets of small, medium and large problems from the benchmark. Additionally, the core MMAS is enhanced with an optional local search routine so that the any-time performance of variants with and without this component can be studied and assessed.

Comparing absolute results against those published in the literature is outside the scope of this study. Rather, the study is motivated by understanding how relative differences in results given by the MMAS on a particular problem can be attributed to differences in the course lecture ordering.

Significance testing is applied to the any-time performance of the MMAS variants. An analysis is provided both of the importance of the model features, as well as patterns of performance between permutations. For the final model training set, a strong correlation is found between the similarity of permutation and relative performance. Features are shown to be most informative for the courses near the beginning and end of a permutation. Among the smaller problems in the ITC test set, the learnt approach improves relative timetabling performance compared to a random baseline, suggesting that there is potential in further developing the idea to tackle its largest problems.

Section 3.1.1 offers a review of the ACO literature relevant to this chapter. It concentrates on the order of course lecture assignments and constraint handling. Methodology is covered in Section 3.2, with Section 3.2.1 recounting the development of each component of the MMAS. These include the solution representation, penalty scheme, pheromone update procedure, dynamic hard constraint handling and smart function evaluations. The section continues with the local search methodology (Section 3.2.2), and finally the proposed approach for learning and predicting the course assignment order using a pipeline of regression model (Section 3.2.3) and genetic algorithm (Section 3.2.4). Section 3.3 gives details of the experimental setup as they relate to the aforementioned pipeline — firstly for a simplified proof of concept model and then a more refined model. This section also includes the mapping procedure as well as specifics of the problem instances, parameter settings and CPU budgets used. Results for the final model are revealed in Section 3.4. Two benchmark experiments, in which the final system is compared against a random baseline (both with and without local search), are then outlined along with results and analysis. Conclusions are drawn in Section 3.5 before some directions for further work are suggested.

3.1.1 Background

3.1.1.1 ACO

Algorithm 3: A basic ACO

```
1  Input: Parameters $\rho$, $k$, problem instance
2  Output: Best solution found
3  Initialise full construction graph
4  while termination criteria not met do
5      for each ant do
6          Begin path at start node
7          while ant not reached end node do
8              Choose next node probabilistically based on state transition rules
9          Evaluate solution encoded by completed path
10     Optionally improve iteration best solution using local search
11     Apply offline pheromone update
```

Algorithm 3 outlines the steps of a basic ACO, where $0 \leq \rho < 1$ is the 'evaporation' parameter used to uniformly geometrically decrease pheromone levels in line 11, and k is the number of virtual ants looped over in line 5. While these steps describe a generic ACO, domain-specific variations and enhancements have been developed in the literature.

In Socha, 2002, the construction graph for the proposed MMAS was a 2-D lattice of nodes representing lectures $\times$ periods, with one extra node at each end to serve as start and end points. A trivially small example of this, with 3 lectures $\times$ 2 periods, is illustrated in Figure 3.1. Node 7 in this case encodes an assignment of lecture 3 to period 2.

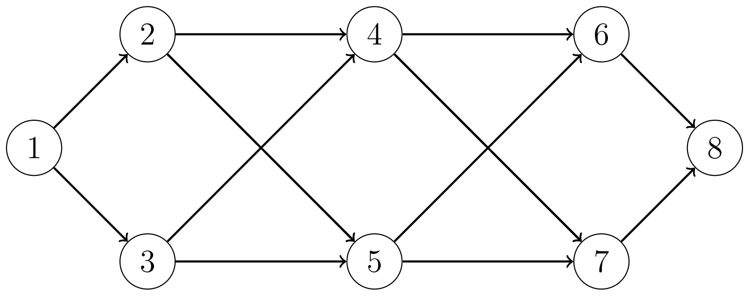


Figure 3.1: 2-D construction graph with three lectures (arranged horizontally) × 2 periods (vertically), plus start (1) and end (8) nodes.

Virtual ants construct paths from start to end, meaning that the sequencing of lecture assignments is fixed. An attempt was made to pre-order lectures by 'hardest to assign' first by way of the following:

$$c(l, l') = \begin{cases} 1, & \text{if lectures } l \text{ and } l\text{' have students in common} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

Where $c(l, l')$ is a binary function that takes pair of distinct lectures and returns student commonality.

$$d(l) = |\{l' \in \mathcal{L} \setminus \{l\} \mid c(l, l') \neq 0\}|. \tag{3.2}$$

Where $d(l)$ is a function of a lecture that returns the count of distinct lectures it shares students with. $\mathcal{L}$ is the set of all lectures. A total order $\prec$ on lectures is then defined:

$$\begin{aligned} l \prec l' \Leftrightarrow \; & d(l) > d(l') \\ & \vee \; d(l) = d(l') \wedge [f(l) < f(l')]. \end{aligned} \tag{3.3}$$

Where $f : \mathcal{L} \rightarrow \mathbb{N}$ is an injective function invoked only for breaking ties.

This is a valid and simple way of defining 'hardness to assign', which nonetheless exploits only one source of available information — namely student commonality.

The 2-D structure also makes this graph less susceptible to combinatorial explosion when scaling. These beneficial features led to the very same architecture being adopted in Ayob and Jaradat, 2009 and, it is implied, the same method of ordering lectures. This total ordering was likewise adopted for a more recent UCTP ACO in Badoni et al., 2023, in which a novel student grouping method was used to produce competitive results for the ITC2007 track 2 benchmark.

Socha et al., 2003 attempted a direct comparison of ACS and MMAS for the timetabling problem. The differences between the two algorithms in this context are subtle but critical:

- While MMAS relies solely on stigmergic information, ACS also incorporates heuristic information. This information is determined by evaluating the constraint violations caused by making a particular next assignment, given the partial assignments made in the path so far. For the purposes of this calculation, hard and soft constraint violations are weighted parametrically.
- In many ACO implementations, the solutions found by the basic algorithm are enhanced by way of a local search improvement phase. In the case of MMAS, only the solution with the fewest constraint violations is chosen, which is then improved upon until either a local minimum is encountered or a time limit elapses. ACS however runs its improvement routine on all candidate solutions generated, but varies the number of steps according to how good the starting solution is.
- As originally described, the procedure by which pheromones are updated differ between ACS and MMAS. In the former, virtual ants apply local updates on edges during construction of a path. After each iteration, the single best performing ant (*iteration best*) then applies a global pheromone updating rule. In the latter, a dynamic mix occurs whereby one of either the global best ant or the iteration best ant update pheromones on their respective trails. In practice, update procedures are often modified. For example in Matijas et al., 2009, which is discussed in further detail later, either the global best ant or

the iteration best ant are used in a static ratio of 5% to 95%. Further, not all edges in a path are updated uniformly, but according to a measure of the goodness of each individual assignment.

- Perhaps the most fundamental difference between these variants though is that, in MMAS, the pheromone levels are bounded between parameters τ_{min} and τ_{max}. This mitigates against premature convergence to potentially sub-optimal solutions by allowing for more sustained exploration of the search space. The extent of this can be controlled by the tuning the bounding parameters.

In Socha et al., 2003 — with both algorithms implemented faithfully as outlined above — MMAS outperformed ACS on all the benchmark problem instances tested. However, the authors posited that the high cost of the ACS local search routine may have been partly responsible for this disparity. In attempting a more equitable comparison, the authors tweaked the algorithms in order to make their operational complexity more similar. While this had the effect of bringing the quality of results closer together, on average MMAS still remained the better performer.

In a further comparative study, Socha et al., 2003 found that MMAS comprehensively out-performed ACS on a set of UCTP instances of varying size. These instances were created by the Paechter problem generator (as referred to in Lewis and Paechter, 2005). It was concluded that the ACS had made inefficient use both of heuristic information and local search.

3.1.1.2 Course assignment order

Matijas et al., 2009 investigated the laboratory exercises timetabling problem (LETP) — a subset of the general UCTP. Efficiency of the construction graph was again cited as a cardinal concern, particularly as the authors' aim was to implement their system for real world scheduling at their institution, the University of Zagreb. Construction of the graph here differed from Ayob and Jaradat, 2009 with the introduction of 'dock nodes' — an ordered pair of room and period. The construction graph was tripartite, with independent sets L of lectures, D of dock nodes and S of students. Edges were used to connect nodes in L with D and in D with S, yielding an upper bound on the number of edges of $(l+s)pr$, where l, p, r and s were used to denote numbers of lectures, periods, rooms and students respectively. Pre-processing typically reduced the graph by eliminating edges known *a priori* to be invalid. The issue of maintaining feasibility was handled by so-called ‘constraint fences’, which were particular to each individual virtual ant. While the construction graph in essence remained static, certain edges were rendered invisible to virtual ants whenever crossing them would render a partial assignment infeasible in that particular path. This was achieved by forcing the heuristic value to zero as appropriate. The order in which the events — laboratory exercises in this study but equivalently, lectures — were scheduled was not fixed, but reset before each new iteration. The new order was determined with reference to the best path from the previous iteration. Events having more unscheduled students in this path were given higher priority in the next iteration.

The idea of dynamic lecture ordering also appears in Mayer et al., 2012, whose approach earned 4th place for the ITC2007 track 2 (post enrollment). Here, pheromones were associated with nodes rather than edges. This is permissible because, while the order of lecture assignments may impact directly upon the efficacy of an algorithm, it has no meaning in the context of a solution’s finalised timetable. Therefore nodes, along with their pheromone values, can be shuffled during a run. The approach taken was to generate a uniform random permutation of lectures, π^e, at each iteration. The priority of periods and rooms were also randomised, but with weights corresponding to their current pheromone values. In this way, the probabilistic element of the ACO was shifted and the assignments were technically made in a greedy deterministic fashion. The naive randomisation of π^e would appear less sophisticated than in Matijas et al., 2009 and yet there is uncertainty in the literature around the best method and even the importance of lecture ordering.

Patrick and Godswill, 2016 are one of the few research teams to have used the ITC2007 track 3 benchmark in their ACO-based approach to timetabling as a single-objective problem. Of particular interest for this review is their initial phase construction graph, which was bipartite and fully connected. One part was made up of lecture nodes while the other comprised combined room-period nodes. Trails began at a random lecture and there was no fixed ordering of lecture assignments. The authors accepted in their conclusions that the expressiveness afforded by this representation was worth the higher compute time, also finding that 8 was the optimal number of virtual ants for their system. A recent ACO interpretation by Fotovvati and Mirghaderi, 2023 retained the same style of bipartite construction graph, while introducing a dynamic bias to the pheromone update rules which was found to aid convergence.

Thepphakorn and Pongcharoen, 2012 also experimented on the ITC2007 using ACO. Their approach to ordering invoked graph colouring based heuristics, namely *random ordering* (RO), *largest enrolment first* (LE), *largest degree first* (LD), *largest coloured degree first* (LCD), *largest weighted degree first* (LWD), and *saturation degree* (SD) (Burke et al., 2007). These heuristics work by determining a priority level for each event (equivalently, course or lecture) and ordering accordingly. LE, for example, attaches higher priority to course lectures that have a larger number of enrolled students, whereas the authors' proposed method — *largest unpermitted period degree first* (LUPD) — uses the number of unavailable periods in a similar fashion. Employing an ACO variant known as rank-based ant system (AS-Rank) (Bullnheimer et al., 1999), the construction graph was determined heuristically in an initialisation phase before the optimisation was allowed to proceed as normal. Six heuristic techniques were tested: RO and LE, as well as LUPD — alone and combined in series and in parallel with LE. The percentage of solutions that were feasible over a limited-iteration AS-Rank run was used as the primary performance metric. Results showed superior performance of the hybridised LUPD+LE heuristics, both in terms of the feasibility metric and speed of computation.

Other practitioners using ACOs have implied the use of similar simple heuristics for event ordering. In Lutuksin et al., 2009, the brief description is found: *"sort courses (C) according to the priority of events and student sizes"*, with no further detail provided. Meanwhile, in Munirah et al., 2019, experimental results were presented with and without *"priority"*, where, *"priority is given on several tests to* *determine the allocation of course timetabling for a different large number of students to be scheduled into the prescribed time".*

3.1.1.3 Constraint handling

As the UCTP is a highly constrained problem, another vital aspect of any ACO solver is (hard) constraint handling. Blum and Roli, 2003 listed three high-level approaches to dealing with an infeasible solution returned by a metaheuristic:

1. Discard the solution as useless.
2. Apply an artificially heavy penalty to the solution.
3. Use appropriate operators to repair the solution.

In Patrick and Godswill, 2016, illegal moves that may have been needed to complete a solution were only made available to a virtual ant once no legal alternatives remained. This is one of several types of possible constraint relaxation approaches. Permitting illegal moves, even as a last resort, can result in infeasible timetables during the optimisation process.

Nguyen et al., 2016 related that in many ant colony applications, hard constraints are subject to a full relaxation. At the end of an iteration, any constructed solutions found to be infeasible are then penalised heavily. This approach opens up connectivity across the entire search space while making individual solutions easier to construct. However, overall efficiency losses may be incurred as the search process can take longer to locate feasible areas of interest. To combat this performance issue, the authors — using an ACO on an optimal crop and water allocation problem — adopted an approach at the other extreme. For each trail node visited, all subsequent nodes that would cause a hard constraint violation by virtue of a dependency were effectively eliminated from the construction graph in real time. The authors referred to the procedure as a dynamic decision variable (DDV), and its effect was to concentrate the search exclusively in the space of feasible solutions. Cross-problem domain applicability is evidenced in Golding et al., 2017, in which the authors incorporated DDV in a multi-objective ant colony modelling global food security.

Another concept which appears in the ACO for timetabling work by Djamarus and Ku-Mahamud, 2009 is that of negative pheromones. Positive pheromones are traditionally used in ACOs to boost the probability of choosing paths leading to preferred solutions. Their negative counterpart was used here to discourage virtual ants from choosing paths that led to 'dead ends' in the solution construction. Chmait and Challita, 2013 elaborated further on the idea as it related to their ACO treatment of the examination scheduling problem. At the point of assignment of an exam j that immediately followed an exam i, checks were made not only for conflicts between the placements of i and j, but also potential conflicts with all other exams yet to be assigned. Where conflicts were uncovered, a negative pheromone was incorporated into a local update procedure performed by virtual ants at each step of their construction trail. The risk of running out of feasible placement options later in construction was therefore pre-emptively mitigated.

3.1.1.4 Summary

It is apparent that the twin aspects of lecture assignment order and constraint handling can complement each other in a well-designed ACO. A beneficial lecture ordering can decrease the amount of naturally occurring hard constraint violations and thus the computational load required to handle them. This section has reviewed various approaches to both elements, as applied to both real world and simulated timetabling problems as well as other related problem domains. The following sections cover the technical details of the proposed MMAS, supplementary components and experimentation.

3.2 Methodology

For ease of reference, this section begins with a high level overview of the entire proposed system, illustrated by two schematics. The first schematic, Figure 3.2, depicts the training process. Beginning with a set of small training problems, all of the possible course permutations are queued up. For each distinct permutation, the MMAS then solves the corresponding problem. A performance measure can thereby be attributed to each permutation for each problem. These target values are then used to train a regression model, denoted for convenience by TO. The full rationale behind TO is given in Section 3.2.3, while the generation of its training set is described later in Section 3.3.2.

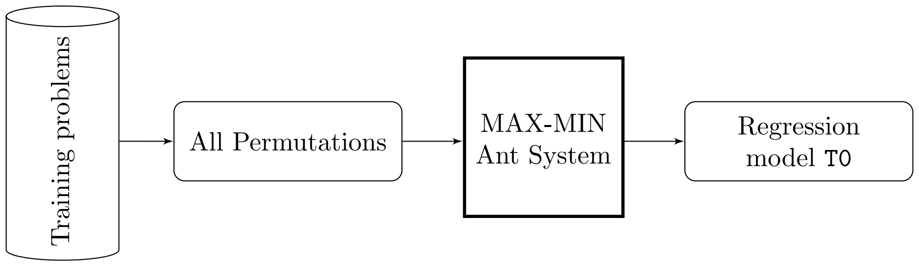


Figure 3.2: A schematic of the training phase.

The second schematic, Figure 3.3, illustrates the benchmark experiment phase, in which unseen ITC2007 problems are solved. Having learned how to identify a good permutation of courses, TO is used to inform the operation of a basic genetic algorithm[1] (details of which are given in Section 3.2.4). The permutation space of the problem is mapped to the equivalent space of the learned permutations. The genetic algorithm searches this space to locate a promising permutation. The mapping, TO, and the genetic algorithm are referred to in combination as the Permutation Predictor. Set against this approach is a baseline comparator known as Random Permutation, which has no learned behaviour and merely selects a permutation at random. The MMAS solves the problem as before. For both Permutation Predictor and Random Permutation, local search can be enabled or disabled as part of the MMAS, meaning that, in total, there are four variants tested. These are denoted PP-LS, PP, RP-LS, and RP, as laid out in Figure 3.3.

3.2.1 MMAS design

3.2.1.1 Solution representation

In contrast to the 2-D construction graph used in Socha, 2002, the initial (proof of concept) design used a 3-D lattice of rooms × periods × lectures, plus start and end nodes. A 3-D architecture contains additional information and was described

[1] A simple greedy search is sufficient for the proof of concept stage and so replaces the genetic algorithm for the proof of concept version only.

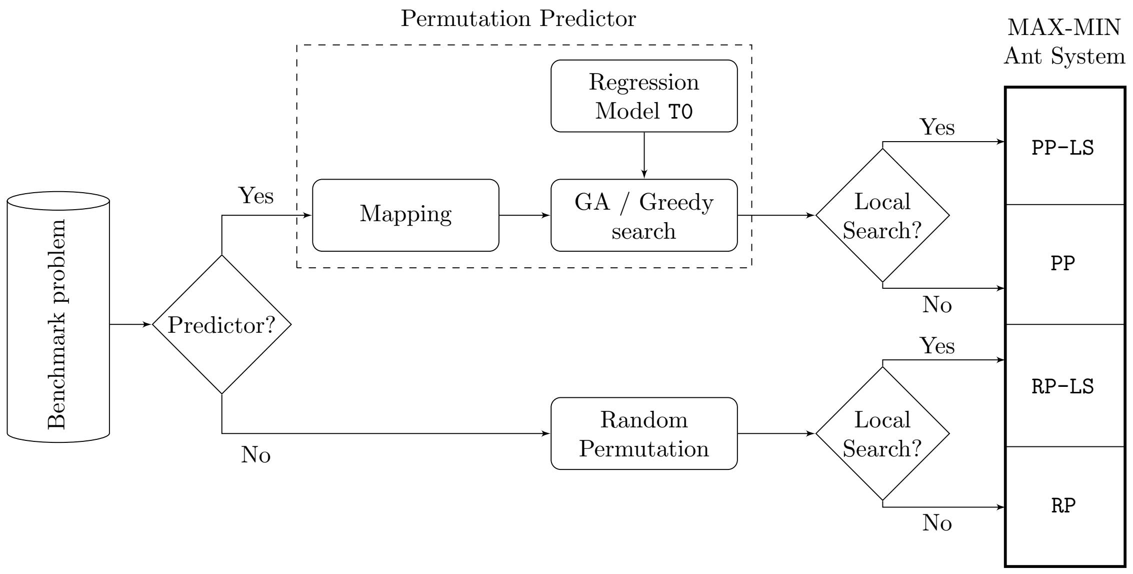


Figure 3.3: A schematic for the benchmark experiment phase. The mapping, `T0`, and the genetic algorithm comprise the proposed Permutation Predictor. Random Permutation is used as a naive baseline comparator. Experiments are conducted with and without local search, giving a total of four experimental MMAS variants in all (`PP-LS`, `PP`, `RP-LS` and `RP`).

in Mayer et al., 2012 as a more *"expressive"* option than the graph in Figure 3.1, but with a cost trade-off. In gaining a more rigid, comprehensive embedding of hard constraints, a data structure of higher combinatorial complexity is required, which can impact on efficiency. Below, some reasons are given as to why this trade-off was initially accepted, and how the cost was mitigated.

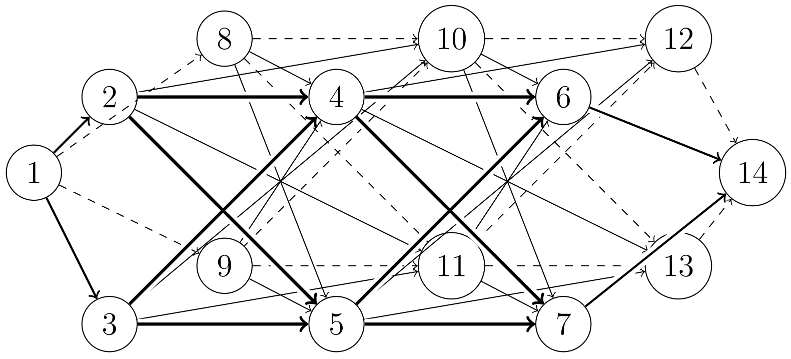


Figure 3.4: 3-D construction graph with 3 lectures (arranged horizontally) × 2 periods (vertically) × 2 rooms (front to back), plus start (1) and end (14) nodes.

Figure 3.4 shows the (full) 3-D composition for a toy example, with lectures on the horizontal (left to right), periods on the vertical (top to bottom) and rooms on the mutually orthogonal plane (front to back). A direct ordered encoding was used as shown such that, here, node 7 denotes an assignment of lecture 3 to period 2, room 1. The layer of nodes depicted with dashed edges (8, 9, 10, 11, 12, 13) constitute room 2 assignments, and nodes 1 and 14 are start and end nodes respectively. In general, this graph has $lpr+2$ nodes and an upper bound of $2pr+p^2r^2(l-1)$ edges for the given architecture. Inequality (3.4) gives an upper bound on the density of the adjacency matrix and, in these applications, this ratio was typically observed to be less than 0.10.

$$\frac{2pr+p^2r^2(l-1)}{(lpr+2)^2} \leq 0.25. \tag{3.4}$$

Computational savings were therefore made by storing the construction graph as a sparse matrix data structure. Pre-processing was invoked on this to remove edges from all nodes representing unavailable assignments in the problem instance, thereby guaranteeing satisfaction of **H4**. Further savings were attempted by decreasing the graph size dynamically. At each step in a virtual ant path, edges connected to the following nodes were removed: The previous lecture $\times$ room layer and the previous period $\times$ room layer, to ensure feasibility with regard to hard constraints **H1** and **H2** respectively. In certain problems that are highly constrained by unavailable periods, this could sometimes cause virtual ants to reach dead ends when constructing a path. There are several ways to handle this possibility — the proof of concept method being to abort the partial path and enforce an unguided restart. Algorithm 4 outlines the initial MMAS — note the instruction to "restart current path" on line 15.

It was only when experimenting with larger problems that two major pitfalls of this first design became evident. These are the memory issues caused by the highly connected construction graph, as well as the recurring failure of restarts. An improved representation was therefore devised for the final design, and the path restart replaced with a more nuanced strategy.

This final model construction graph is depicted in Figure 3.5 as a 3-D matrix plus start and end nodes. Its dimensions are equal to the cardinalities of the sets of periods ($|\mathcal{P}|$), lectures ($|\mathcal{L}|$) and rooms ($|\mathcal{R}|$). Rather than connect nodes by edges, an edgeless graph is employed instead, in which the pheromone is deposited directly on nodes. It is noted in Mayer et al., 2012 that this treatment — known as a permutation-based ACO — removes the complexity and memory issues associated

Algorithm 4: Proof of concept MMAS with path restart

```
1  Input: Parameters $\tau_{min}$, $\tau_{max}$ (bounding), $\rho$ (evaporation), $q$ (deposition
     scaling factor), $k$ (number of virtual ants), problem instance, course
     permutation
2  Output: Best timetable solution found, by SCV value.
3  Initialise full construction graph MatrixH
4  Remove unavailables
5  MatrixP $\leftarrow$ MatrixH$\times$ $\tau_{max}$
6  for iterations do
7      Initialise empty data structure antPaths to store ant paths
8      for each ant do
9          Begin path at start node
10         Initialise dynamic copy MatPCopy of MatrixP
11         Initialise empty CurClashList
12         while ant not reached end node do
13             Remove edges of nodes in CurClashList from MatPCopy
14             if no possible next move then
15                 Restart current path
16             else
17                 Select next move probabilistically
18             Append next move to current path
19             Remove edges of previous lecture $\times$ room layer of nodes from
                 MatPCopy
20             Remove edges of period $\times$ room layer of nodes from MatPCopy
21             Add nodes representing future curriculum clashes to
                 CurClashList
22         Save ant path to antPaths
23     Evaluate SCV for all paths and select update path according to schedule
24     Pheromone, evaporation update procedure on MatrixP and bound new
         values by $\tau_{min}$, $\tau_{max}$
```

with handling a large matrix of edge values. Each virtual ant begins its trail at the start node, s, before visiting exactly one node in each lecture plane and terminating at node e. Every node in the main matrix corresponds to a distinct placing of a lecture, and thus a trail encodes a complete solution timetable.

3.2.1.2 Penalty scheme

The original ITC2007 scoring regime is extended to encompass infeasible solutions too. The SCV therefore incorporates a supplementary heavy penalty for every hard constraint violation. In the case of **H1**, **H2** and **H4**, no such violations are possible, as explained. In the case of **H3** and **H5**, a penalty of 20,000 points is incurred for

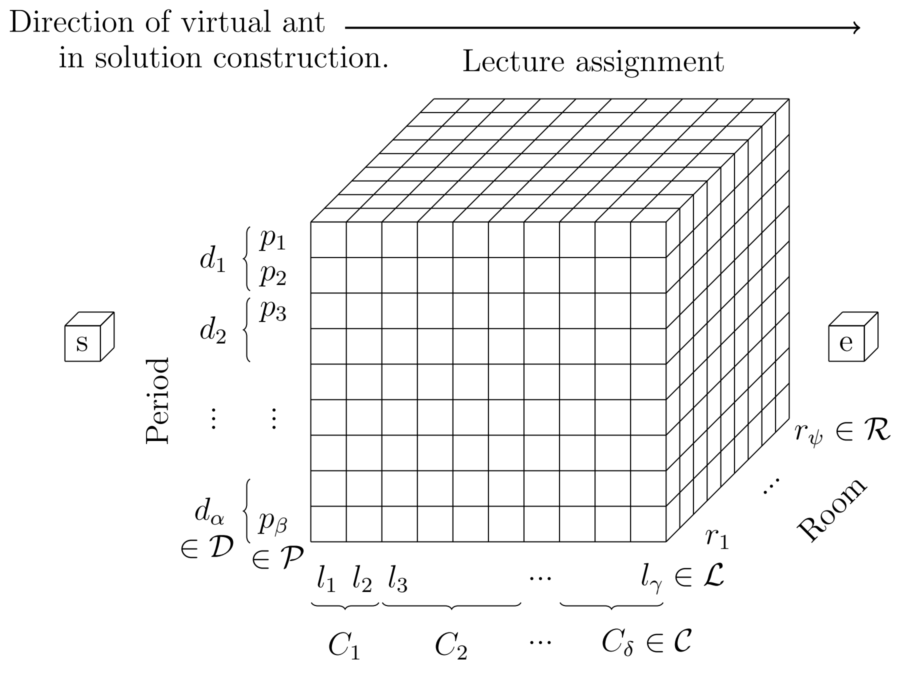

Direction of virtual ant in solution construction.
Lecture assignment
d_1
p_1
p_2
d_2
p_3
s
e
Period
$d_\alpha \in \mathcal{D}$
$p_\beta \in \mathcal{P}$
$r_\psi \in \mathcal{R}$
Room
r_1
l_1 l_2 l_3 ⋯ $l_\gamma \in \mathcal{L}$
C_1 C_2 ⋯ $C_\delta \in \mathcal{C}$

Figure 3.5: The MMAS construction graph as used in the final model, where $\{d_1 \dots d_\alpha\} \in \mathcal{D}$ is the set of days, $\{p_1 \dots p_\beta\} \in \mathcal{P}$ is the set of periods, $\{l_1 \dots l_\gamma\} \in \mathcal{L}$ is the set of lectures, $\{C_1 \dots C_\delta\} \in \mathcal{C}$ is the set of courses, $\{r_1 \dots r_\psi\} \in \mathcal{R}$ is the set of rooms and s and e are start and end nodes respectively. Cubes in the diagram represent nodes in an edgeless graph, which is stored as a 3-D matrix. Additionally, $\mathcal{T}$ is the set of teachers and $\mathcal{N}$ denotes the set of unavailable periods. $\{U_1 \dots U_\theta\} \in \mathcal{U}$ is the set of curricula, where courses are members of one or more curricula.

every pairwise curriculum or teacher clash respectively. The value of this penalty is such that a feasible timetable has a lower cost than an infeasible one with high probability, regardless of how great the number of soft constraint violations.

3.2.1.3 Pheromone update

Pheromone update is achieved by an elite update strategy based on the original MMAS description. While k virtual ants independently complete k trails in a given iteration, only one solution (assigned to a variable `updater`) is used in each iteration to update the pheromone values. This may either be the highest quality solution found in the most recent iteration (`iterBest`) or the highest quality solution found during the search history thus far (`globBest`). The choice of which is determined by a schedule as outlined in Table 3.1. For the first 5% of the iteration run, `iterBest` is used exclusively. From then onwards, `globBest` gradually begins to predominate according to a piecewise function of the iteration number. The purpose of this

Table 3.1: The schedule of solution selection for update in MMAS.

Period of run	`globBest` frequency	`iterBest` frequency
0% - 5%	0	1
5% - 15%	1/5	4/5
15% - 25%	1/3	2/3
25% - 50%	1/2	1/2
50% - 100%	1	0

function is to slowly increase intensification following initial exploration of the search space.

In the proposed system a CPU time budget is specified. When the budget has been exceeded the ongoing iteration is completed before termination. To ensure that the schedule in Table 3.1 runs its course, an adaptive re-scaling factor is introduced. This compresses or expands the schedule in real time based on the number of iterations forecast to complete before the CPU time limit is reached.

The formula for the (unbounded) update of a pheromone value h_i is given in (3.5).

$$h_i := \begin{cases} (1-\rho)h_i + \frac{q}{SCV}, & \text{if node } i \text{ is visited in trail} \\ (1-\rho)h_i, & \text{otherwise.} \end{cases} \tag{3.5}$$

Where $0 \leq \rho < 1$ is the 'evaporation' parameter and q is a deposition scaling factor.

For the MMAS to function as intended, the deposition quantity $\frac{q}{SCV}$ ought to be of the same order of magnitude as the evaporation. For this reason, and for experimental consistency, $\frac{q}{SCV}$ is fixed against ρ by tuning q relative to the problem instance.

In order to estimate the order of magnitude of SCV for initial feasible solutions, a faster, unguided variant of the MMAS with no pheromone update, but enhanced with local search operators, is run. Taking the mean, μ_{SCV}, of the first 5 feasible solutions found, $q := 2.5 \times \rho \times \mu_{SCV}$. This ensures that feasible solutions exert a stronger influence on the update procedure compared to infeasible ones.

The update procedure is completed in accordance with (3.6). Here, upper and lower bounds are imposed on the values assigned in (3.5).

$$h_i := \begin{cases} \tau_{max}, & \text{if } h_i > \tau_{max} \\ \tau_{min}, & \text{if } h_i < \tau_{min} \\ h_i, & \text{otherwise.} \end{cases} \tag{3.6}$$

All pheromone values are initialised to the value of τ_{max} at the start of the MMAS to promote exploration (as per line 5 of Algorithm 4).

3.2.1.4 Dynamic hard constraint handling

For each course, periods linked to potential violations of hard constraint **H4** are known *a priori* from the problem inputs. **H4** is therefore handled in a static fashion by setting pheromone values to zero for all relevant nodes. Hard constraints **H2**, **H3** and **H5**, however, have dependencies on previous lecture assignments, which require a dynamic constraint handling approach during trail construction. Below, an assessment is made of two of the methods discussed in Section 3.1.1.

Full relaxation In the first case, the dynamic handling of hard constraints was fully relaxed in order to guarantee complete assignment of lectures. Moves to nodes invoking hard constraint violations were permitted, and the completed solution was penalised heavily according to the regime outlined. Initial experiments indicated that, unless q was re-tuned to the magnitude of the greatest possible SCV values, this search tended to stagnate in the infeasible solution space. Setting a high value for q was undesirable though as q (and hence $\frac{q}{SCV}$) then became increasingly disproportionate and unbalanced as the search moved into more feasible regions.

Zero-tolerance In the second case, at each trail step at which an ant assigned a lecture l_b from course C_i to a period p_a and room r_c, we considered the set of nodes encoding p_a for all lectures $l_{b+1} \dots l_{|\mathcal{L}|}$ yet to be assigned, whose pheromone value was nonzero. Of this set, the following subsets were reassigned to zero: In respect of **H2**: All with room index c. For **H3**: All whose course belonged to one or more curricula of which C_i was also a member. For **H5**: All whose course shared the same teacher as C_i. This rigid prescription on ant moves guaranteed no violations of the hard constraints listed above. This offered performance benefits by restricting the search entirely to the domain of feasible solutions — provided that all lectures could be assigned, i.e. **H1** was not violated. In certain problems that are highly constrained by unavailable periods, it was too often the case that virtual ants encountered 'dead ends', where every potential move in the next lecture plane had zero probability. Zero-tolerance was therefore considered overly suppressive as a stand-alone strategy.

A workable compromise between these two extremes was therefore proposed. The deployment of *dynamic artificial penalties* guarantees complete lecture assignment while severely discouraging illegal moves. As in the case of zero-tolerance above, the set of newly prohibited nodes is collected at each trail step. Rather than using zero as the re-assignment value though, a novel, positive value, τ_{shelf}, is defined between zero and τ_{min}:

$$\tau_{shelf} = \tau_{min}/(|\mathcal{P}| \times |\mathcal{R}| - 1). \tag{3.7}$$

The proposed formula fixes a worst-case bound on the cumulative probability of making an illegal move such that it never exceeds that of making a legal move. Virtual ants can now distinguish between nodes that are illegal and nodes that are either under performing or as yet unexplored. To assess the efficacy of this idea, a comparison was made with the fully relaxed approach. For both variants, 30 repetitions of 6 benchmark problems were run, using a budget of 500,000 function evaluations. The problems chosen were `comp01`, `comp04`, `comp11`, `comp17`, `comp18` and `comp19`. The fully relaxed approach failed in every case to find feasible solutions, whereas the *dynamic artificial penalty* using τ_{shelf} approach was always successful in this regard. Due to parameter q being fixed, the highest SCV values offer negligible adjustments to the pheromone matrix, leading to difficulty for the former in escaping the infeasible search space. The benefits of a successful dynamic constraint handling approach is to move the system search into the more exclusive feasible space more quickly, reducing the need for an adaptive or re-calibrated q, and therefore dynamic artificial penalties using τ_{shelf} is approved as the default constraint handling technique. With regard to the implementation, this also enables the flagging of hard constraint violations in real time, leading to significant efficiency gains as outlined in the next section.

3.2.1.5 Smart function evaluations

By understanding the context around the solutions, not every one need be evaluated in full. Partial function evaluations are known to be effective in reducing overall computation time. When a virtual ant constructs a solution, it carries an associated feasibility flag, which is passed to the function evaluator. Depending on this and other circumstances, three outcomes are possible:

1. A **full function evaluation** is executed and the SCV returned. If flagged as feasible, virtual ant solution 1 of k is always fully evaluated, as are any subsequent solutions which improve on the current best SCV value found in this iteration.

2. A **partial function evaluation** is attempted before being aborted. For all solutions subsequent to the first full evaluation, violations of individual constraints are calculated in sequence from least computationally expensive to most, with a running total kept of the cost. As soon as it becomes evident a solution cannot be the `iterBest`, the evaluation is halted and discarded.

3. **No evaluation** is computed. This occurs when a solution is carrying an infeasibility flag and at least one of the other solutions in this iteration is feasible and therefore automatically preferable.

This strategy promotes efficient use of the CPU time budget by accelerating the iteration rate.

3.2.2 Local search routine

The local search routine is domain-specific and can be enabled or disabled in accordance with the variant being studied. When activated, the routine tries to improve upon `updater` (the solution chosen to update the pheromone values with), at each iteration where `updater` is feasible. Local search includes three of the perturbation operators described in chapter 5.3 of Müller, 2009: `timeMove`, `roomMove` and `lectureMove`. In brief, these operators select a random lecture and attempt to move it to either a new period, room or some combination of the two respectively. This set is augmented with a swap operator `periodRoomSwap` which attempts to swap the room and period of two randomly chosen lectures. The local search routine is parameterised by `attemptsLS` and `iterLS`. It begins by choosing one of the four operators (`timeMove`, `roomMove`, `lectureMove`, `periodRoomSwap`), with equal probability. It then attempts to find a new feasible solution within the induced neighbourhood, to a maximum of `attemptsLS` attempts. In the case of the first three operators, the search proceeds sequentially following each failed attempt. For example in `timeMove`, if moving randomly chosen lecture l_i to period 17 proves to be infeasible, it is followed by an attempt to move the same lecture to period 18. Upon reaching the final period, the process cycles back to period 1. If a new feasible solution is identified within the threshold attempts, it is returned and evaluated. For computational efficiency, partial function evaluations are once again used. If the new solution improves upon the current best SCV for the local search, it becomes the starting solution for the next step. `iterLS` steps are run before re-assigning `updater` as the best new solution found.

3.2.3 Regression model `T0`

`T0` is one part of the Permutation Predictor, as shown in Figure 3.3. It incorporates a set of decision trees trained on an experimentally generated set of training data. This data is comprised of artificially small-sized problems, for which all permutations can be practically evaluated — full details about the data are given in Section 3.3.2. A decision tree approach serves a useful purpose here as this form of machine learner is transparent, human readable and easy to interpret, while also being relatively quick to train. Its simple representation, employing logical rules, offers an intuitive and cost-effective way to map to larger problem instances.

3.2.4 Genetic Algorithm

Given an unseen problem, a search can be run over its course permutation space for permutations of high fitness. Fitness here corresponds to the normalised predicted performance of that permutation under `TO`. For the proof of concept experiments, a simple greedy search was sufficient, but in the final model, the genetic algorithm is used. In the following section, full details are given on the genetic algorithm and the rest of the experimental setup. Before this, however, the proof of concept stage is briefly described, in which a more limited version of the final system was tested. Its purpose was to demonstrate and establish the viability of the idea before making refinements.

3.3 Experimental setup

The overarching aim in this chapter is to develop an MMAS to test the effect that permuting lecture order in the construction graph has on solution quality. In first proving the concept, a simplified UCTP formulation was used — one that omits both teachers and multiple curricula membership. The following Section 3.3.1 describes this proof of concept model in depth, while Section 3.3.2 provides detail on the final model. Table 3.2 summarises the key differences between the two models, as well as highlighting the key novelties in the final model.

3.3.1 Proof of concept

For this proof of concept approach, experiments were carried out using the toy problem (shown in Figure 2.2) and 13 other similarly sized problem instances that were reverse engineered by hand. By design, all had at least one known perfect solution — meaning an optimum at SCV = 0. The number of courses was fixed to 4

Table 3.2: A summary of the differences between the proof of concept model and the final model. Key novelties in the final model are indicated with asterisks (*). A column for future work is also preemptively included.

	Proof of concept	Final model	Future work
Construction graph	3-D lattice (Figure 3.4.)	Permutation-based (Figure 3.5.)	Permutation-based
Constraint handling	Unguided reset for dead ends	Partial relaxation using τ_{shelf}*	Partial relaxation using τ_{shelf}
Training data source	Manually generated	Semi-automated instance generator*	Fully automated instance generator
Training instance size	4-course	5-course	Up to 8-course
Target values	Mean SCV found within a function evaluation budget	Mean number of function evaluations taken to reach SCV = 0, or a timeout	Mean number of function evaluations taken to reach SCV = 0
Mapping	Average over quartiles	Average over quintiles*	Average over smaller divisions
Problem description	ITC2007 Track 3 without teachers and multiple curricula membership	ITC2007 Track 3	Other

as this kept the size of the permutation space manageable at $4! = 24$. The number of days and periods were similarly fixed across the set, at 5 and 20 respectively. Other factors such as minimum working days, number of students per course were independently varied between instances.

Table 3.3 displays some principal characteristics of these 14 training problem instances, which are named `PIX`, $\texttt{X} \in \{1 \ldots 14\}$. For each training instance, every permutation was tested over 30 repetitions, with the number of iterations set to 400. As well as this limit being time-effective, an iteration run longer than this would have increased the chances of the algorithm arriving at perfect solutions regardless of the permutation. As the performance measure is the best SCV, such an outcome was undesirable as it would have made it harder to discern between permutations.

Instance	Rooms (caps.)	$\|\mathcal{U}\|$	Course ID	Lectures	Curriculum	$mwd(C_i)$	Unavailable periods
PI1	2 (32, 50)	2	1	3	1	3	0
			2	3	1	2	4
			3	5	2	4	4
			4	5	1	4	0
PI2	2 (32, 50)	2	1	3	1	3	0
			2	3	1	2	2
			3	5	2	4	4
			4	5	1	4	0
PI3	3 (25, 35, 50)	3	1	4	2	3	6
			2	5	1	3	3
			3	3	2	2	2
			4	2	3	1	1
PI4	2 (32, 50)	2	1	3	1	2	0
			2	2	1	2	2
			3	5	2	4	4
			4	4	1	3	0
PI5	2 (32, 50)	2	1	3	1	3	0
			2	3	1	2	4
			3	5	2	4	4
			4	5	1	4	0
PI6	3 (32, 50, 25)	2	1	2	1	2	4
			2	2	1	1	3
			3	3	1	3	4
			4	2	1	2	7
PI7	2 (32, 50)	2	1	3	1	3	2
			2	2	2	2	1
			3	3	1	4	0
			4	4	1	3	5
PI8	4 (50, 40, 35, 20)	2	1	4	1	2	2
			2	2	1	1	4
			3	5	2	4	1
			4	3	1	2	7
PI9	2 (32, 50)	2	1	2	2	1	2
			2	5	1	2	5
			3	3	1	3	3
			4	6	1	2	0
PI10	4 (50, 40, 35, 20)	2	1	3	1	3	11
			2	2	1	3	6
			3	3	1	2	8
			4	4	2	2	12
PI11	3 (40, 50, 20)	2	1	4	1	3	8
			2	2	1	1	3
			3	4	2	2	0
			4	5	2	3	10
PI12	3 (25, 35, 50)	3	1	2	3	1	9
			2	3	2	2	6
			3	4	2	3	0
			4	6	1	4	3
PI13	3 (25, 35, 50)	3	1	3	3	2	2
			2	5	1	3	4
			3	5	2	4	1
			4	7	2	5	6
PI14	3 (25, 35, 50)	3	1	3	3	1	8
			2	4	2	3	6
			3	5	2	2	2
			4	6	1	4	0

Table 3.3: The proof of concept training set of 14 problem instances, where $|\mathcal{U}|$ is the number of curricula and $mwd(C_i)$ is the minimum working days requirement for a course i. Characteristics shown are number of rooms (with respective capacities) and number of curricula. Then for course ID $\in \{1, \ldots, 4\}$; number of lectures, curriculum membership, minimum working days and number of unavailable periods.

3.3.1.1 Permutation testing

Figure 3.6 is an illustrative triple plot showing results for the permutation [4 3 2 1] — which is written for ease as 4321 — of `PI1`. The top row of plots show the iteration best SCVs as black points and the global best SCV as a red line. The single best and worst repetitions are shown. The gradual shift from exploration to exploitation of the feasible space is observable, as well as the variation in end result due to the stochastic nature of MMAS. The bottom plot shows convergence of the global best SCV for all 30 of the repetitions.

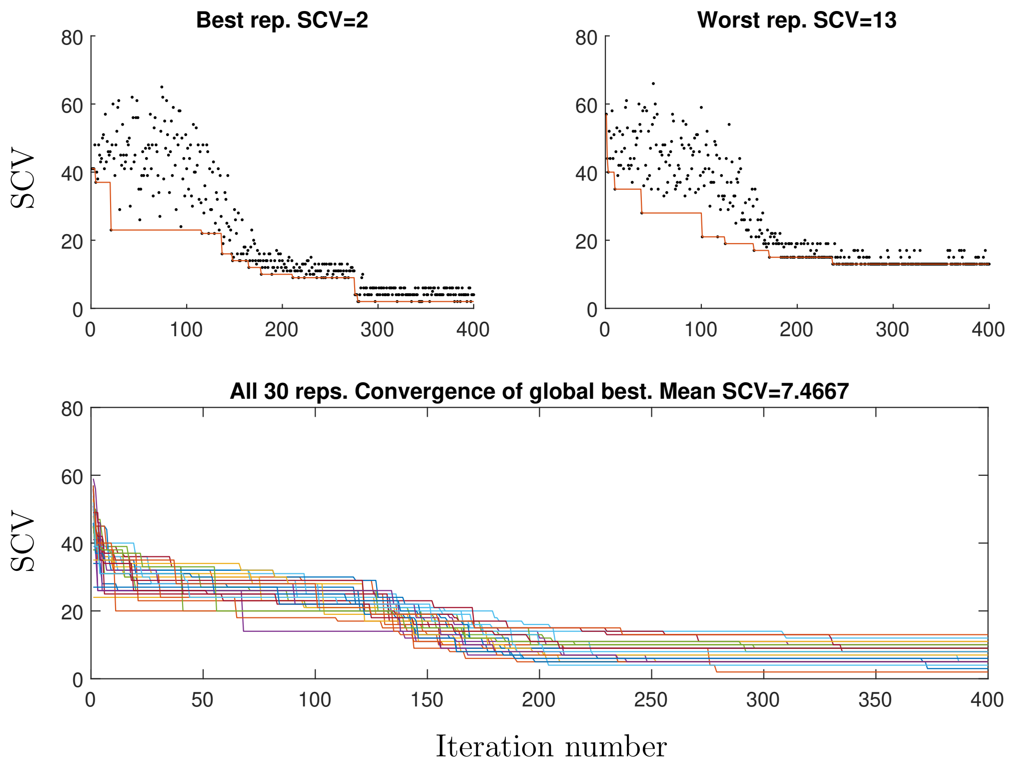


Figure 3.6: Convergence plots for various repetitions of permutation 4321, `PI1`. The top row plots show the best and worst performing repetition, respectively, while the bottom plot shows results for all repetitions. Lines trace the global best while points (only on the top plots) denote the iteration best.

The distribution of this global best SCV at termination is also displayed as a boxplot in Figure 3.7, alongside those obtained from the other 23 permutations. The first intuition from Figure 3.7 is that, all else being equal, changing the order of lectures by permuting the courses has a marked effect on solution quality. For `PI1`, the best performing permutation (by $\hat{\mu}$, the mean best SCV found over the sample repetitions) was the course ordering 2314, with $\hat{\mu} = 2.8$. The worst performing

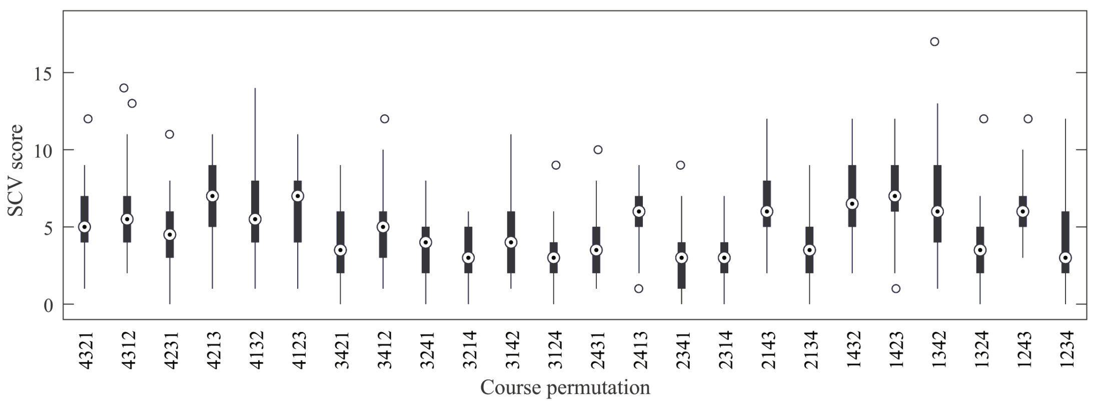
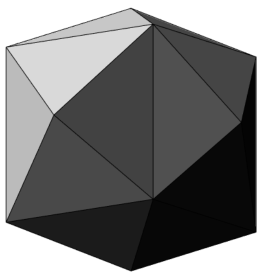


Figure 3.7: Boxplot of sample distributions (30 samples) of the best SCV found over 24 course permutations of `PI1`. The central mark indicates the median, while the top and bottom edges of the box indicate the 75th and 25th percentiles respectively. The whiskers cover all remaining data points not considered outliers, with outliers shown as circles.

was 1423 with $\hat{\mu} = 6.83$. Other instances yielded similar levels of variation across permutations as this instance.

One way to inspect the results for an instance is by sorting its permutations by $\hat{\mu}$ value. However a more revealing visualisation can be achieved by taking inspiration from the permutohedron (Schoute, 1911). Such an object of order n is an $(n-1)$—dimensional polytope in which each of its $n!$ vertices corresponds to a distinct permutation on the first n natural numbers. Two vertices are connected by an edge if and only if their corresponding permutations are neighbours according to some defined neighbourhood relation. As this proof of concept stage uses $n = 4$ courses, the corresponding object is a 3-D polyhedron. In the original permutohedron, the polytope of order 4 forms a truncated octahedron. In the given visualisation, its dual — a tetrakis hexahedron — is used, so that permutations map to faces rather than vertices. Faces share a common edge *if and only if* one of the corresponding permutations can be reached from the other using a swap operator on positionally adjacent elements. For example, the face mapping 1234 has three edges which it shares respectively with the faces mapping 2134, 1324 and 1243. For each permutation of an instance, $\hat{\mu}$ (the mean performance over the repetitions) is converted by normalisation to a greyscale value. The corresponding face of the polytope is then shaded according to this value.

Figure 3.8a shows the visualisation for PI10[2]. There is evidence of order, in that neighbouring permutations show incremental differences in average solution quality rather than sizeable disparities. For comparison and contrast, Figure 3.8b shows the disordered structure generated by a random shading. While PI10 provides the finest example of order, all instances in the set exhibit greater order than would be expected for a random shading.


(a) PI10 permutations: Normalised sample means.

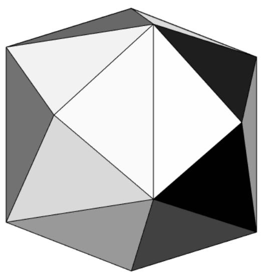

(b) $X_{face} \sim U\{0,1\}$

Figure 3.8: A visualisation based on the conceptual permutohedron. Each polyhedral face corresponds to a permutation on the first four natural numbers. Two faces share an edge if and only if their permutations can be generated from one another using a swap operator on two positionally adjacent elements. Faces are shaded according to a greyscale map where 0 = black (best performing permutation) and 1 = white (worst). Values used in (a) are the results obtained for PI10. Values used in (b) are randomly generated from a uniform distribution, for comparison purposes only.

3.3.1.2 Training a predictor

Having established the existence of order or structure in the performance of different permutations, a predictor was developed for use on unseen instances.

