Iterated Local Search for the examination timetabling problem with constructive-based initial solution*

Synim Selimi[1], Labeat Arbneshi[1], Kadri Sylejmani[1✉], and Nysret Musliu[2]

[1] Faculty of Electrical and Computer Engineering, University of Prishtina, Kosova
[2] Databases and Artificial Intelligence Group, TU Wien, Austria
synim.selimi@student.uni-pr.edu,
[labeat.arbneshi,kadri.sylejmani]@uni-pr.edu,
musliu@dbai.tuwien.ac.at

Abstract. In this extended abstract we present an approach and solution for the examination timetabling problem based on the Iterated Local Search metaheuristic. Initially we introduce a flat data remodeling of the given problem instances. The proposed constructive approach then uses precalculated heuristic information to construct a feasible initial solution from the refined instance data. The neighborhood structure consists of two operators, one to reallocate only rooms and the other to change room-period tuples for course examination events. The algorithm also applies a perturbation mechanism, which is tuned to guide the search for optimal solutions out of the local optima. The presented approach has been preliminarily tested against some smaller test sets existing in the literature, where it has shown that it is able to produce optimal results for some of the instances.

Keywords: Examination Timetabling, Constructive Heuristic, Iterated Local Search.

1 Introduction and problem formulation

The examination timetabling process at universities is an overly complex combinatorial problem that has been extensively researched, resulting in a substantial body of literature. Qu et al. [1] present a survey of the algorithmic methodologies and the respective variants of the problem formulation. Many of the algorithms in literature solve the variant of the examination timetabling problem that was introduced in the International Timetabling Competition in 2007 (ITC2007), which is described thoroughly by McCollum et al. [2]. ITC2007 is mainly inspired by the model of British universities for examination timetabling, and it introduces 12 test instances (tagged as ITC2007 dataset) which are quite challenging. To date, none of the instances have been solved to optimality. In this paper, we tackle a recent reformulation of the problem by

* The work on this paper was supported by the HERAS+ program within the project entitled "Automated Examination Timetabling in the Faculty of Electrical and Computer Engineering - University of Prishtina".

2

Battistutta et al. [3], which is based on the practicalities of the examination timetabling process at Italian universities.

The original formulation of a real-world examination timetabling problem carried out by Battistutta et al. [3] is the variant that we take on in this paper. In the following section we outline the essential problem entities and constraints, but we refer the reader to Battistutta et al. [3] for more in-depth details on the problem formulation.

The problem defines the following entities:

Courses, Exams and Events – Each course requires scheduling one or more exams within an examination term. In addition, each exam can be organized multiple times and can consist of one or two parts, depending on whether it has a written part, an oral part or both. Two-part exams must be scheduled sequentially.

Rooms - The respective exam events might require rooms of specified size. Rooms are classified into three size categories: small, medium, and large. Room combinations are annotated as room sets and are predefined as fixed sets in the problem instances.

Days, Timeslots, Periods - The number of available periods is the total number of time slots per day multiplied by the number of days in a term. For example, in a two-week time span (i.e., examination term) there are 20 available periods given two time slots per day (e.g., 09:00 and 14:00) during working days.

Curricula – Contains the courses with the same students (e.g., courses of a given study program). There are two categories of curricula, namely Primary and Secondary. The first holds the set of courses in the current semester, while the latter has the set of courses in previous semesters. The level of conflicts between assignments of events belonging to Primary and Secondary set is outlined below.

The hard constraints are:

Room request - For each exam event (written or oral) there is a specific number and type of the room/s that must be assigned. Also, if any oral exam requires a room, then a single room of any type must be assigned.

Room occupation - During each period, a given room cannot be assigned to more than one exam event.

Hard conflicts - Two events (written or oral) cannot be assigned to the same period, if they: (1) are primary courses in the same curricula (i.e., same semester), (2) have the same teacher, or (3) have an explicitly written down constraint forbidding them to be assigned in the same period.

Precedence – Requires that two events of the same course are strictly scheduled one after the other when: (1) events are part of two separate exams of the same course and (2) events are part of the same two-part exam (written and oral).

Unavailability - An exam event or room might be explicitly declared not to be scheduled in some specific periods.

The soft constraints are:

Soft conflicts - Two events that are assigned to the same period are in soft conflict if they: (1) belong to the same set of courses either in a primary-secondary or secondary-secondary relationship and (2) have an explicitly declared undesirable constraint not to be scheduled at the same time.