The features used were a combination of various course attributes (as given by the column headings in Table 3.3) and the relative positions of courses within a permutation. For this proof of concept stage, detail is omitted and 20 features are nominally f1 ... f20. However, a full explanation of how features are derived and defined in the final model is included in Section 3.3.2.1.

[2] A video animation showing a rotating polytope for PI10 can be seen at: https://youtu.be/psETpqmvwS8.

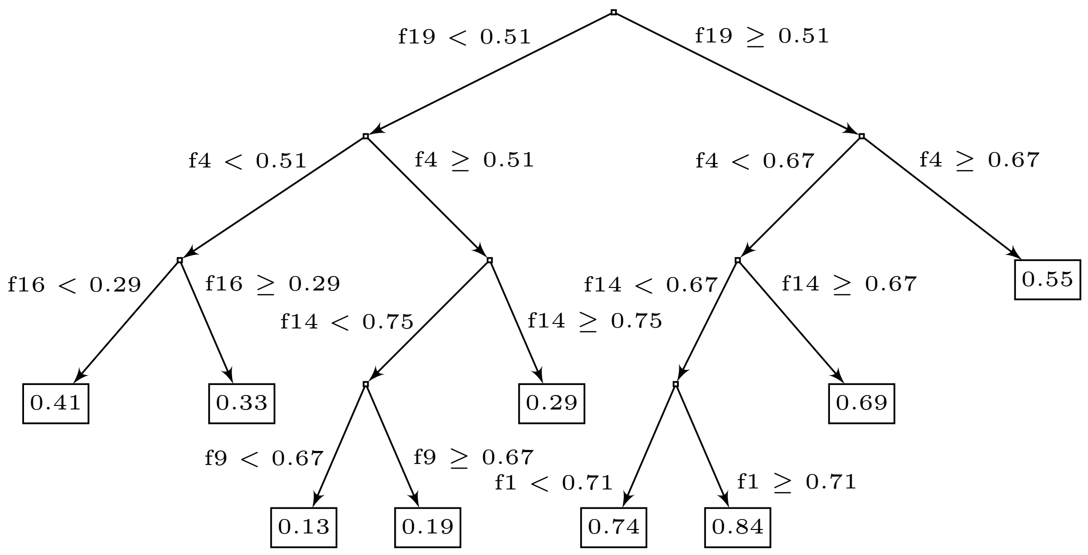
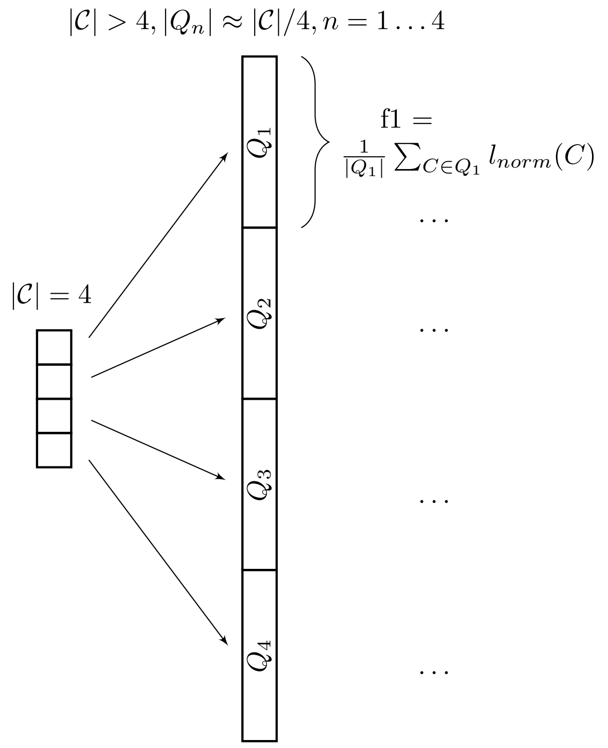


Figure 3.9: The proof of concept decision tree generated from a 20-feature (nominally f1 ... f20) training set. Prediction values at leaf nodes indicate how well a permutation characterised by the relevant branch is expected to perform relative to all others. Lower valued predictions represent better predicted performance.

The proof of concept training set consisted of the 14 instances $\times$ 24 permutations = 336 20-feature vectors. All values were subjected to min-max feature scaling. The target feature for supervised learning was the mean predicted SCV, $\hat{\mu}$, which was normalised in the same manner.

A decision tree was constructed using the standard CART node-splitting technique (Breiman et al., 1983) and 10 fold cross-validation. Minimum leaf size was used for termination and the tree was optimised over this parameter. The resultant decision tree is visualised in Figure 3.9. The R^2 value for this predictor over the proof of concept training set is 0.67 (indicating moderate rather than absolute predictive power) while the optimised minimum leaf size is 28. In avoiding overfitting, a fairly coarse tree was produced, with nine single-value predictor leaf nodes. These range in value from 0.13 to 0.84 and, as the problem is one of minimisation, paths to these leaf nodes represent high quality and low quality permutation performance respectively.

In the next step, the predictive power of the decision tree was exploited for an unseen test problem — `comp01` from the ITC2007 track 3 benchmark. This problem comprises 30 courses (a total of 160 lectures), 14 curricula and 6 rooms. It is larger in all respects than the proof of concept training set problems, and thus a form of transfer learning or scaling was needed.



$|\mathcal{C}| > 4, |Q_n| \approx |\mathcal{C}|/4, n = 1 \dots 4$
$|\mathcal{C}| = 4$
Q_1
Q_2
Q_3
Q_4
f1 = $\frac{1}{|Q_1|}\sum_{C \in Q_1} l_{norm}(C)$
$\dots$
$\dots$
$\dots$
$\dots$

Figure 3.10: The mapping from 4-course proof of concept training problem to a generalised larger problem instance. For illustrative purposes, feature f1 is the number of lectures in the first ordinal course, when $|\mathcal{C}| = 4$. When $|\mathcal{C}| > 4$ as in the larger problem, f1 is derived through averaging the lecture count over the courses in Q_1, the first quartile. For this purpose, $l_{norm}(C)$ is the number of lectures in course C, normalised over those for the entire instance.

The proposed procedure for mapping from the 4-course proof of concept training problem to the generalised larger case is illustrated in Figure 3.10. The course assignment positions of the larger problem are divided into four sets ($Q_1 \dots Q_4$) of equal integer-valued length (or approximately equal where $4 \nmid |\mathcal{C}|$), based on a composition of the course cardinality. Feature values relating to course 1 in the smaller problem map to the average value of the courses in quartile 1 of the larger problem, and likewise for Q_2, Q_3, Q_4.

A division into 4 equal integer parts is not possible for `comp01`, so the split is made according to the composition [8 7 7 8]. The proof of concept system was tested on `comp01` in the case of both the best and worst leaf node predictions, which are 0.13 and 0.84 respectively. Table 3.4 gives two permutations on 30 courses whose mapped feature values satisfy all node split conditions for the relevant branch of the decision tree, for both the high quality prediction (top) and low (bottom). In

Q_1								Q_2							Q_3							Q_4							
1	3	2	4	5	6	7	8	13	20	29	16	17	18	19	9	10	11	14	15	12	24	23	21	25	26	27	28	22	30

Q_1								Q_2							Q_3							Q_4							
23	21	22	26	27	28	30	15	25	14	24	12	9	10	11	20	29	16	17	18	19	13	5	6	7	8	2	4	3	1

Table 3.4: Two permutations on the 30 courses of `comp01`. If the mapping works as intended, the decision tree suggests that the top permutation will perform strongly and the bottom permutation poorly.

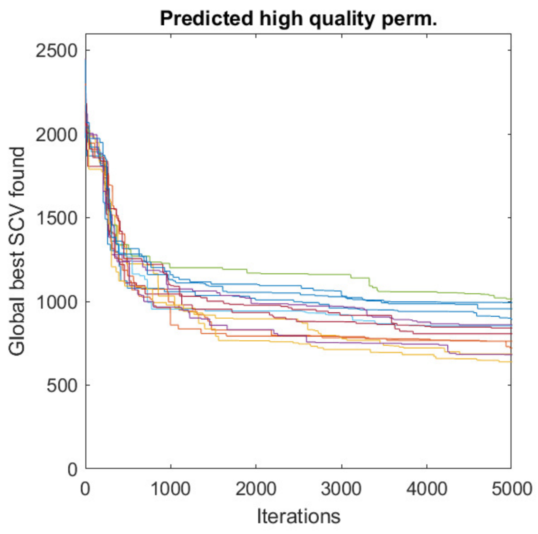


Figure 3.11: Convergence of the global best SCV for 15 repetitions, in the cases of a predicted high and low quality permutation on `comp01`.

this proof of concept version, these permutations were found using a simple greedy search.

For each permutation, 15 repetitions of the MMAS were run over 5,000 iterations. The pheromone update schedule in Table 3.1 was expanded proportionally by a factor of 10 in order to better serve the increased search space and iteration count. This meant switching to exclusive use of the global best solution for update, for example, at iteration 2,500 and not iteration 250 as for the smaller instances. Plots for the convergence of the global best SCV for all 15 repetitions are shown in Figure 3.11.

The sample mean SCV in the case of the predicted high quality permutation is 833.7, while that for the low quality prediction is 1671.3. A paired t-test rejects the null hypothesis that pairwise difference between the best SCV values found by repetitions of the first and second permutations has a mean of zero, with a p-value of 2.82×10^{-11}. Thus there is some evidence to suggest that the actual performance of permutations on a benchmark problem may correlate with their predicted performance according to the learned approach. The remainder of this section describes the experimental setup of the more refined model, starting with the building of a new set of training data.

3.3.2 Final model training data

In order to build up a new set of training instances for the final model, a problem instance generator is proposed. This generator is designed to reverse engineer small instances that have at least one perfect solution. The number of courses is increased to five, for greater granularity. This is still small enough, though, that the entire permutation space of size $5! = 120$ can be evaluated for the complete set over multiple repetitions in reasonable time. All final model training instances have 5 days per week and 4 periods per day. Their other characteristics, such as number of lectures and unavailable periods, are determined by drawing integers from appropriately sized discrete distributions. The generated problems can also be manually adjusted to increase the tightness of constraints as required. The generator can therefore be described as semi-automated. To distinguish the final model training set from the proof of concept training set, the new instances are named $\texttt{PIXN}, \texttt{X} \in \mathbb{Z}$. The precise steps of the generator are as follows:

1. The number of lectures in each of the five courses are set by taking independent samples from a Gaussian distribution $\mathcal{N}(4, 1.3^2)$, truncated at 0.5 and 7.5, and rounding to the nearest integer.

2. The minimum working days value for each course is chosen by sampling uniformly from the integer set $\{1, \ldots, min(|\mathcal{D}|, |\mathcal{L}|)\}$.

3. A distribution of numbers of parent curricula over courses is chosen randomly from a prescribed list. The required number of curricula are then randomly sampled in the case of each course.

4. A sample from another prescribed list is used to assign teachers. Any configuration of assignments involving 3, 4 or 5 teachers is possible.

5. Student numbers for each curriculum are determined by independent samples from the discrete uniform distribution on {1,60}.

6. The number of rooms is sampled from the discrete uniform distribution on {2,5}. Similar distributions on $\{min(stud(C_i)),80\}$ and $\{max(stud(C_i)),80\}$ are used to sample the corresponding capacities for $(|\mathcal{R}| - 1)$ rooms and one potentially larger room respectively.

7. The distribution of unavailable periods over courses is found by sampling the set of integers from 0 to 7, with an incremental bias towards lower values. Periods are then uniformly sampled the relevant number of times without replacement.

8. The core MMAS is used to solve the problem.

9. Problems are discarded as trivial if a perfect solution is found too quickly. They are also discarded if no perfect solution can be found.

10. Steps 1. to 9. are repeated to build a set of problems.

11. Constraints are manually adjusted or tightened across the set in order to promote diversity, without invalidating the known perfect solutions in any case.

Table 3.5: Characteristics of the final model training set of small problem instances. Columns from left to right show name, number of lectures, curricula, teachers, rooms and unavailable periods (for each of courses 1 to 5).

Name	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{R}\rvert$	$unav(C_i), i = 1\ldots5$
PI41N	23	3	4	3	{8,5,2,0,6}
PI58N	19	3	4	3	{3,0,8,4,1}
PI60N	23	3	4	2	{0,3,2,5,4}
PI62N	23	3	4	4	{5,2,1,4,10}
PI72N	18	5	3	3	{5,1,2,4,0}
PI86N	19	4	4	2	{0,1,4,4,1}
PI93N	24	3	3	3	{0,3,8,5,1}
PI94N	20	3	3	2	{0,8,3,4,1}
PI152N	23	5	3	3	{5,0,4,1,8}
PI212N	23	3	5	2	{2,0,4,0,0}
PI214N	25	4	4	5	{1,0,0,3,4}

Table 3.6: A list of eight feature types, defined for a course C_i. The properties *mwd*(), *stud*(), *teach*() and *unav*() are the minimum working days, number of enrolled students, teacher ID and unavailable periods for a given course, while *cap*() designates the capacity of a given room.

Feature type	Human intuition	Definition (prior to normalisation)
`lecs`	Number of lectures	$\lvert C_i \rvert$
`cur`	Number of curricula	$\lvert\{X \in \mathcal{U} : C_i \in X\}\rvert$
`mwd`	Minimum working days	$mwd(C_i)$
`sturms`	Over-enrolment	$\frac{1}{\lvert\mathcal{R}\rvert} \sum_{r \in \mathcal{R}} \max(stud(C_i) - cap(r), 0)$
`unav`	Uavailable periods	$\lvert unav(C_i)\rvert$
`curConf`	Curriculum conflicts	$\sum_{j \neq i} \lvert\{X \in \mathcal{U} : C_i \in X\} \cap \{Y \in \mathcal{U} : C_j \in Y\}\rvert$
`unavConf`	Unavailablity conflicts	$\sum_{j \neq i} \lvert unav(C_i) \cap unav(C_j)\rvert$
`teachConf`	Teacher conflicts	$\sum_{j \neq i} \lvert teach(C_i) \cap teach(C_j)\rvert$

3.3.2.1 Feature types and composite features

Eight feature types[3] defined in Table 3.6 are taken either directly from the problem descriptions or derived from inexpensive summations. Min-max feature scaling is applied to these formulae in order to normalise the values between 0 and 1.

Feature types `lecs`, `cur`, `mwd` and `unav` record the relative number of lectures, parent curricula, minimum working days required and unavailability constraints respectively. These values are inherent to each course. Inter-course relationships are captured by the feature types `curConf`, `unavConf` and `teachConf`. These record the conflict density, or intersection size, between parent curricula, unavailable constraints and teacher allocations respectively. Finally, the value $stud(C_i)$ alone is uninformative as a feature type without context of the room capacities, and so a compound measure `sturms` is introduced. The `sturms` feature type is a metric relating student numbers to room size. A `sturms` value of 0 implies that a course can be allocated to any room without incurring an **S1** penalty. A higher value indicates that a course is over-enroled relative to at least one room.

When permuting courses, we are interested not only in attributes common to courses, but the relative positions of courses within the permutation. To take account of this, the eight feature types in Table 3.6 are composited with position indices $1 \ldots 5$ and the resultant vectors concatenated. This yields a single feature vector of length $8 \times 5 = 40$ for each permutation. By way of a naming convention,

[3] The term 'feature type' is used as distinct from 'feature' — the former is a raw attribute of a course whereas the latter also incorporates the position of that course within a permutation.

`3lecs` is the composite feature whose value is the `lecs` value for the course assigned in position 3 of 5, for example. The final model training data set therefore consists of 11 instances $\times$ 120 permutations = 1320 40-feature vectors.

3.3.2.2 Target values

The target for supervised learning is the sample mean (over 15 repetitions) of the number of full function evaluations taken to find a perfect solution. If no SCV = 0 has been found within a certain timeout, the number of full function evaluations completed to that point is taken instead. The target values are normalised as before. A complete list of other parameter settings is provided in Section 3.3.4.

3.3.2.3 Mapping

In order to make predictions about larger unseen problems where $|\mathcal{C}| > 5$, the following mapping, from unseen problem permutation space to its equivalent in `T0`, is proposed:

$$\left(\frac{1}{|\mathcal{Q}_i|} \sum_{C \in \mathcal{Q}_i} \texttt{feat} \right) \rightarrow \texttt{ifeat} \tag{3.8}$$

for $i = 1 \ldots 5$, where $\mathcal{Q}_i$ is the set of courses comprising the i^{th} quintile of a permutation, by position, `feat` is the normalised value of a generic feature type, and `ifeat` is the subsequent composite positional feature, for example `3lecs`. By way of this averaging, a standard 40-feature vector is obtained irrespective of the value of $|\mathcal{C}|$. Every such vector has an associated fitness — namely the normalised prediction value returned by the regression model `T0`.

3.3.3 Problem Instances

For the benchmark experiments, 14 problems are chosen from the ITC2007 track 3 set, to encompass a range of size and complexity. The problems are divided into three groups: Small, medium and large, based on the number of courses. The groups and their characteristics are shown in Table 3.7.

Table 3.7: Characteristics of 14 of the ITC2007 track 3 problem instances, ordered by number of courses, and headed using the notation introduced in Figure 3.5. From left to right: Group (Small, Medium or Large), number of courses, lectures, curricula, teachers, days per week, timeslots (or periods per day), rooms and unavailable periods.

Name	Grp.	$\|\mathcal{C}\|$	$\|\mathcal{L}\|$	$\|\mathcal{U}\|$	$\|\mathcal{T}\|$	$\|\mathcal{D}\|$	$\|t\|$	$\|\mathcal{R}\|$	$\|\mathcal{N}\|$
`comp01`	S	30	160	14	24	5	6	6	53
`comp11`	S	30	162	13	24	5	9	5	94
`comp18`	S	47	138	52	47	6	6	9	594
`comp05`	S	54	152	139	47	6	6	9	771
`comp03`	M	72	251	68	61	5	5	16	382
`comp19`	M	74	277	66	66	5	5	16	475
`comp09`	M	76	279	75	68	5	5	18	405
`comp04`	M	79	286	57	70	5	5	18	396
`comp12`	M	88	218	150	74	6	6	11	1368
`comp17`	M	99	339	70	80	5	5	17	548
`comp06`	L	108	361	70	87	5	5	18	632
`comp10`	L	115	370	67	88	5	5	18	694
`comp20`	L	121	390	78	95	5	5	19	691
`comp07`	L	131	434	77	99	5	5	20	667

3.3.4 Parameterisation

Table 3.8 lists all parameter settings used for the various components in this study.

3.3.5 CPU budgets

The CPU timeout for finding a perfect solution during the final model training process is 150 seconds. For the benchmark experiments, the budget is set at 2,000, 3,000 and 4,000 seconds for the small, medium and large problems respectively. The proposed system is implemented using Matlab (version 2021a) and run on an AMD Ryzen 9 5900X 12-core CPU @ 3.20GHz. The code incorporates parallelisation such that each repetition is allocated to a single core.

3.4 Results and analysis

In this section, results are first explored from the (final model) training of the Permutation Predictor, upon which the two subsequent benchmark experiments rely. Further results and analysis are then provided for these benchmark experiments.

MAX-MIN ant system		
Phase	**1**	**2**
k (number of 'ants')	8	
ρ	0.035	
τ_{min}	0.004	
τ_{max}	10	
Repetitions	15	30
`iterLS`	n/a	10
`attemptsLS`	n/a	12

Genetic Algorithm	
Population	60
Encoding	Permutation
Initalisation	Random
Selection	Truncation (best 50%)
Crossover	Partially Matched
Crossover rate	1
Mutation	Swap
Mutation rate	$1/(2\|\mathcal{L}\|)$
Fitness	Normalised predicted value returned by `T0`
Termination	750 generations

Regression model `T0`	
Node-splitting	CART (Breiman et al., 1983)
Loss function	Mean squared error (0.0373)
Cross-validation	5 folds
Termination	Minimum leaf size (15)

Table 3.8: Parameter and other settings used in this study. The two phases of the MMAS refer to the final model training, and benchmark experiment phases respectively.

3.4.1 Final model training set permutation results

Figure 3.12 displays the 40 individual composite features (arranged in a grid of their constituent type/position) and their estimated importance in `T0`. Where peaks of importance occur, these tend to be where the position of the feature type in permutation is either 1 or 5, while central positions 2, 3 and 4 exhibit lower importance. Feature types `curConf` and `unav` are most illustrative of this profile, but aspects of the trend can be seen across all feature types. This breakdown of results suggests that the choice of course assignment is crucial in both the early and late nodes of the construction graph as virtual ants pass through it, but less so in the middle.

In Figures 3.13a and 3.13b, the data shown in Figure 3.12 is averaged out along each of its lateral axes. The feature with the lowest estimated importance (zero) is `4teachConf` and, Figure 3.13b makes explicit, features of type `teachConf` are the second least important for splitting across all positions. As the number of final model training set courses is fixed at 5, it is noted that there is no integer partition of 5 that contains more than two unique parts. This is equivalent to the

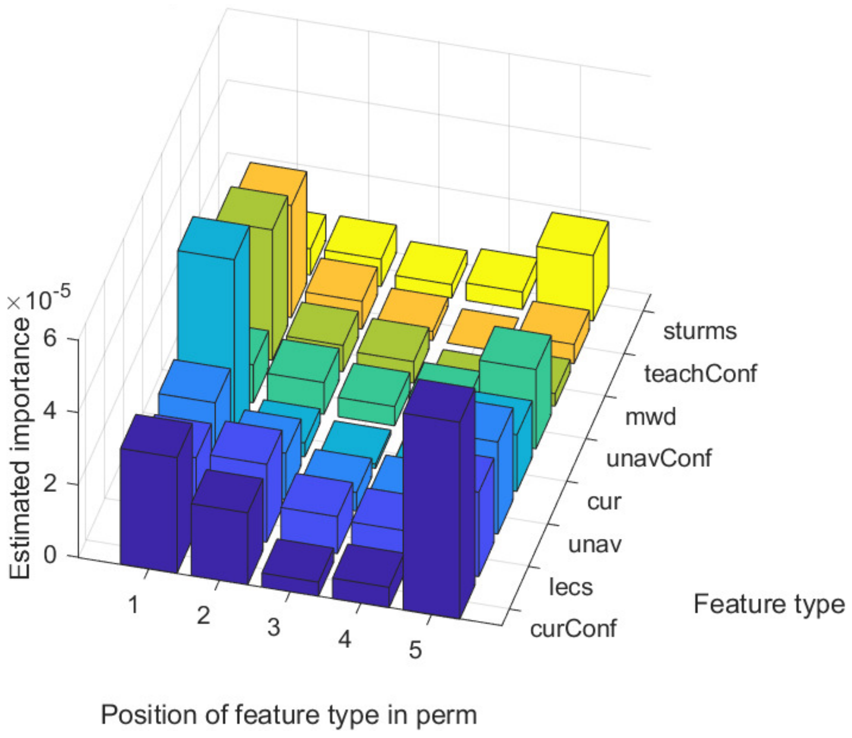


Figure 3.12: A 3-D bar chart showing the estimated importance of all 40 features used in `T0`. The features are arranged in a grid of feature type (ordered by mean estimated importance) vs. position of feature type in permutation. The estimated importance of a feature is calculated by summing the change in mean squared error at each relevant splitting node and dividing by the number of splitting nodes. The final value plotted is an average over all folds.

number of discrete values the normalised `teachConf` feature type can take, and helps to explain the low informational value of features of this type for the training set.

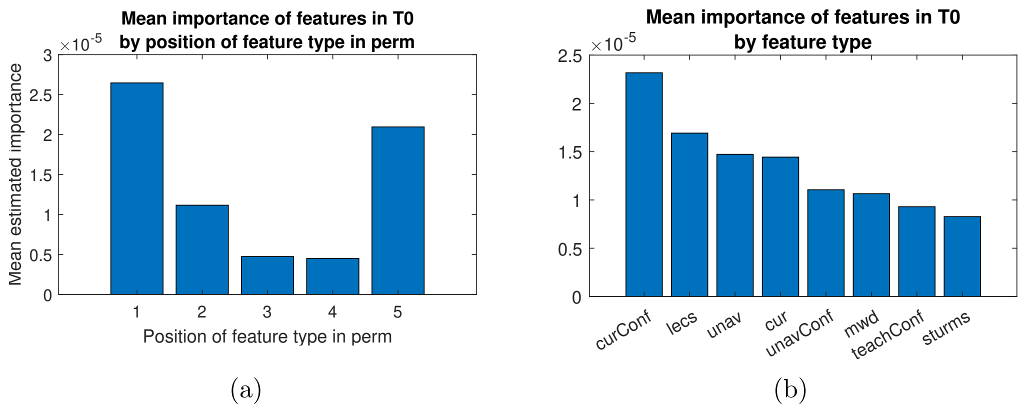


Figure 3.13: Averaged values for the estimated importance of the features in `T0`, by position of feature type in permutation (a) and by feature type (b). Initial calculations are made in the same manner as for Figure 3.12.

The elevated importance of the first and last course assignment is clearly seen in Figure 3.13a. Indeed, analysis of the splitting nodes from the top three levels of the decision trees show that a large majority (83%) split on features from positions 1 or 5 in the permutation.

Purely where the feature type is concerned, Figure 3.13b shows that `curConf` is estimated to be the most important overall. `curConf` feature types appear in 31% of the nodes using for splitting in the top three levels of the trees, compared to 20% for the next most popular feature type (`lecs`). It is interesting to note that a (differently defined) measure of curriculum conflicts was found in Rosa-Rivera et al., 2021 to be one of the four most important features (of 149 proposed) for determining the 'complexity' of a timetabling instance. The closest equivalent metric to `lecs` was likewise in the top four in that study. This strengthens the findings in this section, as increased complexity is associated with increased difficulty in finding good solutions. As discussed, the value ranges for some estimated low-information features are naturally restricted by the modest size of the final model training instances.

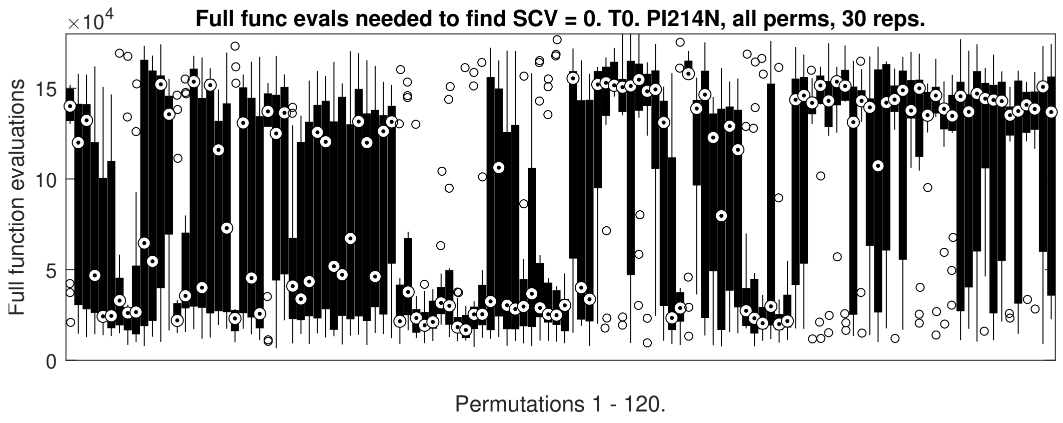


Figure 3.14: Distributions (population size = 30) of full function evaluations required to find a perfect solution for all 5! = 120 permutations (ordered 54321 to 12345) of final model training instance `PI214N` in `T0`. The boxplot shows median, interquartile range, non-outlier max/min and outliers.

An illustrative boxplot, Figure 3.14, shows the distributions of full function evaluations obtained for one of the final model training instances ('`PI214N`'). The permutations (nominally 1 to 120) are ordered from 54321 to 12345, left to right on the x-axis. It can clearly be observed that certain clusters of permutations converge to a perfect solution more efficiently than others. In the example given, index permutations 41-51 are some of the best for locating a perfect solution quickly. Eight of the permutations in this block assign course 4 first while assigning course 3 in position 3 or later. At the other end of the spectrum, a cluster of poor results is evident in index permutations 65-72. Notably, these permutations all assign course 3 first, while course 4 comes later. However, three permutations in the high performing block also assign course 3 first, emphasising that the roles of both absolute and relative course positions are complex. While solving the permutation problem is not synonymous with solving the UCTP itself, this plot demonstrates how a well-informed choice of permutation can enable the MMAS to locate the optimal solution — even without local search enhancement — where it may have been unable or unlikely to otherwise[4].

Beyond the example case `PI214N`, such variation in permutation-related performance was noted across all training problems, establishing in the broadest sense that course lecture ordering has a marked effect on performance. Further analysis of the remainder of final model training instances is possible by considering, as before, a form of permutohedron — this time (5 - 1)-dimensional — in which the median performance of permutations are associated with distinct vertices. If one permutation can be reached from another by swapping a pair of courses in adjacent positions, then the two are neighbours and share an edge. The permutation 12345 is a neighbour of 21345, for example. This adjacency concept offers an insight into how similarly to one another neighbouring permutations perform. A distance metric, ψ, is defined as the number of adjacent swaps required to transform one permutation to another — equivalent to the shortest path between vertices in the permutohedron. The mean absolute difference in median performance between permutations at distance ψ is denoted by δ. In Figure 3.15, these two variables are considered over all permutation pairs for nine of the final model training instances.

An important observation is that all of the plots exhibit positive correlation, with some showing monotonic increase. Not only therefore does the course ordering affect results in the core MMAS, but there is an inherent structure in which similar permutations can be expected on average (while noting a generally wide variance) to deliver similar performances. While this association broadly holds true, it is not

[4] This is the case for the training problems. No optimal solutions were reached by any means for the benchmark problems in this chapter.

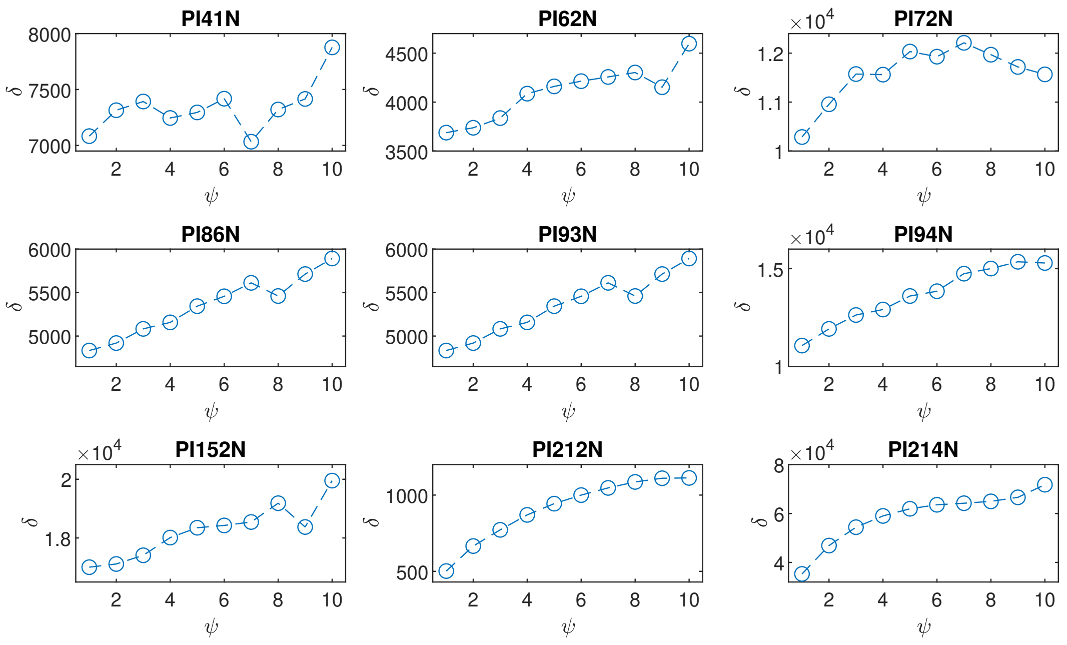


Figure 3.15: Plots of the similarity of course permutation vs. the difference in their median performance, for all pairs of permutations in nine of the final model training set instances. ψ is the pairwise distance between permutations in the permutohedron, where distance is defined as the shortest path between the two based on making adjacent course swaps. δ is the mean absolute difference in median performance (where performance is measured by SCV) between all pairs of permutations at distance ψ.

universal however. Exceptions can be seen in plots for `PI41N` and `PI72N`, while the non-monotonic plots for `PI62N`, `PI86N`, `PI93N` and `PI152N` exhibit a characteristic dip in δ value at $\psi = 8$ or 9. While overall trends are strong, the permutation space and performance space do not conform to a single consistent mapping across all training problems. The following section presents results from the genetic algorithm component of the Permutation Predictor.

3.4.2 Genetic Algorithm results

Figure 3.16 illustrates the evolution of population fitness for a genetic algorithm run on benchmark problem `comp17`, framed as a maximisation problem. As fitness is a normalised value, it can never exceed 1. In practise, the maximum fitness values — dictated by the decision tree — are slightly lower. In this example run, a median permutation fitness of 0.81 is reached within 10 generations and not improved upon thereafter. The speed of convergence shown in Figure 3.16 is typical across the set of all problem instances. While the size of the permutation decision space is large for

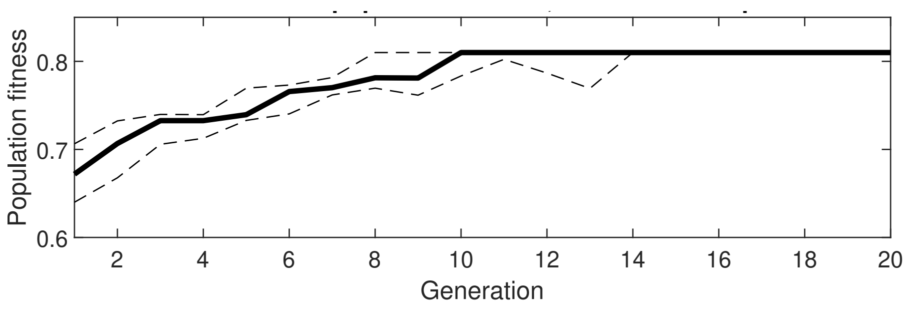


Figure 3.16: Fitness (where fitness corresponds to predicted performance in the MMAS) of a population of 60 individual permutations vs. generation number (truncated at 20) for an example genetic algorithm run on `comp17`. The plot shows median and interquartile range.

the unseen problems ($|\mathcal{C}|!$ where $30 \leq |\mathcal{C}| \leq 131$), its fitness landscape is simplified by the discrete decision trees and a low granularity mapping. This ensures that regions of the fittest permutations are relatively easy and quick to locate.

3.4.3 Benchmark Experiment 1 - `PP` vs. `RP`

In benchmark experiment 1, `PP` (the Permutation Predictor) was compared against `RP` (Random Permutation). Each repetition of `PP` ordered the courses in the construction graph using a different permutation returned by the Permutation Predictor. In `RP`, the order of course assignment was determined by uniform random sampling of the permutation space, without the benefit of any prior learning. The aforementioned CPU time budget was used as a termination criterion.

Rather than returning singular results at the termination point, any-time performance graphs are provided over the entire CPU time budget. The progress of the two variants has been plotted together in different colours (black — `RP`, green — `PP`). Across the 14 ITC2007 problems tested, distinct patterns in these traces were observed, and some representative plots are presented here for discussion. In assessing the efficacy of `PP` on the core MMAS, analysis is focused firstly on the characteristic shapes of the convergence plots. Robust statistical testing is then applied. The Wilcoxon rank sum test is used to determine statistical significance between the performances of `PP` and `RP` at each discrete time step.

3.4.3.1 Any-time performance comparison

Figure 3.17 shows the convergence traces for `RP` and `PP` for medium problem `comp12`.

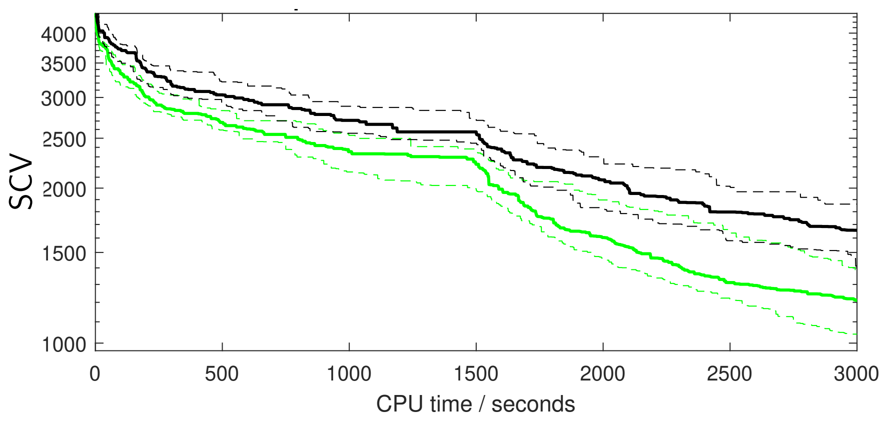


Figure 3.17: `comp12`: Progress of the baseline `RP` (black) vs. the Permutation Predictor `PP` (green), with median and interquartile range shown.

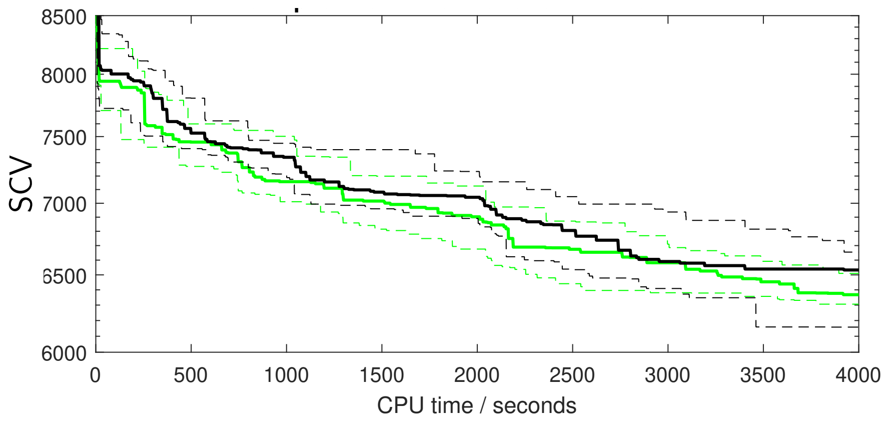


Figure 3.18: `comp20`: Progress of the baseline `RP` (black) vs. the Permutation Predictor `PP` (green), with median and interquartile range shown.

It can be seen that `PP` achieves better median results than the `RP` baseline for the entirety of the time budget. Another notable aspect in this plot is the 'knee' in the curves at 1,500 seconds. This is caused by the rigid update schedule, which switches to exclusive use of `globBest` at 50% of the run. This instance may benefit from an increased intensification earlier in the run — something that could be achieved by a more dynamic update schedule.

Example results for a large problem are given in Figure 3.18, which shows `comp20`. Here, the median performance of `PP` is again superior for the majority of the run, although the relative difference between traces is smaller and `PP` is overtaken at points.

In the given plots, as well as the majority of the small problems, `PP` is successful in beating the any-time performance of the baseline. However, in a minority of problems the converse is true — Figure 3.19 shows an example in `comp04`.

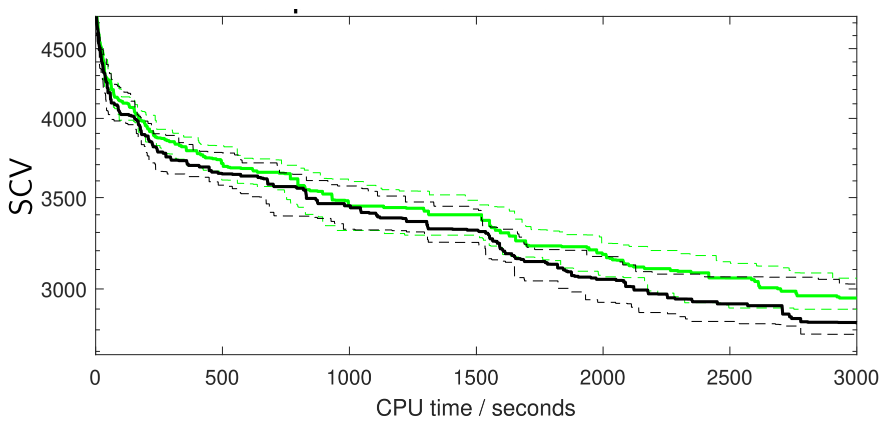


Figure 3.19: `comp04`: Progress of the baseline `RP` (black) vs. the Permutation Predictor `PP` (green), with median and interquartile range shown.

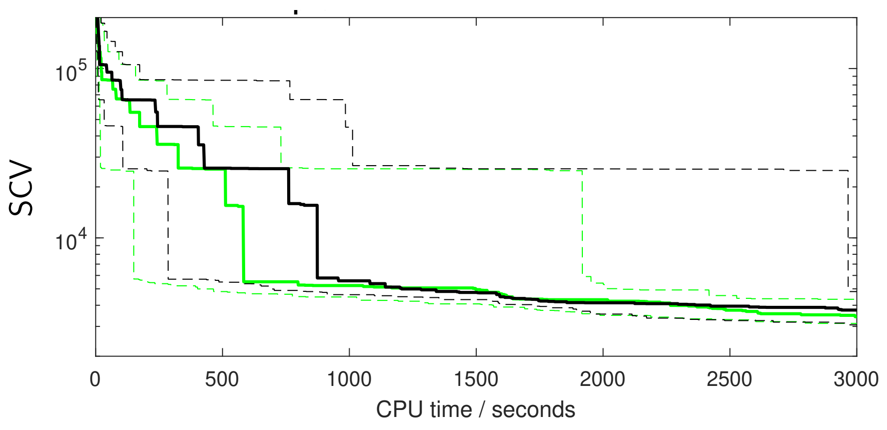


Figure 3.20: `comp19`: Progress of the baseline `RP` (black) vs. the Permutation Predictor `PP` (green), with median and interquartile range shown.

The final plot provided is Figure 3.20, which shows results for medium-sized problem `comp19`. This problem is an anomaly in the sense that its underlying structure due to the interaction of its constraints proved particularly difficult for the MMAS to navigate — the time taken to escape the region of infeasible solutions was relatively long. This is evidenced by the high SCV values (in the range 10^4 to 10^5), large interquartile ranges and jagged contours of the median traces in the initial portion of the run. The median for `PP` is seen to touch the feasible space first, at around 600 seconds. Once within the feasible space, the two variants progress at similar rates.

3.4.3.2 Significance testing

Statistical significance testing is provided by way of a two-sided Wilcoxon rank sum test. The test was applied at every discrete time step for the duration of the result

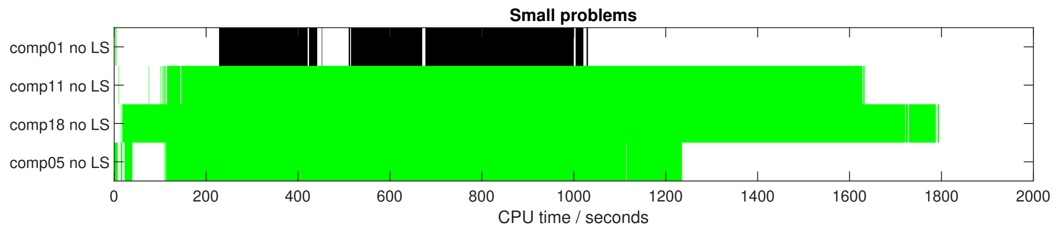


Figure 3.21: Results for Wilcoxon rank sum significance testing comparing the sample sets (size 30) of global best SCV values at each discrete time step of the run, for the small problems. Significantly superior performance ($p < 0.05$) of a particular variant is indicated in green (for PP) or black (RP). No shading indicates that no significant difference was found between the variants at that point in time.

runs for the sample sets produced by RP and PP. Using these statistics, Figures 3.21 (small problems), 3.22 (medium) and 3.23 (large) provide a visual guide to which of PP (green) and RP (black) performed significantly better during an optimisation run and over which time period(s).

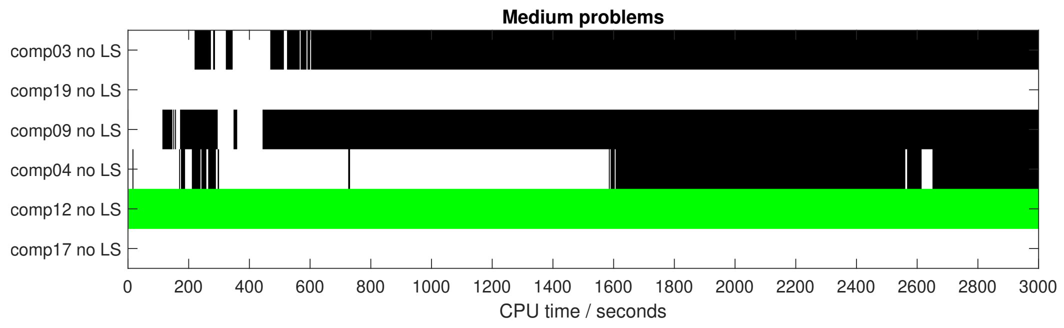


Figure 3.22: Results for significance testing for the medium problems, using the same visualisation as described in Figure 3.21.

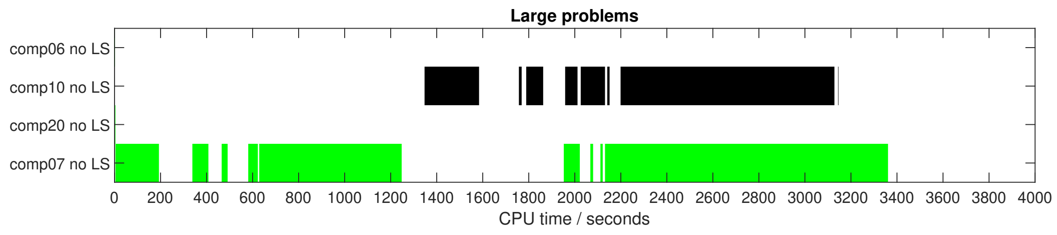


Figure 3.23: Results for significance testing for the large problems, using the same visualisation as described in Figure 3.21.

Figure 3.21 shows results for the small problems. The abundance of contiguous green shading confirms that, for 3 out of the 4 problems in this subset, the Permutation Predictor PP found permutations that significantly improved any-time performance. While the smallest instance, comp01, appears to be an exception, it should be noted that the inferior performance of PP in the initial phase becomes insignificant at the point at which globBest is exclusively used for pheromone update. Further, from around 1,600 seconds, the median value for the PP trace indeed overtakes its RP comparator. Figures 3.22 (medium problems) and 3.23 (large problems) show that as the problem size increases, the statistical test results become more mixed.

In summary of benchmark experiment 1, the Permutation Predictor approach — which has at its heart the mapping used for scaling in (3.8) — showed some promise for the core MMAS. This is particularly encouraging given the modest size of the final model training set and instances. Good performance is noted on the subset of small problems, which ranged from 6 to 10.8 times the size of the final model training set problems (by number of courses). Complex feature relationships of the medium and large problems, whose sizes were 14.4 – 26.2× greater than the final model training problems, proved harder to capture. This may indicate the importance of similarity in size or structure between training and test problems, in the sense that the diversity of relationships between variables and constraints in larger problems must be sufficiently represented within the smaller training problems. Restricting final model training size to 5-course problems was a practical necessity in enabling computation of all $5! = 120$ permutations over multiple repetitions, with the trade-off that some features — particularly teachConf — could only take a limited number of discrete values.

3.4.4 Benchmark Experiment 2 - PP-LS vs. RP-LS

The second experiment repeated the protocol from benchmark experiment 1, except with local search activated during the ACO search in all cases, enabling a comparison between PP-LS and RP-LS. The purpose of this experiment was to assess whether any performance gains (or losses) seen for the core MMAS would be flattened out by the application of the local search routine. In the same fashion as Section 3.4.3.2, Figures 3.24, 3.25 and 3.26 show the (local search-enhanced) results of significance testing for the small, medium and large problems respectively — this time with blue representing PP-LS and red representing RP-LS.

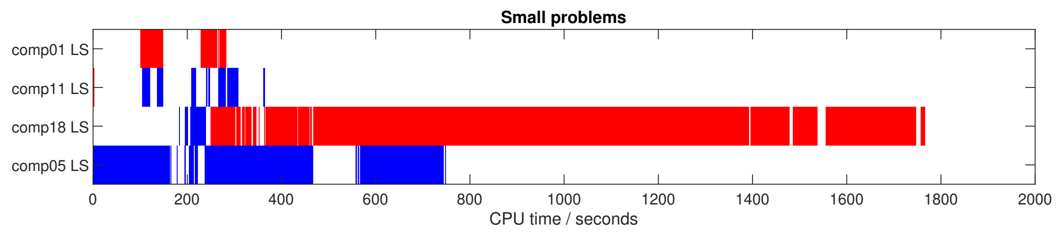


Figure 3.24: Results for Wilcoxon rank sum significance testing comparing the sample sets (size 30) of global best SCV values at each discrete time step of the run, for the small problems. Significantly superior performance ($p < 0.05$) of a particular variant is indicated in blue (for `PP-LS`) or red (`RP-LS`). No shading indicates that no significant difference was found between the variants at that point in time.

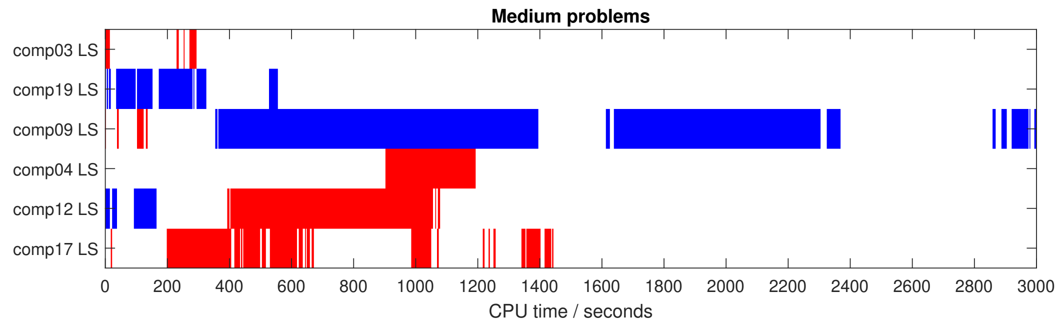


Figure 3.25: Results for significance testing for the medium problems, using the same visualisation as described in Figure 3.24.

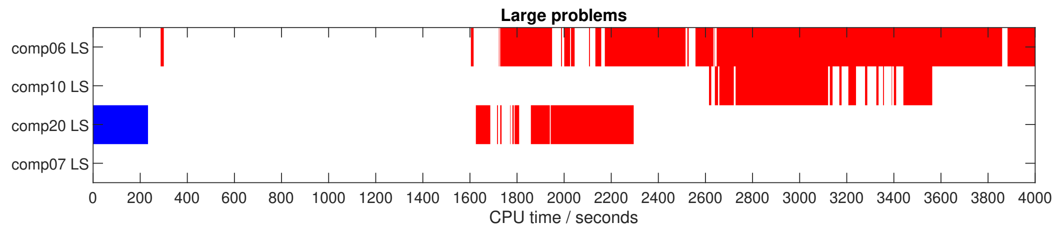
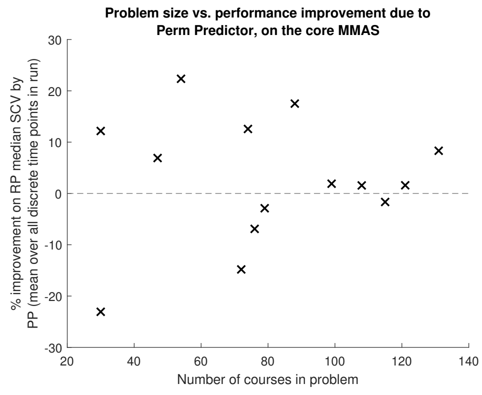
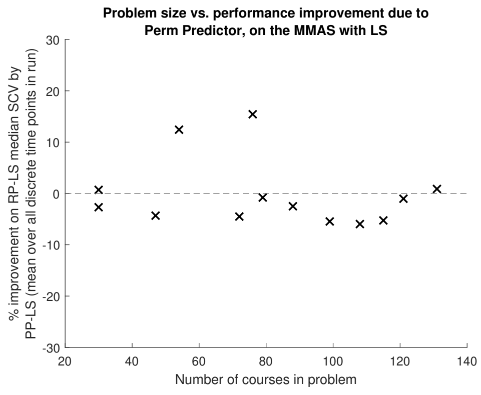


Figure 3.26: Results for significance testing for the large problems, using the same visualisation as described in Figure 3.24.

Across all problems, one key observation about the extent of significant difference between the local search variants is that they are typically less sustained than for no local search. This can be observed in the areas of no shading, which are greater. After a time, the effects of powerful local search operators tend to flatten out any differences attributed to the permutation choice using the core MMAS. There is, however, a consistent exception to this trend. In all four cases where no significant difference was found at almost any point in benchmark experiment 1 (`comp06`, `comp17`, `comp19`, `comp20`), one of the local search variants significantly out-performs the other for at least a portion of the run. This is an interesting result, as — treating the lack of significance in the non-local search variants as controlling



(a)



(b)

Figure 3.27: ITC2007 problem size (by number of courses) vs. % improvement of `PP` over `RP` in benchmark experiment 1 (a) and % improvement of `PP-LS` over `RP-LS` in benchmark experiment 2 (b). The % improvement metric is calculated on the median SCV achieved by each variant at a common discrete time step, averaged over all time steps in a run, providing a single data point for each problem in each plot.

for the effects of the Permutation Predictor on the core MMAS alone — it suggests the choice of permutation has the potential to impact the efficacy of the local search routine in some cases too.

For the small subset of four problems in which one local search variant was significantly superior for a substantial portion of time (`comps05`, `comp18`, `comp09`, `comp06`), only one (`comp05`) returned the same winner (the Permutation Predictor) as for the non-local search case. The key finding in this section is that there is little evidence to suggest that the introduction of local search preserves any performance gains, or indeed losses, attributable to the Permutation Predictor.

Figure 3.27 brings together the performance of all 14 problems, by giving a single averaged percentage improvement metric to each, for benchmark experiment 1 (Figure 3.27a) and 2 (Figure 3.27b). For benchmark experiment 1, it can be seen that in the majority (9 out of 14) of problem instances, use of the Permutation Predictor resulted in improved performance on average and in one case by as much as 22%. Further, considering only sizeable gains through the use of the Permutation Predictor of over 5% and likewise losses of over 5%, there are twice as many problems in the former category than in the latter, which is promising. It is also noted that the greatest gains are clustered at the lower end of problem size, where the resolution of the quintiles used for scaling is higher and the magnitude of the problem is closer to that of the final model training set. In the plot for benchmark experiment 2, the average distance to zero is lower across all points — highlighting the flattening effect of local search — with most being negative as the regression model was not trained on an local search-enhanced MMAS.

3.5 Conclusions and further work

The MMAS system developed was capable of finding feasible solutions to all problems investigated under the ITC2007 track 3 formulation. Smart partial function evaluations enabled efficient use of computational budget. Having considered two traditional methods of hard constraint handling in ACOs — full relaxation and zero-tolerance — neither were found to be suitable for the MMAS in this problem domain. Therefore a novel compromise was proposed. The dynamic deployment of artificial penalties, reducing the appropriate pheromone values to τ_{shelf}, guaranteed complete solutions while helping guide the search towards feasibility.

Using this core MMAS system (i.e. without local search enhancement) and a small training set, it was established that permuting the order of course assignments can influence the quality of solutions and the cost of finding them. Furthermore, across the majority of these 5-course problems, there was a strong positive correlation between similarity of permutation and average similarity of performance — where performance was measured as the number of full function evaluations needed to find a perfect solution.

Human-interpretable features were used in a machine learning pipeline. Out of these, `curConf` (a measure of how much a course's curriculum membership conflicted with that of other courses) was estimated to be the most important based on the final model training data. 'Number of lectures' and 'number of unavailable periods' also ranked highly. Furthermore, feature types relating to assignments in the first or last position of a permutation consistently ranked as more important than those in the middle, suggesting that the choice of course assignment is more critical towards the beginning and end of the ACO construction graph.

The major novelty was in using this learnt information (rather than deterministic/stochastic heuristics) for permuting course lectures in unseen problems. In attempting to scale from the final model training set to ITC2007 benchmark problems, the permutations predicted by the machine learner and mapping generally led to improved results (compared to a random permutation baseline) in the small problems ($|\mathcal{C}| < 55$). Results were more mixed as the problem size increased and feature relationships became harder to capture and transfer. This is reflected in the analysis in Section 3.4.3.2. In essence, it is difficult to represent the complexities of the largest benchmark (or, by extension, real world) problems in a training set of such limited proportions.

When a local search routine was integrated with the core MMAS, two key observations were made. First, performance gains due to a good choice of permutation in the core MMAS did not necessarily hold once local search was applied. Second, for some problems, the choice of permutation appeared to induce significant any-time performance differences between the two local search variants where there was none between the two non-local search variants. This suggests that, while in many cases local search flattened out the differences due to permutation choice, in at least 4 of the 14 problems tested, the effectiveness of local search itself was influenced by permutation choice.

In terms of the Permutation Predictor, further work will involve enlarging the set of training problems. With the benefit of greater computing power and parallelisation, the number of courses in these problems could also be increased. This would further increase the granularity for some feature values. The mapping given by expression 3.8, that showed some promise, nonetheless dilutes information by way of averaging over each quintile. Therefore a more sophisticated and expressive mapping could be sought. As an extension, non-human interpretable features could be extracted and studied. With regard to the local search findings, further research is also needed to understand the effect of the permutation choice on local search performance as it relates to different structures of problem. In addition, the ratio of local search routine calls to iteration (fixed at 1:1 here) could also be varied.

No evidence has been provided as to how generally applicable the Permutation Predictor may be to variants of ACO other than the MMAS developed in this work. The premise of assigning ‘difficult’ lectures first is nonetheless well established. On this basis, a permutation that is shown to deliver good results in the given MMAS may be expected to confer similar advantages when applied within other ACO variants.

In this chapter, the UCTP was evaluated as a single-objective problem and solved by way of an ACO. In real world applications, however, a timetable decision maker may wish to prioritise the satisfaction of one type of soft constraint over another. Furthermore, striving to achieve the single-objective optimum as defined (such as SCV $= 0$ in the case of `comp11`) may neglect other, more intangible, elements of a timetable which are also of interest to a decision maker. In the following chapter, the problem is therefore extended to one with many objectives, in which the SCV is decomposed instead into a vector of objective scores.

4. A Many-Objective Optimiser for the UCTP

In this chapter, a domain-specific many-objective optimiser is developed, based on constructive heuristics and a modified non-dominated sorting genetic algorithm III (NSGA-III) (Deb and Jain, 2013), in which the violations of different constraints are cast as separate objectives to be minimised concurrently. Results show that feasible solutions can be attained consistently in a first phase and that a targeted objective can be optimised to zero (where possible) in a second phase. A set of non-dominated solutions is returned, representing a well-spread approximation to the Pareto front, from which a decision maker could ultimately choose according to *a posteriori* preferences.

4.1 Introduction

The evaluation of timetables in this chapter represents a departure both from the work in the previous chapter and the majority of research on timetabling. Rather than a scalar cost, timetables are evaluated in terms of a vector of objective scores, each of which represents the cost of an individual constraint.

The term 'multi-objective optimisation' is generally reserved for optimisation problems with a modest number of objectives, whose results can be visualised using 2-D or 3-D plots (Karami and Dariane, 2022). A recent example of this is the timetabling approach of Budi Darmawan et al., 2019, which comprised two bespoke objectives, one of which was minimising the loss of revenue due to empty seats.

More recently popularised in the literature is the many-objective approach, for a higher-dimensional objective space. This chapter explores this idea by casting four (and initially more) UCTP constraints as separate objectives. The motivation is to evolve a set of mutually non-dominating solutions that approximate the Pareto front, thereby giving a decision maker a set of high-quality timetables to select from. In highly complex real world timetabling, this is likely to be the preferred approach. One reason is that, in contrast to some artificial problems, a solution that satisfies all constraints is unlikely to exist in a real world case, or to be discoverable in an acceptable time. At the same time, an insight can be gained into the inherent trade-offs and relationships between objectives. Visualising this information is valuable to an experienced user who can then make the most appropriate decision for their problem.

Formally, many-objective optimisation is expressed by the minimisation in (4.1) and mappings in (4.2):

$$\min_{x \in X}(f_1(x), f_2(x), \ldots, f_k(x)) \tag{4.1}$$

$$\begin{aligned} f : X &\mapsto (\mathbb{Z}_0^+)^k \\ x &\mapsto \begin{pmatrix} f_1(x) \\ f_2(x) \\ \ldots \\ f_k(x) \end{pmatrix} \end{aligned} \tag{4.2}$$

Where x is a design vector, X is the feasible solution set, f_i is a function that evaluates objective i, and $k \geq 4$ is the number of objectives.

The primary method of comparison between objective vectors is the Pareto dominance relation, as used (for instance) in NSGA-III. A solution $x_1 \in X$ dominates $x_2 \in X$ *if and only if* the following conditions hold:

$$\begin{aligned} &\forall i \in \{1, \ldots, k\}, f_i(x_1) \leq f_i(x_2) \\ &\exists i \in \{1, \ldots, k\}, f_i(x_1) < f_i(x_2) \end{aligned} \tag{4.3}$$

For efficiency, the implementation in this chapter incorporates δ-evaluators, as suggested in Geiger, 2012. Ultimately, phase one of the system aims to find feasible starting solutions, which are then used to initialise the genetic algorithm in phase two. Computational efficiency, simplicity of design and keeping the parameter count to a minimum are also key motivating factors throughout.

Section 4.2 provides some background work before Section 4.3 details the methodology and optimiser development. Section 4.4 describes the experimental set-up, both for the two-phase system and a preliminary one-phase system. Two-phase results are presented and discussed in Section 4.5, while Section 4.6 presents some conclusions.

4.2 Background

While results have been published for many solvers for the ITC2007 benchmark, the vast majority treat the problem as a single-objective minimisation, as suggested by the original competition rules.

It is noted in Hafsa et al., 2021 that this single-objective approach predominates in educational scheduling more generally, despite the existence of often numerous and conflicting objectives. The authors consider a 3–objective professional training scheduling problem with some similarities to the UCTP. Objectives are typically multi-variable functions of constraint violations and other statistical data. The performance of NSGA-II was compared against NSGA-III, with the former found to be superior on all metrics except speed. However, the parameter values were tuned only for NSGA-II, and we note that the ITC2007 problem has a higher-dimensional objective space which may be tackled better by NSGA-III. Other differences between the ITC2007 and the problem in Hafsa et al., 2021 must also be recognised, such as the timescale of the former (repeating week-long blocks rather than months or years), its requirement to assign all events, and its lack of precedence constraints.

A more apropos comparison may be made with Geiger, 2009, in which the many-objective nature of the UCTP and ITC2007 benchmark was considered. A trajectory search was carried out by selecting a small number of lectures and reassigning them.

Various acceptance criteria were relied upon for the new evaluations. In both of the two variants proposed, decision maker preferences were assumed *a priori* and implied by the cost function. This was defined as either the standard weighted sum of violations or the Chebyshev distance (Cantrell, 2000) to a reference point (the origin). Using the latter resulted in a more uniform spread of scores across individual objectives. However, as a trajectory search, a set of non-dominated solutions was neither sought nor returned in the author's work.

To the best of our knowledge, there are as yet no published results for the benchmark that attempt to approximate the 4-D Pareto set in the absence of decision maker preferences. The following section, on methodology, outlines the system proposed to achieve this. While NSGA-III lies at the core of this system, several modifications and augmentations were required to make it operable and competitive for the problem domain. The development and reasoning behind the different components of the system — such as heuristic initialisation, perturbation operators and δ-evaluations — are described in the methodology.

4.3 Methodology

Figure 4.1 provides a high-level sequence diagram of the final many-objective system. All pictured components are outlined and included as part of this chapter, with the exception of 'genotype diversity', which is not incorporated until Chapter 5.

4.3.1 Encoding

Initial work was carried out using the platEMO package, version 3.1 (Tian et al., 2017). This is an optimisation suite containing MATLAB implementations of over 20 single- and 100 multi/many-objective algorithms. While these algorithms are coded in their most naive form, platEMO is designed to be highly user-extendable. The work described in this chapter involved customising and enhancing standard algorithms with domain-specific knowledge. For this, a small number of platEMO modules were retained in a modified form, while most of the codebase was built from scratch.

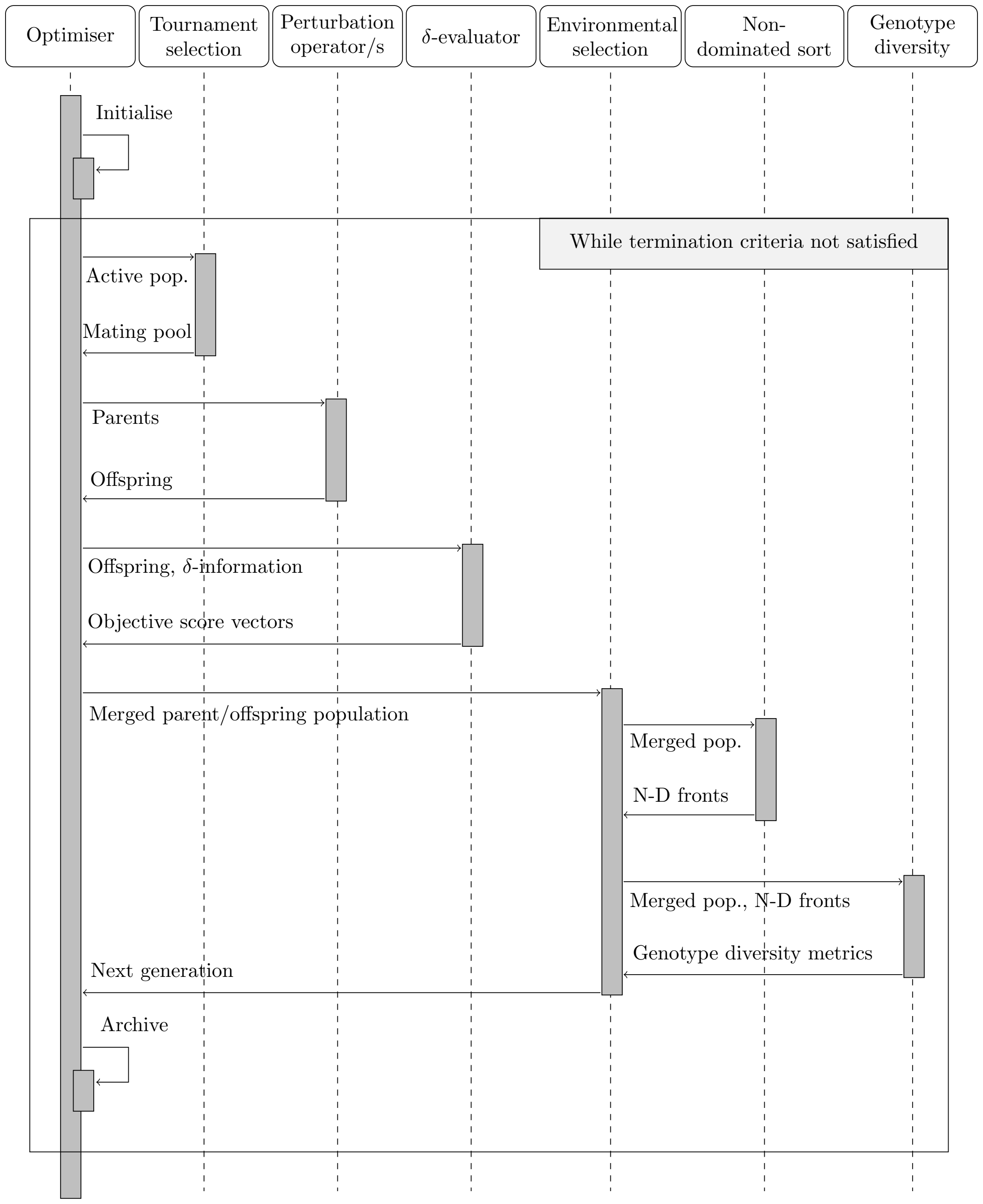


Figure 4.1: A sequence diagram outlining the essential components in the many-objective optimiser. 'Phase one' includes the initialisation, while 'phase two' refers to everything inside the main optimiser loop. Note that the genotype diversity routine does not form part of the work in this chapter, but is developed and incorporated in the following chapter.

The first task was re-encoding the problem instances, by converting each problem from its original `.ctt` file format to a 2-D indexed cell array data structure. Most of this structure is given to the indexing of course, teacher, minimum working days, number of students, curriculum membership and unavailable periods, for every lecture. The last two rows of the cell array include a matrix of the room IDs and capacities, as well as various metadata. Some elements of the problem are also stored as alternative representations (such as curriculum membership as a Boolean matrix and a cumulative lecture count per course) to aid the efficiency of certain recurrent functions.

Solutions — the timetables themselves — must also be encoded. This is a design choice with serious implications for the efficacy of any evolutionary algorithm used. A number of direct approaches were considered. The first attempt used a chromosome of length $|\mathcal{L}|$ and a integer-valued encoding such that each assignment took a value of:

$$(r_i - 1)|t| + t_i \tag{4.4}$$

Where r_i is the room ID and t_i is the timeslot, drawn from the set t. This encoding was somewhat advantageous both in terms of its low memory footprint and the time complexity of the required function evaluators. The drawback, however, lay with the lower-resolution fitness landscape induced and the lack of direct connectivity between, for example, *places* composed of the same timeslot but different rooms.

The second solution encoding therefore used a tuple as a more expressive way to represent an individual lecture assignment:

$$\langle d_i, p_i, r_i \rangle \tag{4.5}$$

Where d_i and p_i are the day and period respectively and the element-wise length of the chromosome became $3 \times |\mathcal{L}|$. Disadvantages of this encoding include the larger data structure required, increased time complexity as well as the potential for epistatic effects caused by interactions between elements within tuples. Nonetheless, the induced search landscape grants connectivity between days, periods and rooms as individual entities, allowing for the design of potentially more nuanced genetic operators. Each element within a gene resonates with a particular soft constraint. For example, perturbing d_i affects the number of unique days that course lectures are held on, and therefore the violation score of **S2**. Varying r_i has a direct effect on the two constraints relating to physical location, namely **S1** and **S4**. Finally, p_i, in conjunction with d_i, influences **S3**. Compliance with **H1** (all lectures must be assigned) is also ensured by the 1:1 lecture:gene ratio.

4.3.2 Initialisation

The initialisation constitutes phase one of a two-phase optimisation. In phase one, the aim is to produce a population of solutions that is as close to fully feasible as practicable. A number of strategies were trialled. These included uniform random sampling, uniform random sampling excluding pre-defined unavailable periods (those in set $\mathcal{N}$), and seeding with solutions that had been pre-optimised in one of the objectives only.

The most promising approach however involved the use of constructive heuristics as discussed in Section 2.3.1.2. Two broad categories of constructive heuristics have been proposed in the literature (Pillay and Özcan, 2019). *Static* heuristics require lectures to be sorted by some metric, where this fixed ordering then determines the sequence of assignments. *Dynamic* heuristics involve recalculating the metric values after each assignment, thus providing greater adaptive potential. In both cases, the chosen metric is intended as a measure of 'difficulty to assign'.

The static heuristics *largest enrolment* (LE) and *largest degree* (LD) and the dynamic heuristic *saturation degree* (SD) were tested on the ITC2007 benchmark. LE relies on the number of enrolled students for its metric. Lectures with a larger number of students take priority. LD, as described for the generic case in Pillay and Özcan, 2019, uses the number of potential clashes a lecture has with other lectures resulting from commonality of students. Since explicit student sectioning is not a feature of the ITC2007 benchmark, the metric is redefined analogously as: The sum total of lectures that have either a curriculum or a teacher in common with the lecture being assessed. Priority is given to lectures with higher numbers of potential clashes in this respect. The metric for SD is the number of available feasible places, i.e. those that would not cause a hard constraint violation at the point of assignment. The lecture with the lowest value at each decision point is chosen for assignment. Across all heuristics, ties are broken at random.

Once a lecture has been chosen on the basis of its metric value, a place is randomly selected from the set of feasible places currently available to that lecture. If no feasible place exists, an infeasible place (excluding pre-defined unavailable periods in set $\mathcal{N}$) is chosen at random instead. A secondary, period-based heuristic is suggested in Pillay and Özcan, 2019 as an optional, more discriminatory, alternative to random sampling. The proposed system declines to include this with the following justification: Any infeasible solutions that may have been constructed in phase one stand to be quickly bred out of the population by an inherent hard constraint handling mechanism. The extra expense of a period-based heuristic was therefore found to outweigh the marginal gains in feasibility rate.

In testing LE, LD and SD, 10 independent repetitions were carried out for each problem instance. In each repetition, 100 timetable solutions were constructed, the number chosen as being a reasonable magnitude of population size. The primary quality measures to consider are the proportion of solutions that are feasible, and the relative speed of obtaining them. As with all experiments in this thesis, the computation was performed on a 12-core Ryzen9 with 32GB RAM, base clock speed 3.8GHz. The wall clock speed shown here resulted from using a single core and no parallelisation. Figure 4.2 shows the results for the three heuristics over the 21 instances.

SD achieved superior feasibility rates for every instance, while being computationally dearer. At the scale of a population size of 100, this additional time cost amounts to no more than a few seconds. More pertinently, all SD rates are 0.99 or higher, with the exception of the 3 instances. These were `comp02` (0.81), `comp05` (0.12) and `comp19` (0.56). In the case of the infeasible constructions for these

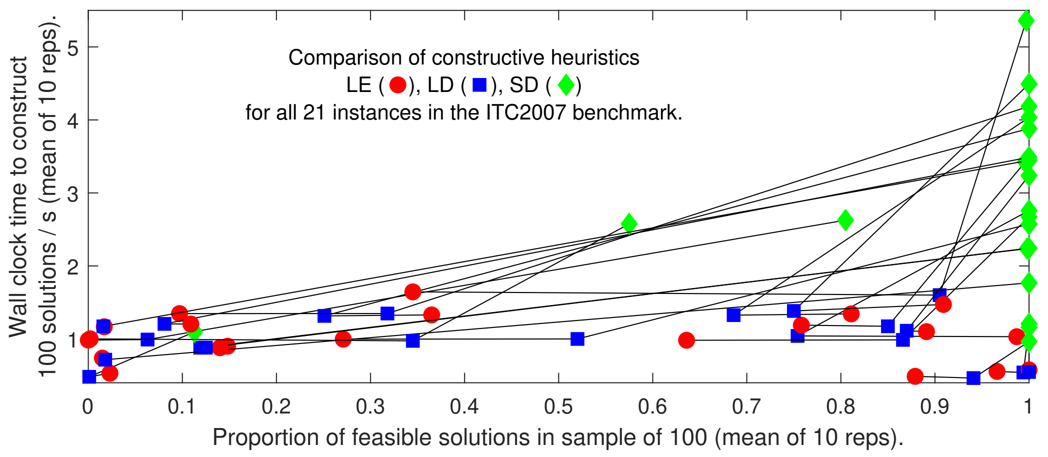


Figure 4.2: Performance comparison of 3 constructive heuristics. Lines connect results for common instances.

problems, the average distance to zero for each hard constraint objective is given in Table 4.1.

Table 4.1: Average distance to feasibility for each hard constraint objective, over the infeasible subset of the 4000 solutions constructed by SD.

Instance	**H2**	**H3**	**H4**	**H5**
`comp02`	0.31	3.12	0	0.21
`comp05`	0.19	18.63	0	0.21
`comp19`	0.49	3.81	0	0.21

These show that in a minority of cases where SD fails to achieve a near-perfect feasibility rate, the expected violations of hard constraints in the infeasible subset are nonetheless low. In particular, **H4** is zero in all cases.

4.3.2.1 Search space size and sample bias

Besides feasibility and speed, other factors may be worth considering in the assessment of an initial population created by a constructive heuristic. The percentage of unique individuals in the sample is one example. In the aforementioned tests, 100% uniqueness was achieved across all instances and all heuristics on this measure. Additionally, some measure of dispersion or dissimilarity between individuals may be important in terms of the subsequent exploratory reach of the optimiser.

The thoughts above can be consolidated in the following question: Does the set of feasible solutions produced by a constructive heuristic represent a uniform sample of the feasible solution space, or does the process introduce certain biases?

To investigate this, it is proposed to completely enumerate the feasible search space of a very small problem. Then, hypothesis testing can be applied to samples generated from the constructive heuristic. To reverse engineer such a problem of sufficient yet manageable size, some intuition is first needed around the relationship between problem variables and the size of the feasible search space. As the ITC2007 formulation is a combinatoric problem, this space is theoretically finite and countable. In the case of the competition instances, the size of the feasible solution set is presumed to be a vanishingly small proportion of the universe of all possible complete assignments. A closed analytical formula for the former is hard to obtain owing to the complex inter-dependencies of hard constraints. Sampling this entire universe using the Monte Carlo method is likewise not practicable due to the extreme rarity of feasible solutions. It is, however, possible to decompose the search space and apply a mixture of exact and approximation methods in order to reach an estimated upper bound for the size of the feasible set. In providing an exposition of this hybrid method, we make use of a toy example, `smallA`, whose characteristics are given in Table 4.2.

Table 4.2: Characteristics of a toy problem instance, using the same notation as in Table 3.7

Name	$\vert\mathcal{C}\vert$	$\vert\mathcal{L}\vert$	$\vert\mathcal{U}\vert$	$\vert\mathcal{T}\vert$	$\vert\mathcal{D}\vert$	$\vert t\vert$	$\vert\mathcal{R}\vert$	$\vert\mathcal{N}\vert$
`smallA`	5	15	3	3	5	4	2	25

When all $|\mathcal{L}|$ lectures are assigned to a day, timeslot and room, the total number of combinations is given by:

$$(|\mathcal{D}| \times |t| \times |\mathcal{R}|)^{|\mathcal{L}|} \tag{4.6}$$

This equates to 1.07×10^{24}. To enumerate the subset of this universe that has no hard constraint violations relating to unavailability, the formula is adapted so that pre-defined unavailable periods are subtracted across all lectures. This set is labelled **H4**c to denote that its members are compliant with constraint **H4**. The size in the `smallA` example, by way of an exact calculation, is 4.69×10^{21}. Considering now **H4***c* as its own universe, Figure 4.3 shows set relations between the remaining hard constraints. The suffix v here denotes the violation of a particular constraint, from $\{\mathbf{H2}, \mathbf{H3}, \mathbf{H5}\}$. Any combination of these constraint violations is possible within a single solution. The probability of sampling a feasible solution from the $\mathbf{H4}c$ probability space is denoted by:

$$P((\mathbf{H2}v \cup \mathbf{H3}v \cup \mathbf{H5}v)') \tag{4.7}$$

Where $'$ denotes the complement of a set. Notwithstanding the timing of respective unavailable periods, in general any lecture can cause a room clash (**H2**) violation with any other, whereas violations of **H3** or **H5** can only result from pairs of lectures which share the same curriculum or teacher respectively. This, coupled with the observation that typically $|\mathcal{R}| \ll |\mathcal{U}|$ in the ITC2007 problems, gives an intuition that $\mathbf{H5}v$ may often be the largest of the three sets in Figure 4.3. Hybrid approximation methods may therefore deliver more accurate and efficient estimates for the other two sets. A mathematical lower bound for the size of $(\mathbf{H3}v \cup \mathbf{H5}v)$ is given by $\mathbf{HX}v$, where $\mathbf{HX}v = max(\mathbf{H3}v, \mathbf{H5}v)$. This bound is equal to the exact value when one of $\mathbf{H3}v$ or $\mathbf{H5}v$ is a subset of the other in solution space, which in practical problems is rare. $1 - P(\mathbf{HX}v)$ nonetheless gives a loose upper bound for the probability of the shaded region in Figure 4.3. Monte Carlo sampling can be applied to this shaded region to infer the relative size of the feasible region as defined by (4.7).

To estimate $P(\mathbf{H5}v)$, the probability of at least one teacher clash occurring, the set can be decomposed further. Since lectures have a maximum of one teacher each, the probability of a clash involving teacher $T_i \in \mathcal{T}$ is independent of those involving $T_j \in \mathcal{T}$, $i \neq j$. Therefore, the probability of at least one teacher clash occurring somewhere in a solution is:

$$P(\mathbf{H5}v) = 1 - \prod^{T \in \mathcal{T}} P(T_{clashFree}) \tag{4.8}$$

An exact formula for $P(T_{clashFree})$ can be applied in the case where a course-to-teacher ratio is 1:1. When this ratio is higher (i.e. a single teacher takes multiple courses), a closed analytical formula is not necessarily available due to inconsistent

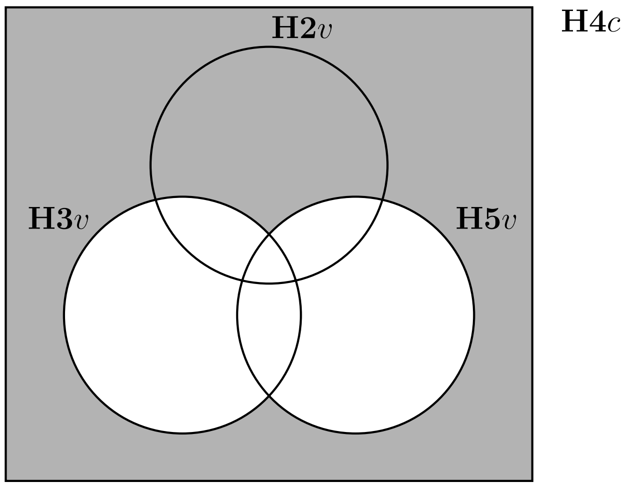


Figure 4.3: A Venn diagram — sets are labelled first by a particular hard constraint $\{\mathbf{H2}, \mathbf{H3}, \mathbf{H5}\}$, with the suffix c denoting compliance and v denoting at least one violation. Ultimately, a Monte Carlo estimate is run on the shaded region to determine the relative size of the feasible space.

patterns of unavailability between these courses. Instead, the Monte Carlo method can be used on the subset of course lectures sharing this teacher. If these multiple courses have the same pattern of unavailable periods (i.e. $unav(C_i) = unav(C_j) = \cdots = unav(C_n)$), they can be aggregated and the exact formula used, although this is a rarity in practise. For `smallA`, the results for the decomposition by teacher are given in Table 4.3. Equation (4.8) then gives an estimate for $P(\mathbf{H5}v)$ of 0.89.

Table 4.3: Probabilities of teacher clashes, by teacher, for `smallA`, where * indicates an exact result. Estimates are returned where a closed analytical formula is unavailable.

$T \in \mathcal{T}$	$P(T_{clashFree})$
1	0.67*
2	0.56
3	0.29

When estimating the probabilities of curriculum clashes, decomposing by individual curricula is not workable. This is because courses may belong to multiple curricula, inducing probabilistic dependencies. Instead, a curriculum co-occurrence graph can be used. In such a graph, each curriculum is represented by a node. The co-occurrence of two curricula within a particular course is represented by an edge. Multi-edges are removed for simplicity. Figure 4.4 shows the graph for `smallA`, which consists of two components.

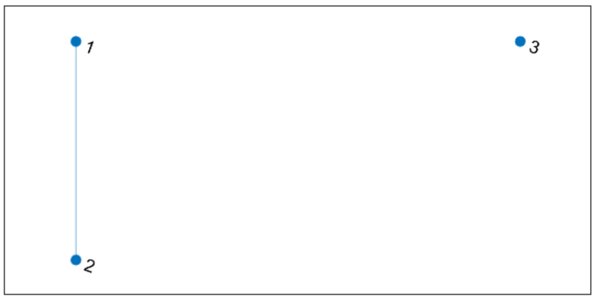


Figure 4.4: The curriculum co-occurrence graph for `smallA`, in which nodes represent curricula, and edges their co-occurrences.

The probability of a clash involving a curriculum in some component k_i is independent of that in component $k_j, i \neq j$. Components are therefore analogous to teachers in the previous exposition, and the corresponding exact and inexact approaches may be applied. Results for `smallA` are given in Table 4.4.

Table 4.4: Probabilities of curriculum component clashes, by component, for `smallA`, where * indicates an exact result. Estimates are returned where a closed analytical formula is unavailable.

Curricula component k	$P(k_{clashFree})$
1 {1, 2}	0.10
2 { 3 }	0.46*

Thus, the estimate for $P(\mathbf{H3}v)$, and consequently $P(\mathbf{HX}v)$ too, is $1 - (0.10 \times 0.46) = 0.95$. The estimated lower bound for the probability of the remainder of the universe, or the shaded region in Figure 4.3, is 0.05, which implies 2.2×10^{20} solutions.

A sample of solutions from the shaded region in Figure 4.3 can be obtained by way of a stochastic heuristic. This incorporates rejection sampling in the case where a solution fails during construction. Figure 4.5 shows convergence of the Monte Carlo estimate for the feasible proportion of the shaded region.

The final estimated upper bound reached for the size of the `smallA` feasible space, as in (4.7), is 0.89% of $\mathbf{H4}c$, or 4.2×10^{19} solutions. As `smallA` is a sufficiently small problem, this result can be empirically checked by applying the Monte Carlo method to the entire $\mathbf{H4}c$ space with a sample budget of half a million. Figure 4.6

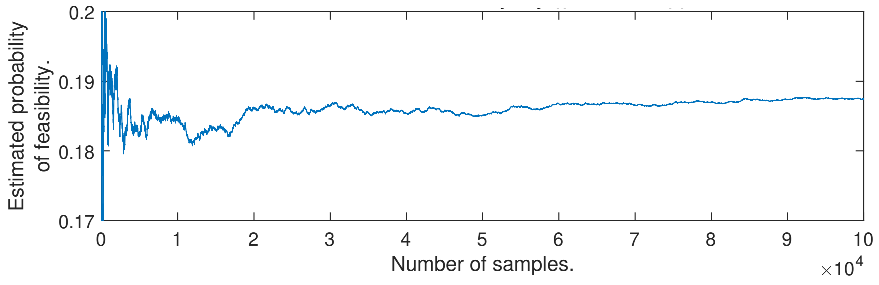


Figure 4.5: `smallA`: Monte Carlo estimate convergence on the shaded region represented by $(\mathbf{H4}v \cup \mathbf{H3}v)'$.

shows this convergence trace. The result obtained, 0.39%, is shown to be both within the estimated bound and of the same order of magnitude.

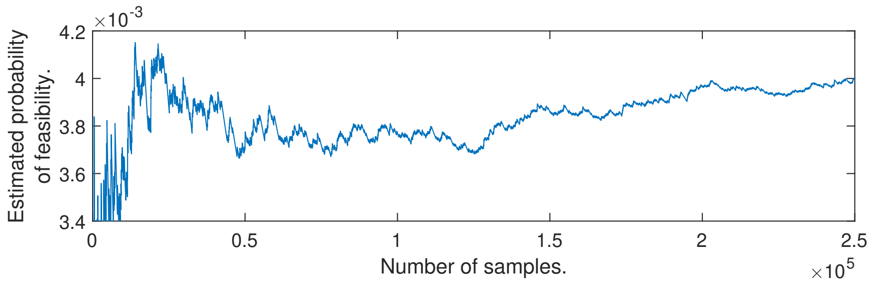


Figure 4.6: `smallA`: Monte Carlo estimate convergence on the complete universe of assignments.

The decomposition method developed above can now be applied to one of the smallest problems in the benchmark, `comp11`. The cardinality of all possible complete assignments here is $225^{162} \approx 1.13 \times 10^{382}$. Of these, 1.91×10^{376} are compliant with constraint **H4**. Figure 4.7 gives a breakdown of values of $P(T_{clashFree})$ for $T \in \mathcal{T}$. From this, $P(\mathbf{H5}v) \approx 1 - 2.9 \times 10^{-7}$.

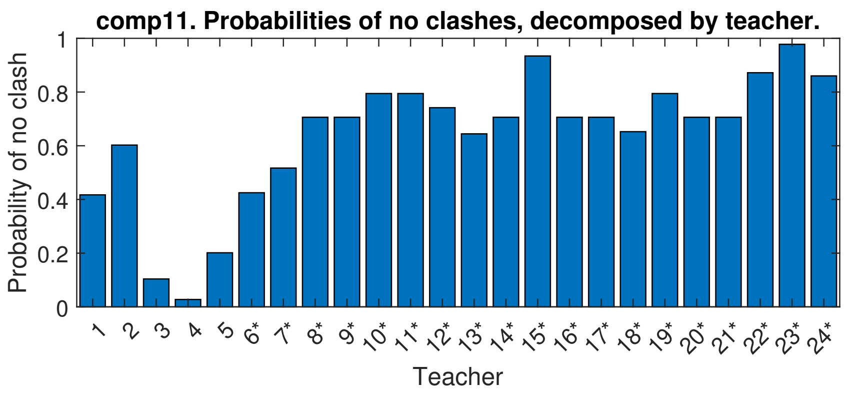


Figure 4.7: Probabilities of teacher clashes, by teacher, for `comp11`, where * indicates an exact result. Estimates are returned where a closed analytical formula is unavailable.

The curricula co-occurrence graph for `comp11` is shown in Figure 4.8, with the values of $P(k_{clashFree})$ for corresponding component k given in Table 4.5. For estimated values, sample budgets were varied according to the size and complexity of the component. $P(\mathbf{H3}v) \approx 1 - 1.24 \times 10^{-22}$. This is also consequently the value of $P(\mathbf{HX}v)$, meaning the upper bound estimate for the shaded region probability is 1.24×10^{-22}.

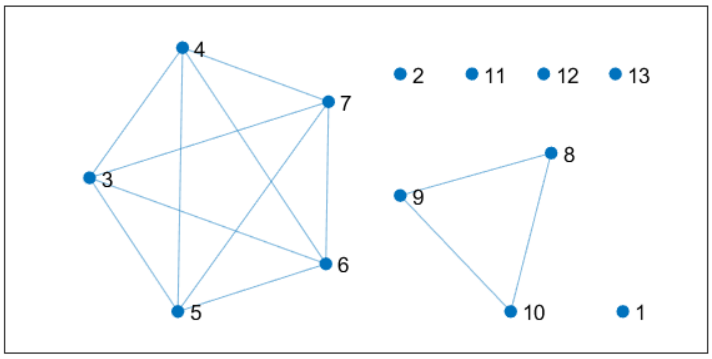


Figure 4.8: The curriculum co-occurrence graph for `comp11`, in which nodes represent curricula, and edges their co-occurrences.

Table 4.5: Probabilities of curriculum component clashes, by component, for `comp11`, where * indicates an exact result. Estimates are returned where a closed analytical formula is unavailable.

Curricula component k	$P(k_{clashFree})$
1 {1}	0.0059
2 {2}	0.0066*
3 {3, 4, 5, 6, 7}	1.75×10^{-7}
4 {8, 9, 10}	1.54×10^{-8}
5 {11}	0.0718*
6 {12}	0.0885
7 {13}	0.1873

Monte Carlo sampling of the shaded region does not yield directly useful results in the case of `comp11` (or other benchmark problems) due to the extreme rarity of feasible solutions. Within this region, the identifying attribute of a feasible solution is that it comprises $|\mathcal{L}| = 162$ unique placements. Figure 4.9 shows a histogram of the number of unique placements in 100,000 non-rejected samples. A fitted Gaussian can be used to approximate the probability of a feasible solution, which is given by the integral bounded by 161.5—162.5 in the extreme right-hand tail. This value is 1.1286×10^{-24}.

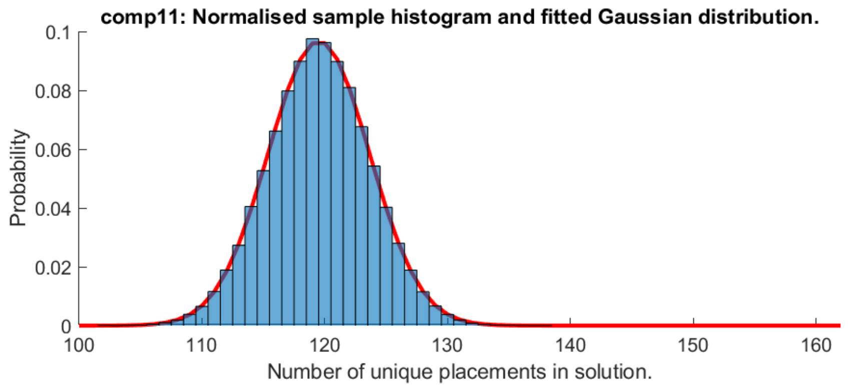


Figure 4.9: `comp11`: Sample histogram and fitted Gaussian for Monte Carlo sampling of the shaded region $(\mathbf{H4}v \cup \mathbf{H3}v)'$, used to estimate the probability of rare event feasible solutions.

Using this value and those obtained previously for `comp11`; an estimate for the upper bound of the feasible space, expressed as a probability $P(\mathbf{H4}c)$, is 1.39×10^{-46}. This corresponds to a cardinality for the feasible solution set of 2.65×10^{330}.

This hybrid approximation method gives a useful intuition about relative search space sizes in the problem domain. Neither `comp11`, nor even `smallA` are suitable for complete enumeration, due to their astronomical search spaces. For this purpose, and guided by the techniques in this section, a more appropriately sized problem, `smallB`, was created instead. It has the following characteristics:

Table 4.6: Characteristics of `smallB`, using the same notation as in Table 3.7

Name	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{D}\rvert$	$\lvert t\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{N}\rvert$
`smallB`	3	6	2	2	5	4	2	26

A total of 5,488,320 non-equivalent[1] feasible solutions were systematically identified and nominally indexed. 100 million feasible solutions were then sampled using the SD constructive heuristic.

Figure 4.10 shows the distribution of this sample, as a histogram with bin sizes of 10,000. Figure 4.11 displays the same data but ordered by ascending bin count. A χ^2 goodness of fit test returns a p-value of 0, leading to a rejection of the

[1] Equivalent solutions are solutions whose quality is invariant to the re-indexing of certain lectures or rooms. The idea is formalised in Section 5.1.2.2 of the following chapter.

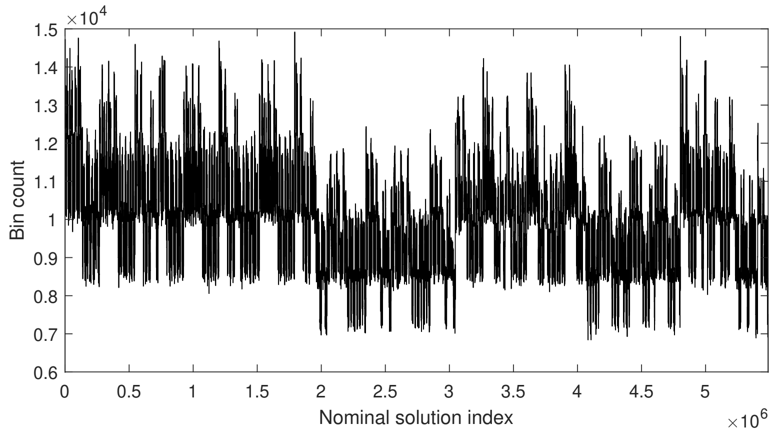


Figure 4.10: A histogram showing the distribution of 100 million feasible solutions to `smallB` generated by the SD constructive heuristic. A bin width of 10,000 solutions is used and solutions are indexed nominally according to a systematic enumeration of the feasible solution space.

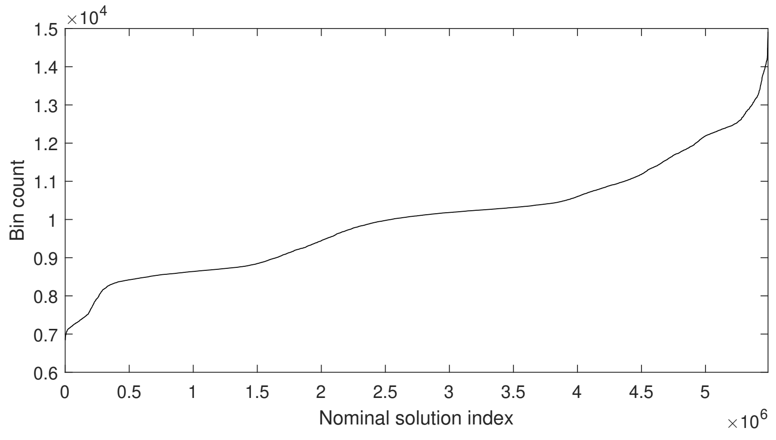


Figure 4.11: A histogram showing the distribution of 100 million feasible solutions to `smallB` generated by the SD constructive heuristic (the same sample data as in Figure 4.10). A bin width of 10,000 solutions is used and indices are reordered by increasing bin count.

null hypothesis that the sample data follows a uniform distribution. Evidence for this can be also gleamed by cursory inspection of Figure 4.10, in which some blocks of solutions visibly display a higher probability of occurrence than other adjacent blocks. Self-similar patterns also emerge in this histogram when viewed at higher scales.

Analysing two examples at opposite extremes gives an insight into why this variation exists. There were six unique solutions that did not appear in the sample at all, and three that tied for the highest bin count of 62. In the former case, following the pathway of construction revealed a consistent fact at each step. The choice of lecture placement had a minimal effect on the number of valid placements remaining for other lectures. That is to say, a high number of potential placements were left open, creating more potential branches for the heuristic to follow. The chance of following any one such branch at a given assignment point was therefore reduced. This occurrence generally stems from a lecture from course C_i being placed in a period p, for which unavailability constraints are already applied across a maximal number of the courses in $\{\mathcal{C} \setminus C_i\}$. The converse phenomenon was responsible in the case of high probability solutions.

Corresponding χ^2 tests on samples by static heuristics LD and LE led to a rejection of the null hypothesis in the same way as for SD, meaning that all three heuristics produce non-uniform samples. As problem size scales, the effects of these biases are exacerbated. In `smallB`, $|\mathcal{L}| \ll |\mathcal{R}| \times |\mathcal{P}|$. In benchmark and real world problems, the relative difference between these quantities is smaller, and some feasible solutions may be completely unreachable by way of a static heuristic. Further analysis is needed to ascertain whether this may also be the case with dynamic heuristics, although their mechanism of iterative sorting is understood to confer an advantage in terms of opening up pathways to more solutions.

For these reasons, the versatility of SD is preferred, even though it is acknowledged not to produce a uniform sample. Once initialised by SD, the optimiser proceeds to phase two, the constituent parts of which are outlined across the following sections 4.3.3 to 4.3.8.

4.3.3 Core algorithm

NSGA-III is a successful evolutionary algorithm that supports many-objective optimisation with constraints (Deb and Jain, 2013). It is an extension to the popular NSGA-II algorithm, which was originally conceived for lower-dimensional objective spaces (Deb et al., 2002). Due to the many-objective nature of the problem, NSGA-III serves as an appropriate core around which to develop phase two, albeit with a number of modifications.

4.3.4 Selection and constraint handling

Alongside the initialised population, the SD heuristic implementation returns an array of feasibility flags, toggled during construction. The property `con` holds the flag associated with each solution, with a `true` value indicating at least one violation of a hard constraint. For the first generation only, scores for the four soft constraint objectives are then calculated in full. Two-way tournament selection is used to select a mating pool. Randomly paired candidate solutions are first compared on their `con` property, with the lower value indicating the winner. Feasible solutions are thereby given priority. Should the `con` values be equal, the sum of the objective scores is used as a tie-breaking fitness measure.

One natural consequence of utilising the `con` property in this way is that infeasible solutions can only decrease monotonically as a proportion of the active population over time. Figure 4.12 illustrates this on a toy problem solved using an unmodified NSGA-III. The red line shows that, once feasible solutions enter the population, they cannot be replaced except by other, better, feasible solutions. Once the infeasible timetables have been entirely bred out, the optimiser rapidly converges to a perfect score, as shown by the blue line which indicates the estimated hypervolume (Zitzler and Künzli, 2004).

Hypervolume (or the hypervolume indicator) is a useful performance metric for optimisation problems with more than one objective, and is used regularly throughout these chapters. The metric quantifies the volume of the objective space that is dominated by a set of solutions, relative to an appropriate reference point. As such, it captures a sense of both the convergence and coverage of the solution set, with a higher hypervolume value representing better performance.

Despite the trivial nature of `smallA` and a sizeable population of 364, a full 120 generations were required movement occurred in the plots as shown. One conclusion from this is that the standard NSGA-III operators are largely ineffective for the UCTP. In the next section, some better-suited operators are proposed.

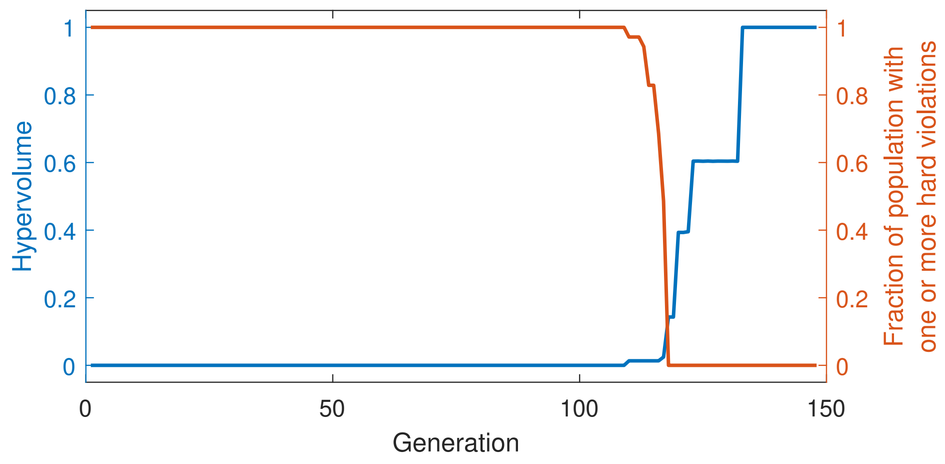
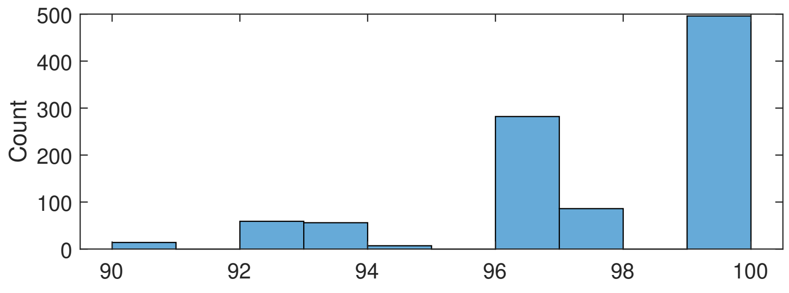


Figure 4.12: Active population statistics — hypervolume (blue line), and fraction of infeasibility within the population (red line) — for an unmodified NSGA-III on the instance `smallA` with population size 364 and default parameters. Note that only feasible solutions contribute towards the hypervolume measurement.