3

Algorithm 1 Construction of the initial solution

```
1:  procedure Solve(instance)
2:      solution <- {}
3:      cInstance <- InstanceLevelHeuristic(instance) // Algorithm 2
4:      for course in cInstance.courses
5:          cCourse <- CourseLevelHeuristic(course) // Algorithm 3
6:          solution[course] <- AssignRoomPeriod(cCourse)
7:          cInstance <- PropagateConstraints(cInstance)
8:      return solution
```

Preferences - There is a set of constraints explicitly declaring that specific combinations of events and periods or rooms and periods are undesired or preferred.
Distances - For some pairs of events there are constraints requiring certain minimum and/or maximum distances between their respective assigned periods, such as: (1) distinct parts of the same exam have a minimum and a maximum distance, declared explicitly for each course, (2) different exams of the same course must be separated for a minimum number of periods, (3) if two courses are part of the same curriculum, there should be a minimum separation between the first event of the respective exams, and (4) there could be explicit constraints requiring certain distances between specific events.

Note that the weights of the violation for distinct types of soft conflicts are left to be specified by the end user.

2 Solution approach

2.1 Preprocessing, search space and cost function

In the preprocessing phase we do a complete flat data remodeling of the given instances to ensure that all relevant data is reduced and contained within its corresponding course instance. This is accomplished by moving and restructuring information about courses and constraints from scattered entities into a self-contained course event entity with information about allowed rooms/periods, number of events, event dependencies, etc. This step makes the use of constructive heuristics and constraint propagation techniques much easier as described in Section 2.2 (Initial solution).

A state in the search space is represented by two vectors, where, for each exam event, the first one stores the rooms and the second one stores the assigned period. An exam can only be placed in a certain room and period if it satisfies all hard constraints that are related to periods, rooms, curricula, and teachers.

The cost (fitness) function is the weighted sum of all penalties that occur when a room-period assignment does not fulfill the associated soft constraints.

4

Algorithm 2 Applying heuristic information at the instance level

```
1:   procedure InstanceLevelHeuristic(instance)
2:       inst <- Copy(instance)
3:       if instance has more courses with multiple exams
4:           inst.courses <- GroupAndAssignByExamNumber(inst.courses)
5:       else if instance has more courses with multiple parts
6:           inst.courses <- GroupAndAssignByParts(inst.courses)
7:       else if instance has more flat courses
8:           inst.courses <- ReorderComplexCoursesFirst(inst.courses)
9:       return inst
```

Algorithm 3 Applying heuristic information at the course level

```
1:   procedure CourseLevelHeuristic(course)
2:       c= Copy(course)
3:       if course required simple rooms
4:         c.possibleRooms <- Shuffle(c.possibleRooms)
5:       if course has multiple exams
6:         c.possiblePeriods <-
                 DistributeExamPeriods(c.possiblePeriods)
7:       else if course has multiple parts
8:         c.possiblePeriods <-
                 DistributePartPeriods(c.possiblePeriods)
9:       return c
```

2.2 **Initial solution**

Following the preprocessing phase, our approach ensures that we can consistently generate an initial solution for all instances in reasonable time by using multiple instance-level and course-level constructive heuristics. These heuristics guide the assignment of courses, periods and rooms based on situational information, for e.g., the number of exams per course, the type of exams (written, oral), the course event complexity (composite rooms, combination of multiple two-part exams), etc. While the approach proposed by Battistutta et al. [3] looks through a search space with infeasible states, our approach always starts with and explores within the feasible region of the search space.

Algorithm 1 displays the general approach to generating an initial solution. Allowed room-period tuples per course are defined during preprocessing and then updated from *PropagateConstraints* after each allocation. *AssignRoomPeriod* selects the first allowed room-period tuple for a given course event. Algorithm 2 illustrates the use of constructive heuristics to guide the generation of the initial solution.

The heuristic mechanisms described below guide the order of course assignments and the order of period-room allocations to always satisfy hard constraints promptly with little to no backtracking.

DistributeExamPeriods(possiblePeriods) – distributes and reorders assignable periods for course events with a calibrated distance between event exams.