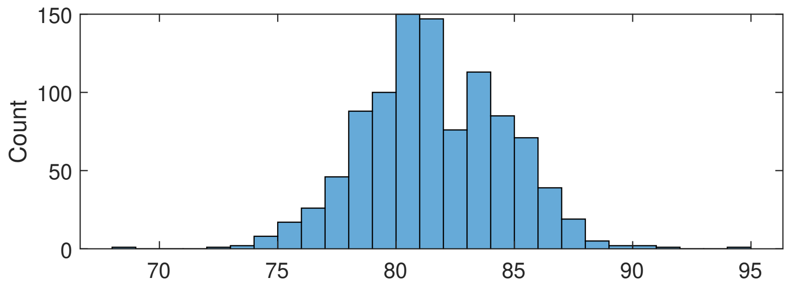
4.3.5 Genetic operators

For a real-valued encoding in a continuous space, NSGA-III traditionally uses simulated binary crossover (SBX) and polynomial mutation as its genetic operators. For the discrete problem at hand, adaptations were first made to both genetic operators to ensure the preservation of integrality in decision space. Further investigation determined that, with no meaningful ordering apparent for entities such as days or periods, traditional polynomial mutation is not necessarily well suited for this problem domain. Similarly, standard SBX carries the risk of degenerating timetables by recombining promising subsets in an injudicious way, thereby worsening the overall solution quality.

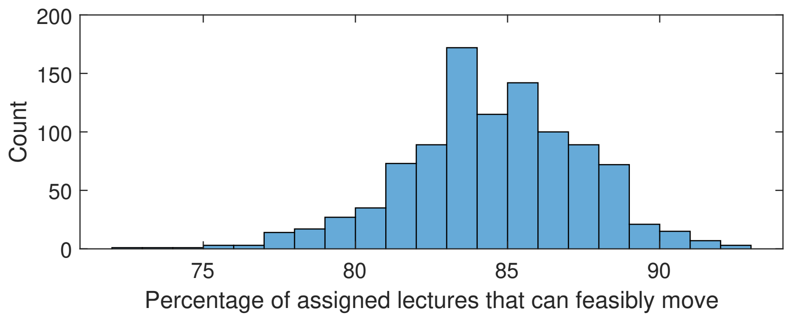
In further tests, uniform crossover and uniform random mutation were trialled in place of the original operators. Uniform crossover in particular, however, was found to cause large jumps in the solution space, diminishing the ability of the optimiser to exploit its more immediate neighbourhood. In complex, combinatorial timetabling problems, a successful crossover operator requires domain-specific knowledge and can be computationally expensive. For simplicity, the proposed approach therefore dispenses with crossover, devoting the entirety of its operator budget instead to a guided mutator. In developing this mutator, the following test was conducted:



(a) comp01



(b) comp02



(c) comp12

Figure 4.13: Histogram of the percentage of assigned lectures with at least one feasible move available, for instances comp01, comp02 and comp12. The sample is the first 1000 feasible solutions constructed by *saturation degree*.

1,000 feasible solutions were constructed using SD. For each assigned lecture of each solution, a check was made on the number of places it could be reassigned to without violating the overall solution feasibility. For some assignments, there were no feasibility-preserving moves available. The histograms in Figure 4.13 show (for comp01, comp02 and comp12 respectively) the distribution of percentages of assigned lectures, over each 1,000-solution sample, that had at least one such available feasible move.

For all problems tested, the distributions demonstrate that the expected number of available feasible moves for a starting solution is generally high. The optimiser can be guided, therefore, by imbuing the first version mutator, known as MuPF, with a preference for feasible moves where they exist. Algorithm 5 outlines this ‘preference for feasibility’ mutator, abbreviated as `MuPF`. In line 4 of the pseudo-code, all potential feasible moves are identified in advance of conducting a random uniform sample of that set. While it would be cheaper to simply take the first feasible move found during identification, the systematic nature of the process makes it implicitly biased. The (small) additional computational expense is deemed acceptable in order not to bias the move selection.

Algorithm 5: Preference for feasibility (`MuPF`) mutation operator

```
1 Input: One starting solution (feasible or infeasible)
2 Output: One mutated solution
3 Randomly select a lecture, $l_i$, to mutate
4 Identify the set of places, $feasMoves(l_i)$, to which $i$ can be reassigned
    without violating the feasibility of the solution
5 if $feasMoves(l_i) = \varnothing$ then
6     Reassign lecture $l_i$ to a new randomly chosen place, excluding pre-defined
        unavailable periods (set $\mathcal{N}$)
7 else
8     Reassign lecture $l_i$ to a place from $feasMoves(l_i)$, chosen at random
```

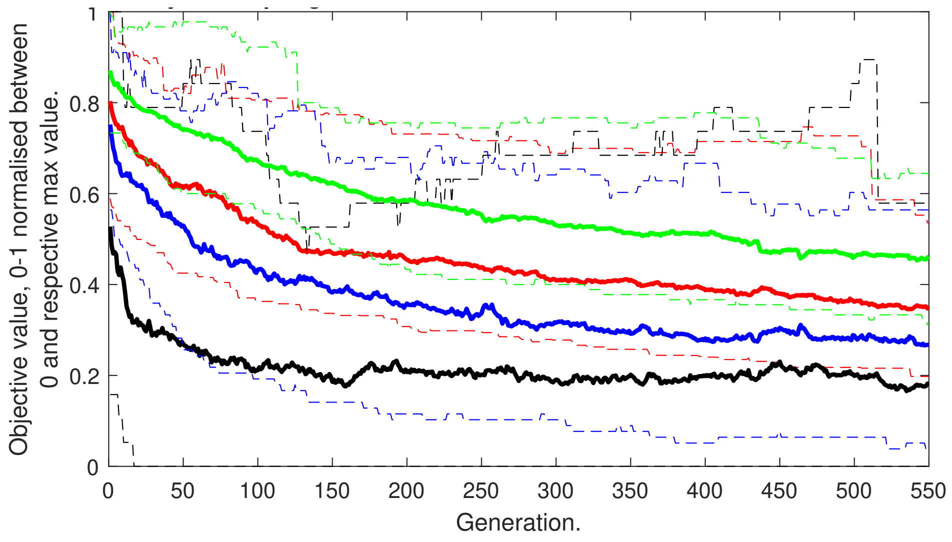


Figure 4.14: Red: **S1**. Black: **S2**. Blue: **S3**. Green: **S4**. Normalised minimum (lower dashed lines), mean (solid lines) and maximum (upper dashed lines) values of individual soft constraint objectives in the active population, for generations 1 to 550, using the proposed `MuPF` mutator for `comp01` and 10 repetitions.

Using this mutator, a test run was performed on `comp01`. Figure 4.14 shows traces of the population minimum, mean and maximum for the normalised individual objective scores as they evolved over 550 generations. Over the course of this run, the minimum values of objectives [**S1**, **S2**, **S3**, **S4**] improved from [1599, 15, 88,

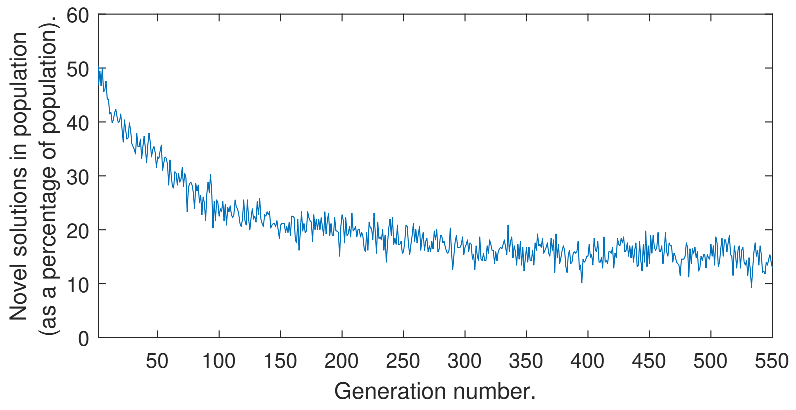


Figure 4.15: `comp01`: Novel solutions (with regard to the entire search history) in the active population using the base optimiser with mutator `MuPF`. Generations 2 to 550 are shown.

66] to [537, 0, 6, 28] respectively. The inference is that, regardless of its simplicity, the perturbations introduced by `MuPF` effectively facilitate the search process, even when operating within a limited budget. Figure 4.15 gives a measure of the novelty within the active population for this run. A healthy degree of churn is in evidence. Towards the end of the run, around 15% of each successive population is made up of previously unseen solutions.

Further tests emphasised the large relative contribution that **S1** often makes to a scalarised objective score. A modification to the mutator, in which sufficient room capacity is taken into account, was proposed to better target this objective. Denoted `MuPFPR`, this mutator has the same initial preference for placements that preserve feasibility, but appends a secondary preference for placements in rooms whose capacity is sufficient for the number of enrolled students. The pseudo-code is given in Algorithm 6.

A comparison between `MuPF` and `MuPFPR` is displayed in Figure 4.16. A run on `comp01` was carried out with a function evaluation budget of 2 million. The extra room-related guidance provided by `MuPFPR`, shown as a blue trace, helped drive the convergence rate for the **S1** objective (in the top left tile), at no significant detriment to objectives **S2** or **S3** (which are temporal rather than spatial). **S4** is also not significantly compromised in this particular example, but in general has a greater potential to conflict with **S1** due to both being room-related.

Algorithm 6: Preference for feasibility, preference for room (`MuPFPR`) mutation operator

```
1  Input: One starting solution (feasible or infeasible)
2  Output: One mutated solution
3  Randomly select a lecture, $l_i$, to mutate
4  Identify the set of places, feasMoves($l_i$), to which $l_i$ can be reassigned
     without violating the feasibility of the solution
5  if feasMoves($l_i$) $= \varnothing$ then
6      Reassign $l_i$ to a new randomly chosen place in any room with sufficiently
         high capacity but excluding pre-defined unavailable periods (set $\mathcal{N}$)
7  else
8      if feasMoves($l_i$) $\cap$ sufficientRooms($l_i$) $= \varnothing$ then
9          Reassign lecture $i$ to a place randomly chosen from feasMoves($l_i$)
10     else
11         Reassign $l_i$ to a place randomly chosen from the given non-empty
             intersection
```

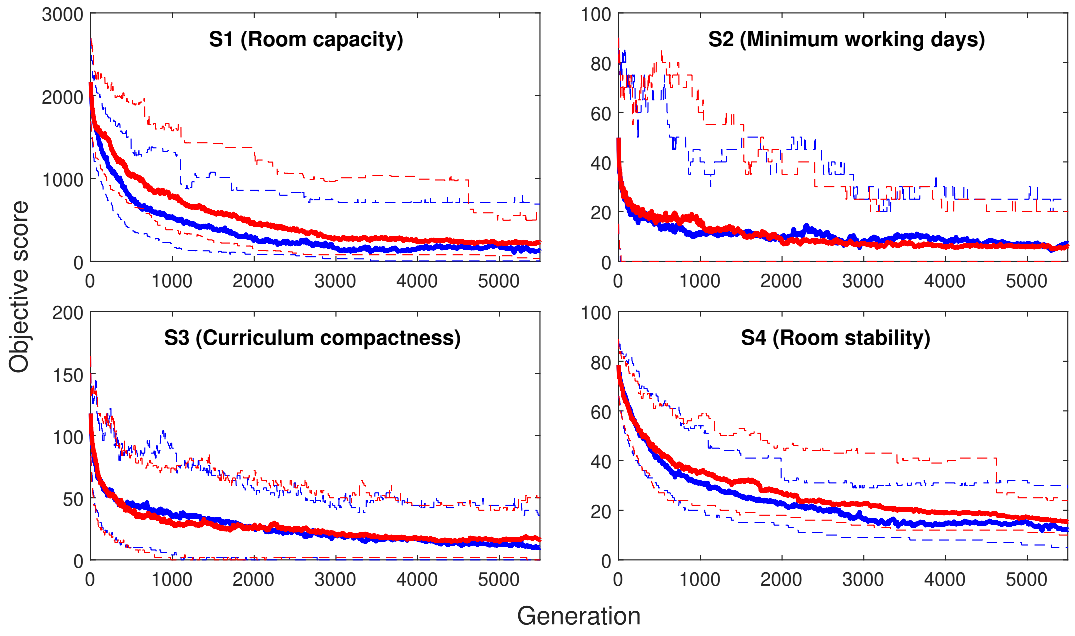


Figure 4.16: Red: Mutator `MuPF`. Blue: `MuPFPR`. Traces shown are the minimum (lower dashed lines), mean (solid lines) and maximum (upper dashed lines) objective scores from each generation for a single repetition of `comp01` with 2 million function evaluations (or approximately 5,500 generations with a population of 364).

Incorporated into the mutation process is an implicit feasibility checker. A violation flag, `conMutation`, is toggled if and only if `feasMoves`$(i) = \varnothing$. The returned `con` property for that child is generally given by (`conParent` $\vee$ `conMutation`) — except in the case when the parent solution is infeasible and the mutation (in isolation) is feasible. In such a case, the status of the child is unknown and a full evaluation of the hard constraints must be called. The rarity of this happening ensures that, in practise, the hard constraint evaluators seldom need to be executed at all — an example of a time-saving partial evaluation. The following section details how δ-evaluations are used to make similar savings when calculating the soft constraint objectives.

4.3.6 δ-evaluations

δ-evaluators can speed up the function evaluations by an order of magnitude, enabling experimentation with higher budgets. This is achieved by computing only the small areas of inter-generational change rather than evaluating an entire solution. The concept was suggested for timetabling as far back as Ross et al., 1994. More recently, Geiger, 2012 reported expending 13 million function evaluations in 10 minutes on a single CPU core for `comp01` by incorporating δ-evaluations, and yet many authors neglect to take advantage of the strategy. Profiling of the system suggested that 50% of run time was used on function evaluations, making it a prime candidate for streamlining.

The process by which the proposed δ-evaluators obviate the need for full evaluations is as follows: The ID of the lecture to be perturbed is recorded. The value contributed to the parent objective score by the assignment of this lecture is calculated. This value is subtracted from the objective score of that parent, which is known *a priori* from the previous generation. Lastly, the contribution of the new assignment in the child solution is added. Objective **S1** is best suited for a fast δ implementation, due to the fact that the value contributed by an individual lecture is independent of those from other lectures. For the remainder of the objectives, interactions between the lecture being perturbed and various other lectures must also be accounted for — specifically, those from the same course (for **S2** and **S4**), or those with a common curriculum (**S3**). Combined over four objectives, the δ-evaluators nonetheless offer a sizeable time saving over their fully executed counterparts, as illustrated in Figure 4.17.

While the run time of a full evaluator scales with the number of lectures, the δ run time scales with the number of mutations — due to the resulting combinatorial

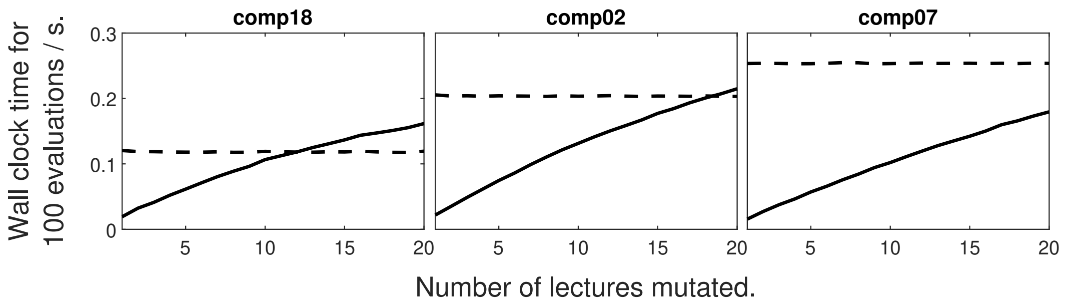


Figure 4.17: A comparison of the estimated time complexity (mean of 10 repetitions) for the combined δ-evaluators (solid lines) vs. full evaluation (dashed lines), for a small (`comp18`), medium (`comp02`) and large (`comp07`) sized problem and a variable number of mutations.

interactions. Under a single lecture mutation, the δ-evaluator gives the largest time savings, by multiples of 6.3, 10.7 and 13.2 for the respective problems shown. Additionally, the likelihood of returning a feasible solution decreases exponentially with multiple mutations under `MuPFPR`. For these reasons, `MuPFPR` is restricted to one execution per solution per generation.

4.3.7 Non-dominated sorting

At the heart of NSGA-III (and other many-objective optimisers) is the dominance relation on objective scores, which is used to sort a merged parent/offspring population into non-dominated fronts. The efficient non-dominated sort with sequential search (ENS-SS) is used (Zhang et al., 2015). The hard constraint handling procedure mandates that any solution with a `con` flag value `true` is automatically dominated by all feasible solutions, regardless of the quality of its objective vector. The only way, therefore, in which such a solution can be admitted into the next generation is if the cardinality of the feasible solution set is less than the active population size. This in turn reaffirms the following about phase two: If a given generation is fully feasible, all subsequent generations are also fully feasible. To promote diversity, NSGA-III also associates solutions with rays passing through a set of `popSize` uniformly distributed points on the 4-dimensional unit hyperplane. The normal-boundary intersection method with two layers is used to obtain these coordinates. `popSize` is a geometrically constrained approximation to the desired, user-input population size, `setPopSize`.

4.3.8 Archiving

An external archiving routine, implementing the ND-Tree structure (Jaszkiewicz and Lust, 2018, Fieldsend, 2020), is included inside the main optimisation loop. Its purpose is to update and maintain the set of non-dominated solutions found over the course of the search. The archive is passive, meaning that it plays no active role in the optimisation process.

4.3.9 Summary

In this section, the rationale and description of each algorithmic step for a many-objective solver is given. An overview of how these steps fit into the two-phase system is given by the sequence diagram in Figure 4.1, in which initialisation is followed by optimisation. A integer-valued encoding is used, with a chromosome of length $|\mathcal{L}| \times \langle d_i, p_i, r_i \rangle$ (Section 4.3.1). Initialisation is by *saturation degree* constructive heuristic, with infeasible solutions permitted in the first generation. Two-way tournament selection is used as in Section 4.3.4. Perturbation is by mutation-only, using the operator `MuPFPR` exclusively (Section 4.3.5). In replacing individuals for the next generation, the traditional NSGA-III approach is used. Non-dominated sorting is employed until the size of a front causes the population size to be exceeded, at which point, closeness to a set of decomposition rays is used (Section 4.3.7) until the population quota is satisfied. Genotype measures are not incorporated in this chapter. At each generation, a non-dominated archive is maintained (Section 4.3.8) and the complete search history is also archived for *post hoc* system analysis.

In the next section, details of two stages of experimentation are laid out. The latter is based on the two-phase system, while initial tests were also carried out using a single-phase model in which hard constraints were relaxed.

4.4 Experiments

4.4.1 Relaxed hard constraints experiments

Before arriving at the two-phase model, preliminary tests were carried out into the viability of solving the many-objective problem in a single phase. Barring **H1** and **H4**, whose satisfaction was guaranteed by the encoding scheme, all other hard constraints were relaxed. The violation count of each was cast as an additional objective, giving rise to a 7-dimensional space and no `con` property.

When initiated with a random population, this algorithm exhibited poor convergence. In a second round of tests, the initial random population was instead heavily seeded with solutions that had been pre-optimised to zero in one or two objectives only. Figure 4.18 provides an example of an initial population in which a large majority of the 200 solutions have no **S1** violations[2].

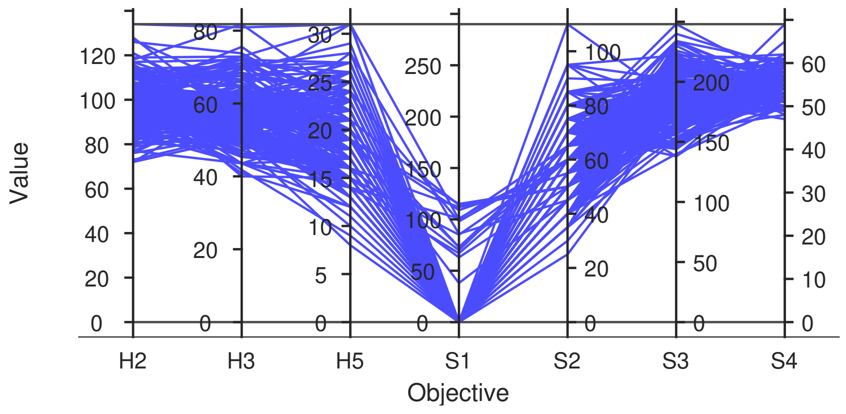


Figure 4.18: The 7-objective evaluation of an initial population (size 200) of solutions to `comp01`, in which the majority of solutions have a zero **S1** score.

Figure 4.19 shows the evolved population at termination, while Figure 4.20 shows a trace of the hypervolume[3]. The gradient at termination indicates that further significant improvement was unlikely. Zero scores for **S1** were not achieved consistently across the entire population, despite the initial and deliberate bias

[2] **S1** is arguably the simplest of the objectives to pre-optimise and has been used here for illustrative purposes. Similar experiments were attempted in which **S2** ... **S4** were similarly pre-optimised. In all cases, the evolution was hampered by genetic drift, causing poor convergence.

[3] Note that the apparent non-monotonicity of the hypervolume is due to estimation error, as high-dimensional hypervolume is expensive to calculate exactly — details of the methods used are elucidated in Section 4.4.2.

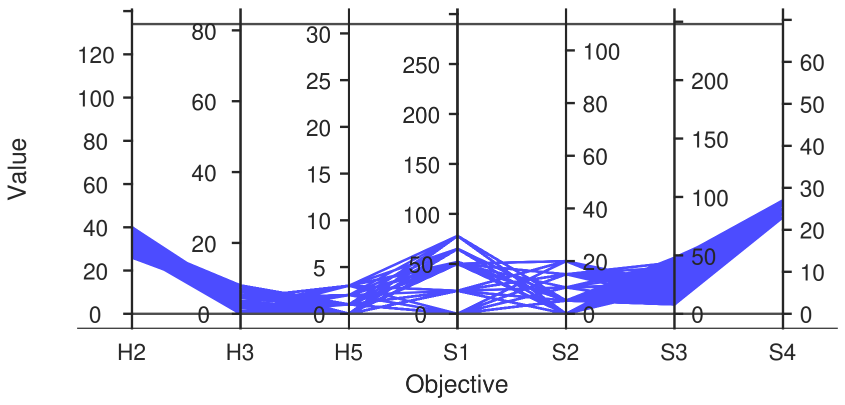


Figure 4.19: The 7-objective evaluation of an active population of solutions to `comp01` at termination, after 250,000 function evaluations (approximately 1,250 generations). The optimiser was initiated using the population shown in Figure 4.18 and uses the same axes ranges for consistency.

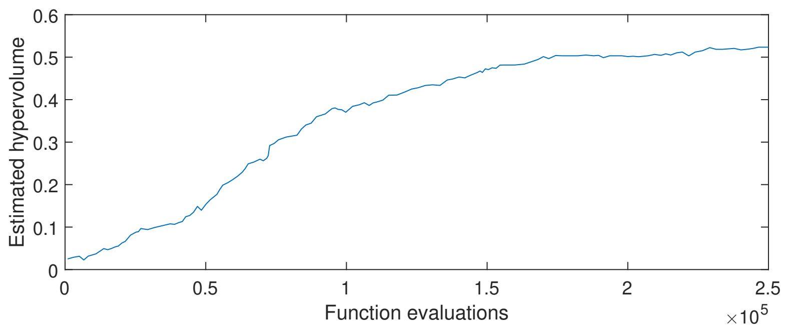


Figure 4.20: The progression of the hypervolume over the course of an optimisation run that evolved the initial population from Figure 4.18 into the final population from Figure 4.19.

towards this objective. Most pertinently, the relaxation of the hard constraints meant that, throughout this batch of runs, feasible solutions were almost never located.

These preliminary findings strengthened the argument for a two-phase approach as developed in Section 4.3: The first strand of which is that feasibility rates could be improved by tackling the hard constraints in advance of the soft. Secondly, a greater selection pressure could be brought to bear by the dominance relation if the dimensionality of the objective space were reduced from seven to four. The next section lays out the main experiments using this two-phase system.

4.4.2 Two-phase system experiments

Each run of the optimiser was allocated to a single core of the Ryzen9 machine, as per the original ITC2007 stipulation. Parallelisation was used only across independent repetitions. In the absence of the original CPU benchmarking program, termination was after 600 seconds wall clock time, which was the limit intended by the competition, and `setPopSize` = 100. For each problem instance[4], 30 repetitions were carried out by varying the random seed. The results are reported in terms of the following performance metrics:

- The number, at termination, of unique solutions in the non-dominated archive. This cardinality is reported in both the objective space and design space, as these values may differ owing to the many-to-one mapping. All equal quality designs are retained.
- A Monte Carlo estimate of the hypervolume indicator, for which 1,000,000 samples are used and objective scores are normalised. The hypervolume is a useful indicator by which to gauge performance, as it captures a sense of both spread and convergence, and has the property of being strictly monotonic with respect to Pareto dominance (Bader and Zitzler, 2011). Theoretical upper bounds on the maximum objective scores are used as its reference point coordinates. The method of derivation for these is given under the following heading.

4.4.2.1 Hypervolume reference points

By contriving a worst case scenario on each objective, coordinates of a theoretical worst point can be loosely approximated. Potential interactions between objectives and curricula are ignored, as are pre-defined unavailable periods. This ensures a conservative estimate, suitable for use as the hypervolume reference point. In the case of each objective, the value is determined by way of the following steps:

For **S1**: Sort lectures by number of enrolled students, in descending order. Replicate each room capacity value $|\mathcal{P}|$ times, and sort in ascending order. Pair lectures to rooms accordingly, beginning with the most popular lecture and smallest room.

For **S2**: Distribute lectures across the smallest number of days possible, given by $\lceil(|\mathcal{L}|/t)\rceil$.

[4] Under the UD2 configuration, `comp03` is identical to `comp15`, so the latter was omitted.

For **S3**: The maximum number of 'isolated' periods is given by $\lceil (t/2) \times |\mathcal{D}| \rceil$. In a loop over curricula 1:$|\mathcal{U}|$, compare this value to the number of lectures. If the value is less than or equal to, increment a running total by the difference. Otherwise, increment by the number of lectures in the curriculum.

For **S4**: Distribute lectures across the greatest number of rooms as possible, given by $min(|\mathcal{R}|, |\mathcal{L}|)$.

Table 4.7 shows the calculations for all benchmark instances.

Table 4.7: Coordinates of hypervolume reference points for benchmark problems `comp01` to `comp21`, as upper bounds on the worst point.

`comp`	**S1**	**S2**	**S3**	**S4**
01	3,606	360	294	124
02	14,072	815	1,704	201
03	11,160	720	1,536	179
04	8,151	665	1,130	207
05	11,102	475	2,954	98
06	10,632	990	1,668	253
07	9,760	1,150	1,806	303
08	7,711	700	1,166	238
09	9,269	720	1,492	203
10	9,296	1,020	1,610	255
11	3,196	335	500	103
12	4,533	650	3,102	130
13	10,668	670	1,292	226
14	7,138	830	1,392	190
15	11,160	720	1,536	179
16	9,354	985	1,650	258
17	10,229	930	1,608	240
18	2,638	455	954	91
19	10,108	760	1,386	203
20	12,338	1,070	1,874	269
21	10,687	860	1,666	233

4.5 Results

Table 4.8 shows the results in terms of the key performance metrics and additional statistics, alongside results from Geiger, 2009. In Figure 4.21, parallel coordinate plots show the spread of non-dominated solutions for the single best performing repetition, for ten problems. In ten other problems, the **S1** dimension was successfully

Table 4.8: Results from 30 independent repetitions. b_s is the best scalarised solution score found over all repetitions, while $b_s(\mathbf{S1}, \mathbf{S2}, \mathbf{S3}, \mathbf{S4})$ gives a decomposition over the objective scores (averaged over all unique objective vectors whose sum is b_s). $\mathcal{A}$ is the final archive of non-dominated solutions, where sets of unique vectors in objective or decision space are distinguished by subscripts $_o$ and $_d$ respectively. Cardinalities for both are given as median values. $hv(\mathcal{A}_o)$ is the (mean) hypervolume of $\mathcal{A}_o$. The best scalarised results from the two approaches in Geiger, 2009 are given as G1 (Threshold Accepting with 1% threshold) and G2 (reference point based).

	Proposed approach					Others	
Instance	b_s	$b_s(\mathbf{S1}, \mathbf{S2}, \mathbf{S3}, \mathbf{S4})$	$\lvert\mathcal{A}_o\rvert$	$\lvert\mathcal{A}_d\rvert$	$hv(\mathcal{A}_o)$	G1	G2
`comp01`	11	(4, 0, 4, 3)	10.5	7,492	0.959	5	10
`comp02`	200	(0, 45, 129, 26)	15.5	396	0.792	108	134
`comp03`	162	(0, 52.5, 92, 17.5)	17	850	0.831	115	154
`comp04`	92	(0, 6.7, 65.3, 20)	16.5	481.5	0.853	67	90
`comp05`	564	(0, 151.7, 403.3, 9)	331.5	669	0.732	408	611
`comp06`	167	(0, 15, 104, 48)	15.5	233	0.777	94	159
`comp07`	183	(0, 20, 90, 73)	17	125.5	0.710	56	155
`comp08`	108	(0, 0, 74, 34)	13.5	300.5	0.810	75	120
`comp09`	158	(0, 40, 94, 24)	24	623	0.821	153	197
`comp10`	133	(0, 20, 60, 53)	15.5	179.5	0.763	66	104
`comp11`	0	(0, 0, 0, 0)	2	45,453	0.981	0	0
`comp12`	515	(2, 210, 292, 11)	412	735.5	0.759	430	660
`comp13`	131	(0, 30, 84, 17)	20	390	0.832	101	133
`comp14`	125	(0, 20, 90, 15)	19	1,288.5	0.866	88	120
`comp16`	149	(0, 20, 82, 47)	16	139.5	0.787	n/a	n/a
`comp17`	182	(0, 15, 130, 37)	17	243.5	0.779	n/a	n/a
`comp18`	116	(0, 30, 78, 8)	44.5	1,372.5	0.884	n/a	n/a
`comp19`	180	(0, 37.5, 126, 16.5)	16.5	703	0.798	n/a	n/a
`comp20`	200	(0, 25, 119, 56)	138.5	408	0.751	n/a	n/a
`comp21`	213	(0, 35, 148, 30)	17.5	240.5	0.770	n/a	n/a

collapsed to zero, meaning they can be visualised in 3-D objective space, as per Figure 4.22.[5]

In terms of run time, expensive components such as non-dominated sort meant that the proposed method was found to execute fewer total function calls in 600 seconds than many of its single-objective competitors. Despite this, scalarised results were seen to approach those of single-objective solvers on some problems which is encouraging — `comp11` in particular was solved to optimality. With regard to the individual objective scores, the targeted operator `MuPFPR` was capable of rapidly optimising **S1** to zero across the board (except for `comp01` where the value of **S1** in the optimal solution is known to be 4, and `comp12`). These gains were not made

[5] Problem `comp11` is excluded from this set of plots, as a perfect score was attained.

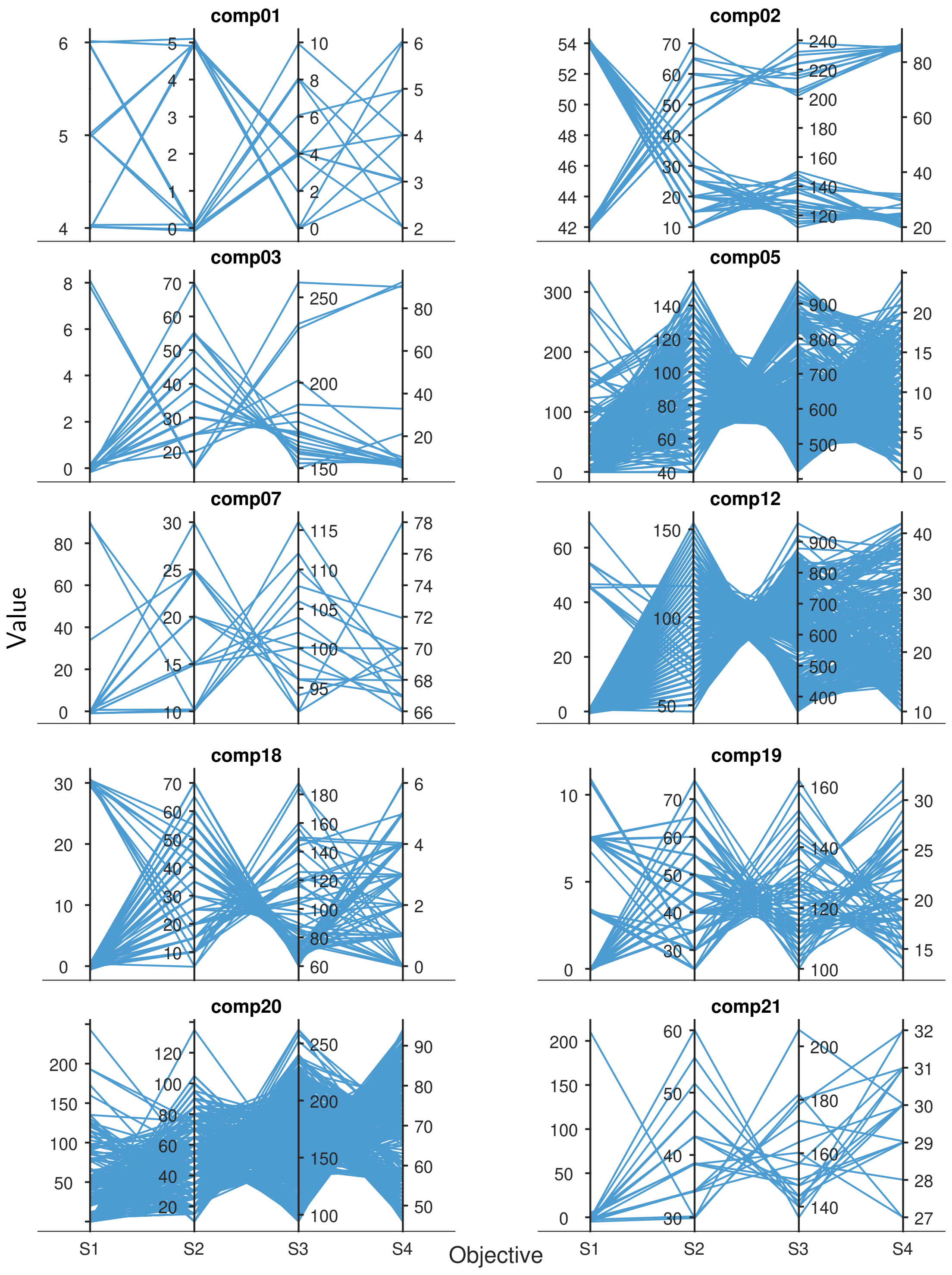


Figure 4.21: Non-dominated solution sets found during single repetitions for the ten problems `comp01`, `comp02`, `comp03`, `comp05`, `comp07`, `comp12`, `comp18`, `comp19`, `comp20`, `comp21` shown as parallel coordinate plots. Each plot window shows the repetition that achieved the greatest hypervolume.

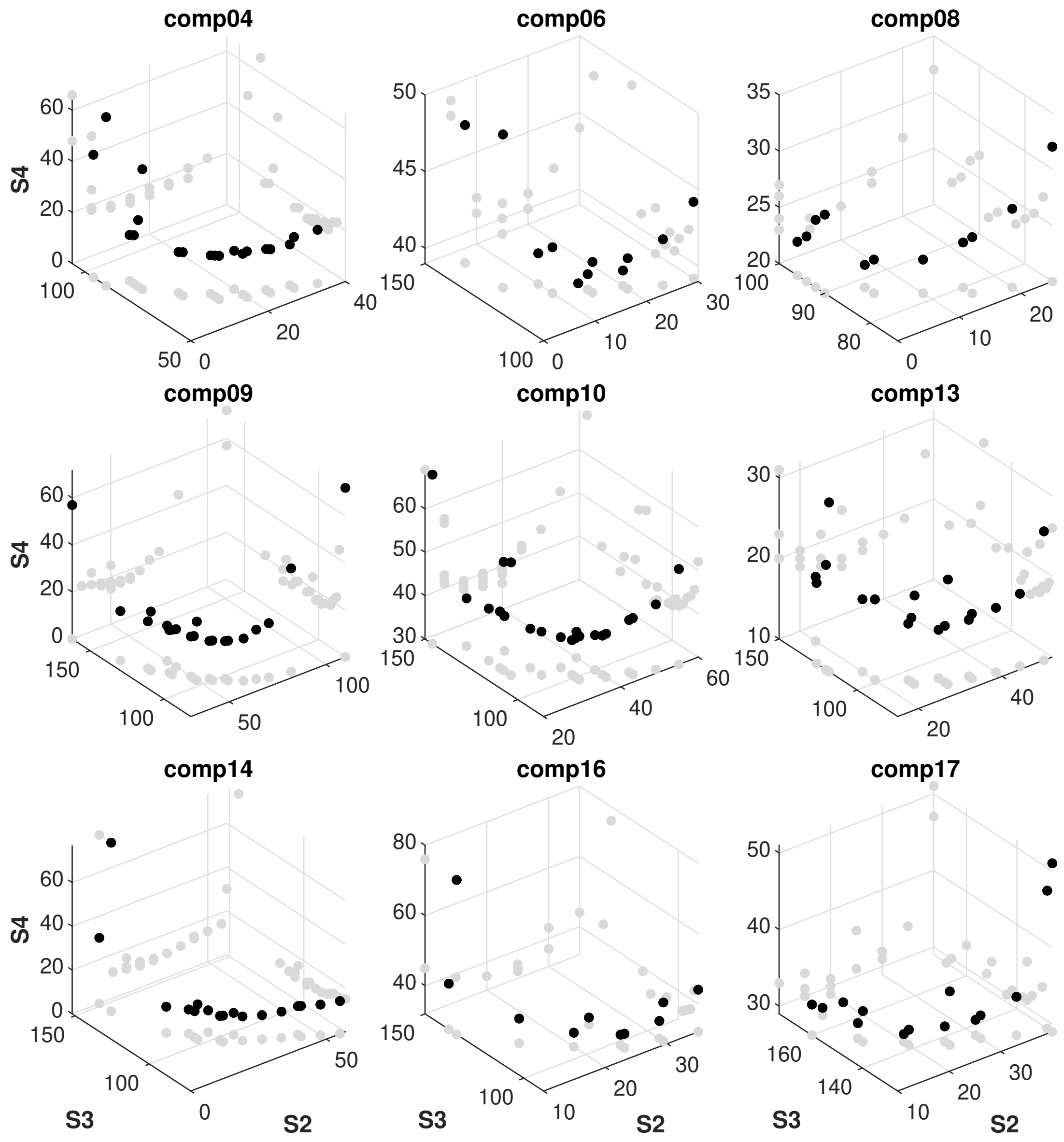


Figure 4.22: Non-dominated solution sets (black points) in (**S2**,**S3**,**S4**)-space, found during single repetitions for nine problems in which the fourth objective, **S1**, was consistently optimised to zero. Grey points show the projection of the sets onto the coordinate axes. For each instance, the data is taken from the repetition that achieved the greatest hypervolume.

at the expense of other objectives however, which showed improvement without exception during the runs. This suggests that additional bespoke operators, targeted at these objectives, may be a promising next step in striving to closer approximate the true Pareto front.

The results of Geiger, 2009 are included as a point of reference because of the matching benchmark and the attention given to soft constraints as individual objectives within the algorithm. A comparison with the reference point based approach (G2 in Table 4.8), shows competitive or superior performance in terms of scalarised scores, although this claim is weakened by the CPU benchmarking discrepancy as mentioned in Section 4.4.2. It is important to note that a direct statistical comparison is not possible as, despite some similarities in process, Geiger, 2009 ultimately returns a single scalarised solution. The major point of differentiation and novelty in this chapter is that the approach returns a Pareto front approximation set (with sizes and spreads of those sets varying with the complexity or size of the instances.) Achieving this in a comparable timescale while also maintaining competitiveness on the preferred metric of Geiger, 2009 is therefore encouraging.

The original NSGA-III offers arguably more of a like-for-like comparison, but it must be understood in the context of the problem domain. When the original NSGA-III was run under under default settings and a matching evaluation budget, no feasible solutions were returned for any of the problem instances. In NSGA-III, random (rather than heuristic) initialisation leads to a first generation of wholly infeasible timetables. From here onwards, and as discussed in Section 4.3.5, traditional operators do not respect the hard constraints, meaning that the populations never migrate away from infeasible regions, even if soft constraint violations are gradually improved upon. Thus, while a formal statistical comparison between the proposed algorithm and NSGA-III is also not appropriate, it can be stated that the modifications made in the former are overwhelmingly suitable and beneficial for the UCTP since feasible solutions are consistently found.

The approach appears relatively problem-agnostic, in contrast to, for example, Geiger, 2012 whose results show high variance across problems. Most importantly, it works under the assumption of *a posteriori* decision maker preferences. Different areas and extremes of the Pareto front are therefore explored simultaneously and a well-spread set of non-dominated solutions can be delivered, as shown in Figures 4.21 and 4.22. The hypervolume indicator values in Table 4.8 also provide evidence for this, with nine problems achieving a mean of 0.8 or higher. Figure 4.21 also helps to understand the trade-offs between objectives. Conflict can be seen between **S2** and **S3**, for example. This may be expected, as **S2** depends on spreading lectures more widely over time (days), while **S3** relies on forcing lectures closer together in time (periods).

As lower absolute objective scores are achieved, the cardinality $|\mathcal{A}_o|$ naturally tends to decrease, as in `comp01` (median 10.5) and `comp11` (median 2). This tension can be understood by the proximity of the front to the origin and consequent sparsity of distinct points on the 4-D integer lattice.

$\mathcal{A}_d$ is the set of unique decision vectors, discovered at any point during the optimisation run, that evaluate to the objective vectors stored in the final non-dominated archive. The set $\mathcal{A}_d$ is obtained by retrospectively scanning the complete history archive, retrieving the necessary data and removing duplicates. The observation $|\mathcal{A}_d| \gg |\mathcal{A}_o|$ interestingly highlights the extent to which multiple designs map to a common objective point. This particular facet of the problem domain will be analysed further in the next chapter.

4.6 Conclusions

In a departure from the single-objective treatment of the ITC2007 timetabling problem, this chapter proposed a two-phase, many-objective optimiser based on a modified NSGA-III in which hard constraints are handled procedurally and soft constraints are cast as objectives.

It is effectively parameterless — save for `setPopSize` and termination criteria which are pragmatic user choices — and therefore does not require or depend upon tuning. The value 2 in the 2-way tournament selection operator is a commonly used default. The time cost associated with many-objective algorithms was alleviated by prudent use of δ-evaluators. The various strategies for speeding up (or by-passing) the calculation of objective scores was successful in yielding inexpensive evaluations. However, mitigation against the cost of non-dominated sort was only partial. The algorithm unsurprisingly had a lower execution rate for function calls than many single-objective solvers. Comparing its performance on an equal function evaluation budget rather than a time budget would be enlightening, as the gradients in Figure 4.16 suggest further gains are available.

A simple guided mutation operator reduced the otherwise large violation contributions caused by over-filling rooms (constraint **S1**) to zero wherever this was possible (i.e. in all problem instances except `comp01` and `comp12`). Selection and non-dominated sorting ensured convergence of the other objectives as well as feasibility of solutions, while a quick start was guaranteed by the SD constructive heuristic.

Some avenues for future work include increasing the convergence speed of the remaining three objectives by widening the pool of targeted operators. If the mutator is interpreted as a neighbourhood, a more systematic exploration may be possible. Figure 4.13 gives an intuition about the size of such a neighbourhood. An adaptive element may be added to phase two to select from such a pool based on the state of the current population or trajectory of the evolution. Alternatively, objectives that reach optimality may be aggregated with `con` so that any solutions sub-optimal in this objective will thereafter be automatically dominated. Further analysis will also help characterise the trade-offs between the objectives. By their definitions, **S1**/**S4** and **S2**/**S3** represent the two pairs with the greatest potential to conflict. The large cardinalities of the decision space solution sets suggests that genotype diversity could also play a useful role in the selection process.

In the next chapter, the system is developed further. Redundancy in the search space is examined, leading to a modification of the encoding scheme. Aforementioned genotype diversity measures, alluded to in schematic Figure 4.1 but absent in this chapter, are introduced and tested. The effects of additional perturbation operators and a modified tournament selection operator on run time and performance are assessed. Finally, a robustness metric is designed and inserted as an additional objective.

5. Genotype Diversity, Enhanced Operators and Robustness

When using evolutionary algorithms to solve the UCTP as a many-objective problem, measures aimed at encouraging population diversity are commonly applied in the objective space (Preuss et al., 2010). Difficulties can arise when the search encounters plateau regions, caused by multiple designs evaluating to a common objective vector. The work in this chapter seeks to address this through making subtle, but impactful, adjustments to the algorithmic components. These include the non-dominated sort and its leverage of genotype information, as well as the perturbation and selection operators. The enhancements made, which lead to improved diversity through better exploration, are important for the final section, in which robustness is then added as a new objective. Chapter 5 is therefore laid out in three distinct parts:

In the first part (Section 5.1) the phenomenon of plateaus is illustrated with a specialised tree diagram. An enhanced diversity measure that includes genotype crowding is proposed in order to alleviate the issue shown. A *standard form* encoding is also developed to handle solution equivalence and reduce metric entropy. Four metrics and a baseline are tested, using the optimiser developed in Chapter 4. Hypervolume is the primary performance measure. As an additional integrated selection criterion in the non-dominated sort, behind dominance and phenotype diversity, genotype diversity measures are shown to improve performance. Hamming distance is the most successful of the metrics tested.

In the second part (Section 5.2) the pool of operators is extended to include those with stronger perturbations. Analysis is conducted into aspects of performance of each, such as feasibility preservation and run time. The tournament selection routine is also modified to take advantage of symmetry within the problem. A variant of this operator is proposed that is motivated by discouraging the proliferation of equivalent or recycled solutions. While no significant improvement in overall hypervolume is achieved by the use of this operator, increases are seen in the final set cardinality instead. This suggests that a more granular rendering of the Pareto approximation set can be achieved. A larger set of choices could then be made available to a decision maker, without compromising on timetable quality.

In the third part (Section 5.3) a fifth objective — robustness — is introduced to the optimiser. To facilitate this, some realistic disruption scenarios are defined. A metric is then devised in order to capture the deterioration in timetables when re-evaluated subject to various disruptions to the problem. Robustness is an important but often overlooked quality in real world timetabling, in which disruptions can occur regularly. Given the many-to-one mapping that exists in the UCTP model between designs and evaluations, a new objective representing robustness could be invaluable to a decision maker in discriminating between timetables of otherwise equal quality. The many-objective approach developed in this thesis provides the ideal set-up for analysing what trade-offs may be present between individual quality objectives and the robustness. It is therefore proposed to generate solutions as part of a 5-dimensional Pareto approximation set. When using a fixed disruption scenario, perfectly robust solutions are discoverable for the majority of benchmark instances. These solutions are also shown to be more robust to generalised disruptions when compared to those returned from the 4-objective system.

5.1 Diversity measures

5.1.1 Background

It has long been recognised that maintaining a diverse population of solutions is important in genetic algorithms tasked with navigating vast and fragmented search landscapes. This is as essential for timetabling as it is for other problem domains. Deb, 2001 cites diversity maintenance as one of the two key functions a good multi-objective optimiser must perform, the other being to converge as close as possible to the true Pareto front. Diversity and convergence may also at times conflict with one another. In Myszkowski and Laszczyk, 2021, a selection operator based on diversity is tested on a scheduling problem with the aim of nudging the search process towards unexplored ‘gaps’ and increasing the spread of solutions to encompass these regions. A weakness was found in relation to constrained problems, in that processing power may be wasted in the exploration of infeasible solutions, reducing the likelihood of convergence within the time budget.

In a recent survey of university course timetabling practises, Oude Vrielink et al., 2019 related work in Alkan and Özcan, 2003 highlighting the opposite problem. While exploring feasible solutions with a GA, the authors noted that premature convergence is a common occurrence, caused by individual solutions becoming too similar. Some techniques to counter this are mentioned, such as hypermutation or maintaining parallel sub-populations of solutions. A conclusion drawn from experimentation was that prohibiting non-unique offspring from joining the next population during the process of replacement was one successful way to maintain diversity.

A variety of encoding schemes have been proposed for timetabling, including indirect and direct. In the latter category, using a multi-dimensional logical array or its integer indices is a common approach (see for example Abdullah and Turabieh, 2008). However, depending on the semantics of the problem description, this style of encoding has the potential to invite degeneracy whereby multiple encodings represent solutions that are in some way equivalent to one another.

It is important to note that in timetabling, as in other forms of scheduling problem, the genotype-to-phenotype mapping is often many-to-one. Laszczyk and Myszkowski, 2019 notes that there are many ways to rearrange variable assignments that are not on a critical path and therefore do not alter the evaluated quality of that solution. This can cause a perverse outcome when enforcing diversity on the phenotype level, as solutions that occupy promising positions in genotype space can sometimes be unfairly removed. Plateau regions in phenotype space were explicitly recognised by Salehi and Doncieux, 2022 as one of three challenges (along with distance metric bias and evolvability traps) faced by quality-diversity (QD) optimisers. These are diversity-promoting search processes that seek to maximise the coverage of their solutions, usually by operating diversity preservation techniques in the phenotype space. Similar considerations underpin other established approaches, such as NSGA-III (Deb and Jain, 2013). In the original formulation of NSGA-III the preferential selection of solutions for the next generation is made primarily on their relative positions according to the dominance relation. Solutions whose objective vectors are not dominated by any others are the first to be accepted. If this non-dominated front (or some subsequent front) should cause the population size to be exceeded, a second criterion may be called upon to discriminate between its mutually non-dominated solutions. The simplest approach, as seen in Greenfield, 2003, is to randomise this choice. While this mitigates against the introduction of unwanted selection bias, it also fails to make use of information about solution spread. NSGA-III leverages this with a phenotype diversity strategy built around the projection of evenly distributed rays into objective space, yet it takes no account of how solution genotypes may be spread. As discussed in Deb, 2001, proximity in one space does not guarantee proximity in the other, particularly in many-variable, non-linear or highly complex problems.

A suggestion offered in Chen et al., 2013 and Chen and Chou, 2017, with regard to an air crew scheduling problem, was to select one portion of the required solutions using a phenotype crowding measure, and the remainder using an analogous genotype measure. For the latter, pairwise Hamming distances were used in order to ascertain the least crowded designs. A parameter was used to define the ratio between portions.

Conceptually, the use of genotype measures is not restricted to GAs. In Pérez et al., 2021, a novel harmony search approach was presented, in which traditional non-dominated sorting and phenotype crowding measures were enhanced by a genotype crowding measure to encourage dispersion in the design space. The authors argued that this extra criterion was useful in improving refined search (exploitation) in promising areas of the solution space for multi-objective problems.

As discussed, different ideas exist for the implementation of genotype crowding. One example is Deb and Tiwari, 2008, who proposed a multi-purpose 'omni' optimiser. When encountering a partial front and requiring further discriminatory power, this optimiser compared crowding measures in both spaces before selecting just one for use. For each solution in phenotype space, a loop was executed through the objective axes. The nearest lateral neighbours were found and the distance added to a sum total. Boundary points were considered infinitely distant. The same process was applied to the variable axes in genotype space, except that boundary points were given a one-sided distance, then doubled. In both spaces, the values were normalised and averaged. For each solution, if either distance exceeded the average, the maximum of the two was accepted, otherwise the minimum was accepted. This nearest neighbour approach in dual spaces was found to drive convergence through the promotion of diversity.

As per the normalisation used above, care must be taken with the intrinsic meaning of any variables used in the representation of a solution. In a real world engineering problem such as Desai and Williamson, 2009, values of design variables such as battery power and motor power are either on a continuum and/or have a natural ordering. The authors use a multi-objective GA, with integrated genotype diversity measure, to find Pareto optimal solutions to a hybrid electric vehicle problem. Because of these design variable properties, a normalised Euclidean distance was appropriate. In the timetabling problem under consideration here this may not be valid, as some variables do not possess such an obvious natural ordering.

The next sections are laid out as follows. In Section 5.1.2, the methodology is explained, including the proposed framework for choosing solutions from a truncated front. This comprises an investigation into metric entropy, leading to a proposed *standard form* encoding to deal with solution equivalence, and distance metrics. Experimental protocol and parameter settings are given in Section 5.1.3. Section 5.1.4

provides analysis and statistical testing of the experimental data, before conclusions on this part of the chapter are summarised in Section 5.1.5.

5.1.2 Methodology

The system developed in Chapter 4 is used both to initiate and optimise a population of solutions, but with the addition of a genotype diversity routine (as preempted in the schematic in Figure 4.1). While the perturbation operator in this system is denoted `MuPFPR`, the more generalised reassignment of a lecture l_i to some new randomly-chosen place within a solution x is denoted by the function $\mathit{reassign}(x,l_i)$. Additional relevant terminology and notation is summarised below.

Subsets of rooms with identical capacities are referred to as *equal room sets* and are contained by $\mathcal{E}$. The variable `editDist` is the value of the edit, or Levenshtein, distance between two given solution encodings. In context, this is defined as the minimum number of *reassign* moves required to transform one solution, or its equivalent, into the other. The population size parameter is denoted `popSize`. Any further notation is introduced at the point of use.

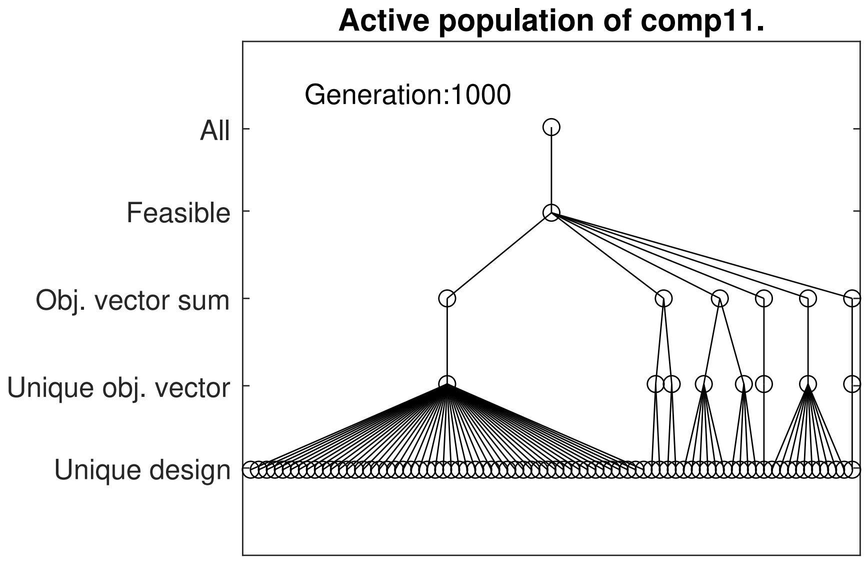


Figure 5.1: A novel visualisation to illustrate uniqueness within an evolutionary population. Shown is a snapshot of the active population at generation 1,000 for `comp11`, using the baseline solver. Each node corresponds to a solution subset in the particular space labelled on the left.

The timetabling problem domain and ITC2007 benchmark are known to exhibit a many-to-one mapping from the genotype to the phenotype space. That is to say, typically, a large array of unique solution designs evaluate to a common vector of objective scores. Figure 5.1 provides a novel way of illustrating this phenomenon. Taken from a snapshot (at Generation 1,000) of the optimiser run on `comp11` with `popSize` = 84, the tree graph breaks down the active population according to different aspects of uniqueness. Each node in the lowest level corresponds to a unique design vector. An animated form[1] is proposed for observing the changes over time. In the snapshot shown in Figure 5.1, All 84 population members are unique, yet collectively they map to only 8 distinct objective vectors, as represented by the level above in the tree. The exclusive use of a small-perturbation operator may exacerbate this issue for some problem instances due to the difficulty in escaping from local optima in the search space. The example instance in Figure 5.1 has a known optimum at [**S1**, **S2**, **S3**, **S4**] = [0, 0, 0, 0] (Hao and Benlic, 2011) (note though that not all problem instances have a perfect solution). As the population converges towards this singular point, the approximation front also becomes naturally constrained by the geometry of the 4-dimensional integer lattice in objective space. It is common therefore in `comp11` and other problems to encounter 'plateau' regions in the search landscape, where small changes in the designs have no effect on solution quality. The methodology is driven by this insight. Figure 5.1 serves also to emphasise the potential wealth of information at the genotype level that could usefully be leveraged when selecting solutions for the next generation. By developing a suitable genotype crowding routine and identifying the most dissimilar amongst designs, the aim is to help the search process escape plateau regions by encouraging exploration.

In our solver, non-dominated sorting is carried out on the union of parent and offspring populations. The leading front of solutions that are not dominated by any others (according to the dominance relation on objective vectors) are accepted first into the next generation. This priority continues to be applied to each subsequent non-dominated front. The first front that causes the number of admitted solutions to exceed the population size is known as a split, or truncated, front. The original NSGA-III algorithm provides a mechanism by which to discriminate between solutions in such a front. A set of reference points, usually identical in size to the population

[1] An example for `comp13` can be seen at: `https://www.youtube.com/watch?v=V70_b1w3KB0`.

Algorithm 7: Genotype diversity: Choosing procedure for truncated front

```
1  Inputs: $S_t$ (set of normalised solutions in accepted and last fronts
     combined), $F_t$ (set of normalised solutions in truncated front), $Z^r$
     (normalised reference points), $\pi(x)$ (associated ray/s for a solution or set of
     solutions $x$), $K$ = popSize - $|\{S_t \setminus F_t\}|$ (number of solutions required from
     the truncated front to make up the population quota), $dist$ (some pairwise
     distance metric on genotypes)
2  Output: $K$ solutions from truncated front $F_t$
3  $\rho \leftarrow \pi(x) : x \in \{S_t \setminus F_t\}$          // Get the rays associated with the
     solutions in the non-truncated front/s.
4  f $\leftarrow \pi(x) : x \in F_t$     // Get the rays associated with the solutions
     in the truncated front.
5  quota $\leftarrow \alpha : \alpha \subset$ f$, |\alpha| = K, \text{argmin}_\alpha[var(bins\{\rho \cup \alpha\})]$      // Choose the
     set of K rays in f that, when combined with $\rho$, gives the most
     even spread (lowest possible variance) over the rays.  The
     $bins$ function returns the vector of counts per ray.
6  for rayIdx = 1:$|Z^r|$                           // Loop through the rays.
7  do
8      if $bins$\{quota\}(rayIdx) $= 0$ then
9          Accept no new solutions for this ray.
10     else if $bins$\{quota\}(rayIdx) $=$ $bins$\{f\}(rayIdx) then
11         Accept all $x \in F_t$ associated with this ray.
12     else
                                           // Invoke genotype crowding.
13         Compute $dist(x_i, x_j)$ for all $x \in F_t$ associated with this ray, $i > j$.
14         Accept a set of solutions (size $bins$\{quota\}(rayIdx)) with the
             greatest minimum distance metrics (tie break at random).
```

size, are defined in objective space, through which decomposition rays are extended. Taking the perpendicular Euclidean distance, solutions are associated with their nearest ray. The algorithm then accepts as many solutions as are needed to satisfy the next population quota, giving preference to those associated with the most underrepresented rays. The motivation behind this is to maintain diversity in objective space. However, it is quite possible for rays to have no associated solutions or for multiple solutions to eventually cluster around a few rays. Furthermore, no genotype information whatsoever is incorporated into this process of selection.

Algorithm 7, the genotype diversity routine, is proposed here to replace lines 16 and 17 of Algorithm 1 pseudo-code in the original NSGA-III (Deb and Jain, 2013). It outlines the procedure for choosing solutions from a truncated front. In our Algorithm 7, line 3 obtains the rays associated with solutions in the previously accepted, non-truncated fronts. Line 4 does the same but for the truncated front. Line 5 selects a vector of counts of rays from the truncated front such that the variance of the combined counts (of solutions associated with rays in both the non-truncated and truncated fronts) is minimised.

By way of an example, consider a bi-objective problem with two decision variables and population size of seven. Figure 5.2 shows five solutions comprising a non-truncated front as black crosses. These are associated with seven rays as per the vector [2 2 1 0 0 0 0]. Two more solutions are required from a truncated front whose corresponding ray bin counts (solutions shown as red circles) are [1 2 2 3 0 1 0]. One solution from each of ray 4 and 6 (shown as solid lines) would be selected in this case, as this is the choice that yields the most even spread, [2 2 1 1 0 1 0] (corresponding to the minimal variance of 0.67), in the now-completed next population. Lines 6-14 loop through the rays, calling the index value from the quota derived above, [0 0 0 1 0 1 0]. For each ray, the quantity of accepted solutions is either none (the *if* clause in line 8, or rays 1, 2, 3, 5, 7 in the example), all (the *else* clause in line 10, or ray 6 in the example) or some (the *else* clause in line 12, or ray 4 in the example). In the last case, genotype crowding is invoked to determine which of the three solutions (enclosed by the dotted box in the objective space plot) to choose. The corresponding locations of the three points in decision space are shown as green triangles in the right window of Figure 5.2. Preference is given to the solution/s that have the highest genotypic dissimilarity to their closest neighbour, as defined by a distance metric (generalised as *dist*) in line 13. In the Figure 5.2 example, Euclidean distance is used for clarity of illustration, with the most dissimilar of the three solutions shown enclosed by a dotted box.

A traditional distance metric in this scenario, particularly for binary encodings, is the Hamming distance. This can be applied to the real encoding used here in an analogous way. However, a potential drawback exists in relation to the $\langle d, p, r \rangle$ style of encoding. Different designs can represent solutions that are, in practical terms, equivalent. This is not an explicit consideration within the original ITC2007 rules, however there are good real world motivations for acknowledging equivalence.

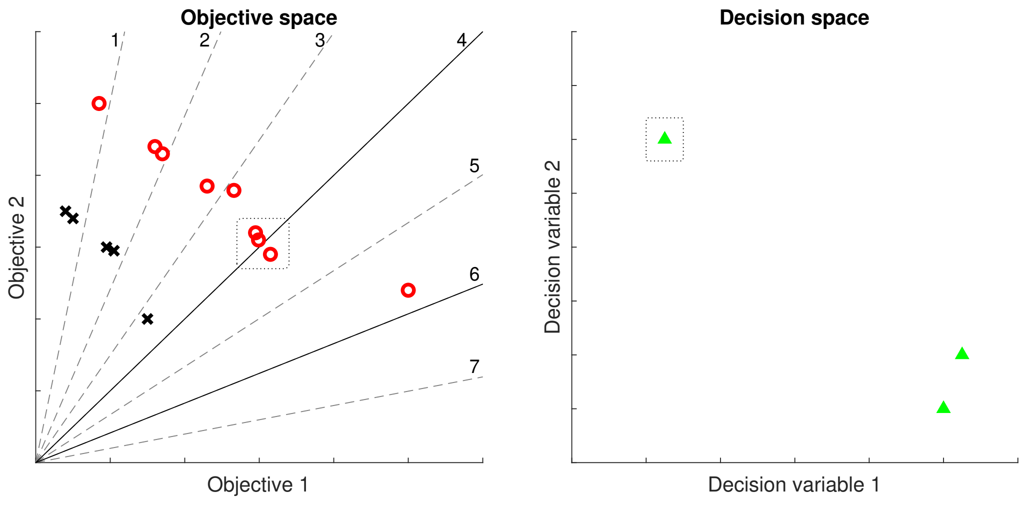


Figure 5.2: An illustrative example, for a bi-objective problem, of the integrated genotype diversity routine for choosing solutions from a truncated front.

Firstly, if a teacher is allocated four timeslot/room pairs for their four lectures, then the nominal ordering of these lectures is unimportant. The teacher will decide how best to spread the delivery of course material across the discontiguous time available to them. Secondly, other than capacity, rooms in the ITC2007 formulation have no secondary distinguishing features. Therefore, rooms of equal capacity can be considered equivalent and interchangeable.

In the following section, we propose a *standard form* conversion for the encoding, in which equivalent solutions have a single design. We also outline a set of distance metrics for use by Algorithm 7. These variants are empirically assessed to ascertain the best performing metric.

5.1.2.1 Metric entropy in equivalent solutions

To understand and quantify the theoretical weakness of Hamming distance as a metric on this solution encoding, a sample is taken according to the procedure in Algorithm 8. For each of `sampleSize` = 10,000 sample pairs, lines 4–7 select a random problem instance and generate an initial feasible solution, x, using the *saturation degree* heuristic. The function *equivalent*() renders a solution into some equivalent form. In this case, lectures within courses and rooms within *equal room sets* are both permuted at random. Line 9 selects an `editDist` between 0 and `maxMoves` = 100. `editDist` defines the number of single-lecture operations to be carried out in sequence. Inside the loop in lines 12–22, rejection sampling is employed so that only feasible solutions are accepted, thus the path traced from x to x^{new} is contained within a connected feasible subspace. No lecture can be operated on more than once. When all perturbations are complete, line 23 performs *equivalent*() on the new solution, x^{new}, before the distance metric ($Hamm$) is computed on the pair (x, x^{new}). Figure 5.3 shows the correlation between `editDist` and $Hamm$ for one such sample. The Pearson's coefficient is 0.434, implying a weak positive correlation. The estimated mean $Hamm$ value for an `editDist` of zero is positive (0.64) when ideally it too would be zero. $Hamm$ demonstrably cannot account for solution equivalence or near-equivalence when used on this encoding. For these experiments, it is therefore proposed that solutions are transformed into a *standard form.* The aim of this is to increase the utility of cheap pairwise distance metrics such as $Hamm$ by reducing the entropy.

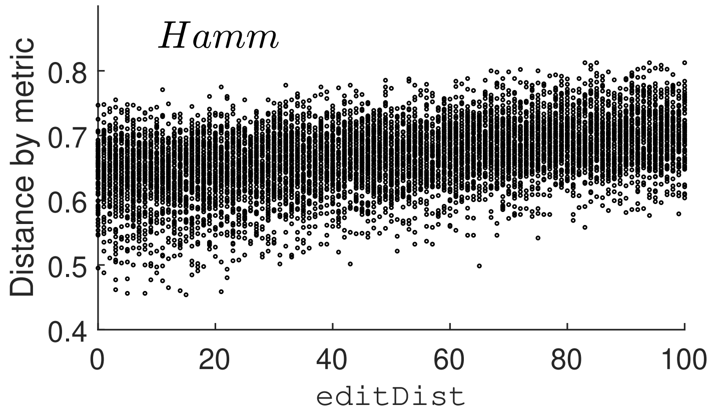


Figure 5.3: Distance metric $Hamm$ (Hamming) vs. `editDist` for a sample using Algorithm 8. Sample settings: `sampleSize` = 10,000, `maxMoves` = 100, *equivalent*() randomly shuffles lectures within courses and rooms within *equal room sets.*

5.1.2.2 Standard form encoding

Figure 5.4 provides a step-by-step exposition of the conversion of an offspring solution to the *standard form.* The example has also been chosen to highlight how a small perturbation can lead to a significantly different looking genome. Understanding this unavoidable quirk of the conversion process will inform the design of more bespoke, albeit expensive, distance metrics. The example problem is an artificially generated

Algorithm 8: Sampling procedure for pairs of solutions and their distances according to some chosen metric

```
1  Inputs: sampleSize, maxMoves, $equivalent()$
2  Output: Pairs of feasible solutions $(x, x^{new})$ and their distances
3  for 1:sampleSize do
4      PI $\sim U\{1, 21\}$    // Sample a problem instance number from the
        discrete uniform distribution on 1 to 21
5      repeat
6          $x \leftarrow SDinit($PI$, 1)$    // Initialize one solution using the
            saturation degree heuristic
7      until $x$ is feasible
8      $x \leftarrow equivalent(x)$
9      editDist $\sim U\{1,$maxMoves$\}$
10     lecsMoved $\leftarrow \varnothing$
11     $x^{temp} \leftarrow x$
12     for 1:editDist do
13         validMove $\leftarrow$ false
14         while validMove = false do
15             lecToMove $\sim U\{\mathcal{L} \setminus$ lecsMoved$\}$    // Choose a lecture to
                move, from the set of all lectures excluding those
                already moved, with uniform probability
16             $x^{new} \leftarrow reassign(x^{temp},$ lecToMove$)$
17             if $x^{new}$ is infeasible then
18                 Reject $x^{new}$
19             else
20                 validMove $\leftarrow$ true
21                 $x^{temp} \leftarrow x^{new}$
22                 lecsMoved $\leftarrow \{$lecsMoved $\cup$ lecToMove$\}$
23     $x^{new} \leftarrow equivalent(x^{new})$
24     $dist_i(x, x^{new})$    // Compute solution distance, where $dist_i$ is a
        user-defined distance metric (with nominal index $i$) such as
        $Hamm$
```

toy instance, Toy3R. Amongst its other characteristics, it has 3 rooms (capacities 50, 20, 20) and 5 courses (lecture counts 4, 3, 2, 2, 4).

Solution 1 is a starting feasible parent solution, already encoded in *standard form*.

Solution 2a) is a hypothetical offspring of **Solution 1**, having undergone an operation to feasibly reassign one lecture. Nominal lecture 1 has been moved from $\langle d, p, r\rangle = \langle 1, 4, 2\rangle$ to $\langle 5, 4, 3\rangle$.

Solution 2b) represents the first step in its conversion. Here, only courses of perturbed lectures need to be considered — in this case, course 1. This is another

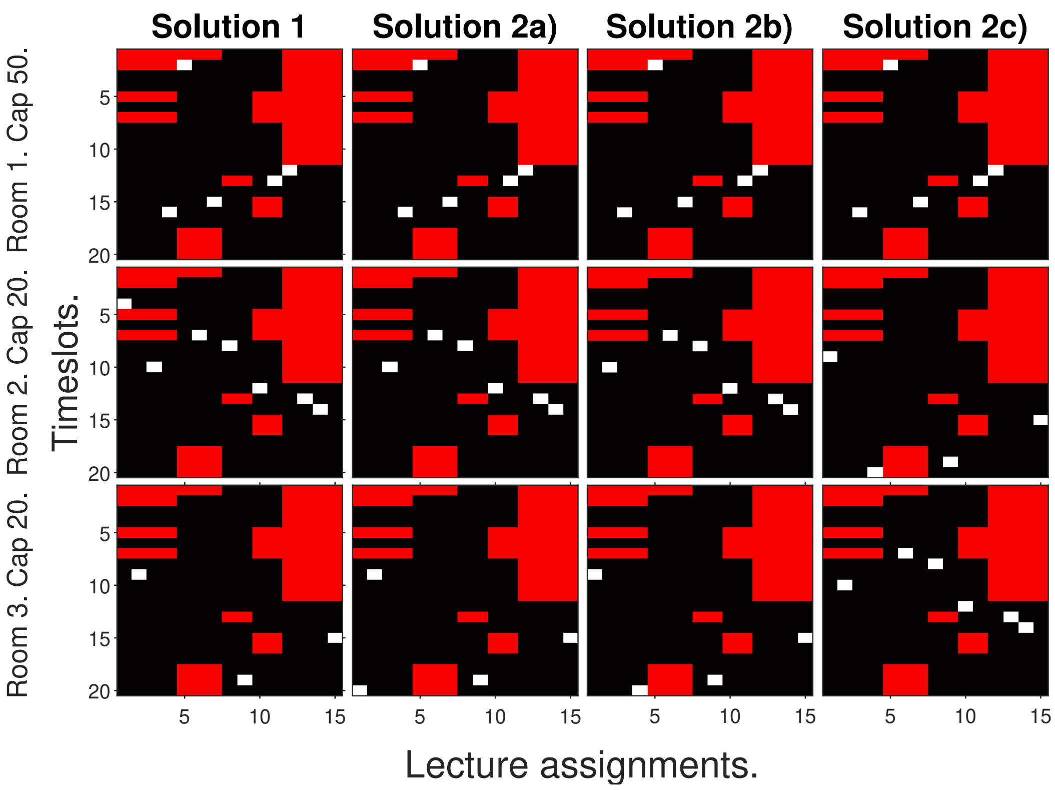


Figure 5.4: A visualisation of 4 solutions to the instance `Toy3R`. Each vertical grid of 3 plot windows is a single solution, with each of those windows representing a distinct room. Red shading shows *a priori* unavailable periods while black shows *a priori* available periods. White indicates the timeslot and room of each lecture assignment, as derived from its encoded tuple $\langle d, p, r \rangle$. Solutions 2a), 2b) and 2c) are all equivalent and have an `editDist` of 1 from Solution 1. Solutions 1 and 2c) are the only solutions in the proposed *standard form*.

δ-evaluation approach which helps to reduce compute time. All lectures in course 1 are re-sorted by timeslot, with the earliest temporal lecture taking the first nominal position.

Solution 2c). In the second and final step, the rooms are considered. As before, potential treatment only applies to the *equal room sets* affected by the initial perturbation. Here, *equal room set* $\{1\}$ has not been changed and so only *equal room set* $\{2, 3\}$ is handled. Within an *equal room set*, rooms are re-sorted according to the nominal assignment position of the lectures scheduled in them. For example, in **2b)**, room 3 hosts lecture 1. As this is a lower minimum nominal lecture than in room 2, room 3 is sorted ahead of room 2 in *standard form*. Any empty rooms found in this step are pushed to the end of the sort. Empty rooms are not explicit in a solution encoding and so the ordering of equally-sized empty rooms is immaterial.

In totality, the procedure is kept from being too computationally cumbersome by a combination of the (typically small) size of the *equal room sets*, the perturbation operator which only acts on a single lecture, and the aforementioned δ-evaluation approaches.

5.1.2.3 Distance metrics

Algorithm 8 demonstrates that, for a given input `editDist`, it is possible to efficiently sample a pair of solutions having that distance. The inverse of this function, however, can involve higher complexity. It may not be viable to obtain exact values of `editDist` for unseen pairs of solutions inside the optimiser loop. We therefore propose two surrogate functions, $SepM$ and $AggM$, as approximations to `editDist`. These metrics, along with traditional pairwise measures $Hamm$ and $City$, make up a total of four that are used to generate experimental results in this study. As the purpose of the metrics is for determining priority on solution selection, it is the relative values that are important rather than the absolute. A description of each of the four metrics, as applied to a pair of feasible *standard form* solutions a and b, is given below.

1. $Hamm$. The fraction of elements in a that differ from their pairwise counterparts in b.

2. $City$. The Cityblock (alternatively known as Manhattan) distance, which is given by: $\sum_{i=1}^{3|\mathcal{L}|} |a_i - b_i|$.

3. $SepM$. This metric uses estimates of the number of separate moves required between timeslots (`timeslotMoves`) and *equal room sets* (`equalRoomMoves`) to transform a into b, as set out in the following formulae:

$$\texttt{timeslotMoves} = \sum_{k=1}^{|\mathcal{C}|} |l \in C_k| - |(l \in C_k)_t^a \cap (l \in C_k)_t^b| \tag{5.1}$$

$$\texttt{equalRoomMoves} = \sum_{k=1}^{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{E}|} (|\mathbf{rs}_{ki}^a - \mathbf{rs}_{ki}^b|)/2 \tag{5.2}$$

$$SepM = \texttt{timeslotMoves} + \texttt{roomSetMoves} \tag{5.3}$$

Where $()_t$ denotes the assigned timeslots of a set of lectures. $\mathbf{rs}_k$ is a vector giving the distribution of assignments of course C_k lectures over the *equal room sets*. The second subscript i denotes its i^{th} element.

4. *AggM*. The final metric is based on aggregating lectures by course and rooms by *equal room set*.

$$\sum_{k=1}^{|\mathcal{C}|} \left[|l \in C_k| - \sum_{i=1}^{|\mathcal{E}|\times|t|} (\mathbf{trs}^a_{ki} \wedge \mathbf{trs}^b_{ki}) \right] \tag{5.4}$$

Where $\mathbf{trs}_k$ is a flattened Boolean vector of length $|\mathcal{E}| \times |t|$, indicating the *equal room sets* and timeslots in which assignments have been made, for course C_k. Its i^{th} element is denoted by the second subscript i, as before.

As a baseline to empirically compare against the performance of these four distance metrics, the base system from Chapter 4 — before inclusion of genotype crowding — is used. This variant is labelled *Base*.

5.1.3 Experimental setup

The experiment proceeds as follows: 24 repetitions were run (by varying the random seed) for each of the 21 benchmark problem instances and for each of the 4 distance metric variants plus baseline. Each repetition is independent and allocated to a single CPU core of a 12-core Ryzen9 with 32GB RAM, base clock speed 3.8GHz. The population size was fixed at 84, and each run had a budget of 1,000,000 function evaluations. The optimiser is written in Matlab and Java.

At every 50th generation across all runs, a snapshot is taken and the hypervolume of the current archive of non-dominated solutions is estimated using the Monte Carlo method with 1,000,000 samples. The reference points used to define the hypervolumes are retained from Table 4.7.

5.1.4 Results and analysis

The results section begins with an analysis of the properties of *Hamm*, *City*, *SepM* and *AggM*, and how closely each of these distance metrics was found to correlate with `editDist`.

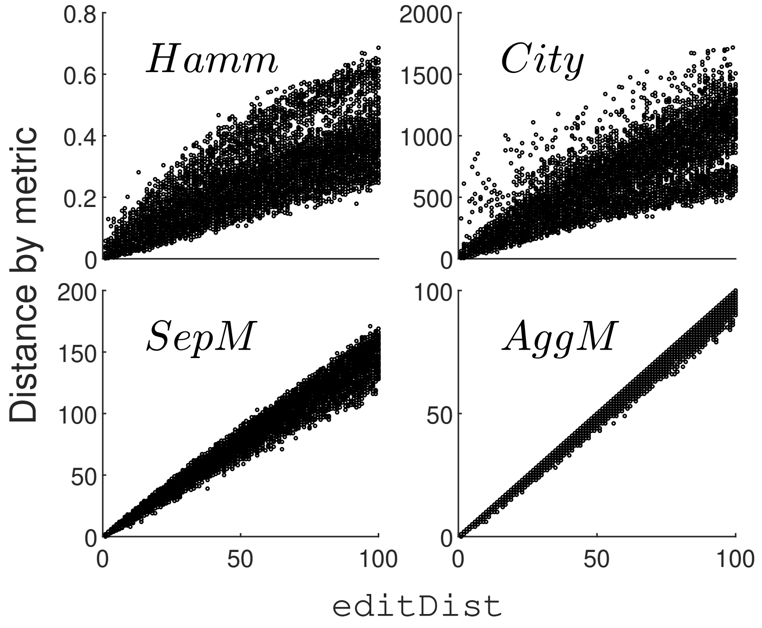


Figure 5.5: The four distance metrics vs. `editDist` for four samples using Algorithm 8. Sample settings: `sampleSize` = 10,000, `maxMoves` = 100, *equivalent*() converts a solution into *standard form*.

Figure 5.5 displays the results of four samples using the sampling procedure in Algorithm 8. Settings `sampleSize` and `maxMoves` are the same as for Figure 5.3, but *equivalent*() is used here to call the *standard form* conversion function. Pearson's correlation coefficients are 0.829 (*Hamm*), 0.857 (*City*), 0.986 (*SepM*), 0.998 (*AggM*). The effect of the *standard form* encoding scheme on *Hamm* is a significant strengthening of correlation, as highlighted by comparing Figures 5.3 and 5.5. Correlation of the other pairwise metric, *City*, is similarly strengthened by the use of *standard form* encoding, however sizeable variability remains. There is a lack of natural ordering for variables such as days or periods in the timetabling problem, meaning pairwise distance metrics that rely on absolute values can be deceptive.

Even for *Hamm* with *standard form*, variability can be an issue, as illustrated by the `Toy3R` solutions in Figure 5.4. Here, **Solution 2c)** only differs from **Solution 1** in the perturbation of one lecture, yet the pair have a relatively large *Hamm* distance of 0.33, due in no small part to the switching of rooms 2 and 3. One motivation behind the novel metrics was therefore to reduce the metric variance in solutions that are nearly equivalent. *SepM* and *AggM* both have this characteristic, exemplified by their high correlation coefficients. Out of the four, *AggM* proved to be the strongest overall surrogate for `editDist`.

In order to analyse results from the optimiser runs, an omnibus test — the Friedman test — was performed. The five variants were considered as distinct treatments and a blocking factor used over the repetitions of each problem instance. The Friedman test examines the null hypothesis, H_0, that the treatment effects are all the same, versus the alternative hypothesis that they are not, under minimal assumptions about the data. Repeat tests were carried out for the hypervolume data at each snapshot. Figure 5.6 shows the progression of the p-value over the 1,000,000 function evaluations of the runs.

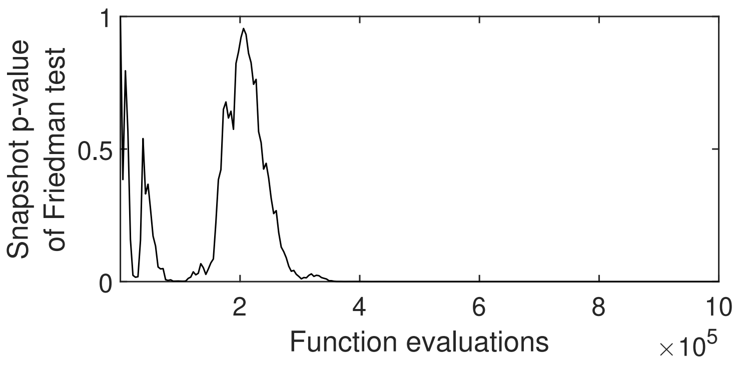


Figure 5.6: A trace of p-values obtained by independent Friedman tests at each snapshot of the runs. H_0: The variant effects are the same. H_1: The variant effects differ.

The plot indicates no significant difference between variants at the start of the run. Results remain mixed until around $2{\times}10^5$ evaluations. In this early stage of a run, it is likely that the search is predominantly exploratory. The number of plateau regions encountered by the optimiser here is low, resulting in fewer invocations. In addition, it is easier to find neighbourhood solutions that improve the current state of the Pareto front approximation early in a run. Because of this, even on the rare occasions that the genotype distance metrics are called, the advantages they confer early in a run may be less pronounced relative to later on.

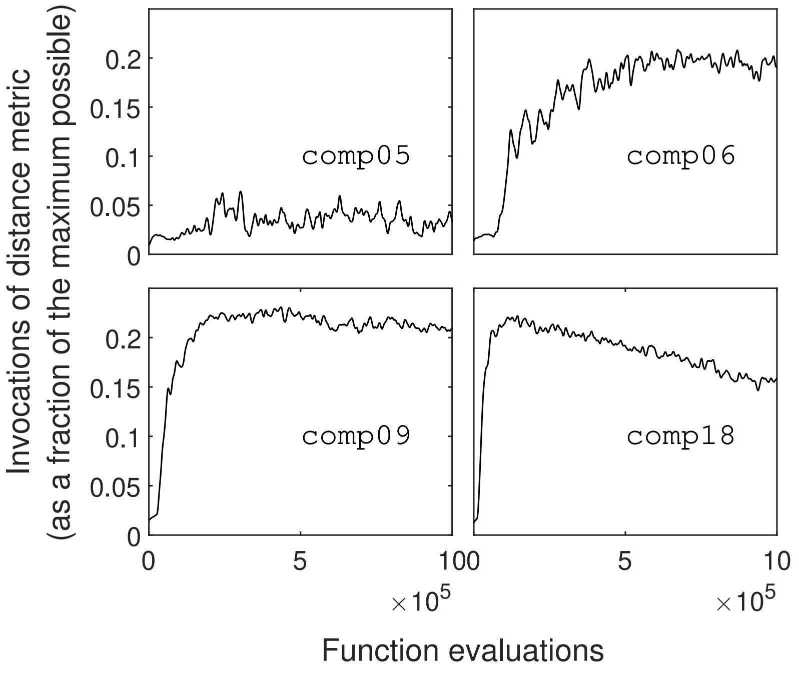


Figure 5.7: Normalised number of distance metric invocations vs. function evaluations for 4 benchmark problems, using *Hamm*. Data has been averaged over 24 repetitions and smoothed using a Gaussian-weighted moving average filter of window length 200.

To confirm these intuitions about the changing frequency of distance metric invocations over time, Figure 5.7 displays the relevant data for four representative problem instances using the *Hamm* variant. The plots show how frequently line 13 is called in Algorithm 7. Low numbers of invocations at the start are generally followed by a steep increase, as shown in the initial gradients of all plots barring `comp05`. Problem `comp05` has a particularly large and intractable search landscape, meaning the rate of early increase here, while still evident, is shallower. As the runs progress, the rate of invocations per time begins to slow down, as seen for `comp06`. Most problems follow this trend (suggesting that there are still gains to be made after the 1,000,000 function evaluation cut-off) or reach a peak before levelling off, as in `comp09`. Only one problem (`comp18`) displays a clear downwards trend after peaking. This may be explainable by the diminishing returns of trying to further evolve a population that has achieved a hypervolume closely approaching its theoretical maximum. As a note on the y-axis scale in Figure 5.7, the normalised maximum value of 1 can only be reached if all offspring solutions are part of the same leading non-dominated front as their parents and all solutions in the union set

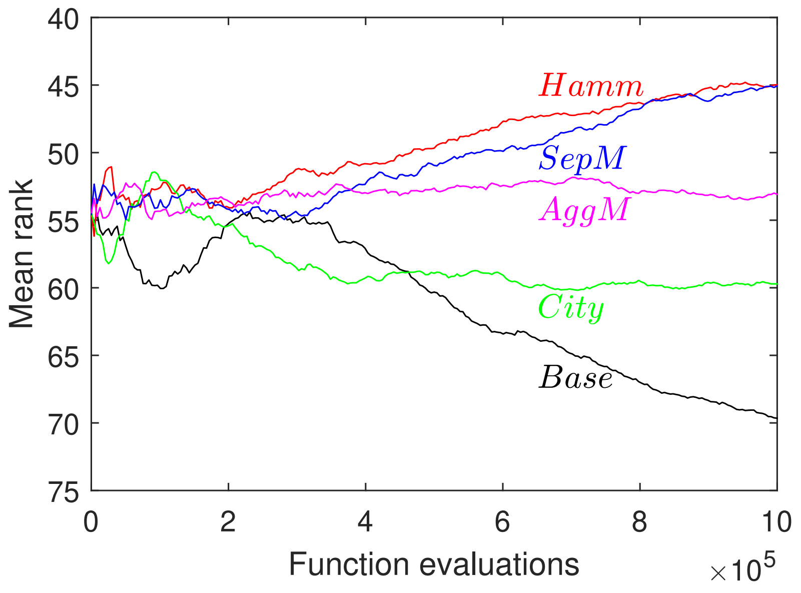


Figure 5.8: *Post hoc* analysis on the performance by hypervolume of individual variants using the Bonferroni method.

are associated with a single ray. This is an extremely unlikely scenario, hence the peaks occur at a substantially lower normalised value of around 0.2.

Figure 5.6 shows that in the latter part of the run, from around 3.5×10^5 onwards, there is consistent evidence to reject H_0 at the 5% significance level. *Post hoc* analysis is therefore conducted to investigate which of the individual variants may be outperforming the others. Figure 5.8 shows the mean ranks (by instance/repetition row) of the variants over the course of the run.

As the role of genotype crowding assumes greater prominence over time, there is a clear divergence in the performance of the variants. *Base* is outranked consistently almost from the start, while *Hamm* proves to be the highest achiever. *SepM* comes a close second, narrowing the gap on *Hamm* towards the end of the budget. Using the Bonferroni method to mitigate against family-wise errors, statistically significant difference is not found between any of the individual pairwise comparisons. However, the use of a more liberal test — Fisher's Least Squared Difference — returns significant differences in two cases: *Base* vs. *Hamm* and *Base* vs. *SepM*. The p-values at the final snapshot are 0.019 and 0.021 respectively.

Figure 5.9 provides a closer look at these two most promising variants, *Hamm* and *SepM*. The mean hypervolumes are plotted against function evaluations for

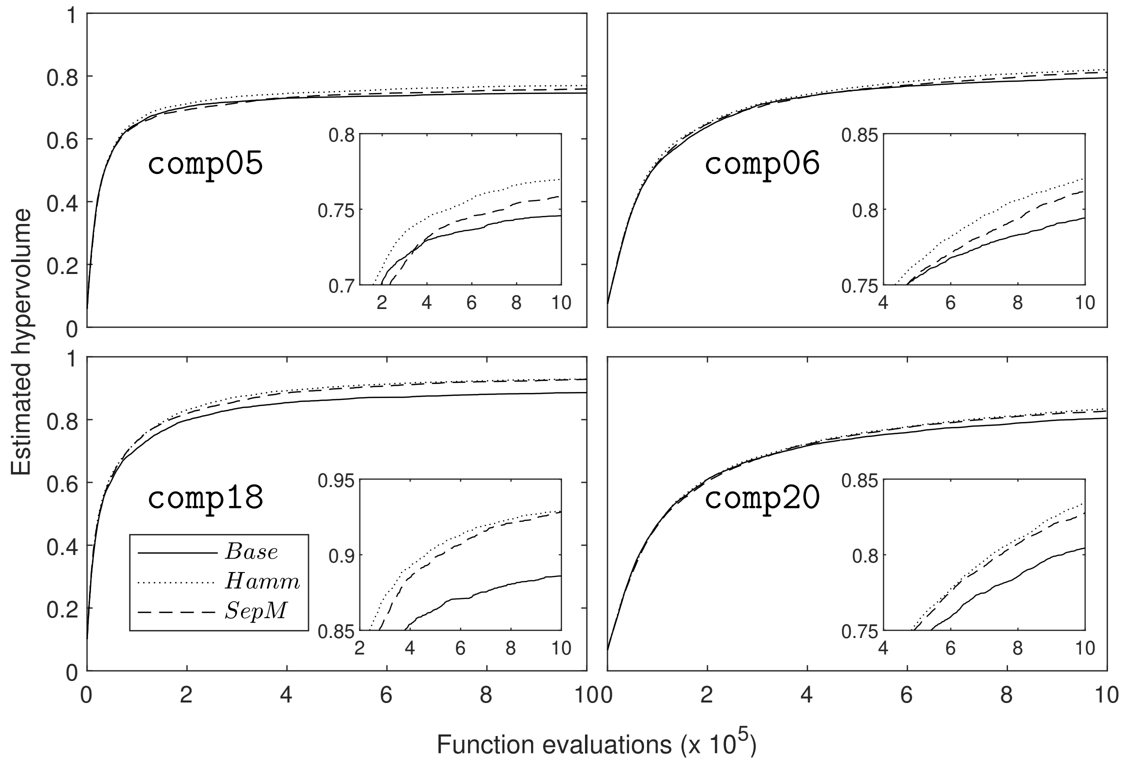


Figure 5.9: Performance of the two highest ranked variants (*Hamm* and *SepM*) and the baseline, *Base*, for four benchmark problems. Traces shown are mean averages of estimated hypervolume over 24 repetitions. Inset plots show cropped portions of the main plots.

four problems. While *SepM* is successful, note that it does not hold an any-time advantage over the baseline, as illustrated by the cross-over in the traces shown in the inset plot for `comp05` at around 350,000 evaluations. Both *Base* and *SepM* are any-time dominated by *Hamm* across the ranges shown in the inset plots. Both genotype crowding measures finish the runs with higher gradients than the baseline, suggesting they would continue to offer better economy if the run budgets were to be increased.

While hypervolume is the primary performance metric, some consideration may be given to time complexity too. Experiments were run to a function evaluation budget so that results could be judged independently of this. It is noted however that *Hamm* was the cheapest (non-baseline) variant to compute, and therefore dominated the others on both performance and speed.

Another interesting aspect to look at is the number of non-dominated solutions and designs found by the optimiser using the different variants. In respect of the (objective space) cardinality of the non-dominated solution set at termination, *Base* returned lower cardinality values on average than the tested variants. Table 5.1 gives

Table 5.1: For five variants, the mean sizes (over 24 repetitions) of the archives of non-dominated objective vectors at termination, rounded to the nearest integer. The largest raw averages are shaded.

Instance	*Base*	*Hamm*	*SepM*	*City*	*AggM*
`comp01`	11	16	11	11	11
`comp02`	23	54	50	49	30
`comp03`	17	46	39	42	26
`comp04`	16	36	36	30	25
`comp05`	291	371	399	334	339
`comp06`	17	69	62	53	38
`comp07`	18	59	50	53	37
`comp08`	14	30	27	24	19
`comp09`	26	94	91	66	44
`comp10`	18	39	39	39	26
`comp11`	2	2	2	3	2
`comp12`	421	390	390	442	300
`comp13`	20	66	64	50	43
`comp14`	20	52	38	50	33
`comp16`	15	55	50	38	28
`comp17`	22	70	69	63	30
`comp18`	45	110	99	105	90
`comp19`	28	49	33	42	30
`comp20`	105	257	240	230	228
`comp21`	20	68	58	51	28

the mean averages across the instances. $Hamm$ returned most of the highest average set sizes, and was significantly different from *Base*. The superior hypervolumes achieved by $Hamm$ suggest that the Pareto front approximations were closer to the true Pareto front, better spread, or both, compared to the baseline. It is perhaps the improved spread that gave rise to, on average, greater numbers of non-dominated solutions.

More counter-intuitively, perhaps, are the counts of unique designs mapping to these final non-dominated solutions, the mean values of which are given in Table 5.2. Here, the converse was true — counts for *Base* were the highest, and significantly higher than any of the other variants under pairwise statistical tests. The smaller number of non-dominated solutions in the approximation front for *Base* were found to have, on average, greater numbers of discovered designs mapping to them. One reason for this may be that *Base*, by definition, lacks a genotype-based mechanism to aid the escape from plateau regions. It can therefore spend more time idling through (and archiving) designs that do not offer a path to improved evaluations,

Table 5.2: For five variants, the mean number of discovered unique designs (over 24 repetitions) that cumulatively map to the objective vectors in the final non-dominated archives. The largest averages are shaded.

Instance	*Base*	*Hamm*	*SepM*	*City*	*AggM*
`comp01`	6,843	3,706	3,624	3,474	4,288
`comp02`	2,484	745	798	702	882
`comp03`	1,765	859	740	854	960
`comp04`	988	353	496	345	614
`comp05`	1,735	604	661	598	661
`comp06`	861	444	420	494	530
`comp07`	699	401	346	284	438
`comp08`	753	420	419	247	319
`comp09`	658	390	403	322	661
`comp10`	851	382	482	474	500
`comp11`	26,521	14,623	13,626	13,277	12,862
`comp12`	838	630	650	728	582
`comp13`	840	416	388	353	581
`comp14`	1,785	1,140	1,212	1,054	1,149
`comp16`	771	414	370	342	465
`comp17`	1,009	419	505	551	831
`comp18`	1,087	294	317	324	544
`comp19`	1,675	920	625	681	1,447
`comp20`	754	634	601	610	677
`comp21`	858	493	586	492	727

leading to the higher counts observed. It may also be the case that plateau regions of lower fitness are naturally larger than those of high fitness. With *Hamm* and other variants, however, the frequent discovery of new non-dominated solutions has the potential to remove one or more previously non-dominated solutions from the approximation front. Whenever this action occurs, tens, hundreds or even thousands of designs are also instantly subtracted from the count.

In relation to the wider hypothesis that the performance of a distance metric is proportional to its closeness of approximation to `editDist`, the experiments do not provide enough evidence to accept this. *AggM*, which was most highly correlated to `editDist`, was the third best performing variant, while *Hamm* delivered the best results despite having a lower correlation in this respect.

5.1.5 Conclusions on genotype diversity

In this section, the known many-to-one mapping between the genotype and phenotype space in the UCTP was scrutinised. In problems with this characteristic, an evolutionary search may have difficulty navigating plateau regions in the objective space. Using a many-objective optimiser, further adaptations were implemented in which genotype distance measures were used as part of the process for choosing solutions from a truncated non-dominated front. Two novel distance metrics were proposed and compared against two traditional metrics and a baseline. A novel *standard form* encoding conversion was also proposed in order to reduce the entropy of the traditional metrics. It was found that a genotype Hamming distance provided the best performance gains in this context, based on attained hypervolumes and controlling for cross-problem variation. A number of insights follow from this. $Hamm$ is a proportional measure of the number of index positions in two chromosomes in which the representational symbols differ. As each lecture assignment is represented by a 3-tuple, $Hamm$ is implicitly more granular than `editDist` (which counts any number of changes to the tuple as a single move). The notion of a shortest path through connected feasible space, as encapsulated by `editDist`, was found not to be as important as originally hypothesised. Therefore $AggM$ and $SepM$, both of which closely approximate `editDist`, do not necessarily confer advantages. Furthermore, $Hamm$ does not assume or impose any natural ordering on the variables where none exists, unlike the other variant $City$.

The *standard form* encoding was motivated by reducing entropy, increasing information capture, and ultimately making the distance metrics more effective. In addition, *standard form* reduced the risk of equivalent solutions emerging in the population, at an acceptable computational cost, while also offering practical benefits for the work on tournament selection that follows.

Further work may focus on alternative approaches for incorporating genotype distance into the system. Despite being essentially a fourth tie-breaking criterion in the non-dominated sort (after feasibility, dominance and phenotype crowding), the data on number of invocations (Figure 5.7) showed the role that it plays is significant and tends to increase over time.

5.2 Operators

In this section, some alternative operators are introduced to the system in order to investigate the benefits to performance versus the repercussions for run time. This is motivated by a desire to address premature convergence and homogeneity within the population. To this end, the new operators feature larger perturbations.

5.2.1 Destroy-repair

Premature convergence can result from becoming trapped in a local optimum, from which the original operators do not offer means of escape, or through other forms of stagnation. A destroy-repair operator is proposed for whenever this occurs. The first solution created by this operator is accepted — even if it is of lower quality. Two potential indicators of stagnation (which are equivalent to one another) could include:

1. No change to the non-dominated archive for `stagThresh` generations, where `stagThresh` is a parameter representing the stagnation threshold. This value may be determined adaptively or derived from other input parameters such as the budget.

2. No increase in the hypervolume of the non-dominated archive for `stagThresh` generations.

In practise, as hypervolume is expensive to calculate or estimate, any real time computations are only executed at intervals of g generations. Therefore criterion 2 is less precise as it can only indicate no improvement for some number of generations in the range g to $2g - 1$. The first criterion is therefore preferred for implementation purposes. At each generation, the non-dominated archive is inspected. It is not sufficient to check only the cardinality, as solutions may have exited and entered the archive in equal numbers, altering the hypervolume but not the archive size. Therefore, the cardinality and, if necessary, the set members are checked. A counter, `gensUnchanged`, is incremented if no change has been found since the previous inspection. Once `gensUnchanged` reaches the threshold, the destroy-repair procedure is triggered, as outlined in Algorithm 9.

Algorithm 9: Destroy-repair operator

```
1  Inputs: Current population, stagThresh, gensUnchanged, percDestroy
2  Output: New population
3  $n = \lceil percDestroy/100 \times |\mathcal{L}| \rceil$   // Compute the number of lectures to perturb
4  if gensUnchanged = stagThresh then
5      forall $x \in$ Current population do
6          if $x$ is feasible then
7              $l_1, l_2, \ldots, l_n \overset{\text{iid}}{\sim} U\{1, |\mathcal{L}|\}$
8              Unassign lectures $l_1, l_2, \ldots, l_n$ in $x$
9              Reassign lectures $l_1, l_2, \ldots, l_n$ in $x$ using SD heuristic
10     gensUnchanged ← 0
11 New population ← Current population
```

To determine an order of magnitude for `stagThresh`, diagnostic data from previous optimiser runs is used. Figure 5.10 shows how the counter variable `gensUnchanged` changes with generation number, for four different problem runs.

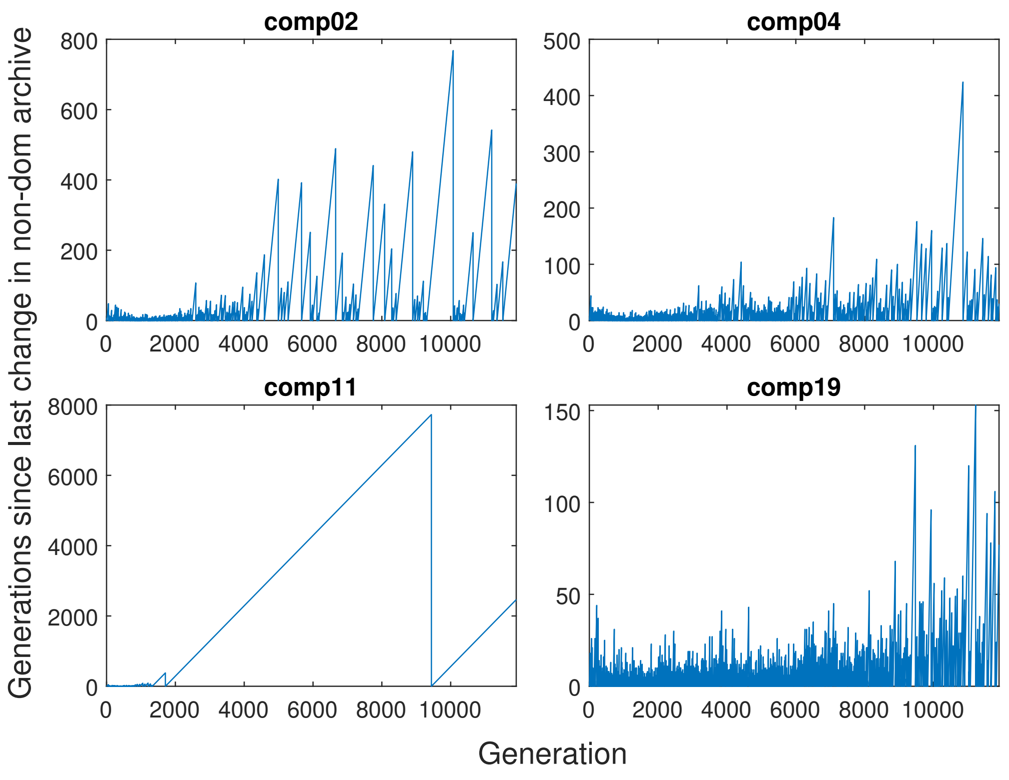


Figure 5.10: The number of generations since the last update to the non-dominated archive, for four example runs on problems `comp02`, `comp04`, `comp11` and `comp19`.

Table 5.3: A small grid search on destroy-repair parameters `stagThresh` and `percDestroy` for `comp04`. Feasibility percentages are mean values attained from 10 repetitions, where the first execution of the operator is relied upon to generate the results.

`stagThresh`	500	500	500	500	500
`percDestroy`	10	20	30	40	50
Proportion of feasible solutions in new population	0.18	0.04	0.00	0.00	0.00

In all cases, the greatest peaks occur in the latter stages of a run. The heights of these peaks are problem-dependent, with the most tractable instance `comp11` showing no improvement for as many as ∼8,000 generations. Based on this information, `comp04` was chosen for experimentation and an *ad hoc* value of `stagThresh` = 500 used. Further tests were then conducted on this instance in which the second parameter `percDestroy` — controlling the percentage of lectures to be operated on — was varied between 10 and 50. Table 5.3 shows the effects of `percDestroy` on the feasibility of the new population.

Note that the placements of the fixed lecture assignments in these tests have not been determined by a direct heuristic, but through many iterations of the evolutionary algorithm itself. SD is therefore limited in the feasible places it can reassign the 'destroyed' lectures to, as highly evolved populations can be more sensitive to the effects of reassignment. The upshot is that, even operating on as few as 20% of the lectures in `comp04`, almost none of the resultant solutions are feasible, as rates drop away rapidly. Setting a much higher value for `percDestroy` (approaching 100) would allow SD a blanker canvas on which to construct new solutions, but at the cost of discarding most of the progress made by the optimiser — more akin to a wholesale restart. A prudent value should be large enough to reinvigorate the search while not introducing excessive infeasibility.

In a second batch of tests, focusing only on `percDestory`, values of this parameter were varied across a narrower range of 0% – 20%. All instances barring `comp15` were tested. It is hypothesised that the more mature (i.e. evolved) a population is, the less adept the destroy-repair operator will be at returning feasible solutions. Using a set of 1,000,000 function evaluation budget runs, snapshot populations were extracted from the first and last generations. The repair-destroy procedure was applied to each, over 30 repetitions, with the mean results plotted in Figure 5.11.

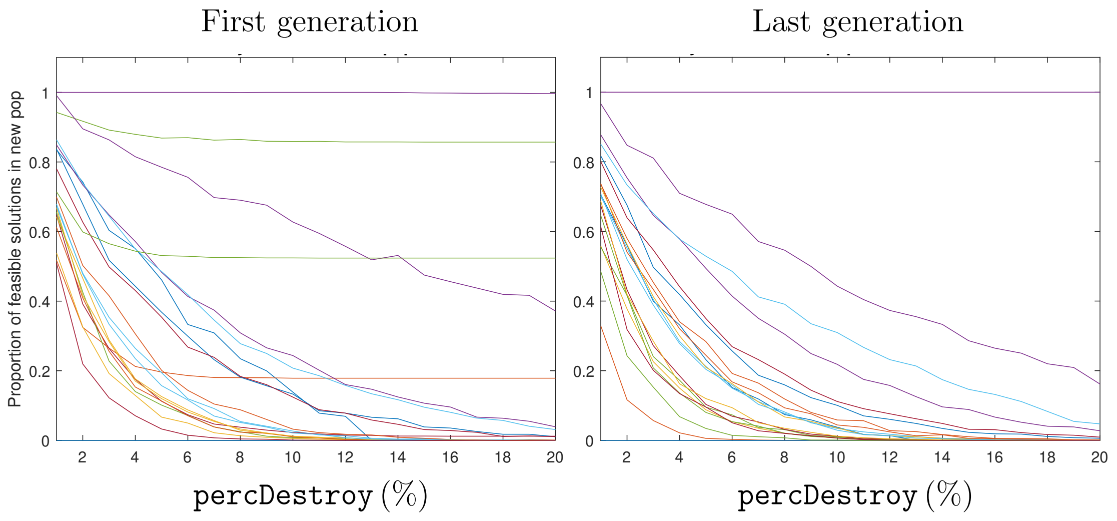


Figure 5.11: Test results from snapshot populations extracted from first generations (left plot window) and last generations (right plot window) of 1,000,000 function evaluation budget runs. Across 20 instances `comp01`, ..., `comp21` (shown in different colours and omitting `comp15`), parameter `percDestroy` was varied and the feasibility of the new population recorded. Traces show the mean values from 30 repetitions.

In both initial and pre-optimised populations, the `comp11` trace — shown as a horizontal purple line — is invariant to changes in `percDestroy`. In the left plot only, traces for instances `comp02`, `comp05` and `comp19` are also relatively unchanging from `percDestroy` $= 6$ onwards — levelling out at proportions of 0.18, 0.86 and 0.52 respectively. All three of these show significantly diminished returns in the right plot. It is notable and perhaps not coincidental that these instances are some of the hardest in the benchmark according to complexity metrics suggested in Rosa-Rivera et al., 2021. For the majority of the problems, the optimisation state of the population used does not effect significant differences in the traces as shown.

Including infeasible solutions under the umbrella term ‘worsening moves’ and allowing their temporary acceptance is problematic for a few reasons. Tournament selection ensures that such solutions will be bred out of the population fairly quickly, meaning that any beneficial traction is lost. Alternatively, feasibility repair functions may be required, which are considered overly expensive. Should such a function be invoked, it is also not clear whether this would simply walk the solution back to its original pre-perturbation state, thus rendering the computation redundant. The graphs in Figure 5.11 confirm that, even when perturbations are limited to 1% of the lectures (2.8 lectures as a benchmark average), the benchmark average feasibility rate in the latter stages of a run is only 0.68. Moreover, the two parameters `stagThresh` and `percDestroy` are shown to be highly sensitive and problem-dependent. For these reasons, the destroy-repair operator, as described, was not adopted for the optimiser.

5.2.2 Perturbation

In this section, further perturbation operators are analysed in the context of feasibility preservation. Three initial operators are defined as follows:

`swapPlace`: A pair of lectures are chosen at random and their places (i.e. room, day and timeslot) are swapped. A tabu matrix, size $|\mathcal{L}| \times |\mathcal{L}|$, is maintained so that pairs of lectures from the same course cannot be selected and neither can any pairs previously found to result in infeasible swaps.

`tsSwitch`: A pair of timeslots are chosen at random and all lectures in the first timeslot are switched with all lectures in the second.

`tsCirc`: A pair of timeslots t_i and t_j are chosen at random, where $i < j$. A circular shift is performed on the timeslot variable of all lectures originally assigned to t_i, t_j and all ordinal timeslots in between. The direction of the circular shift, up or down, is determined at random with equal probability.

`swapPlace` and `tsCirc` are adaptations of operators from Abdullah and Turabieh, 2012, while `tsSwitch` corresponds to operator **Nbs3** from the same authors. The operators are parameterised by a variable labelled `attempts`. This controls the number of attempts that will be made to return a feasible solution, before the operator defaults to `MuPFPR`.

In an initial batch of tests, populations from prior runs were once again extracted from both the first and last generations, for all instances barring `comp15`. The parameter `attempts` was varied from 1 to 100 and the three operators applied. Over 24 repetitions, the number of perturbed lectures was recorded. Defaulting was switched off, so that if a feasible solution was not obtained within the threshold attempts, a value of 0 was recorded. Figure 5.12 shows traces for the means.

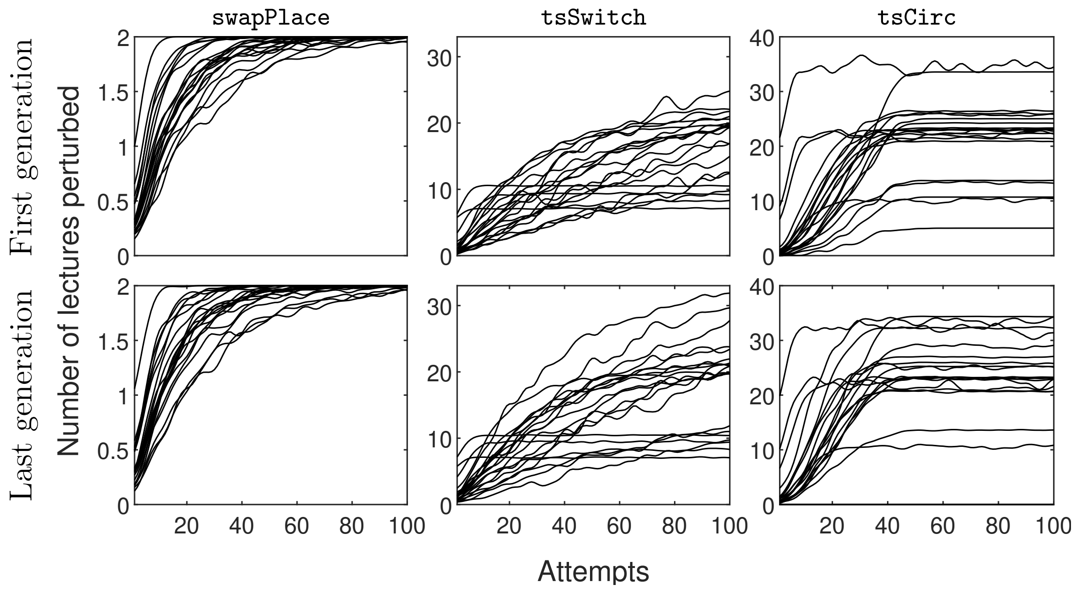


Figure 5.12: Test results from snapshot populations extracted from first generations (top row) and last generations (bottom row) of 1,000,000 function evaluation budget runs. Across instances `comp01`, ..., `comp21` (shown as black lines, and omitting `comp15`), parameter `attempts` was varied and the number of lectures perturbed was recorded. Traces show the mean values from 24 repetitions and have been smoothed using a Gaussian filter.

The key observations are:

1. Increasing the value of `attempts` tends to increase the number of lectures that are moved. This reflects the fact that, on average, a larger number of individuals are perturbed feasibly due to greater opportunity.

2. As the time complexity of `attempts` is linear, the best economy is represented by the steepest gradients in Figure 5.12. These are generally found in the range 1–30, after which diminishing returns are evident. 30 could therefore be considered a reasonable *ad hoc* value for `attempts`.

3. The evolutionary maturity of the populations did not impact significantly upon results. On average, the numbers of lectures perturbed were similar whether the populations in question were initial (the top row of plots) or pre-optimised (the bottom row). The operator least affected by this factor was `swapPlace`. Due to this consistency throughout the run, and for its relatively high feasibility returns within a limited number of attempts, `swapPlace` is focused on exclusively for the remainder of this section.

One method to increase the chances of feasible returns from `swapPlace` is to embed some degree of hard constraint checking into the tabu matrix. Doing so would mean that fewer attempts are expended on case-by-case feasibility checking. Figure 5.13 illustrates how the search space of pairs contracts as each hard constraint type is forbidden in turn. An initial solution is generated for `comp07`. Figure 5.13a) shows the space induced by the original tabu matrix, in which swaps are only prohibited between lectures in the same course. The constraint violation counts due to **H1** and **H2** are invariant with `swapPlace`. The remaining hard constraints are sorted according to time complexity. Figure 5.13b) shows the induced space when the least complex of these, **H4**, is prohibited. In Figures 5.13c) and 5.13d), violating pairs are prohibited for **H5** and **H3** respectively. The last figure gives a good insight into the sparsity of feasible swaps available even for an initial solution. Run time analysis on a wider set of problems showed, however, that pre-filtering **H5** and **H3** violations was too CPU-intensive. Therefore, only **H4** was incorporated into the final tabu matrix, with other violations handled case by case.

While an increase in the proportion of returned feasible solutions is welcome, some insight is needed around how likely these are to be improvements. Therefore, another batch of tests was conducted with a focus this time on the soft constraint violations. Parameter `attempts` was set to 100 in order to ensure a near-total feasibility rate and to generate a usable quantity of data. For all instances, populations (size 84) were extracted from previous runs at a series of progress points ranging from generation 1 to generation 4000. Operator `swapPlace` was applied to all individuals over 24 repetitions. The scalarised objective score was used as a cheap indicative measure of quality, and the net improvement or deterioration recorded for each perturbed solution. Figure 5.14 illustrates the results for `comp20`.

Improving moves are more abundant at the start, which is followed by a deterioration, the rate of which slows over the generations. The median is below zero at all times, and the upper quartile is mostly below zero, which was typical across all problems. Improving moves remain available in the upper tail. In the next

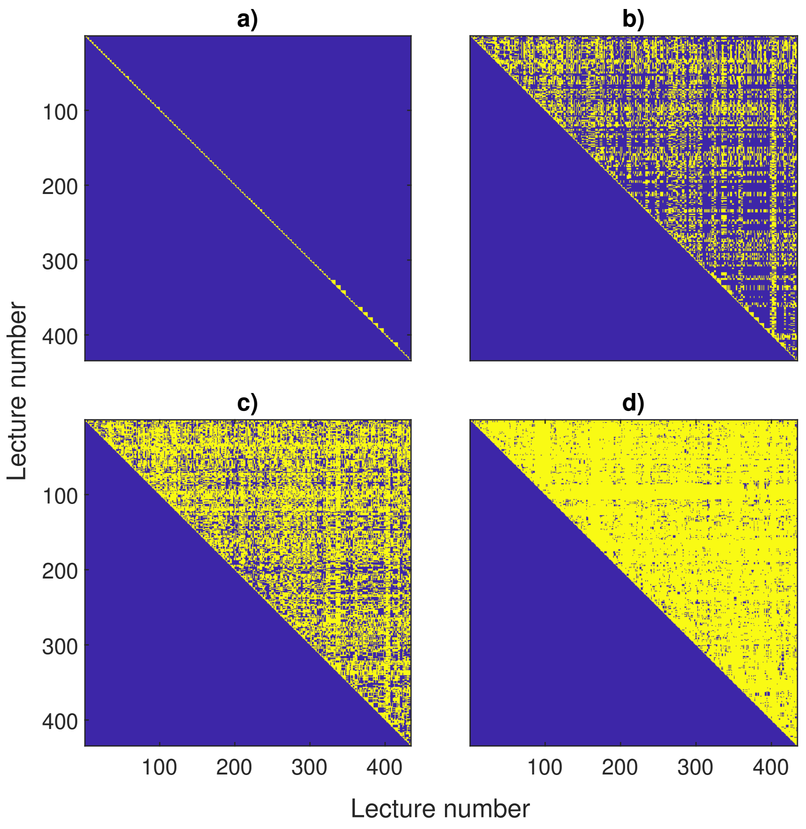


Figure 5.13: Matrix visualisation of pairs of lectures that can be selected by the `swapPlace` operator, for an initial `comp07` solution, over four tabu scenarios. Yellow shading indicates prohibited pairs (upper triangle only). Incremental prohibition scenarios shown are a) pairs in same course, b) pairs that would violate **H4** (unavailable periods), c) pairs that would violate **H5** (teacher clashes), d) pairs that would violate **H3** (curriculum clashes).

investigation, consideration is given to ways in which the distribution can be skewed towards these better moves.

The `swapPlace` operator already includes rudimentary soft constraint-based guidance, on account of common course swaps being prohibited. There are other conditions on a lecture pair that guarantee a redundant swap, however, such as when all of the following are true:

For **S1**, if lectures are assigned to rooms in the same *equal room set*.

For **S2**, if lectures are assigned in the same day.

For **S3**, if the curricula membership of the lectures is identical.

For **S4**, if lectures are assigned in the same room.

The last of these is a stronger condition than the **S1** condition, and therefore subsumes it. An analysis of lecture pairs was carried out to ascertain the likelihood of the three remaining conditions being met. In a sample of 1,000 initialised solutions,

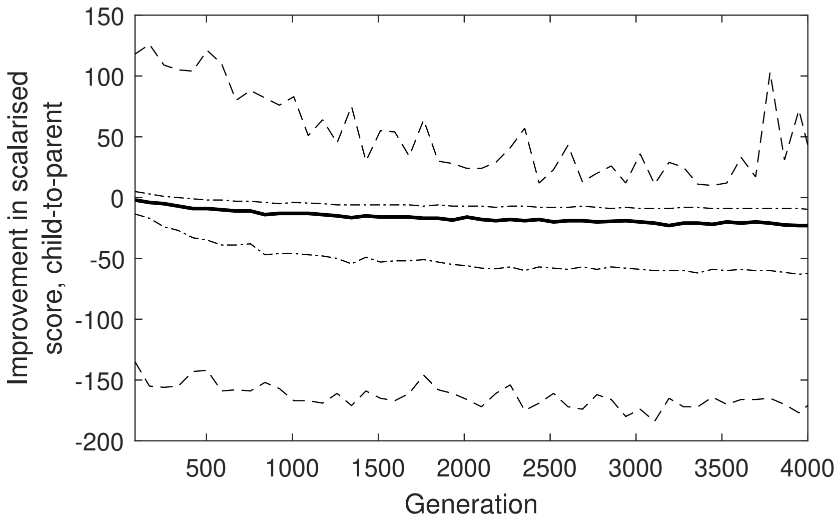


Figure 5.14: Previously run populations for `comp20` from generations 1 to 4000 vs. improvement (by scalarised score) due to the application of `swapPlace`. Median (solid line), lower and upper quartiles (dash-dotted lines) and maximum/minimum (dashed lines) are shown for data from 24 repetitions. Positive values on the y-axis indicate improvement.

the modal number of pairs meeting all conditions for redundancy was 1 for `toy3R`. This toy problem has $\frac{15\times14}{2} = 105$ lecture pairs in total, making the mode less than 1%. For the ITC2007 problem `comp07`, the mode was even lower, at 0. The implication is that these conditions are satisfied together so rarely in practise as to be of little use.

A complementary approach was therefore explored. By prioritising pairs that are in distinct rooms, distinct days and have no curricula in common, would `swapPlace` become more powerful in terms of discovering improving moves? Algorithm 10 outlines how these priorities are embedded into the `swapPlace` operator, where $()_d$, $()_p$, $()_r$ denote the day, period and room that a particular lecture has been assigned to, and $()_{cur}$ is its curricula set. The for-loop populates the tabu matrix with regard to common course lectures (line 7) and **H4** violations (line 9). A promising pair is sampled at random in line 14, before feasibility and priority checks are carried out in line 15. These look at violations of the remaining hard constraints **H5** and **H3**, as well as distinct room, day and curricula properties. Short circuit logic is used to streamline the checks.

Across all problems and different stages of optimisation, pairs satisfying the priority conditions are generally easy to locate. With high probability such a pair

Algorithm 10: `swapPlace` with embedded dissimilarity-based guidance.

```
1  Inputs: $x$ (a solution), attempts
2  Output: lecIdx (lectures to operate on.)
3  Initialise $|\mathcal{L}| \times |\mathcal{L}|$ tabuMatrix with zeroes.
4  lecIdx $\leftarrow \varnothing$
5  for $i = 1 : |\mathcal{L}| - 1$ do
6      for $j = i + 1 : |\mathcal{L}|$ do
7          if $l_i, l_j \in C_a$ then
8              tabuMatrix$(i, j) \leftarrow 1$
9          else if $[(l_i)_p \in unav(C_a)$ & $l_j \in C_a]$ OR $[(l_j)_p \in unav(C_a)$ & $l_i \in C_a]$
            then
10             tabuMatrix(i,j) $\leftarrow 1$
11 pairFound $\leftarrow 0$
12 attempted $\leftarrow 0$
13 while pairFound $= 0$ & attempted $<$ attempts do
14     Randomly sample $i, j$ where $i, j \in \{1, |\mathcal{L}|\}, i > j,$ tabuMatrix$(i, j) = 0$.
15     if swap is feasible & $(l_i)_r \neq (l_j)_r$ & $(l_i)_d \neq (l_j)_d$ & $(l_i)_{cur} \cap (l_j)_{cur} = \varnothing$
        then
16         pairFound $= 1$
17         lecIdx $= [i\ j]$
18     else
19         tabuMatrix$(i, j) \leftarrow 1$
20         attempted $\leftarrow$ attempted $+ 1$
21 if lecIdx $= \varnothing$ then
22     Default to MuPFPR operator.
```

can be found within 9 samples, and the modal number of samples required was 1. Note that not all of these guarantee a feasible swap, however, and a cost of one attempt is incurred (line 20) for every failure.

Default operator `MuPFPR` and `swapPlace` differ both in perturbation size and computational complexity. In the interests of balancing the exploration/exploitation trade-off, as well as regulating the run time, it is proposed to call `swapPlace` more frequently in the early stages of a run, after which `MuPFPR` begins to predominate. The probability of `swapPlace` being called is defined by an exponential decay profile $exp[-10 \times \texttt{genID}/(\texttt{budgetFE}/\texttt{popSize}))]$, where `genID` is the current generation number and `budgetFE` is the function evaluation budget. Initial tests suggested this approach lead to improved hypervolumes, on average, over a majority of problems when compared with exclusive use of `MuPFPR`. Experiments are laid out in the next section in order to formally test this.

5.2.2.1 Experiments on enhanced `swapPlace`

As the culmination of the development of perturbation operator `swapPlace` in this section, an experiment was carried out using four variants defined as follows:

Defa: Only the default (`MuPFPR`) operator is used, as a baseline.

Naiv: Exponential decay profile `swapPlace`. A naive implementation of the operator with no tabu matrix or pair priority.

Uvio: Exponential decay profile `swapPlace`. The tabu matrix filters out common course lectures and **H4** violations.

Upre: Exponential decay profile `swapPlace`. The tabu matrix filters out common course lectures and **H4** violatons. A scheme for preferring certain pairs based on dissimilarity of properties is used, as per Algorithm 10.

Six instances were solved by way of each variant. Experimental settings were `attempts` = 30, repetitions = 24, function evaluations = 1,000,000 (equivalent to 11,904 generations with `popSize` = 84.) Figure 5.15 shows the progression of the mean hypervolumes over the entire runs, with inset windows providing greater detail on the traces as they approach termination. Figure 5.16 meanwhile shows the results of significance testing on the data, for the two most informative variant comparisons: *Uvio* vs. *Upre* and *Uvio* vs. *Defa*.

Analysis of the data reveals a number of key insights. Inclusion of `swapPlace` (in any capacity) resulted in shallower initial gradients of the hypervolume trace. This reflects a greater focus on early exploration of the search space. Delayed benefits are apparent, as the gradients of these variants steepen in the later stages. Variant *Uvio* was the best performer, on average, by termination in four of the six instances (and second best in the remaining two). This suggests that early use of `swapPlace` helps navigate the search to regions from which the exploitative power of `MuPFPR` then has a greater impact. *Upre* did not appear to confer an advantage compared to *Uvio*, as it came last, on average, at termination in half the instances. In fact, the statistical tests visualised in Figure 5.16 indicate the opposite — that *Uvio* performed significantly better than *Upre* in at least two of the problems. Conceptually, prohibiting swaps known not to affect the objectives increases the

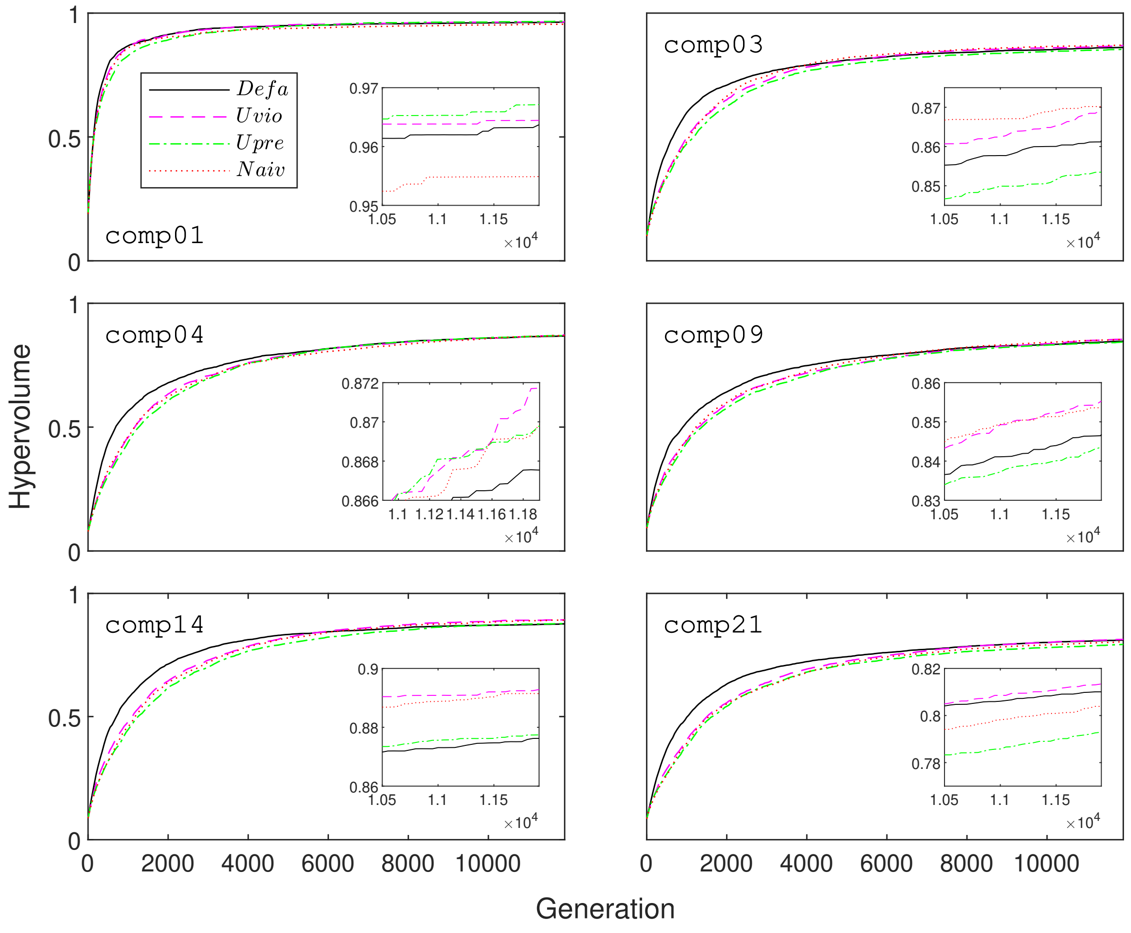


Figure 5.15: Mean hypervolume improvement for four variants *Defa*, *Uvio*, *Upre* and *Naiv* on a set of six instances.

chances of change, though not necessarily in a net positive direction. Therefore, other methods of filtering pairs based on their potential contributions to soft constraint violation counts could be considered to improve *Upre*. In the preliminary work, it was found that filtering out all hard constraints was cost-prohibitive. The experiments showed that, even relatively cheap filtering of **H4** violations added enough time complexity such that the practical needs of the end user may have to be carefully considered in order to justify it. The exponential decay profile played a part in curbing run time (compared to exclusive use of `swapPlace`), while also serving the exploration/exploitation trade-off well. However, more tests are needed to determine the optimal profile for this curve and to what extent this may be influenced by the instance characteristics.

Having considered perturbation operators in this section, the next section looks at the tournament selection operator. The concept of symmetry within solutions,

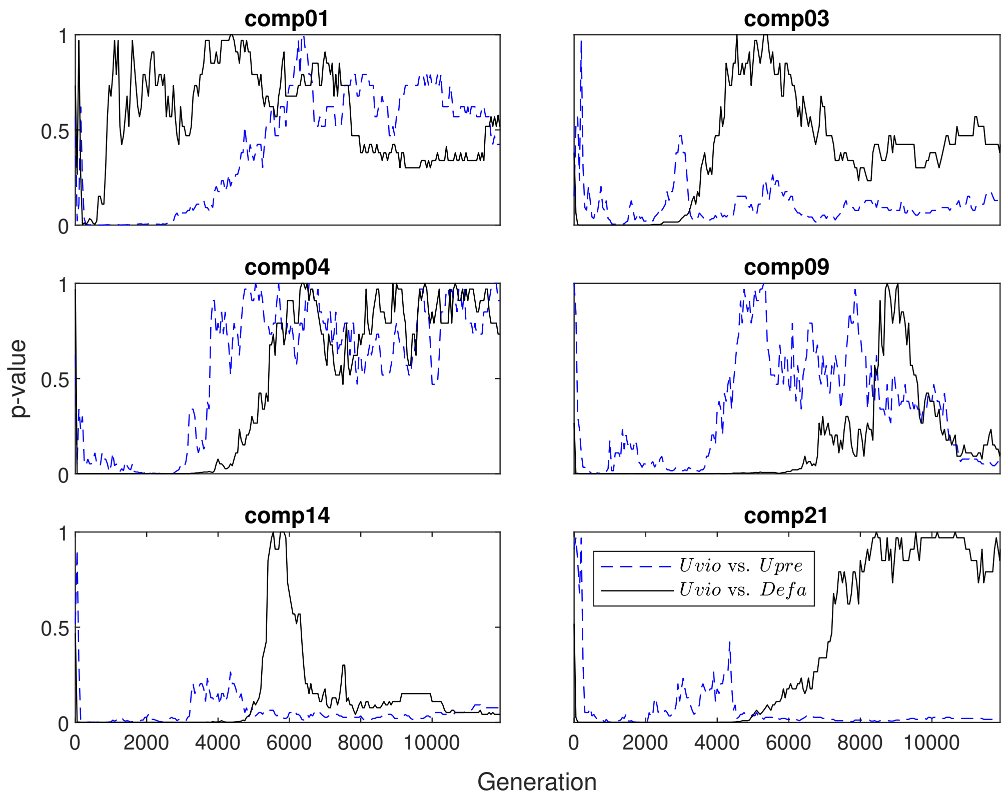


Figure 5.16: Wilcoxon signed rank tests comparing *Uvio* with *Upre* (blue) and *Uvio* with *Defa* (black) for six problems. In all cases, lines show the progression of the p-value over increments of the entire run.

leading to equivalence, as discussed in Chapter 5, is once again invoked. The *standard form* encoding proposed in that chapter is leveraged by an enhanced tournament selection routine.

5.2.3 Tournament selection

One consequence of the confluence of small perturbation operators and standard tournament selection is potential homogeneity of the population. Tournament selection plays a key role in regulating this diversity. In this section, an enhancement is proposed in order to help widen the search. The motivation is to discourage the acceptance of offspring that are equivalent either to their parent, or to some other parent in the population. The *standard form* encoding provides a useful way to do this.

The idea could potentially be taken further, by discriminating against any offspring that are equivalent or identical to any encountered over the entire search history. Some methods by which to achieve this include archiving the complete search history and utilising look-up or hash tables. Both the design vectors and active search history in this problem domain can be large however, and such an approach may have a significant impact on the memory footprint. Moreover, the cost of look-ups may not be justifiable beyond a certain threshold in the history, as new designs appear that are increasingly dissimilar to older ones. Recording the 'age' of each solution is another discriminatory measure by which freshly discovered designs can be given preference (Schmidt and Lipson, 2010). However, it is possible for a solution x to be converted into itself via a cycle of moves, especially if its immediate neighbourhoods all lie on a plateau, meaning this scheme is not without issues either.

The tournament selection used in the optimiser is deterministic 2-way without replacement. This permits a maximum of one identical copy of any given solution in the mating pool. Promoting identical copies of solutions that are, by implication, high quality, is not problematic by itself, as the perturbation stage can reintroduce divergence. However, it is desirable to avoid a proliferation of equivalent solutions as this is anti-exploratory and an inefficient use of compute time.

A property, *replica*, is proposed, as follows: All offspring created in generation $i - 1$ are represented using the *standard form* encoding. The active population in generation i is concatenated with the active population from the previous generation. This set therefore contains $2 \times$ `popSize` designs. The property *replica* of each member is `false` if it is unique within this set, and `true` otherwise. When comparing on the *replica* property, a solution with a `false` value out-ranks one with a `true`. Note that this approach treats solutions that have been successfully carried forward from generation $i - 1$ to generation i as 'replicas' — thereby incorporating an element of 'discrimination by age' too. Note also that no special status or protection is afforded to the 'original' solution in a pair of equivalent or identical solutions. This is justified by the following: If the population of a generation i is largely unchanged from that of generation $i - 1$ then the *replica* property is mostly redundant as a tie-breaker. If the populations differ, this is because new non-dominated solutions have been admitted. Uniqueness in this case is a sign of high quality and thus the favouring of these solutions is encouraged. At the same time, high quality solutions whose *replica* property is `true` will not necessarily be lost as, in the case of an equivalent pair, there is an empirically high chance of one of the two being paired with another *replica* = `true` solution, where *replica* again becomes redundant as a tie-breaker. In this sense, the stochasticity of the tournament selection operator serves to mitigate the issue of discarding good solutions.

5.2.3.1 Experiments on tournament selection

When conducting a tournament, the selection operator compares solutions according to a hierarchy of properties. The first of these is outright feasibility. For the purposes of the experiments in this section, property *replica* is positioned immediately underneath this as the second most important criterion. This variant is denoted *Wrep*, and it is tested against the baseline tournament selection variant, *Tsel*, which omits the *replica* property. Each benchmark problem is run for 24 independent repetitions, with a function evaluation budget of 1,000,000. The primary performance metric is the hypervolume of the non-dominated solution set at termination. Another metric used to provide insight is the number of designs that are unique within the active population history. This is denoted as $|\mathcal{H}|$, where $\mathcal{H}$ is the union of all active populations from first to last generation. A third metric is the cardinality of the final non-dominated archive.

Table 5.4 gives the results in terms of these three metrics. $|\mathcal{H}|$ is essentially a measure of churn in the active population, and results show that *Wrep* increased this quantity, on average, in 18 of the 21 problems tested. In terms of hypervolume, the results were more mixed, with *Wrep* leading to increases, on average, in 14 of the 21 problems. This was the same number as for the metric $|\mathcal{A}_o|$, although these two sets differed by which instances they contained.

In Figure 5.17, the percentage increase in $|\mathcal{H}|$ between *Tsel* and *Wrep* has been calculated and the distributions across repetitions plotted. Most commonly, *Wrep* is seen to increase the churn by between 5–10%, with highly complex problem `comp05` the most striking exception. A greater number of unique designs being accepted into active populations may suggest that higher quality timetables were being found, and

Table 5.4: A comparison of baseline tournament selection operator variant, *Tsel*, against a variant, *Wrep*, that includes the *replica* property as the second criterion after feasibility. $|\mathcal{H}|$, the number of unique designs in the active population history, is given as a mean. The hypervolume of the final non-dominated archive, $hv(\mathcal{A}_o)$, is given as a mean. $|\mathcal{A}_o|$, the cardinality of the final non-dominated archive, is given as a median (due to more outliers). For each metric and problem, the larger of the two values is shaded.

	$\|\mathcal{H}\|$		$hv(\mathcal{A}_o)$		$\|\mathcal{A}_o\|$	
Instance	*Wrep*	*Tsel*	*Wrep*	*Tsel*	*Wrep*	*Tsel*
`comp01`	35,620	33,888	0.960	0.957	9.5	12.5
`comp02`	59,865	61,422	0.837	0.829	71.5	42.5
`comp03`	62,050	57,250	0.858	0.843	45	39.5
`comp04`	59,981	57,778	0.874	0.871	30	28.5
`comp05`	70,129	77,765	0.776	0.772	297.5	383.5
`comp06`	69,044	67,458	0.812	0.819	50	40
`comp07`	81,173	76,799	0.791	0.787	50.5	32.5
`comp08`	64,636	61,606	0.841	0.831	31.5	33
`comp09`	60,051	55,504	0.847	0.847	111.5	86
`comp10`	70,762	68,042	0.818	0.821	47.5	52.5
`comp11`	55,191	55,144	0.973	0.969	2	2
`comp12`	86,428	86,487	0.809	0.797	337	435.5
`comp13`	63,094	60,651	0.840	0.837	58.5	50.5
`comp14`	63,685	62,502	0.880	0.890	55	49.5
`comp15`	59,919	56,624	0.865	0.856	50.5	44.5
`comp16`	71,266	68,319	0.837	0.838	49	46.5
`comp17`	66,875	64,975	0.823	0.822	83	76
`comp18`	42,837	38,575	0.926	0.926	113	116
`comp19`	60,945	57,685	0.826	0.831	60.5	43
`comp20`	90,353	80,876	0.829	0.838	211	182.5
`comp21`	67,623	66,692	0.807	0.810	73.5	65.5

with greater frequency, through the wider exploration afforded by *Wrep*. However, the hypervolume figures do not bear out any such benefits in terms of the quality of the Pareto set approximation — at least not in as much as can be inferred from a unary metric. There was no significant improvement in $hv(\mathcal{A}_o)$ effected by *Wrep* over *Tsel*.

Table 5.5 presents the data as a contingency table, showing the counts where an increase or decrease in $|\mathcal{H}|$ has coincided with an increase or decrease in each of the other two metrics. While a significant relationship could not be shown with hypervolume, a χ^2 test statistic of 4.25 indicated a significant association between $|\mathcal{H}|$ and $|\mathcal{A}_o|$ at the $\alpha = 0.05$ confidence level.

This finding offers up an interesting interpretation. In many cases, the modified tournament selection operator *Wrep* led to a greater churn in the evolving population,

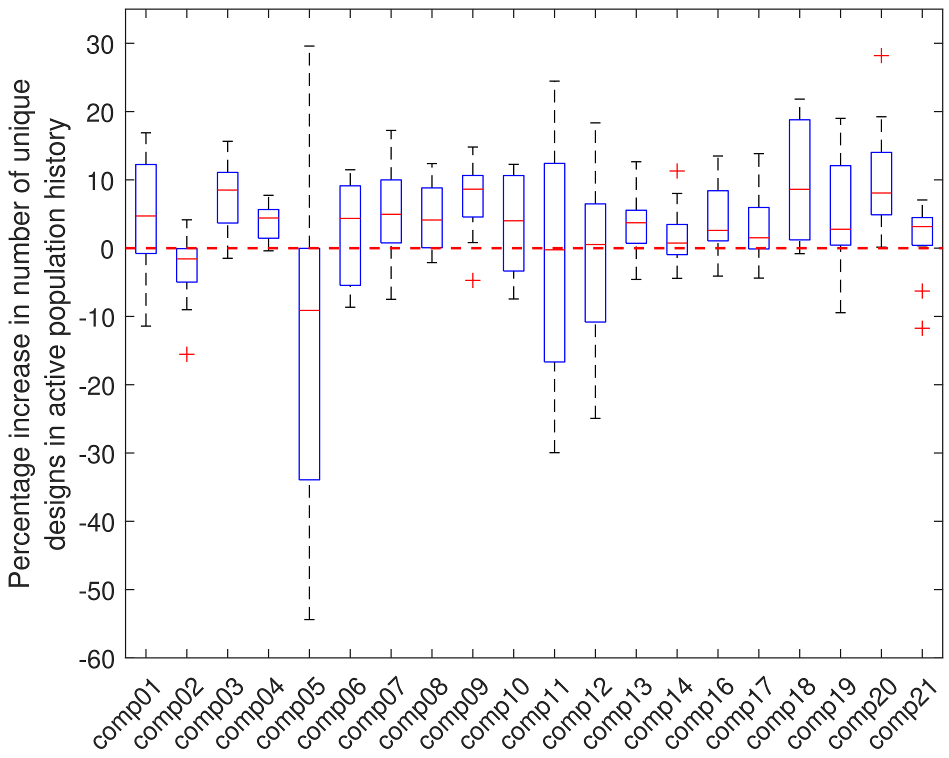


Figure 5.17: A boxplot showing distributions of the percentage increase in $|\mathcal{H}|$, the number of unique designs in the active population history, from the variant *Tsel* to *Wrep*. Boxes show the median and interquartile range, while the whiskers extend to the most extreme points not considered outliers. Outliers are shown as crosses.

but without necessarily increasing the hypervolume covered by the final approximation set. Nevertheless, increased churn correlated with higher cardinality of this set. There are several possible inferences to be drawn. The set returned by *Wrep* may be less well converged on the whole, while spanning a wider range of trade-offs between the objectives. Alternatively, the set may have reached a similar level of convergence, while providing a greater density of points and hence a higher granularity approximation to the front. Areas and extremes of the approximation to the front that are missing or, at best, coarsely drawn using the baseline variant may

Table 5.5: A contingency table for all 252 problem × repetition combinations. Symbols + and - represent an increase or decrease from *Tsel* to *Wrep* respectively, where *Tsel* is the baseline tournament selection operator and *Wrep* is a proposed modification using the *replica* property. $|\mathcal{H}|$ is the number of unique designs within the active population history, $hv(\mathcal{A}_o)$ is the hypervolume of the final non-dominated archive and $|\mathcal{A}_o|$ is its cardinality.

		$hv(\mathcal{A}_o)$		$\lvert\mathcal{A}_o\rvert$	
		+	-	+	-
$\lvert\mathcal{H}\rvert$	+	98	81	104	75
	-	38	35	32	41

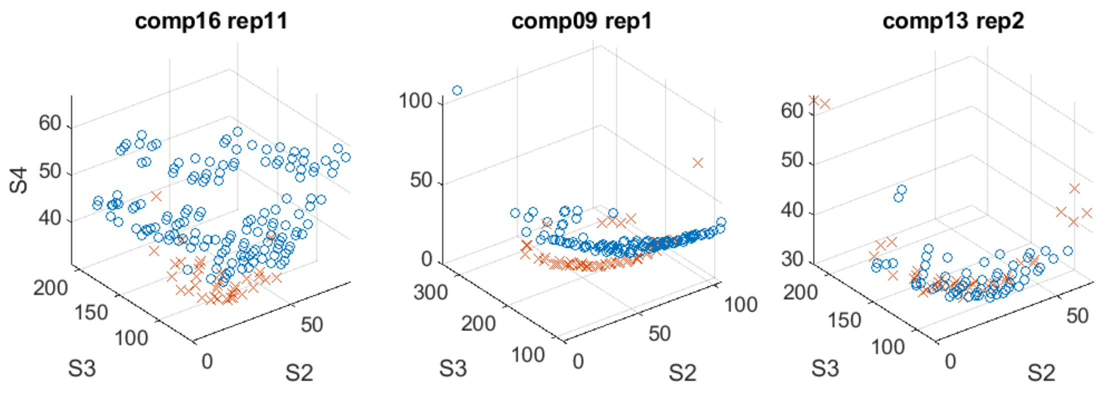


Figure 5.18: Comparisons of the Pareto front approximations generated by *Tsel* (red 'x') and *Wrep* (blue 'o') for repetition 11 of `comp16`, repetition 1 of `comp09` and repetition 2 of `comp13`. All three plots are representative of the case in which the two hypervolumes are approximately equal but the set cardinalities differ, with the *Wrep* variant being the more populous in every case. From left to right, cardinalities in the plot windows are 36 and 146, 54 and 109, and 38 and 57.

be discoverable in more detail by the *Wrep* variant. In Figure 5.18, three pairs of Pareto front approximations are presented to illustrate aspects of these scenarios. The sets, taken from single repetitions of `comp16`, `comp09` and `comp13` were chosen on the basis of the following criteria:

- The hypervolumes are equal, to within a tolerance of 0.2%.
- The cardinalities are unequal.
- The **S1** objective has been collapsed to zero across all points, enabling a 3-D visualisation.

In the first plot, many of the *Wrep* points are at a greater distance from the ideal point than those in *Tsel* and may in many cases be dominated. The larger set does, however, cover greater extremes. In the second and third plots, the fronts occupy positions that are more proximal in Euclidean space. Again, the larger sets offer greater overall coverage and a better resolution of the approximation front. Where trade-offs occur between convergence and cardinality, as in `comp16` repetition 11, valid reasons exist for preferring the larger set. These are summarised in the conclusions section which follows.

5.2.4 Conclusions on operators

An analysis of the benefits and drawbacks of various additional operators was conducted in this section. First, data was interrogated in order to quantify the periods of non-improvement in optimisation runs. This led to an *ad hoc* value for a stagnation threshold parameter, which was then relied upon to trigger a destroy-repair operator. The idea is broadly similar to iterated local search (Lourenço et al., 2010). It was discovered that the *saturation degree* heuristic, which worked well for constructing solutions from scratch, was less effective as a partial repair tool, even when the destruction was limited to a small number of lectures. In seeking to escape local optima with this strategy, a natural trade-off emerged between the need for a sufficiently large alteration to the timetable and the desire to preserve its more beneficial and well-evolved sub-parts. Two newly-introduced parameters, relating to stagnation threshold and the degree of destruction, were found to be highly sensitive and problem-dependent. More insight is required in order to correctly calibrate them.

Another pervasive question in this section revolved around the sequencing of feasibility checks. In the investigation into other perturbation operators, it was found that explicitly checking for all hard violations added too great a computational expense. Instead, a parameter called `attempts` was introduced, and a default routine called if a feasible perturbation was not found within this threshold. A value `attempts` $= 30$ was found to give good economy on the problems studied. More evidence was also found that, in terms of feasibility, the relationship between the state of maturity of the evolved solutions and the activity of the operators is somewhat problem-dependent but in most cases not significant. This suggests that there are more important factors than the progress of the run when considering why and when particular operators should be called.

One aspect that was associated with the state of maturity, however, was the quality of solutions found by the operators (i.e. the soft violation counts). A proposal was made to guide the operators towards choosing higher yielding swaps, based on domain knowledge. This was not easy to achieve, as any conditions on lectures that were inexpensive to check were also rarely satisfied in practise. Conceptually though, the pre-operator filtering of lectures certainly holds promise. A different selective approach was trialled in which lectures were preferred based on the dissimilarity of their assignment properties. While this led to larger changes in the quality of perturbed solutions, these were not always in a net positive direction. Variant *Uvio*, which was something of a compromise between two filtering extremes, was found to be the most successful, outperforming the other variants and the baseline.

Finally, modifications to the tournament selection operator also helped improve the algorithm. By introducing the property *replica* and encouraging greater churn — even sometimes at the expense of dominating solutions from the previous generation — more of the budget was diverted to exploration of the search space. The tangible advantages of this were seen not in the final hypervolume, which was not significantly affected, but rather in the increased granularity of the final approximation front.

In the next section, developments are premised upon the following notion: If a fifth objective of 'robustness' is brought into consideration, then the cardinality and spread shown in Figure 5.18 assumes greater importance. Without sacrificing high quality solutions to an overly-excessive degree, the decision maker can be furnished with a larger pool of options, potentially encompassing a wider range of robustness, from which to choose their preferred timetable. With this justification, the successful *Wrep* variant of the tournament selection operator was therefore adopted.

5.3 Robustness

In real world applications, one or more unforeseen disruptions may occur after a timetable has been drafted. Extra students may belatedly be enrolled into a particular curriculum, or a teacher may alter their unavailability requirements, for example. If the original timetable maintains its quality and feasibility in the face of a set of changes, it is said to be entirely robust to that particular disruption scenario. It is more likely, however, that the quality of a timetable will be compromised in some way by the adjustments to the problem instance.

The best method of finding, or generating, highly robust timetables is an open area of research. Akkan et al., 2020 developed a Simulated Annealing search algorithm for this purpose. The goal was to locate ITC2007 solutions that, once acted upon by random disruption scenarios, could be repaired without significantly degrading their quality. This was termed 'quality robustness'. A second consideration, deemed 'solution robustness', was to ensure the repaired solutions were, at the same time, not excessively different in design to the old ones.

When quantifying robustness, Akkan et al., 2021 noted that estimators are needed. Calculating an exact or 'idealised' metric is not computationally viable because of the high memory and time requirements. This is particularly the case when multiple concurrent disruptions are permitted. The authors investigated slack-based estimators, finding that the 'coefficient of variations of the number of conflict-free available periods per course' was the most effective of the 33 surrogates (11 slack measures $\times$ 3 summary statistics) tested. Extended work in Akkan et al., 2022 concluded that there is no closed-form expression for the 'idealised' robustness metric, due to the large sample sizes required. The authors treated their robustness estimator as stochastic, in the sense that a Pareto 'band' was maintained rather than a Pareto front approximation. This band gave an indication of the probability of a solution existing on the true Pareto front.

In this section, some perturbations to the problem instance, or 'disruptors', are mathematically defined. These are inspired by circumstances that might be encountered in a real world use case. An 'idealised' robustness metric is defined in order to capture a sense of the deterioration in quality of a solution as a reaction to single and/or multiple disruptors. A more computationally viable approximation to this idealised metric is then proposed and ultimately integrated into the optimiser as a fifth objective to be minimised.

5.3.1 Disruptors

A number of principles guide the design of the disruptors. Re-evaluating a timetable subject to a changed instance could result in different values for the objectives, **S1** ... **S4**. In a worst case, it may also affect any combination of **H1** ... **H5**, thereby rendering the solution infeasible. Every constraint, whether hard or soft, that is potentially unstable with regard to the changing of a problem instance should be targeted by at least one disruptor in the pool. Where possible, individual disruptors should act on individual constraints in isolation. In the real world, circumstances may transpire that make the problem easier, rather than harder, to solve. For example, if an extension is built that enlarges the capacity of a room, or if a group of students drop out of a course. Such eventualities are of lesser interest when considering robustness. The disruptors should instead be designed with the aim of constraining the problem more tightly or, at the very least, rearranging the patterns of existing constraints. As a counterbalance to this, disruptors should not act so destructively on a particular problem so as to make it entirely intractable. For maximum insight, the robustness tests should induce deterioration across a full spectrum of ways: exclusive soft constraints, exclusive hard constraints, and both constraint violation types together. By extension, the perturbed problems ought ideally to be solvable by the optimiser.

Table 5.6 provides details of seven disruptors. As a pool, the disruptors can bring about violations in any of six constraint types (**H3**, **H4**, **H5**, **S1**, **S2**, **S3**). The remaining three (**H1**, **H2**, **S4**) are invariant to problem perturbation under the encoding scheme used.

In `dNewCnn`, the curriculum membership of a course is revised in such a way as to increase its student count. Problem instance descriptions do not give student count as an explicit property of curricula. However, to maintain internal consistency of the problem, the student count must be updated in conjunction with the changed curricula. The correct new values could be inferred, in some cases, directly from the data, or by the use of linear algebra. A more restrictive but computationally cheaper method is by the subtraction described in Table 5.6. Note that due to these structural dependencies between students and curricula, `dNewCnn` is the only disruptor that affects multiple violations. `dNewCP` and `dNewTP` both affect unavailability constraint **H4**. The former adds course-specific period prohibitions, while offsetting this by releasing a (smaller) number of previously unavailable periods. The latter blocks out periods in a teacher-specific manner, which may impact across more than one course.

Table 5.6: A pool of seven disruptors used in the calculation of a robustness metric. The identifier is the name of the function used to call a particular disruptor. Each one acts on prescribed entities, such as a course or teacher, according to the mechanism in column 3. The final column shows which type/s of constraint violation can potentially be induced.

Identifier	Acts on	Mechanism	Affects
`dNewCnn`	A randomly chosen course C_i that does not have membership of every possible curriculum.	The curriculum membership of C_i is changed to $\{u : C_j \in u\} \backslash \{u : C_k \in u\}$ and $stud(C_i)$ becomes $stud(C_j) - stud(C_k)$, where $j,k \sim \{1, \mathcal{C}\}, j,k \neq i$, $\{u : C_k \in u\} \subsetneq \{u : C_j \in u\}$ and $stud(C_j) - stud(C_k) > stud(C_i)$.	**H3**, **S1**, **S3**
`dNewCP`	A randomly chosen course C_i where $\lvert unav(C_i)\rvert \geq 1$.	A set of consecutive periods within a single day that were available are made unavailable. Meanwhile, a smaller or equally sized set (if it exists) of consecutive periods within a single day that were unavailable, are made available.	**H4**
`dNewCS`	A randomly chosen curriculum u_i.	All courses in u_i have their student enrolment increased by $\theta \sim \{1, \lfloor \frac{1}{2\lvert\mathcal{C}\rvert} \sum_{i=1}^{\lvert\mathcal{C}\rvert} stud(C_i) \rfloor\}$.	**S1**
`dNewTP`	Courses taken by a randomly chosen teacher, T_i.	Teacher T_i becomes unavailable for a randomly chosen timeslot.	**H4**
`dNewTS`	Two randomly chosen courses C_i and C_j with distinct teachers.	The two teachers of the courses are swapped with one another.	**H5**
`dNewMWD`	Two randomly chosen courses C_i and C_j, where $mwd(C_i) < \lvert\mathcal{D}\rvert$ and $mwd(C_j) > 1$.	$mwd(C_i)$ is increased by $\theta \sim \{1, min(\lvert\mathcal{L}\rvert, \lvert\mathcal{D}\rvert)\}$. $mwd(C_j)$ is decreased by 1.	**S2**
`dNewRC`	Two randomly chosen rooms, r_1 and r_2, where $cap(r_1) \geq cap(r_2)$.	$cap(r_1)$ is decreased by θ_1, where $\theta_1 \sim \{2, \lfloor cap(r_1)/2 \rfloor\}$. $cap(r_2)$ is increased by θ_2 where $\theta_2 \sim \{1, \theta_1 - 1\}$.	**S1**

`dNewCS` and `dNewRC` also target a common constraint (**S1**) but, again, from different perspectives. The former increases the student count for a particular curriculum, up to a limit of half of the mean student count across all courses. The latter acts upon a pair of rooms, decreasing the capacity of the larger by the value of a discrete random variable, while increasing the capacity of the smaller by a lesser amount. `dNewTS` reflects a real world situation in which two teachers swap courses with one another, which may possibly cause teacher clashes in other areas of the timetable. Finally, `dNewMWD` tightens the minimum working days requirement for one course, while loosening it to a lesser degree for a second course.

5.3.2 Idealised robustness metric

In real world timetabling, multiple disruptions may occur concurrently. To model this, a random number of disruptors are selected (without replacement) and executed on the problem in sequence. To respect the principle of problem tractability, this number is capped conservatively at 3. A total of $\sum_{k=1}^{3}\binom{7}{k} = 63$ combinations of disruptors are therefore available. Another feature in real world applications is that no combination of disruptors can be predicted to occur with certainty. Let this random variable be denoted as D. Let R be the random variable representing the outcome of executing a combination of disruptors. The expected outcome of a fixed combination is $E(R|D)$, while a idealised unary robustness metric may be expressed by $E(R)$.

A definition is first required for R in order to quantify the deterioration of a given timetable. Given a feasible solution x to a problem $\mathtt{d}$, $\mathtt{H}_{all}(x, \mathtt{d})$ returns a vector of length five representing the violation counts for [**H1 H2 H3 H4 H5**], while $\mathtt{S}_{all}(x, \mathtt{d})$ gives the objective vector representing violation scores for [**S1 S2 S3 S4**]. The equations (5.5) and (5.6) give the deterioration in solution quality in terms of hard and soft violations respectively, when x is re-evaluated on the perturbed problem `dNew`. A greater value corresponds to a higher deterioration, and $[0]^k$ is the zero vector of length k.

$$\texttt{deteriorationH} = max(\texttt{H}_{all}(x, \texttt{dNew}) - \texttt{H}_{all}(x, \texttt{d}), [0]^5) \tag{5.5}$$

$$\texttt{deteriorationS} = max(\texttt{S}_{all}(x, \texttt{dNew}) - \texttt{S}_{all}(x, \texttt{d}), [0]^4) \tag{5.6}$$

These quantities are brought together in (5.7). The hard constraint differences are multiplied by a coefficient in order to give them prominence relative to the soft constraint differences. The sum of the concatenated vector then yields a measure of robustness for x — under the specific disruption scenario described by the transformation of `d` into `dNew`.

$$\texttt{rob}_x(\texttt{d}, \texttt{dNew}) = \sum_{i=1}^{9} [5 \times \texttt{deteriorationH}, \texttt{deteriorationS}]_i \tag{5.7}$$

In theory, $E(R)$ can be obtained by systematic evaluation of all possible arrangements of `dNew` accompanied by the calculations in (5.5), (5.6) and (5.7), and averaging. As the affected constraint type is known for each disruptor (as per column four of Table 5.6), δ-evaluations can be employed to reduce the computational cost. Even with this mitigation however, calculating the idealised robustness metric is not practicable due to the overwhelming number of distinct problem perturbations that the 63 disruptor combinations can induce.

As a test, an estimator is considered for $E(R)$, in order to first gain an understanding of the discriminatory power of the idealised metric. If the idealised metric shows little differentiation among dissimilar solutions, then the guidance provided for the optimiser is less useful. Such an issue would only be compounded further by the statistical noise introduced by a low quality estimator. In the following test, a large number of samples (far higher than would be viable within a real time optimiser loop) were taken of potential disruption scenarios. This number was set to 20,000. A feasible solution, x, was sampled at random from the complete search history of a previous run. The distribution of the proposed robustness metric resulting from reevaluating solution x over the 20,000 perturbed problems was obtained. Solution x was then resampled a total of 250 times and the corresponding distribution returned each time. The 250 distribution means (which serve as the

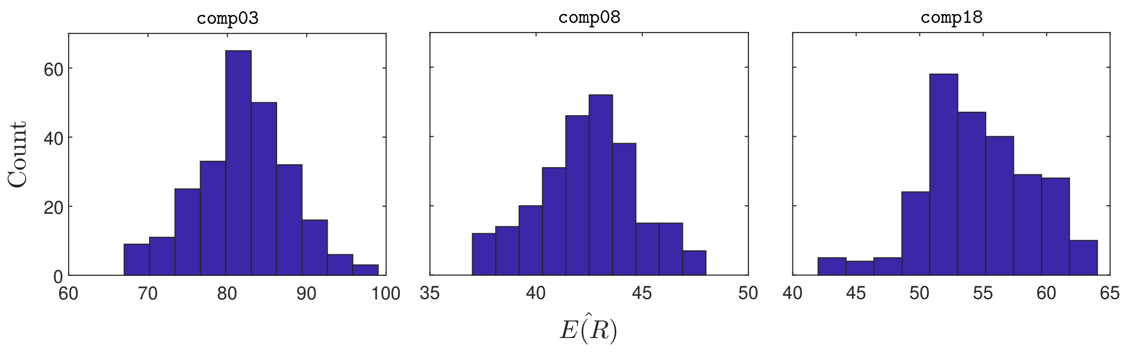


Figure 5.19: $\hat{E(R)}$, an estimate for the idealised robustness metric $E(R)$, is obtained by way of the mean outcome over 20,000 resampled disruption scenarios. The histograms show the distribution of $\hat{E(R)}$ over 250 solutions sampled from previous 4-objective runs, for instances `comp03`, `comp08` and `comp18`.

naive estimates for $E(R)$) are displayed as histograms for three instances in Figure 5.19.

Figure 5.19 serves as intuition for the range of values that $E(R)$ may commonly take. Recall that these samples use data from the 4-objective runs only, and that the left hand tails of these distributions are likely to extend further towards 0 if robustness were being driven deliberately as its own objective. While these plots provide useful insight, real time calculation, or even high quality estimation, of $E(R)$ is too expensive in the context of the optimiser. In the next section, experiments are undertaken using a simpler, computationally viable scheme in order to prove the many-objective robustness concept and analyse trade-offs between the five objectives.

5.3.3 Experiment on fixed scenarios

In this experiment, fixed scenarios are used — in other words, *a priori* knowledge of disruption scenarios is assumed. Initial testing revealed that five was an acceptable trade-off between the sample size of problem perturbations and additional run time. For each problem, five disruption combinations were therefore sampled at random and executed on the instance outside of the optimiser loop. The perturbed instances, $\texttt{dNew}_i, i = 1 \ldots 5$, remained fixed over all repetitions. The aggregation in (5.8) provided the robustness metric.

$$\sum_{i=1}^{5} \texttt{rob}_x(\texttt{d}, \texttt{dNew}_i) \tag{5.8}$$

Table 5.7: Hypervolume reference point coordinates for the fifth objective, robustness, for the formula given by expression 5.8 and fixed sets of five disruptor combinations.

`comp01`	`comp02`	`comp03`	`comp04`	`comp05`	`comp06`	`comp07`
3,200	5,817	6,245	5,321	9,233	8,232	10,624
`comp08`	`comp09`	`comp10`	`comp11`	`comp12`	`comp13`	`comp14`
6,418	9,267	1,434	5,650	1,292	1,797	1,496
`comp15`	`comp16`	`comp17`	`comp18`	`comp19`	`comp20`	`comp21`
10,397	6,082	10,850	6,494	6,178	9,161	5,127

12 repetitions were run for each problem instance with a function evaluation budget of 1,000,000, `setPopSize` = 100. Hypervolume reference coordinates for the fifth objective were derived from a mathematical upper bound. These are listed in Table 5.7.

5.3.4 Results

Results for the hypervolume and cardinality of the non-dominated archive at termination are given in Table 5.8. The left-hand side of the table shows extremes and averages of the individual repetitions. Exploiting the parallelisation of these repetitions across CPU cores, better hypervolumes can be achieved in a run time of the same magnitude by amalgamating the independent repetitions. A combined archive was therefore created by extracting non-dominated solutions from the union of the final archives across all repetitions. Metrics for these combined sets are shown in the right-hand side of the table.

Figure 5.20 shows the improvement of the mean hypervolume over the generations, for all instances. Gradients at termination suggest that even a small budget increase would lead to further gains.

Figures 5.21 and 5.22 show (for `comp01` to `comp10`, and `comp11` to `comp21` respectively) parallel coordinate plots of the combined Pareto approximation sets. At least one perfectly robust solution (i.e. a robustness metric of 0) was found for every instance in these sets, with the exceptions of `comp11`, `comp16` and `comp20` (whose most robust, non-dominated solutions scored 6, 5 and 13 on that objective respectively). In order to better show the trade-offs between robustness and generalised timetable quality, an alternative, lower-dimensional visualisation of the same data is given in Figures 5.23 and 5.24. In these plots, the combined Pareto approximation sets have

Table 5.8: Results for the 5-objective treatment (four soft constraint violation scores plus a robustness metric) using a 1,000,000 function evaluation budget. $hv(\mathcal{A}_o)$ is the hypervolume of the final Pareto approximation set, while $|\mathcal{A}_o|$ is its cardinality. Results are given both by individual repetition, and for the set generated by combining the 12 repetitions.

	By individual repetition					By combined repetitions	
Instance	$hv(\mathcal{A}_o)$		$\|\mathcal{A}_o\|$			$hv(\mathcal{A}_o)$	$\|\mathcal{A}_o\|$
	Best	Mean	Min	Max	Median		
`comp01`	0.976	0.962	7	147	42.5	0.976	31
`comp02`	0.820	0.800	202	861	495	0.857	1,220
`comp03`	0.855	0.823	220	698	305	0.880	697
`comp04`	0.877	0.852	97	335	129	0.893	275
`comp05`	0.767	0.740	1,499	3,289	2,632.5	0.816	4,496
`comp06`	0.829	0.801	108	950	475.5	0.841	428
`comp07`	0.782	0.764	152	724	415.5	0.797	427
`comp08`	0.853	0.826	17	229	60	0.877	98
`comp09`	0.800	0.776	378	1,327	626.5	0.815	599
`comp10`	0.813	0.785	73	928	635	0.821	316
`comp11`	0.976	0.954	628	2,451	1,781.5	0.977	1,527
`comp12`	0.746	0.708	1,743	4,920	3,699	0.772	5,560
`comp13`	0.815	0.780	175	835	480.5	0.825	442
`comp14`	0.873	0.844	142	762	359.5	0.884	501
`comp16`	0.837	0.813	68	442	309.5	0.854	154
`comp17`	0.809	0.775	454	1,024	605.5	0.824	855
`comp18`	0.890	0.866	374	1,102	768	0.914	500
`comp19`	0.860	0.835	35	575	86.5	0.884	212
`comp20`	0.737	0.699	1,068	3,664	2,120	0.747	2,130
`comp21`	0.805	0.783	68	469	254	0.832	391

been projected into 2-dimensional space where the x-axis is the traditional scalarised score (**S1** + **S2** + **S3** + **S4**). A common observation (exemplified best by `comp02`, `comp03`, `comp07`, `comp11`, `comp14`, `comp17`, `comp18` and `comp20`) is that quality and robustness are in conflict.

Characteristic trade-off curves are sketched out for these instances, with the most tightly drawn being that of `comp11`. Seeking a near-optimal solution to this instance means accepting a trade-off with its robustness, with this metric being around 200 or higher. On the other hand, a perfectly robust solution can be chosen, but only if a large deterioration in scalarised score, of around 300 or higher, is accepted. A more complex trade-off picture is shown for `comp05` and `comp12`, in which clustering / disconnected fronts are seen. These two problems are highly constrained. Perhaps not coincidentally, they also returned the two largest combined

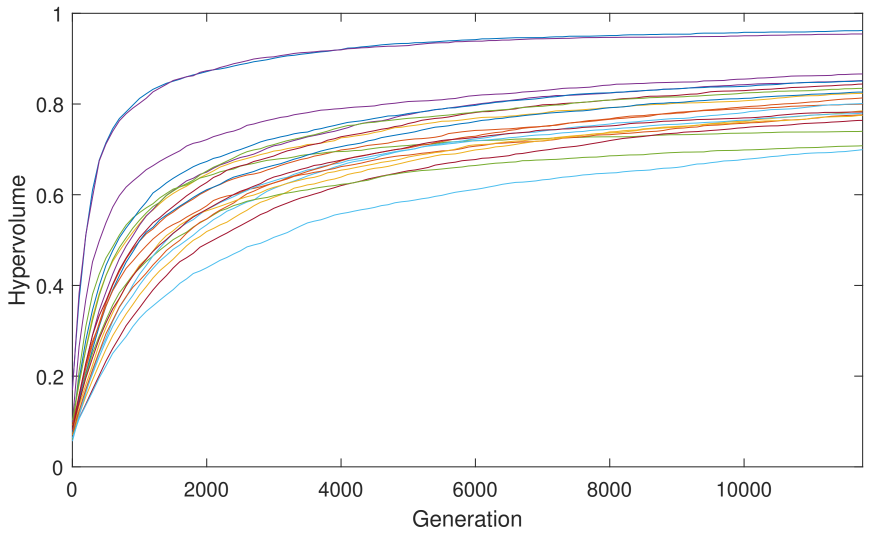


Figure 5.20: Profiles of the hypervolume improvement (as a mean over 12 repetitions) for instances `comp01` to `comp21` under a 5-objective treatment (four soft constraint violation scores plus a robustness metric). Numerical results from the same runs are reproduced in Table 5.8.

archives of all instances in the experiment. This suggests they would benefit more than most instances from an increased budget. Meanwhile, `comp08`, `comp16` and `comp21` show that, for instances where inherent conflicts between objectives are oblique, solutions that are highly robust to the fixed scenarios can be found across a wide range of qualities.

The scenarios used in this experiment were fixed in order to avoid the constant resampling of disruptions. Tests showed that attempting this would not only be prohibitively expensive, but would also introduce an untenable degree of noise to the estimator. However, solutions optimised according to a fixed disruption scenario may, ultimately, be more robust in general when compared to solutions obtained by the 4-objective model. The following section tests this hypothesis analytically.

5.3.5 Analysis of general robustness

In this section, further analysis is carried out on the data from the fixed scenarios experiment. Included in the 5-D Pareto front approximations obtained previously are subsets of solutions in which the robustness score is minimal. For the majority of instances, this minimal value is 0. While quality varies in other regards, any solution with a metric value of 0 can be said to be perfectly robust to the fixed scenario.

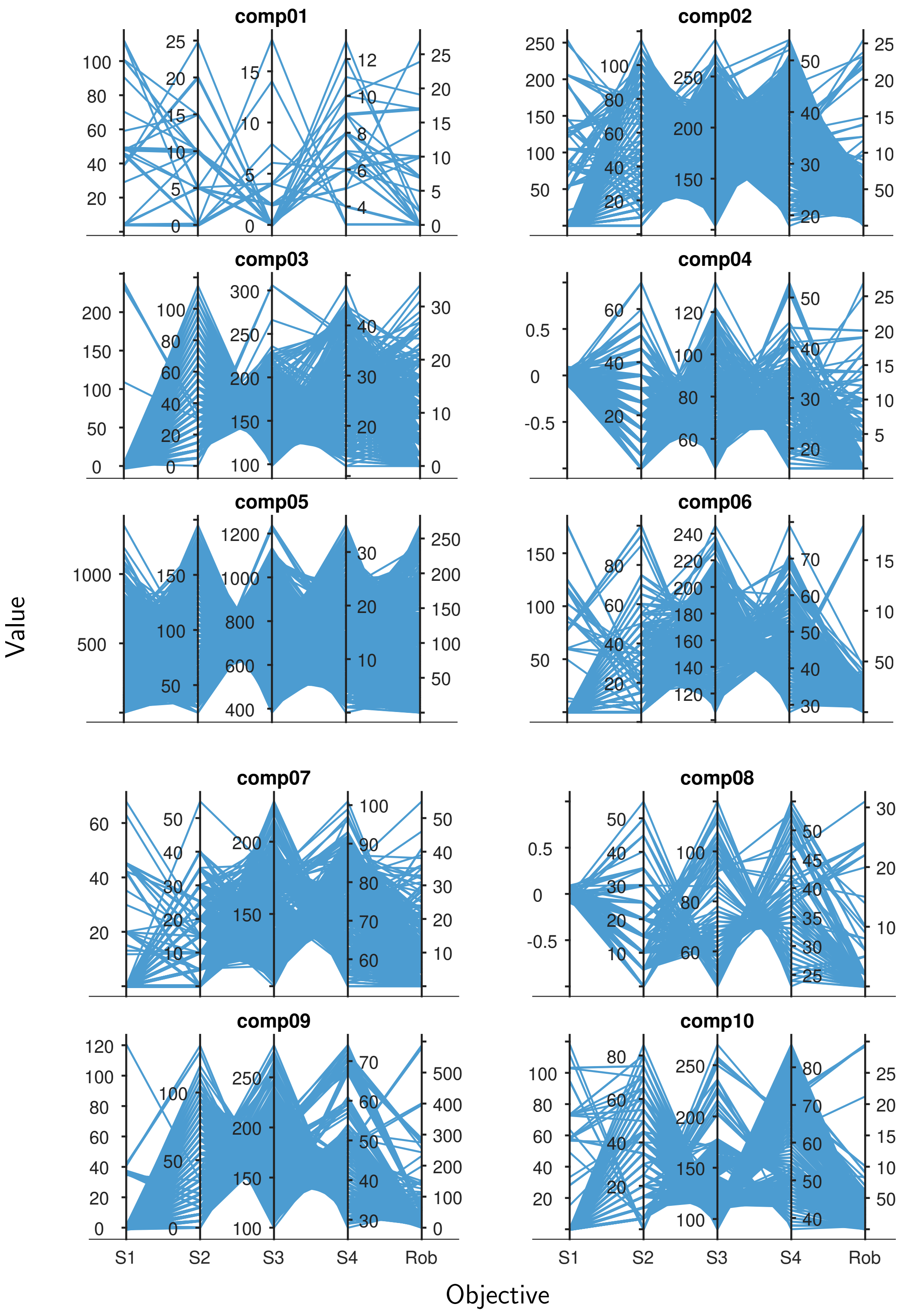


Figure 5.21: Combined Pareto approximation sets for instances `comp01` to `comp10` under a 5-objective treatment.

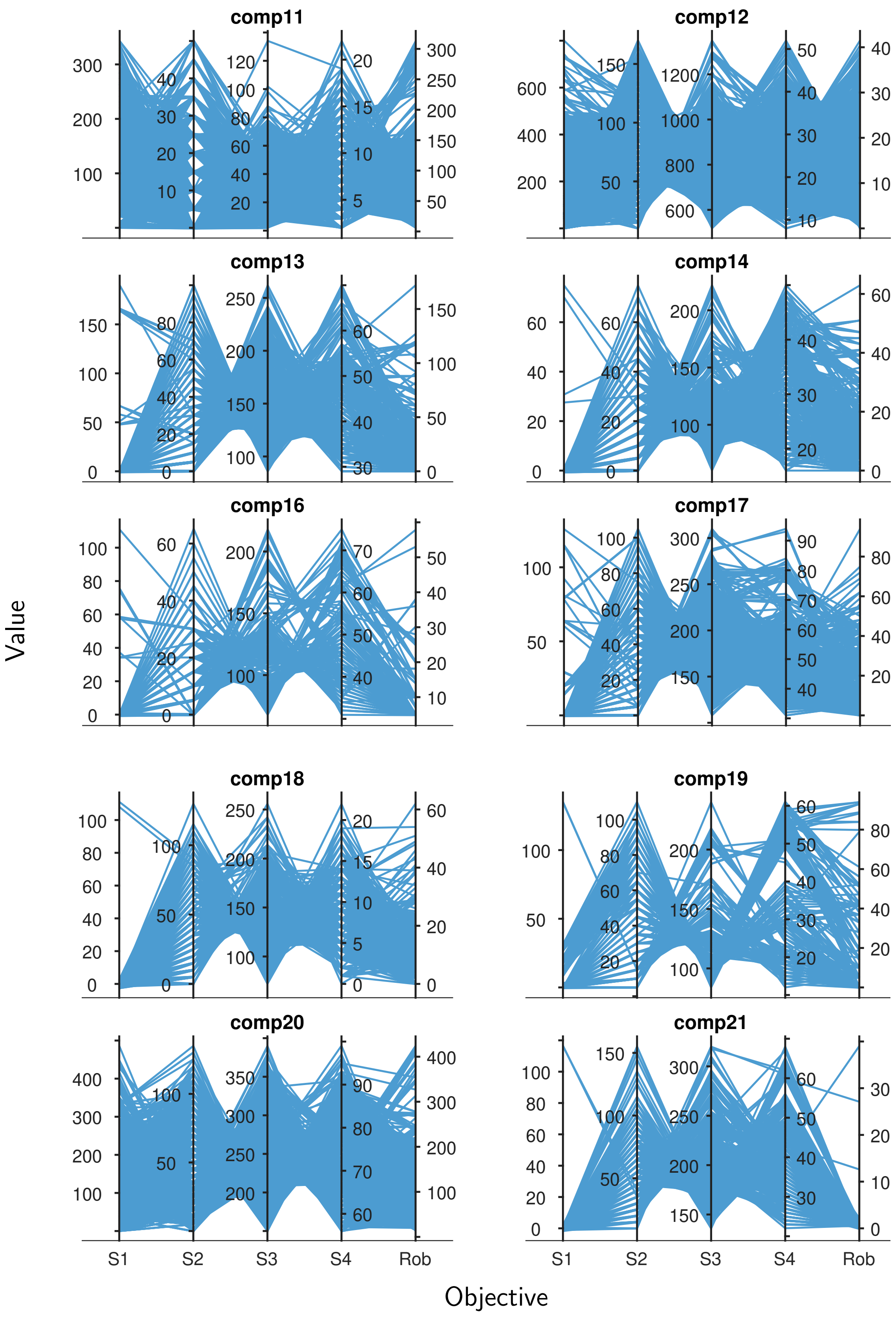


Figure 5.22: Combined Pareto approximation sets for instances `comp11` to `comp21` under a 5-objective treatment (`comp15` excluded).

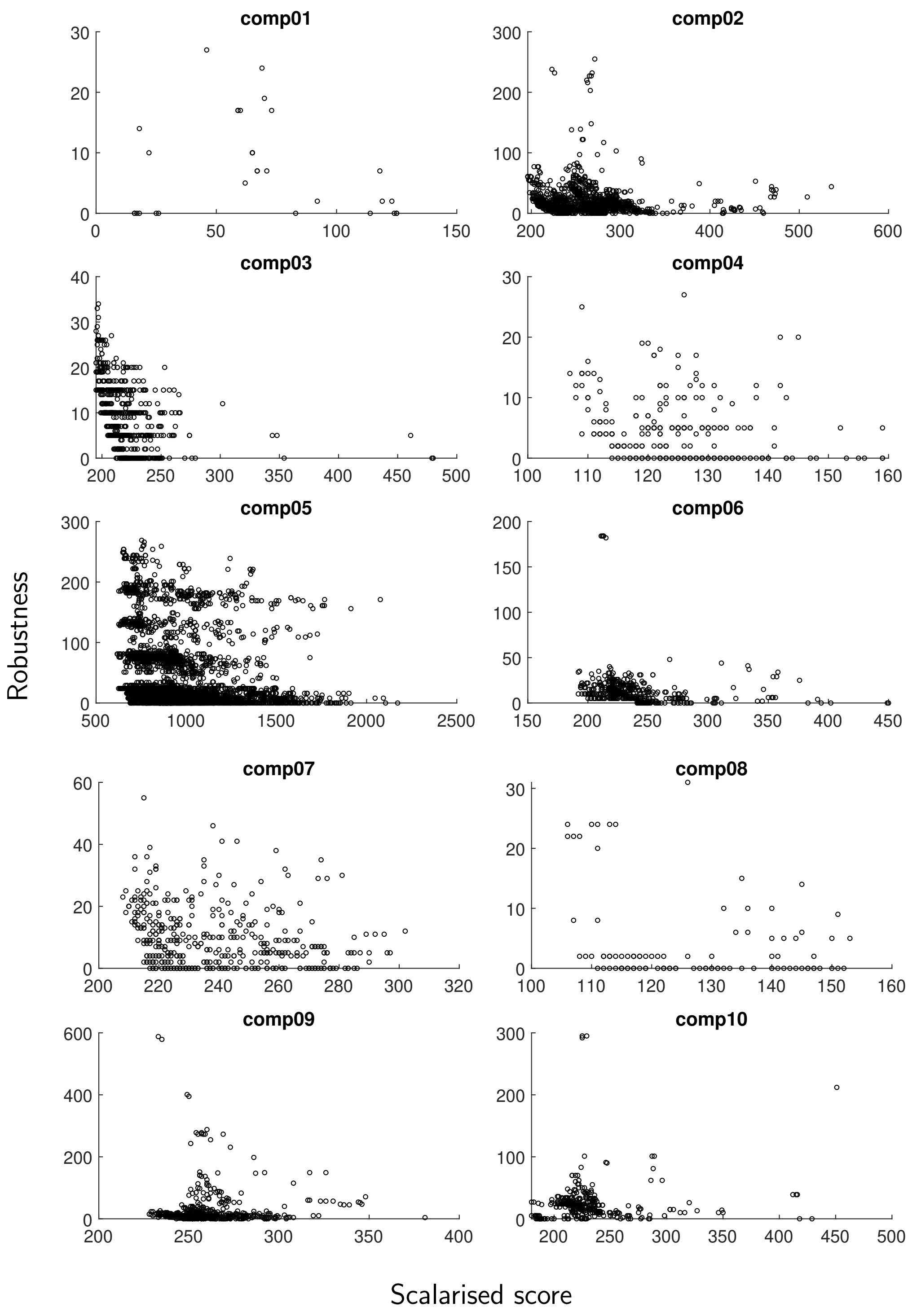


Figure 5.23: The combined Pareto approximation sets for instances `comp01` to `comp10` under a 5-objective treatment, projected into 2-dimensional space. On the x-axis is the scalarised score (or sum of objectives **S1** ... **S4**), while on the y-axis is the robustness metric. This figure presents the same data as in Figure 5.21.

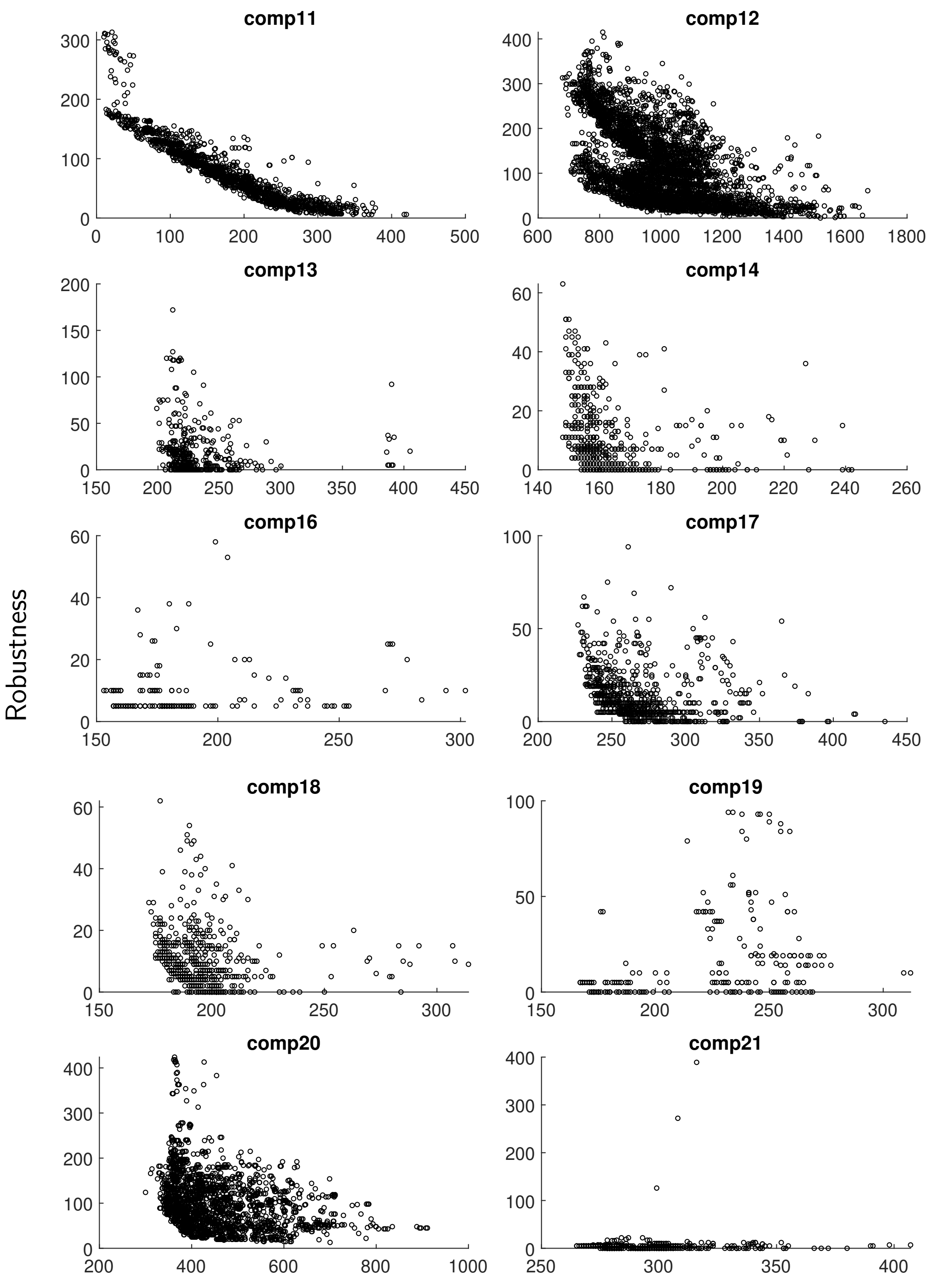


Figure 5.24: The combined Pareto approximation sets for instances `comp11` to `comp21` (excluding `comp15`) under a 5-objective treatment, projected into 2-dimensional space. On the x-axis is the scalarised score (or sum of objectives **S1** ... **S4**), while on the y-axis is the robustness metric. This figure presents the same data as in Figure 5.22.

The question of whether this property implies a greater robustness in general, to any form of unseen disruption or disruptions, is addressed using the following test. For all instances, where possible, 50 perfectly robust solutions, obtained from runs with a 1,000,000 function evaluation budget, were sampled without replacement from the final non-dominated archives. As a baseline comparator, samples were also taken of non-dominated solutions obtained from the 4-objective model, which has no robustness objective at all. These sample sets, also of size 50, were drawn from runs with an equal budget. For each sampled solution, an estimate for $E(R)$, the idealised robustness metric described in Section 5.3.2, is then obtained by way of reevaluations against 10,000 resampled disruption scenarios and averaging.

Table 5.9: The mean, over 50 solutions, of the estimated idealised robustness metric, $(\widehat{E(R)})$, for samples of solutions from the 4-objective optimiser vs. those from the 5-objective optimiser. The lesser means, indicating greater robustness, are shaded.

Instance	$\widehat{E(R)}$ 4-objective	$\widehat{E(R)}$ 5-objective
comp01	47.0	46.6
comp02	122.2	110.8
comp03	86.7	83.0
comp04	48.3	49.5
comp05	223.8	209.9
comp06	67.1	57.0
comp07	48.5	46.2
comp08	45.4	43.7
comp09	50.0	46.2
comp10	45.8	40.9
comp12	67.3	52.4
comp13	61.4	51.5
comp14	37.6	35.2
comp17	45.0	43.5
comp18	56.0	50.3
comp19	79.2	79.6
comp21	56.9	53.5

Table 5.9 shows the mean (over 50 solutions) of the estimated idealised robustness metric, $\widehat{E(R)}$, for each relevant problem, over the two sample universes. Using a blocking factor to account for variability in problem instances, a Friedman test was run on the raw data. A p-value of 0 indicated a rejection of the null hypothesis that the column effects (which universe the samples were drawn from) are the same. In other words, the evidence suggests that being perfectly robust to a specific fixed

disruption scenario makes a solution more robust to generalised disruptions than a solution that has been optimised with no robustness objective at all.

5.3.6 Conclusions on robustness

The concluding work in this chapter looked at the incorporation of a robustness metric as the fifth objective in a many-objective optimiser.

Realistic disruptors were first defined according to a set of guiding principles. A small number of constraints were found to be untouchable by problem disruption, due to the framework and encoding used. The remainder, whether hard or soft, were targeted in specific ways by the mathematical operations of seven different disruptors.

The difficulty of measuring robustness became apparent when devising a metric. The idealised metric was so-named on account of the exorbitant expense required for its calculation. Nonetheless, it provided both a starting point and a reference point for further investigation and later sanity checking. A proposed estimator proved viable in experimental settings but not within the optimiser loop itself. As a remedy for this, instead of a vast pool of possible disruption scenarios, fixed scenarios were utilised instead. By re-assessing solutions against a total of five known disruptor combinations, an evaluation for the fifth objective could be obtained without any of the statistical noise associated with sampling from a larger pool. A degree of variability in the problem perturbation was also retained. Furthermore, the impact on run time was deemed acceptable using this number.

Results showed that the optimiser coped well with an additional objective, with large and well-spread non-dominated sets achieved by termination. The optimiser yielded highly robust solutions for all problems, with the majority of non-dominated sets including multiple distinct timetables that were perfectly robust. As would perhaps be expected, the values for the soft constraints were generally worse than for the 4-objective optimiser with the same budget. A trade-off, best illustrated in the 2-D projections in Figures 5.23 and 5.24, was commonly found between robustness and quality. While `S1` was again well handled, more investigation into operators is needed if these fronts are to show better convergence in the other three quality objectives. It is interesting, however, that no operator within the system was specifically tailored to aid robustness, and yet this objective was optimised most successfully.

In the final section, a determination was made as to essentially how good a surrogate the fixed scenarios metric was for the idealised metric. ‘Perfectly’ robust solutions were compared with solutions optimised in four objectives only, under equal budgets. The outcome was statistically significant. The evidence suggested that even optimising for a limited pool of disruptor combinations could prime solutions to be robust in a more general sense. It should be noted though that the two variants terminated in different regions of the (**S1**,**S2**,**S3**,**S4**)-space, with the Pareto fronts generally being better converged for the 4-objective case, though not always significantly so. While the equal budget comparison yielded a significant and positive result, further study is needed to ascertain whether positioning in 4-D space is a confounding factor associated with robustness too.

In the section that follows, a final summary is given of each chapter, along with insights and suggested directions for future work.

6. Summary and Further Work

This thesis has considered the university course timetabling problem (UCTP), and in particular the curriculum-based model and ITC2007 benchmark. A review of literature in Chapter 2 revealed that the UCTP has been the subject of many decades of research, both as an abstraction and for its practical applications. Interest in educational timetabling has shown a particularly rapid growth since the early 2000s, coinciding with the advent of the International Timetabling Competition, which has encouraged collaboration through standardised formulations. Despite this large body of extant work, automatic timetabling continues to evolve as a field, and gaps in the research remain. In Chapter 2, metaheuristics were identified as some of the most successful, efficient and adaptable techniques for solving the UCTP. One such approach, ant colony optimisation (ACO), showed promise for timetabling, while at the same time posing a number of unanswered questions. Chief amongst these related to the ordering of course lectures within the construction graph. In Chapter 3, a specialised ACO solver was created. The idea of a threshold value in a dynamic constraint handling component — strong enough to discourage virtual ants following unprofitable pathways, yet liberal enough to prevent dead ends — was included as a novel contribution. This proved successful in enabling the virtual ants to construct feasible solutions. The central focus of Chapter 3, however, was the related aspect of lecture ordering. Revisiting the objectives set out in Section 1.1, the question of whether lecture ordering impacted on ultimate timetable quality was answered in the affirmative. More profoundly than this, certain features of problems, and the position of those features within a permutation of lectures, were identified as being more important than others in predicting good orderings. Conflict between curriculum memberships, and the number of overall lectures were examples of highly informative features in this regard. The significance of feature values for courses was also influenced by the position of those courses within the permutation. Courses assigned near the start or the end of a virtual ant's construction path were shown to provide a more critical contribution and thus had to be chosen with greater care than those in the middle. Strong patterns of correlation were also observed between permutation distance (measured by a chain of neighbourhood swap operators) and similarity in performance.

This finding suggested that information from easy-to-obtain features could be exploited in order to learn good permutations for use in an ACO construction graph. A novel pipeline was proposed for this purpose. A training set of small problems was completely enumerated by permutation. The outcomes of these repeated runs were used as target values in a regression model, which was then scaled to larger, unseen problems as a predictor. When scaling to relatively small problems, there were some encouraging results, but the success rate become more mixed when extrapolating to larger problems. Taken as a whole, the ACO was not competitive with state of the art results, but this fact is to overlook the true value of the study, which was to investigate the workings of one of its internal components in relative isolation. Having established the importance of both ordering and more specifically course features and positions, there are several directions for further work that could address some of the limitations of this chapter. While the feature set was deliberately kept simple, feature extraction could be used to discover more elaborate features for use in the training process. Similarly, the features could be made more granular by increasing the course cardinality in the training problems. The reason why this was fixed to 4 (in the proof of concept work) and then 5 (in the final model) is because the permutation space scales exponentially with number of courses, making complete enumeration costly. However, by leveraging greater CPU power, parallelism, or simply allowing for a greater temporal training budget, it is far from inconceivable that problems with 8 courses or more could be processed in this manner. An increased set of larger-sized training problems would also enable more nuanced interplay between constraints to be learned. The wider implications are that learned permutations could potentially outperform those constructed by heuristic or by randomness, as is often the case currently. If such a predictor were to be refined and proven, it could be ported to other types of ACO, or potentially other metaheuristics which rely on the sequential assignment of lectures or other events.

Chapter 4 followed a new direction, with the UCTP being treated as a many-objective problem. This was identified as being another prominent gap in the existing literature, or rather, an approach to timetabling that is only recently beginning to gain traction. As such, the proposed extension and modification of an existing (and originally continuous) many-objective algorithm provides novelty. Previous studies have formulated the UCTP as a multi- or bi-objective problem, or sought to address individual objectives separately before returning scalarised results. In this chapter however, a novel attempt was made to return a set of non-dominated solutions that approximate the Pareto front for the ITC2007. Attention was paid to minimising complexity and parameter count. The solver, which is essentially parameterless and uses a simple mutator but no crossover, was shown to return well-spread solution sets. With the exception of perfectly solvable problems like `comp11`, whose Pareto optimal set comprises a single point in objective space, the shapes and discontinuities of the Pareto fronts for each problem are not known. Published approximations to these are also, until now, lacking. Thus, although there is no direct set-wise comparison to be made for the results in this chapter, published best known scalarised scores can offer useful context. Analogous to shining a light on a single brick in a wall to gauge its position in the dark, some intuition about distance to the optimal solutions can be gained this way. Due to the extra computational overheads present in many-objective optimisers, achieving results close to the optimal or best-known single-objective scores was not expected within the same time budget. Regardless, optimal or near-optimal timetables were returned for two problems, while across the benchmark, results were competitive for some other instances. This is highly encouraging given both the simplicity of the approach, and the fact that a higher focus was placed on the spread/cardinality of the non-dominated set than on pure convergence. The efficacy of a multi-phase solver was reaffirmed in this chapter too.

Tackling constraint types sequentially prevented stagnation in infeasible space. Some interesting findings emerged from the initialisation phase, which could lead to further work. The constructive heuristics studied were shown to perform well, tempered by a failure to return a uniform sample from the solution space. How much of a limitation this is, in the wider context of the optimiser, remains unclear. A direction for future work may involve introducing an adaptive bias to the choice of construction paths such that a more uniform sample could be obtained. The idea of combining or pooling heuristics is also an interesting avenue for investigation. Initialisation was shown to be inexpensive relative to phase two (the main optimiser loop). Thus, if it were established that the initialisation protocol exerted a significant influence on the final results, there is certainly scope to devote more of the computational budget to this opening phase.

In Chapter 5, some issues with the commonly used form of direct encoding were identified. A *standard form* version, which involved making limited and systematic swaps between genes in the chromosome, was proposed. This contracted the search space as well as providing a basis for modification of the tournament selection operator employed later on. While this thesis was limited in scope to direct encodings, the efficacy of constructive heuristics showed that further exploration of indirect encoding schemes could be useful in the main optimisation phase too. The proposed tree-graph visualisation of the active population (Figure 5.1) provided valuable insight into the extent of plateau regions and showed, in its animated video form, how the population evolved through them. This suggested that further discrimination in the genotype space when carrying out non-dominated sorting may be helpful. Distance metrics were invoked for this purpose, with the Hamming distance shown to perform the best. While the Hamming distance is known to perform well on binary encodings, the finding that it was also superior on real encodings is enlightening. Moreover, it is convenient given that the Hamming distance is one of the cheapest metrics to compute. Data on the invocations of the genotype distance metric indicated its importance, despite it being low in the hierarchy of discriminating features called by the non-dominated sort routine. In the approach used, phenotype and genotype distances were integrated — at the lower levels of priority. Future work may look at the intelligent or adaptive weighting of one or the other. Also, while there are unlikely to be any cheaper metrics than the Hamming distance, more sophisticated and insightful distance metrics may be available that are worth the increased cost of their computation.

Chapter 5 also offered an assessment of some other perturbation operators. The prevailing issue with many of these operators was the preservation of solution feasibility. A small perturbation operator like `MuPFPR` (introduced in Section 4.3.4) can encapsulate feasibility checks inexpensively, due to only one lecture being reassigned. Batch reassignments, circular shifts and swaps require more complex feasibility checks. An alternative or enhanced encoding scheme may be able to alleviate this issue, but inevitably not without affecting the topology of the search landscape. The motivation behind exploring larger perturbation operators was to compensate somewhat for the absence of crossover in the genetic algorithm, as well as to encourage escape from local optima. Future study should compare the complexity of bespoke UCTP crossover operators with the feasibility checking required for mutation/perturbation operators. Using a faster, compiled language such as C or C++ could also give a better empirical idea as to what these limits are, as well as allowing for higher budget runs. The study in this section was limited to a destroy-repair operator plus three other operators. Future tests could involve other perturbations based on intelligent filtering of lectures (for example, by how much they contribute to the cost of a timetable). The work on an enhanced `swapPlace` showed that intelligently selecting lectures for operation holds promise. Inferring a preference on the lectures from simple features of the problem is one option. Another possible direction, bringing to mind the approach taken in Chapter 3, would be to harvest large amounts of data from an augmented set of benchmark problems and attempt to learn associations between the placement of lectures and the potential benefits of operating on them. These could then inform the choices made, either in a probabilistic or deterministic fashion. Tournament selection was successfully modified so as to encourage greater exploration. This was achieved through a combination of age and equivalence discrimination. The effects of this were seen in the final non-dominated sets that were, on average, more populous but without sacrificing their quality.

The development of a many-objective optimiser provided the groundwork for a study on robustness as a fifth objective. Disruptions, based on real world scenarios, were defined. All constraint types were accounted for in the action of the disruptors, barring those that were invariant under the encoding scheme. In future work, a different encoding could open up the possibility of more scenarios, such as a specific room being made unavailable. Nonetheless, the proposed disruptors covered a wide range of scenarios involving curricula, students, unavailable periods and more. They were also finely calibrated such that reevaluations on the changed problems would yield a representative mix of both lower quality and infeasible solutions, and not only exclusively the latter. Experiments using fixed scenarios led to two major findings. Firstly, the algorithm found perfectly robust solutions in nearly all cases, without any increase on the previously-used budget. However, this was often at the expense of the soft constraint objectives — **S2**, **S3** and **S4** more so than **S1**. A natural trade-off became apparent in the projected 2-D plots between robustness and quality for some problems, which supports similar findings in Akkan and Gülcü, 2018. The second key finding was that perfect robustness to fixed scenarios implied an improved robustness to generalised scenarios, when compared with solutions optimised in 4 objectives only. Therefore, solutions with greater robustness could be found, on average, without recourse to expensive resampling of disruptions or noisy estimators.

There are several directions for future work as regards robustness in many-objective timetabling. Firstly, the definition of robustness itself could be explored. The proposed metric captured a sense of deterioration in both quality and feasibility, without considering the cost of repair. One fairly crude but simple way to include this would be to return the number of lecture assignments that contribute towards hard constraint violations. An alternative would be to ascertain the minimum number of such lectures needing to be reassigned to restore feasibility. A sense of the consequent impact on soft constraint scores could be factored in too. This would lead to a measure of not only how badly solutions were damaged by disruptions, but also the least-cost option for restoring them. As with any idealised robustness metric, its exact calculation is inevitably prohibitively expensive. Some interesting ideas about noisy objectives and mitigating the chaos induced by such evaluations are presented in Rakshit et al., 2017. For example, the budget for resampling could be increased over the run according to a linear profile. An underlying assumption in this approach is that accuracy becomes more important as the optimiser hones in on a particularly promising region.

Finally, an in-depth analysis of the trade-offs between the five objectives will help inform future operator design. It is known that high quality solutions exist in 4-objective space, and yet some desirable characteristics were sacrificed for good robustness in the 5-objective experiments. It would be enlightening to harvest data on the robustness of high quality and best known solutions in order to further fill the gaps in knowledge on this front. The state of a population and its evolution history at any point in time could also potentially be harnessed for the adaptive selection of newly proposed operators. The ultimate aim would be to drive timetable quality, intelligently and in all dimensions, while maintaining the solid robustness that was successfully identified in this thesis.

Bibliography

H. A. Abbass (2001a). ‘MBO: Marriage in Honey Bees Optimization A Haplometrosis Polygynous Swarming Approach’. In: *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC* 1, pp. 207–214. DOI: 10.1109/cec.2001.934391 (Cited on page 56).

H. A. Abbass (2001b). ‘A Monogenous MBO Approach to Satisfiability’. In: *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA)* (Cited on page 56).

E. A. Abdelhalim and G. A. El Khayat (2016). ‘A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem (UGA)’. In: *Alexandria Engineering Journal* 55.2, pp. 1395–1409. DOI: 10.1016/j.aej.2016.02.017 (Cited on page 66).

S. Abdullah, E. K. Burke and B. McCollum (2005). ‘An Investigation of Variable Neighborhood Search for University Course Timetabling’. In: *Proceedings of the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications*, pp. 413–427 (Cited on page 45).

S. Abdullah, E. K. Burke and B. McCollum (2007). ‘Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for the University Course Timetabling Problem’. In: *Metaheuristics: Progress in Complex Systems Optimization*, pp. 153–169. DOI: 10.1007/978-0-387-71921-4_8 (Cited on pages 25 and 45).

S. Abdullah and H. Turabieh (2008). ‘Generating University Course Timetable Using Genetic Algorithms and Local Search’. In: *International Conference on Convergence Information Technology* 1, pp. 254–260. DOI: 10.1109/ICCIT.2008.379 (Cited on page 158).

S. Abdullah and H. Turabieh (2012). ‘On the use of multi neighbourhood structures within a Tabu-based memetic approach to university timetabling problems’. In: *Information Sciences* 191, pp. 146–168. DOI: 10.1016/j.ins.2011.12.018 (Cited on pages 58, 61, 62 and 183).

A. F. Abouelhamayed, A. S. Mahmoud, T. T. Shaaban, C. Salama and A. H. Yousef (2017). ‘An Enhanced Genetic Algorithm-Based Timetabling System with Incremental Changes’. In: *Proceedings of 11th International Conference on Computer Engineering and Systems (ICCES)*, pp. 122–127. DOI: 10.1109/ICCES.2016.7821985 (Cited on pages 47, 51 and 53).

A. Abuhamdah, M. Ayob, G. Kendall and N. R. Sabar (2014). ‘Population based Local Search for university course timetabling problems’. In: *Applied Intelligence* 40.1, pp. 44–53. DOI: 10.1007/s10489-013-0444-6 (Cited on page 57).

L. N. Ahmed, E. Özcan and A. Kheiri (2015). ‘Solving High School Timetabling Problems Worldwide Using Selection Hyper-heuristics’. In: *Expert Systems with Applications* 42.13, pp. 5463–5471. DOI: 10.1016/j.eswa.2015.02.059 (Cited on page 60).

H. E. Akbulut (2024). ‘A simulated annealing algorithm for the faculty-level university course timetabling problem’. In: *Pamukkale University Journal of Engineering Sciences* 30.1, pp. 17–30. DOI: 10.5505/pajes.2023.00483 (Cited on page 43).

C. Akkan and A. Gülcü (2018). ‘A bi-criteria hybrid Genetic Algorithm with robustness objective for the course timetabling problem’. In: *Computers and Operations Research* 90, pp. 22–32. DOI: 10.1016/j.cor.2017.09.007 (Cited on pages 47, 50, 51, 64 and 220).

C. Akkan, A. Gülcü and Z. Kuş (2020). ‘Search Space Sampling by Simulated Annealing for Identifying Robust Solutions in Course Timetabling’. In: *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–10. DOI: 10.1109/CEC48606.2020.9185823 (Cited on page 198).

C. Akkan, A. Gülcü and Z. Kuş (2021). ‘Slack-based Robustness Estimators for the Curriculum-Based Course Timetabling Problem’. In: *Proceedings of the 13th*

International Conference on the Practice and Theory of Automated Timetabling (PATAT), pp. 147–158 (Cited on page 199).

C. Akkan, A. Gülcü and Z. Kuş (2022). ‘Bi-criteria simulated annealing for the curriculum-based course timetabling problem with robustness approximation’. In: *Journal of Scheduling* 25.4, pp. 477–501. DOI: 10.1007/s10951-022-00722-0 (Cited on page 199).

M. S. Al-Ashhab and A. Abdulrahman (2018). ‘Two-Stage Multi-Objective University Courses Timetabling Using Genetic Algorithms’. In: *International Journal of Engineering and Technology* 10.4, pp. 1102–1111. DOI: 10.21817/ijet/2018/v10i4/181004030 (Cited on page 53).

C. H. Aladag, G. A. Hocaoglu and M. Basaran (2009). ‘The effect of neighborhood structures on tabu search algorithm in solving course timetabling problem.’ In: *Expert Systems with Applications* 36.10, pp. 12,349–12,356. DOI: 10.1016/j.eswa.2009.04.051 (Cited on page 40).

N. Alhuwaishel and M. Hosny (2011). ‘A Hybrid Bees/Demon Optimization Algorithm for Solving the University Course Timetabling Problem’. In: *College of Computer and Information Sciences, King Saud University, Riyadh, Kingdom of Saudi Arabia* (Cited on page 13).

A. Alkan and E. Özcan (2003). ‘Memetic Algorithms for Timetabling’. In: *Proceedings on IEEE Congress on Evolutionary Computation* 3, pp. 1,796–1,802. DOI: 10.1109/CEC.2003.1299890 (Cited on page 158).

J. Almeida, J. R. Figueira, A. P. Francisco and D. Santos (2023). ‘A hybrid meta-heuristic for the generation of feasible large-scale course timetables using instance decomposition’. In: *arXiv*, pp. 1–43. DOI: 10.48550/arXiv.2310.20334 (Cited on page 45).

R. Alvarez-Valdes, E. Crespo and J. M. Tamarit (2002). ‘Design and implementation of a course scheduling system using Tabu Search’. In: *European Journal of Operational Research* 137.3, pp. 512–523. DOI: 10.1016/S0377-2217(01)00091-1 (Cited on pages 40, 41 and 65).

A. Ariyazand, H. Soleimani, F. Etebari and E. Mehdizadeh1 (2022). 'Optimization of a multi-objective university course timetabling problem with a hy-brid WOANSGA-II (Case study: IAU, Robat Karim branch)'. In: *Journal of Industrial Engineering and Management Studies* 10.2, pp. 131–148 (Cited on pages 12 and 61).

S. Asiyaban and Z. Mousavinasab (2012). 'University Course Timetabling using Multi-population Genetic Algorithm Guided with Local Search and Fuzzy Logic.' In: *International Journal of Computers and Technology* 11.10, pp. 3,043–3,050 (Cited on pages 48, 50 and 53).

M. Assi, B. Halawi and R. A. Haraty (2018). 'Genetic Algorithm Analysis using the Graph Coloring Method for Solving the University Timetable Problem'. In: *Procedia Computer Science* 126, pp. 899–906. DOI: 10.1016/j.procS.2018.08.024 (Cited on pages 50 and 52).

M. Atsuta, K. Nonobe and T. Ibaraki (2008). 'ITC-2007 Track 2: An Approach using General CSP Solver'. In: *Proceedings of the Practice and Theory of Automated Timetabling* (Cited on page 61).

P. Avella and I. Vasil'ev (2005). 'A Computational Study of a Cutting Plane Algorithm for University Course Timetabling'. In: *Journal of Scheduling* 8.6, pp. 497–514. DOI: 10.1007/s10951-005-4780-1 (Cited on page 25).

F. H. Awad, A. Al-Kubaisi and M. Mahmood (2022). 'Large-scale timetabling problems with adaptive tabu search'. In: *Journal of Intelligent Systems* 31.1, pp. 168–176. DOI: 10.1515/jisys-2022-0003 (Cited on page 42).

E. Aycan and T. Ayav (2008). 'Solving the course scheduling problem using simulated annealing'. In: *IEEE International Advance Computing Conference (IACC)*, pp. 462–466 (Cited on page 43).

M. Ayob and G. Jaradat (2009). *Hybrid Ant Colony Systems For Course Timetabling Problems.* 2nd Conference on Data Mining and Optimization, pp 120-126 (Cited on pages 73 and 74).

H. Babaei and A. Hadidi (2014). 'A Review of Distributed Multi-Agent Systems Approach to Solve University Course Timetabling Problem'. In: *Advances in*

Computer Science: an International Journal (ACSIJ) 3.5, pp. 19–28 (Cited on page 58).

H. Babaei, J. Karimpour and A. Hadidi (2014). ‘A survey of approaches for university course timetabling problem’. In: *Computers and Industrial Engineering* 86, pp. 43–59 (Cited on page 26).

H. Babaei, J. Karimpour and A. Hadidi (2019). ‘Generating an optimal timetabling for multi-departments common lecturers using hybrid fuzzy and clustering algorithms’. In: *Soft Computing* 23.13, pp. 4,735–4,747 (Cited on page 25).

J. Bader and E. Zitzler (2011). ‘HypE : An algorithm for fast optimization’. In: *Evolutionary Computation* 19.1, pp. 45–76. DOI: `10.1162/EVCO_a_00009` (Cited on page 147).

R. P. Badoni, D. K. Gupta and P. Mishra (2014). ‘A new hybrid algorithm for university course timetabling problem using events based on groupings of students’. In: *Computers & Industrial Engineering* 78, pp. 12–25 (Cited on pages 14 and 28).

R. P. Badoni, S. Kumar, M. Mann, R. P. Mohanty and A. Sarangi (2023). ‘Ant colony optimization algorithm for the university course timetabling problem using events based on groupings of students’. In: *Modeling and Applications in Operations Research* February 2024, pp. 1–36. DOI: `10.1201/9781003462422-1` (Cited on page 73).

M. Banbara, K. Inoue, B. Kaufmann, T. Okimoto, T. Schaub, T. Soh, N. Tamura and P. Wanko (2019). ‘teaspoon: solving the curriculum-based course timetabling problems with answer set programming’. In: *Annals of Operations Research* 275, pp. 3–37. DOI: `10.1007/S10479-018-2757-7` (Cited on page 38).

M. Banbara, T. Soh, N. Tamura, K. Inoue and T. Schaub (2013). ‘Answer set programming as a modeling language for course timetabling’. In: *Theory and Practice of Logic Programming* 13.4-5, pp. 783–798. DOI: `10.1017/S1471068413000495` (Cited on page 38).

A. Bashab, A. O. Ibrahim, E. E. AbedElgabar, M. A. Ismail, A. Elsafi, A. Ahmed and A. Abraham (2020). ‘A systematic mapping study on solving university timetabling problems using meta-heuristic algorithms'. In: 32.23, pp. 17,397–17,432. DOI: 10.1007/s00521-020-05110-3 (Cited on pages 64, 65 and 66).

R. Bellio, S. Ceschia, L. di Gaspero, A. Schaerf and T. Urli (2016). 'Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem'. In: *Computers and Operations Research* 65, pp. 83–92. DOI: 10.1016/j.cor.2015.07.002 (Cited on page 44).

A. Bettinelli, V. Cacchiani, R. Roberti and P. Toth (2015). 'An overview of curriculum-based course timetabling'. In: *TOP* 23.2, pp. 313–349. DOI: 10.1007/s11750-015-0366-z (Cited on page 33).

L. Bianchi, M. Dorigo, L. M. Gambardella and W. J. Gutjahr (2009). 'A survey on metaheuristics for stochastic combinatorial optimization'. In: *Natural Computing* 8.2, pp. 239–287. DOI: 10.1007/s11047-008-9098-4 (Cited on page 39).

C. Blum and A. Roli (2003). 'Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison'. In: *ACM Computing Surveys* 35, pp. 268–308 (Cited on page 77).

A. Bonutti, F. De Cesco, L. di Gaspero and A. Schaerf (2012). 'Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results'. In: *Annals of Operations Research* 194.1, pp. 59–70. DOI: 10.1007/s10479-010-0707-0 (Cited on pages 17 and 18).

L. Breiman, J. Friedman, R. Olshen and C. J. Stone (1983). *Classification and Regression Trees*. Wadsworth International Group (Cited on pages 95 and 103).

D. Brélaz (1979). 'New methods to color the vertices of a graph'. In: *Communications of the ACM* 22.4, pp. 251–256. DOI: 10.1145/359094.359101 (Cited on page 28).

J. A. Breslaw (1976). 'A linear programming solution to the faculty assignment problem'. In: 10.6, pp. 227–230. DOI: 10.1016/0038-0121(76)90008-2 (Cited on page 31).

V. E. Budi Darmawan, Y. W. Chen, A. Larasati, D. Prastyo and A. Dwiastuti (2019). 'Multi-objective Modeling for a Course Timetabling Problem'. In: *Proceedings of the International Conference on Creative Economics, Tourism and Information*

Management (ICCETIM), pp. 10–14. DOI: 10.5220/0009857300100014 (Cited on page 118).

B. Bullnheimer, R. Hartl and C. Strauss (1999). 'A New Rank Based Version of the Ant System: A Computational Study.' In: *Central European Journal for Operations Research and Economics* 7.1, pp. 25–38 (Cited on page 76).

E. Burke, K. Jackson, J. Kingston and R. Weare (1997). 'Automated University Timetabling: The State of the Art'. In: *The Computer Journal* 40.9, 565—571. DOI: 10.1093/comjnl/40.9.565 (Cited on page 12).

E. K. Burke, J. Mareček, A. J. Parkes and H. Rudová (2010). 'Decomposition, Reformulation, and Diving in University Course Timetabling'. In: *Computers & Operations Research* 37.3. DOI: 10.1016/j.cor.2009.02.023 (Cited on pages 31 and 33).

E. K. Burke, J. Mareček, A. J. Parkes and H. Rudová (2012). 'A branch-and-cut procedure for the Udine Course Timetabling problem'. In: *Annals of Operations Research* 194.1, pp. 71–87. DOI: 10.1007/s10479-010-0828-5 (Cited on page 33).

E. K. Burke, B. Mccollum, A. Meisels, S. Petrovic and R. Qu (2007). 'A Graph-Based Hyper-Heuristic for Educational Timetabling Problems'. In: *European Journal of Operational Research* 176, pp. 177–192 (Cited on page 76).

V. Cacchiani, A. Caprara, R. Roberti and P. Toth (2013). 'A new lower bound for curriculum-based course timetabling'. In: *Computers & Operations Research* 40.10, pp. 2,466–2,477. DOI: 10.1016/j.cor.2013.02.010 (Cited on page 33).

C. D. Cantrell (2000). *Modern mathematical methods for physicists and engineers.* Cambridge University Press (Cited on page 121).

M. W. Carter (1989). 'A Lagrangian Relaxation Approach To The Classroom Assignment Problem'. In: *Information Systems and Operational Research (INFOR)* 27.2, pp. 230–246. DOI: 10.1080/03155986.1989.11732094 (Cited on page 31).

S. Ceschia, L. di Gaspero and A. Schaerf (2012). 'Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem'. In: *Computers & Operations Research* 39.7, pp. 1,615–1,624. DOI: 10.1016/j.cor.2011.09.014 (Cited on page 44).

S. Ceschia, L. di Gaspero and A. Schaerf (2023). 'Educational timetabling: Problems, benchmarks, and state-of-the-art results'. In: *European Journal of Operational Research* 308.1, pp. 1–18. DOI: 10.1016/j.ejor.2022.07.011 (Cited on pages 11, 12, 26, 61, 62, 63 and 64).

N. Chahal and D. de Werra (1989). 'An interactive system for constructing timetables on a PC'. In: *European Journal of Operational Research* 40.1, pp. 32–37. DOI: 10.1016/0377-2217(89)90269-5 (Cited on page 29).

A. Chaudhuri and K. De (2010). 'Fuzzy genetic heuristic for university course timetable problem'. In: *International Journal of Advances in Soft Computing and its Applications* 2.1, pp. 100–123 (Cited on page 48).

C. Chen and J. Chou (2017). 'Multiobjective Optimization of Airline Crew Roster Recovery Problems Under Disruption Conditions'. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.1, pp. 133–144. DOI: 10.1109/TSMC.2016.2560130 (Cited on page 159).

C. Chen, T. Liu and J. Chou (2013). 'Integrated Short-Haul Airline Crew Scheduling Using Multiobjective Optimization Genetic Algorithms'. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43, pp. 1,077–1,090. DOI: 10.1109/TSMC.2012.2234943 (Cited on page 159).

M. C. Chen, S. N. Sze, S. L. Goh, N. R. Sabar and G. Kendall (2021). 'A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities'. In: *IEEE Access* 9, pp. 106,515–106,529. DOI: 10.1109/ACCESS.2021.3100613 (Cited on pages 26, 56 and 64).

R. M. Chen and H. F. Shih (2013). 'Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search'. In: *Algorithms* 6.2, pp. 227–244. DOI: 10.3390/a6020227 (Cited on page 55).

W. Chinnasri, S. Krootjohn and N. Sureerattanan (2012). 'Performance Study of Genetic Operators on University Course Timetabling Problem'. In: *International*

Journal of Advancements in Computing Technology 4.20, pp. 61–71. DOI: 10.4156/ijact.vol4.issue20.8 (Cited on pages 47 and 48).

N. Chmait and K. Challita (2013). 'Using Simulated Annealing and Ant-Colony Optimization Algorithms to Solve the Scheduling Problem'. In: *Computer Science and Information Technology* 1.3, pp. 208–224. DOI: 10.13189/csit.2013.010307 (Cited on page 78).

M. Clark, M. Henz and B. Love (2008). 'QuikFix A Repair-based Timetable Solver'. In: *7th International Conference on the Practice and Theory of Automated Timetabling (PATAT)* (Cited on page 61).

A. Colorni, M. Dorigo and V. Maniezzo (1992). *A genetic algorithm to solve the timetable problem.* Technical Report, Politecnico di Milano, Italy (Cited on page 63).

T. B. Cooper and J. H. Kingston (1996). 'The Complexity of Timetable Construction Problems'. In: pp. 283–295 (Cited on page 2).

S. Daskalaki, T. Birbas and E. Housos (2004). 'An integer programming formulation for a case study in university timetabling'. In: *European Journal of Operational Research* 153.1, pp. 117–135. DOI: 10.1016/S0377-2217(03)00103-6 (Cited on page 25).

D. Datta, K. Deb and C. M. Fonseca (2007). 'Multi-Objective Evolutionary Algorithm for University Class Timetabling Problem'. In: *Evolutionary Scheduling*, pp. 197–236 (Cited on page 60).

D. de Werra (1985). 'An introduction to timetabling'. In: 19, pp. 151–162 (Cited on page 28).

K. Deb (2001). *Multiobjective Optimization Using Evolutionary Algorithms.* Wiley, New York (Cited on pages 157 and 159).

K. Deb and H. Jain (2013). 'An Evolutionary Many-Objective Optimization Algorithm Using Reference-point Based Non-dominated Sorting Approach, Part I: Solving Problems with Box Constraints'. In: *IEEE Transactions on Evolutionary Computation* 18.4, pp. 577–601. DOI: 10.1109/TEVC.2013.2281535 (Cited on pages 118, 135, 159 and 163).

K. Deb, A. Pratap, S. Agarwal and T. Meyarivan (2002). 'A fast and elitist multiobjective genetic algorithm: NSGA-II'. In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197. DOI: `10.1109/4235.996017` (Cited on page 135).

K. Deb and S. Tiwari (2008). 'Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization'. In: *European Journal of Operational Research* 185.3, pp. 1,062–1,087. DOI: `https://doi.org/10.1016/j.ejor.2006.06.042` (Cited on page 160).

S. Deris, S. Omatu and H. Ohta (2000). 'Timetable planning using the constraint-based reasoning'. In: *Computers and Operations Research* 27.9, pp. 819–840. DOI: `10.1016/S0305-0548(99)00051-9` (Cited on page 35).

S. Deris, S. Omatu, H. Ohta and P. Saada (1999). 'Incorporating constraint propagation in genetic algorithm for university timetable planning'. In: *Engineering Applications of Artificial Intelligence* 12.3, pp. 241–253. DOI: `10.1016/S0952-1976(99)00007-X` (Cited on page 34).

C. Desai and S. S. Williamson (2009). 'Optimal design of a parallel Hybrid Electric Vehicle using multi-objective genetic algorithms'. In: *2009 IEEE Vehicle Power and Propulsion Conference*, pp. 871–876. DOI: `10.1109/VPPC.2009.5289754` (Cited on page 160).

L. di Gaspero and A. Schaerf (2003). 'Multi-neighbourhood local search with application to course timetabling'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2740, pp. 262–275. DOI: `10.1007/978-3-540-45157-0_17` (Cited on page 41).

M. Dimopoulou and P. Miliotis (2004). 'An automated university course timetabling system developed in a distributed environment : A case study'. In: *European Journal of Operational Research* 153.1, pp. 136–147. DOI: `10.1016/S0377-2217(03)00104-8` (Cited on page 25).

J. J. Dinkel, J. Mote and M. A. Venkataramanan (1989). 'An efficient decision support system for academic course scheduling.' In: *Operations Research* 37.6, pp. 853–864 (Cited on pages 29 and 30).

D. Djamarus and K. R. Ku-Mahamud (2009). 'Heuristic Factors in Ant System Algorithm for Course Timetabling Problem'. In: *Ninth International Conference on Intelligent Systems Design and Applications*, 232—236. DOI: 10.1109/ISDA.2009.62 (Cited on page 78).

M. Dorigo (1992). *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, Italy (Cited on page 54).

G. Dueck (1993). 'The Great Deluge Algorithm and the Record-to-Record Travel'. In: *Journal of Computational Physics* 104.1, pp. 86–92 (Cited on page 56).

G. Dueck and T. Scheuer (1990). 'Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing'. In: *Journal of Computational Physics* 90.1, pp. 161–175. DOI: 10.1016/0021-9991(90)90201-B (Cited on page 44).

J. S. Dyer and J. M. Mulvey (1976). 'The implementation of an integrated optimization/information system for academic departmental planning.' In: *Management Science* 22, pp. 1332–1341 (Cited on page 29).

H. A. Eiselt and G. Laporte (1987). 'Combinatorial Optimization Problems with Soft and Hard Requirements'. In: *Journal of the Operational Research Society* 38.9, pp. 785–795 (Cited on page 26).

J. A. Ferland and S. Roy (1985). 'Timetabling problem for university as assignment of activity to resources'. In: *Computers and Operational Research* 12.2, pp. 207–218 (Cited on page 31).

T. Feutrier, N. Veerapen and M. E. Kessaci (2023). 'When Simpler is Better: Automated Configuration of a University Timetabling Solver'. In: *IEEE Congress on Evolutionary Computation (CEC).* DOI: 10.1109/CEC53210.2023.10253986 (Cited on page 59).

J. E. Fieldsend (2020). 'Data structures for non-dominated sets: Implementations and empirical assessment of two decades of advances'. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 489–497. DOI: 10.1145/3377930.3390150 (Cited on page 144).

L. R. Ford and D. R. Fulkerson (1962). *Flows in Networks.* Princeton University Press (Cited on page 30).

J. Frausto-Solis, F. Alonso-Pecina, M. Larre, C. M. Gonzalez-Segura and J. L. Gomez-Ramos (2006). 'Three heuristics to solve Timetabling'. In: *Proceedings of the 10th WSEAS International Conference on Communications*, pp. 564–570 (Cited on page 1).

M. Fotovvati and S. Mirghaderi (2023). 'Automated University Course Timetabling Using Hyper-heuristic Approach'. In: *Sharif Journal of Industrial Engineering amp; Management* 39.1, pp. 155–167. DOI: `10.24200/j65.2022.57866.2212` (Cited on page 76).

P. Garg (2009). 'A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm'. In: *International Journal of Network Security and Its Applications (IJNSA)* 1.1, pp. 34–42 (Cited on page 26).

L. di Gaspero, A. Schaerf and B. McCollum (2007). 'The Second International Timetabling Competition: Curriculum-based Course Timetabling (Track 3)'. In: *Proceedings of the 1st International Workshop on Scheduling a Scheduling Competition*, pp. 1–21 (Cited on page 14).

M. J. Geiger (2008). 'An application of the Threshold Accepting metaheuristic for curriculum based course timetabling'. In: *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT).* DOI: `10.48550/arXiv.0809.0757` (Cited on page 61).

M. J. Geiger (2009). 'Multi-criteria Curriculum-Based Course Timetabling — A Comparison of a Weighted Sum and a Reference Point Based Approach'. In: *Proceedings of Evolutionary Multi-Criterion Optimization, 5th International Conference (EMO)*, pp. 290–304 (Cited on pages 120, 148, 149 and 152).

M. J. Geiger (2012). 'Applying the threshold accepting metaheuristic to curriculum based course timetabling'. In: *Annals of Operations Research* 194.1, pp. 189–202. DOI: `10.1007/s10479-010-0703-4` (Cited on pages 44, 120, 142 and 152).

S. L. Goh, G. Kendall and N. R. Sabar (2019). ‘Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem’. In: *Journal of the Operational Research Society* 70.6, pp. 873–888. DOI: `10.1080/01605682.2018.1468862` (Cited on page 44).

S. L. Goh, G. Kendall, N. R. Sabar and S. Abdullah (2020). ‘An effective hybrid local search approach for the post enrolment course timetabling problem’. In: *Opsearch* 57.4, pp. 1,131–1,163. DOI: `10.1007/s12597-020-00444-x` (Cited on pages 44 and 45).

P. Golding, S. Kapadia, S. Naylor, J. Schulz, H. R. Maier, U. Lall and M. van der Velde (2017). ‘Framework for minimising the impact of regional shocks on global food security using multi-objective ant colony optimisation’. In: *Environmental Modelling and Software* 95, pp. 303–319. DOI: `10.1016/j.envsoft.2017.06.004` (Cited on page 77).

C. C. L. B. G. Gotlib (1963). ‘The construction of class-teacher timetables’. In: *Proceedings of IFIP Congrass* 62, pp. 73–77 (Cited on page 2).

A. A. Gozali and S. Fujimura (2020). ‘Solving University Course Timetabling Problem Using Multi-Depth Genetic Algorithm’. In: *SHS Web of Conferences* 77.9. DOI: `10.1051/shsconf/20207701001` (Cited on page 61).

A. Grech and J. Main (2005). ‘A case-based reasoning approach to formulating university timetables using genetic algorithms’. In: *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* 3681, pp. 76–83. DOI: `10.1007/11552413_12` (Cited on page 58).

G. R. Greenfield (2003). ‘Evolving aesthetic images using multiobjective optimization’. In: *Proceedings of the IEEE Congress on Evolutionary Computation* 3, pp. 1,903–1,909. DOI: `10.1109/CEC.2003.1299906` (Cited on page 159).

S. S. Habashi and A. H. Yousef (2018). ‘Approach for Timetabling Problems’. In: *Proceedings of IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 259–266 (Cited on page 60).

M. Hafsa, P. Wattebled, J. Jacques and L. Jourdan (2021). ‘A Multi-Objective Evolutionary Approach to Professional Course Timetabling: A Real-World Case

Study'. In: *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 997–1004 (Cited on page 120).

J. K. Hao and U. Benlic (2011). 'Lower Bounds for the ITC-2007 Curriculum-Based Course Timetabling Problem'. In: *European Journal of Operational Research* 212.3, pp. 464–472. DOI: `10.1016/j.ejor.2011.02.019` (Cited on pages 34 and 162).

A. Hertz (1991). 'Tabu search for large scale timetabling problems'. In: *European Journal of Operational Research* 54.1, pp. 39–47 (Cited on page 26).

A. Hertz (1992). 'Finding a feasible course schedule using tabu search'. In: *Discrete Applied Mathematics* 35.3, pp. 255–270 (Cited on page 40).

Z. Houhamdi, B. Athamena, R. Abuzaineddin and M. Muhairat (2019). 'A multi-agent system for course timetable generation'. In: *TEM* 8.1, pp. 211–221. DOI: `10.18421/TEM81-30` (Cited on page 58).

R. Ilyas and Z. Iqbal (2015). 'Study of hybrid approaches used for university course timetable problem (UCTP)'. In: *Proceedings of the 10th IEEE Conference on Industrial Electronics and Applications, (ICIEA)*, pp. 696–701. DOI: `10.1109/ICIEA.2015.7334198` (Cited on pages 26, 53 and 58).

S. Imran Hossain, M. A. Akhand, M. I. Shuvo, N. Siddique and H. Adeli (2019). 'Optimization of University Course Scheduling Problem using Particle Swarm Optimization with Selective Search'. In: *Expert Systems with Applications* 127, pp. 9–24. DOI: `10.1016/j.eswa.2019.02.026` (Cited on page 55).

A. Jaszkiewicz and T. Lust (2018). 'ND-Tree-Based Update : A Fast Algorithm for the Dynamic Nondominance Problem'. In: *IEEE Transactions on Evolutionary Computation* 22.5, pp. 778–791 (Cited on page 144).

N. S. Jat and Y. Shengxiang (2008). 'A memetic algorithm for the university course timetabling problem'. In: *Proceedings of 20th IEEE international conference on tools with artificial intelligence*, pp. 427–433. DOI: `10.1109/ICTAI.2008.126` (Cited on pages 53 and 54).

M. Joudaki, M. Imani and N. Mazhari (2011). 'Using improved Memetic Algorithm and local search to solve University Course Timetabling Problem (UCTTP)'. In:

Procedings of the International Conference on Artificial Intelligence (ICAI) 2 (Cited on pages 39 and 54).

W. Junginger (1986). ‘Timetabling in Germany - a survey’. In: *Interfaces* 16, pp. 66–74 (Cited on page 29).

E. H. Kampke, W. D. S. Rocha, M. C. S. Boeres and M. C. Rangel (2015). ‘A GRASP algorithm with Path Relinking for the University Courses Timetabling Problem’. In: *Proceeding Series of the Brazilian Society for Computational and Applied Mathematics* 3.2. DOI: 10.5540/03.2015.003.02.0108 (Cited on page 31).

E. H. Kampke, L. M. Scheideger, G. R. Mauri and M. C. S. Boeres (2019). ‘A Network Flow Based Construction for a GRASP+SA Algorithm to Solve the University Timetabling Problem’. In: *Lecture Notes in Computer Science* 11621, pp. 215–231. DOI: 10.1007/978-3-030-24302-9_16 (Cited on pages 30 and 35).

F. Karami and A. Dariane (2022). ‘A review and evaluation of multi and many-objective optimization: Methods and algorithms’. In: *Global Journal of Ecology* 7.2, pp. 104–119. DOI: 10.17352/gje.000070 (Cited on page 118).

A. Kiefer, R. F. Hartl and A. Schnell (2017). ‘Adaptive large neighborhood search for the curriculum-based course timetabling problem’. In: *Annals of Operations Research* 252.2, pp. 255–282. DOI: 10.1007/s10479-016-2151-2 (Cited on pages 59, 61 and 62).

S. Kirkpatrick, C. D. Gellat and M. P Vecchi (1983). ‘Optimization by simulated annealing’. In: *Science* 4598.220, pp. 671–680 (Cited on page 44).

M. S. Kohshori, D. Zeynolabedini, M. S. Liri and L. Jadidi (2012). ‘Multi Population Hybrid Genetic Algorithms for University Course Timetabling Problem’. In: *International Journal of Information Technology and Computer Science* 4.6, pp. 1–11. DOI: 10.5815/ijitcs.2012.06.01 (Cited on pages 50 and 53).

P. Kostuch (2005). ‘The university course timetabling problem with a three-phase approach’. In: *Proceedings of the 5th International Conference on the Practise and Theory of Automated Timetabling (PATAT)*, pp. 109–125 (Cited on page 58).

G. Lach and M. E. Lübbecke (2012). ‘Curriculum based course timetabling: New solutions to Udine benchmark instances’. In: *Annals of Operations Research* 194.1, pp. 255–272. DOI: `10.1007/s10479-010-0700-7` (Cited on page 34).

M. Laszczyk and P. B. Myszkowski (2019). ‘Improved selection in evolutionary multi–objective optimization of multi–skill resource–constrained project scheduling problem’. In: *Information Sciences* 481, pp. 412–431. DOI: `https://doi.org/10.1016/j.ins.2019.01.002` (Cited on page 158).

N. L. Lawrie (1969). ‘An integer linear programming model of a school timetabling problem’. In: *The Computer Journal* 12.4, pp. 307–316. DOI: `10.1093/comjnl/12.4.307` (Cited on page 31).

R. Lewis (2008a). ‘A time-dependent metaheuristic algorithm for post enrolment-based course timetabling’. In: *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, (PATAT)*, pp. 1–15 (Cited on page 43).

R. Lewis (2006). *Metaheuristics for University Course Timetabling.* PhD thesis, Napier University (Cited on page 14).

R. Lewis (2008b). ‘A survey of metaheuristic-based techniques for University Timetabling problems’. In: *OR Spectrum* 30.1, pp. 167–190. DOI: `10.1007/s00291-007-0097-0` (Cited on page 26).

R. Lewis and B. Paechter (2002). ‘New Crossover Operators for Timetabling with Evolutionary Algorithms’. In: *Practice* 44, pp. 1–6 (Cited on page 51).

R. Lewis and B. Paechter (2005). ‘Application of the grouping genetic algorithm to University Course Timetabling’. In: *Lecture Notes in Computer Science* 3448, pp. 144–153. DOI: `10.1007/978-3-540-31996-2_14` (Cited on pages 13, 26 and 74).

R. Lewis and B. Paechter (2006). ‘Finding Feasible Timetables Using Group-Based Operators’. In: *IEEE Transactions on Evolutionary Computation* 11.3, pp. 397–413. DOI: `10.1109/tevc.2006.885162` (Cited on page 44).

M. Lindahl, M. Sørensen and T. R. Stidsen (2018). 'A fix-and-optimize matheuristic for university timetabling'. In: *Journal of Heuristics* 24.4, pp. 645–665. DOI: 10.1007/s10732-018-9371-3 (Cited on pages 34, 61 and 62).

L. Lopes and K. Smith-Miles (2010). 'Pitfalls in Instance Generation for Udine Timetabling'. In: *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* 6073, pp. 299–302. DOI: 10.1007/978-3-642-13800-3_31 (Cited on page 44).

H. R. Lourenço, O. C. Martin and T. Stützle (2010). 'Iterated Local Search: Framework and Applications'. In: *Handbook of Metaheuristics*. Springer US, pp. 363–397. DOI: 10.1007/978-1-4419-1665-5_12 (Cited on page 197).

Z. Lü and J. K. Hao (2010). 'Adaptive Tabu Search for course timetabling'. In: *European Journal of Operational Research* 200.1, pp. 235–244. DOI: 10.1016/j.ejor.2008.12.007 (Cited on pages 41, 42 and 44).

Z. Lü, J. K. Hao and F. Glover (2011). 'Neighborhood analysis: A case study on curriculum-based course timetabling'. In: *Journal of Heuristics* 17.2, pp. 97–118. DOI: 10.1007/s10732-010-9128-0 (Cited on page 41).

T. Lutuksin, A. Chainual and P. Pongcharoen (2009). 'Experimental Design and Analysis on Parameter Investigation and Performance Comparison of Ant Algorithms for Course Timetabling Problem'. In: *Naresuan University Engineering Journal* 4.1, pp. 31–38 (Cited on page 76).

V. W. Marek and M. Truszczynski (1999). 'Stable models and an alternative logic programming paradigm'. In: *The Logic Programming Paradigm*, pp. 375–398. DOI: 10.48550/arXiv.cs/9809032 (Cited on page 36).

V. D. Matijas, G. Molnar, M. Cupic, D. Jakobovic and B. Dalbelo Basic (2009). 'University Course Timetabling Using ACO: A Case Study on Laboratory Exercises'. In: *Proceedings of the 14th international conference on Knowledge-based and intelligent information and engineering systems*, pp. 100–110. DOI: 10.1007/978-3-642-15387-7_14 (Cited on pages 73, 74 and 75).

N. R. Maya, J. J. Flores and H. R. Rangel (2016). 'Performance Comparison of Evolutionary Algorithms for University Course Timetabling Problem'. In: *Com-*

putacion y Sistemas 20.4, pp. 623–634. DOI: 10.13053/CyS-20-4-2504 (Cited on page 25).

A. Mayer, C. Nothegger, A. Chwatal and G. Raidl (2012). 'Solving the post enrolment course timetabling problem by ant colony optimization'. In: *Annals of Operations Research* 194.1, pp. 325–339. DOI: 10.1007/s10479-012-1078-5 (Cited on pages 17, 75, 80 and 81).

R. H. McClure and C. E. Wells (1984). 'A mathematical programming model for faculty course assignment'. In: *Decision Science* 15, pp. 409–420 (Cited on page 31).

N. Mladenović and P. Hansen (1997). 'Variable neighborhood search'. In: *Computers Operations Research* 24.11, pp. 1,097–1,100. DOI: https://doi.org/10.1016/S0305-0548(97)00031-2 (Cited on page 45).

M. Mühlenthaler and R. Wanka (2016). 'Fairness in academic course timetabling'. In: *Annals of Operations Research* 239.1, pp. 171–188. DOI: 10.1007/s10479-014-1553-2 (Cited on page 64).

T. Müller (2009). 'ITC2007 solver description: A hybrid approach'. In: *Annals of Operations Research* 172.1, pp. 429–446. DOI: 10.1007/s10479-009-0644-y (Cited on pages 61 and 88).

T. Müller, R. Bartak and H. Rudová (2004). 'Conflict-based statistics'. In: *Proceedings of the EU/ME Workshop on Design and Evaluation of Advanced Hybrid Metaheuristics* 201 (Cited on page 35).

T. Müller, H. Rudová and Z. Müllerová (2018). 'University course timetabling and International Timetabling Competition 2019'. In: *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pp. 5–31 (Cited on pages 11 and 19).

J. M. Mulvey (1982). 'A Classrooms/Time Assignment Model'. In: *European Journal of Operational Research* 9, pp. 64–70 (Cited on page 29).

M. Munirah, M. Mokhairi, F. K. Ahmad, K. Ahmad and M. A. Mohamed (2019). 'University course timetabling model using ant colony optimization algorithm approach'. In: *Indonesian Journal of Electrical Engineering and Computer Science* 13.1, pp. 72–76 (Cited on page 76).

P. B. Myszkowski and M. Laszczyk (2021). 'Diversity based selection for many-objective evolutionary optimisation problems with constraints'. In: *Information Sciences* 546, pp. 665–700. DOI: `10.1016/j.ins.2020.08.118` (Cited on page 158).

J. A. Nelder and R. Mead (1965). 'A Simplex Method for Function Minimization'. In: *The Computer Journal* 7.4, pp. 308–313. DOI: `10.1093/comjnl/7.4.308` (Cited on page 56).

G. A. Neufeld and J. Tartar (1974). 'Graph coloring conditions for the existence of solutions to the timetable problem'. In: *Communications of the ACM* 17.8, pp. 450–453 (Cited on page 27).

D. C. H. Nguyen, H. R. Maier, G. C. Dandy and J. C. Ascough (2016). 'Framework for computationally efficient optimal crop and water allocation using ant colony optimization'. In: *Environmental Modelling and Software* 76, pp. 37–53. DOI: `10.1016/j.envsoft.2015.11.003` (Cited on page 77).

I. Niemelä (1999). 'Logic programs with stable model semantics as a constraint programming paradigm'. In: *Annals of Mathematics and Artificial Intelligence* 25.3/4, pp. 241–273 (Cited on page 36).

H. E. Nouri and O. B. Driss (2016). 'MATP: A multi-agent model for the university timetabling problem'. In: *Advances in Intelligent Systems and Computing* 465, pp. 11–22. DOI: `10.1007/978-3-319-33622-0_2` (Cited on page 58).

J. H. Obit (2010). *Developing novel meta-heuristic, hyper-heuristic and cooperative search for course timetabling problems.* PhD Thesis, Universiti Malaysia Sabah (Cited on page 60).

R. A. Oude Vrielink, E. A. Jansen, E. W. Hans and J. van Hillegersberg (2019). 'Practices in timetabling in higher education institutions: a systematic review'. In: *Annals of Operations Research* 275.1, pp. 145–160. DOI: `10.1007/s10479-017-2688-8` (Cited on pages 2 and 158).

B. Paechter, R. C. Rankin, A. Cumming and T. C. Fogarty (1998). ‘Timetabling the classes of an entire university with an evolutionary algorithm’. In: *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* 1498, pp. 865–874. DOI: `10.1007/bfb0056928` (Cited on page 48).

J. Pandey and A. K. Sharma (2016). ‘Survey on University timetabling problem’. In: *Proceedings of the 3rd International Conference on Computing for Sustainable Global Development*, pp. 160–164 (Cited on page 26).

K. Patrick and Z. Godswill (2016). ‘Greedy Ants Colony Optimization Strategy for Solving the Curriculum Based University Course Timetabling Problem’. In: *British Journal of Mathematics & Computer Science* 14.2, pp. 1–10. DOI: `10.9734/bjmcs/2016/23143` (Cited on pages 75 and 77).

S. Petrovic and E. Burke (2004). ‘University timetabling’. In: *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pp. 1–24 (Cited on page 29).

N. Pillay (2012). ‘Hyper-heuristics for Educational Timetabling’. In: pp. 316–340 (Cited on page 1).

N. Pillay (2016). *A review of hyper-heuristics for educational timetabling.* Vol. 239, pp. 3–38. DOI: `10.1007/s10479-014-1688-1` (Cited on pages 26, 59 and 60).

N. Pillay and E. Özcan (2019). ‘Automated generation of constructive ordering heuristics for educational timetabling’. In: *Annals of Operations Research* 275.1, pp. 181–208. DOI: `10.1007/s10479-017-2625-x` (Cited on pages 124 and 125).

M. Preuss, C. Kausch, C. Bouvy and F. Henrich (2010). ‘Decision Space Diversity Can Be Essential for Solving Multiobjective Real-World Problems’. In: *Lecture Notes in Economics and Mathematical Systems* 634, pp. 367–377. DOI: `10.1007/978-3-642-04045-0_31` (Cited on page 156).

E. Psarra and D. Apostolou (2023). ‘A Combination of Genetic Algorithms and Local Search to Solve a Real Data University Timetable Scheduling Problem’. In: *14th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pp. 1–8. DOI: `10.1109/IISA59645.2023.10345845` (Cited on page 46).

D. M. Pérez, E. A. Portilla-Flores, E. Vega-Alvarado, M. B. Calva-Yañez and G. S. Cervantes (2021). 'A Novel Multi-Objective Harmony Search Algorithm with Pitch Adjustment by Genotype'. In: *Applied Sciences* 11.19, p. 8931. DOI: `10.3390/app11198931` (Cited on page 159).

P. Rakshit, A. Konar and S. Das (2017). 'Noisy evolutionary optimization algorithms – A comprehensive survey'. In: *Swarm and Evolutionary Computation* 33, pp. 18–45. DOI: `10.1016/j.swevo.2016.09.002` (Cited on page 221).

F. de la Rosa-Rivera, J. I. Nunez-Varela, C. A. Puente-Montejano and S. E. Nava-Muñoz (2021). 'Measuring the complexity of university timetabling instances'. In: *Journal of Scheduling* 24.1, pp. 103–121. DOI: `10.1007/s10951-020-00641-y` (Cited on pages 105 and 182).

P. Ross, D. Corne and H. L. Fang (1994). 'Improving evolutionary timetabling with delta evaluation and directed mutation'. In: *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* 866, pp. 556–565. DOI: `10.1007/3-540-58484-6_298` (Cited on page 142).

O. Rossi-Doria, C. Blum, J. Knowles, M. Sampels, K. Socha and B. Paechter (2006). 'A local search for the timetabling problem'. In: *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pp. 124–127 (Cited on page 12).

O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter et al. (2003). 'A comparison of the performance of different metaheuristics on the timetabling problem'. In: *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* 2740, pp. 329–351. DOI: `10.1007/978-3-540-45157-0_22` (Cited on pages 10, 13, 26 and 55).

N. R. Sabar, M. Ayob, G. Kendall and R. Qu (2012). 'A honey-bee mating optimization algorithm for educational timetabling problems'. In: *European Journal of Operational Research* 216.3, pp. 533–543. DOI: `10.1016/j.ejor.2011.08.006` (Cited on page 56).

A. Salehi and S. Doncieux (2022). 'Towards QD-suite: developing a set of benchmarks for Quality-Diversity algorithms'. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* DOI: 10.1145/3520304.3533994 (Cited on page 159).

R. Santiago-Mozos, S. Salcedo-Sanz, M. Deprado-Cumplido and C. Bousoño-Calzón (2005). 'A two-phase heuristic evolutionary algorithm for personalizing course timetables: A case study in a Spanish university'. In: *Computers and Operations Research* 32.7, pp. 1761–1776. DOI: 10.1016/j.cor.2003.11.030 (Cited on page 25).

A. Schaerf (1999). 'Survey of automated timetabling'. In: *Artificial Intelligence Review.* Vol. 13. 2. Artificial Intelligence Review, pp. 87–127. DOI: 10.1023/A:1006576209967 (Cited on pages 17 and 26).

M. D. Schmidt and H. Lipson (2010). 'Age-Fitness Pareto optimization'. In: *Proceedings of the 12th Genetic and Evolutionary Computation Conference*, pp. 543–544. DOI: 10.1145/1830483.1830584 (Cited on page 192).

P. H. Schoute (1911). *Analytic treatment of the polytopes regularly derived from the regular polytopes.* J. Muller in Amsterdam (Cited on page 93).

M. K. Shahvali, M. S. Abadah and H. Sajedi (2011). 'A fuzzy genetic algorithm with local search for university course timetabling'. In: *Proceedings of the Third International Conference on Data Mining and Intelligent Information Technology Applications (ICMiA)*, pp. 250–254 (Cited on page 39).

D. F. Shiau (2011). 'A hybrid particle swarm optimization for a university course scheduling problem with flexible preferences'. In: *Expert Systems with Applications* 38.1, pp. 235–248. DOI: 10.1016/j.eswa.2010.06.051 (Cited on page 55).

W. Shin and J. A. Sullivan (1977). 'Dynamic course scheduling for college faculty via zero-one programming'. In: *Decision Science* 8, pp. 711–721 (Cited on page 31).

B. Sigl, M. Golub and V. Mornarh (2012). 'Solving timetable scheduling problem using genetic algorithms'. In: *Proceedings of the 25th International Conference on Information Technology Interfaces (ITI)*, pp. 519–524 (Cited on page 48).

M. Sniedovich and S. Voß (2006). ‘The corridor method: A dynamic programming inspired metaheuristic’. In: *Control and Cybernetics* 35.3, pp. 551–578. ISSN: 03248569 (Cited on page 34).

K. J. S. M. Socha K. (2002). ‘A MAX-MIN ant system for the university course timetabling problem’. In: *Lecturer notes in computer science* 2463, pp. 1–13 (Cited on pages 55, 71 and 79).

K. Socha, M. Sampels and M. Manfrin (2003). ‘Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art’. In: *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* 2611, pp. 334–345. DOI: 10.1007/3-540-36605-9_31 (Cited on pages 73 and 74).

T. Song, S. Liu, X. Tang, X. Peng and M. Chen (2018). ‘An iterated local search algorithm for the University Course Timetabling Problem’. In: *Applied Soft Computing Journal* 68, pp. 597–608. DOI: 10.1016/j.asoc.2018.04.034 (Cited on pages 13 and 44).

K. Sörensen (2015). ‘Metaheuristics-the metaphor exposed’. In: *International Transactions in Operational Research* 22.1, pp. 3–18. DOI: 10.1111/itor.12001 (Cited on page 56).

S Srinivasan, J. Singh and V. Kumar (2011). ‘Multi-agent based decision Support System using Data Mining and Case Based Reasoning’. In: *International Journal of Computer Science Issues* 8.4, pp. 340–349 (Cited on page 57).

D. Strnad and N. Guid (2007). ‘A multi-agent system for university course timetabling’. In: *Applied Artificial Intelligence* 21.2, pp. 137–153. DOI: 10.1080/08839510601147554 (Cited on page 57).

T. Stutzle and H. H. Hoos (1999). ‘MAX-MIN Ant System’. In: *Future Generation Computer Systems* 16 (8), pp. 889–914. DOI: 10.1016/S0167-739X(00)00043-1 (Cited on page 55).

S. R. Sutar and R. S. Bichkar (2016). ‘Genetic Algorithms based Timetabling using Knowledge Augmented Operators’. In: *International Journal of Computer Science and Information Security* 14.11, pp. 570–579 (Cited on pages 47, 50, 51 and 52).

Suyanto (2010). ‘An Informed Genetic Algorithm for University Course and Student Timetabling Problems’. In: *Lecture Notes in Computer Science* 6114, pp. 229–236 (Cited on page 52).

H. Y. Tarawneh, M. Ayob and Z. Ahmad (2013). ‘A hybrid simulated annealing with solutions memory for curriculum-based course timetabling problem’. In: *Journal of Applied Science* 13.2, pp. 262–269. DOI: 10.3923/jas.2013.262.269 (Cited on page 44).

T. Thepphakorn and P. Pongcharoen (2012). ‘Heuristic ordering for ant colony based timetabling tool’. In: *International Journal of Production Economics* 149, pp. 131–144. DOI: 10.1016/j.ijpe.2013.04.026 (Cited on page 76).

Y. Tian, R. Cheng, X. Zhang and Y. Jin (2017). ‘PlatEMO: A MATLAB Platform for Evolutionary Multi-Objective Optimization’. In: *Computational Intelligence Magazine* 12.4, pp. 73–87. DOI: 10.1109/MCI.2017.2742868 (Cited on page 121).

A. Tripathy (1984). ‘A case in large binary integer linear programming’. In: *Management Science* 30.12, pp. 1473–1489 (Cited on page 31).

A. Tripathy (1992). ‘Computerised decision aid for timetabling - A case analysis’. In: *Discrete Applied Mathematics* 35.3, pp. 313–323 (Cited on page 31).

H. Turabieh, S. Abdullah, B. McCollum and P. McMullan (2010). ‘Fish swarm intelligent algorithm for the course timetabling problem’. In: *Lecture Notes in Computer Science: Lecture Notes in Bioinformatics* 6401, pp. 588–595. DOI: 10.1007/978-3-642-16248-0_80 (Cited on page 56).

UniTime (2023). *UniTime: University timetabling – Comprehensive academic scheduling solutions*. https://www.unitime.org/. [Accessed August 2, 2023] (Cited on page 25).

K. Vermirovsky (2003). *Algorithms for Constraint Satisfaction Problems*. Master Thesis, Facultas Artis Informaticae, Universitas Masarykiana (Cited on page 25).

D. S. Vianna, C. B. Martins, T. J. Lima, M. d. F. D. Vianna and E. B. M. Meza (2020). ‘Hybrid VNS-TS heuristics for University Course Timetabling Problem’. In: *Brazilian Journal of Operations and Production Management* 17.2, pp. 1–20. DOI: 10.14488/BJOPM.2020.014 (Cited on page 45).

J. Wahid and N. M. Hussin (2017). ‘Hybrid harmony search with great deluge for UUM CAS curriculum based course timetabling’. In: *Journal of Telecommunication, Electronic and Computer Engineering* 9.1-2, pp. 33–38 (Cited on pages 25 and 66).

K. Wang, W. Shang, M. Liu, W. Lin and H. Fu (2018). ‘A Greedy and Genetic Fusion Algorithm for Solving Course Timetabling Problem’. In: *Proceedings of the 17th IEEE/ACIS International Conference on Computer and Information Science*, pp. 344–349. DOI: 10.1109/ICIS.2018.8466405 (Cited on page 48).

J. Ward and J. S. Schlipf (2010). ‘Answer Set Programming with Clause Learning’. In: *Lecture Notes in Artificial Intelligence* 2923, pp. 302–313 (Cited on page 38).

B. Webster (2004). *Solving Combinatorial Optimization Problems Using a New Algorithm Based on Gravitational Attraction.* PhD thesis, College of Engineering at Florida Institute of Technology (Cited on page 57).

D. J. A. Welsh and M. B. Powell (1967). ‘An upper bound for the chromatic number of a graph and its application to timetabling problems’. In: *The Computer Journal* 10, pp. 85–86 (Cited on page 27).

A. Wren (1995). ‘Scheduling, timetabling and rostering—a special relationship?’ In: *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT)*, pp. 46–75 (Cited on page 1).

K. Xiang, X. Hu, M. Yu and X. Wang (2024). ‘Exact and heuristic methods for a university course scheduling problem’. In: *Expert Systems with Applications* 248.123383. DOI: 10.1016/j.eswa.2024.123383 (Cited on page 54).

S. Yang and N. S. Jat (2011). ‘Genetic algorithms with guided and local search strategies for university course timetabling’. In: *Transactions on Systems MAN, and Cybernetics-PART C: Applications and Reviews* 41 (1), pp. 93–106 (Cited on page 39).

M. Yoshikawa, K. Kaneko, Y. Nomura and M. Watanabe (1996). ‘A constraint-based high school scheduling system’. In: *IEEE Expert* 11.1, pp. 63–72 (Cited on page 34).

A. H. Yousef, C. Salama, M. Y. Jad, T. El-Gafy, M. Matar and S. S. Habashi (2017). 'A GPU based genetic algorithm solution for the timetabling problem'. In: *Proceedings of the 11th International Conference on Computer Engineering and Systems (ICCES)*, pp. 103–109. DOI: `10.1109/ICCES.2016.7821982` (Cited on pages 52 and 54).

E. Yu and K. S. Sung (2002). 'A genetic algorithm for a university weekly courses timetabling problem'. In: *International Transactions in Operational Research* 9.6, pp. 703–717. DOI: `10.1111/1475-3995.00383` (Cited on page 50).

M. Yusoff and N. Roslan (2019). 'Evaluation of Genetic Algorithm and Hybrid Genetic Algorithm-Hill Climbing with Elitist for Lecturer University Timetabling Problem'. In: *Lecture Notes in Computer Science* 11655, pp. 363–373. DOI: `10.1007/978-3-030-26369-0` (Cited on pages 52, 53 and 66).

L. Zhang and S. Lau (2005). 'Constructing university timetable using constraint satisfaction programming approach'. In: *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC)*, pp. 55–60 (Cited on page 35).

X. Zhang, Y. Tian, R. Cheng and Y. Jin (2015). 'An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization'. In: *Transactions on Evolutionary Computation* 19.2, pp. 201–213 (Cited on page 143).

E. Zitzler and S. Künzli (2004). 'Indicator-based selection in multiobjective search'. In: *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics* 3242, pp. 832–842. DOI: `10.1007/978-3-540-30217-9_84` (Cited on page 136).