5

Algorithm 4 Iterated Local Search

```
procedure SolveWithILS (instance, maxIterations,
hillClimbingIterations, operatorRate, changeRate, changeHomeRate,
perturbRate)
    current <- Solve(instance) // Algorithm 1
    best <- home <- current
    for n from 1 to maxIterations
        p <- random (0,1)
        for h from 1 to hillClimbingIterations
           if p < operatorRate
              neighbor <- ChangeRoom(current)
           else
              numCourses <- changeRate * instance.totalCourses
              for c from 1 to numCourses
                 neighbor <- ChangeRoomPeriod(current)
           if neighbor better than current
              current <- neighbor
        if current better than best
            best <- current
        h <- random (0,1)
        if h < changeHomeRate or current better than home
            home <- current
        numCourses <- perturbRate * instance.totalCourses
        for i from 1 to numCourses
            current <- ChangeRoomPeriod(home)
    return best
```

DistributePartPeriods(possiblePeriods) – distributes and reorders assignable periods for two-part course events with a calibrated distance between event parts.

GroupAndAssignByExamNumber(courses) – groups and reorders by exam number, whereafter preceding exams will be assigned before subsequent exams by default.

GroupAndAssignByParts(courses) – groups and reorders courses by their respective parts (Written or Oral), whereafter Written exams will be always assigned before Oral exams.

ReorderComplexCoursesFirst(courses) – groups and reorders courses by their complexity first, where complexity is defined as the number of events attached to a course due having multiple exams, multiple parts, or both.

6

2.3 Neighborhood structure

Battistutta et al. [3] define the *MoveEvent* operator, which changes the period-room tuple of an event (i.e., a part of an exam). In our case, the neighborhood structure is a variation of *MoveEvent*, which consists of two operators, namely *ChangeRoom* and *ChangeRoomPeriod*, as described below.

ChangeRoom – randomly selects a given exam (including all its events) and moves all corresponding events from their existing room to a new room, provided that all hard constraints are satisfied.

ChangeRoomPeriod – extends the *ChangeRoom* operator with the ability to move exam events from a given room and period to another randomly selected available room and period.

2.4 Iterated Local Search

In Algorithm 4, we present the pseudocode of the proposed approach that is based on the Iterated Local Search (ILS) metaheuristic. The initial solution is constructed by using constructive heuristics and constraint propagation as discussed in Section 2.2. This ILS-based procedure runs a form of hill climbing iteratively and explores the search space using the neighborhood structure described above. Within the iterative phase of ILS, we distinguish between three main steps, as follows.

The exploitation phase runs a hill climbing algorithm, which exploits the search space using our neighborhood structure guided by the parameter `operatorRate`. This parameter defines the selection of one of the existing neighborhood operators.

The selection phase selects the new home depending on the parameter `changeHomeRate`, which determines whether the algorithm has more of an explorative nature (where the current solution is accepted as the new home base, regardless of quality) or exploitative nature (where the current solution becomes the new home base only if its quality is better than the quality of current home base).

The perturbation mechanism, which, based on the parameter `perturbRate`, applies the operator `ChangeRoomPeriod` multiple times to perturb the home solution and consequently avoid the local optima.

3 Computation results

3.1 Data set and parameter tuning

The formulation of the examination timetabling problem from Battistutta et al [3] that we have tackled in this paper has a dataset of 40 instances that is divided into 7 groups. Each group presents timetabling requirements from different study department at selected Italian universities. For our preliminary computational study, we have used the revised version of the dataset [3] and have only selected two of the smaller groups in the dataset (namely group D2 and group D3) that have at most 89 exams, 188 periods and 15 rooms. The experiments have been conducted using an Apple M1 machine with 16 GB of memory.

7

3.2 Comparison results

During experimentation sampled from 100 attempts we managed to generate initial feasible solutions for all instances in 0.3537s on average, with 0.01s being the shortest and 2.58s the longest duration.

The preliminary computational experiments were conducted by running the algorithm 10 times for 30 seconds for each of the instances belonging to the test subsets, namely D2 and D3. All results and generated solutions have been validated with the solution validator (named *examtt toolbox*) provided by Battistutta et al [3]. Table 1 displays the preliminary results, which show promising indications that the algorithm can generate comparable and satisfactory solutions. For the D2 subset, our approach falls behind the approach of Battistutta et. al [3], while for the D3 subset, our approach finds the optimal solutions for 6 instances and is outperformed on the remaining 3 instances.

We cannot draw a direct comparison of the computation time against the approach of Battistutta et. al [3], due to different computing environments, however as shown in Table 1, the approach of Battistutta et. al [3] on average solves the D2 subsets for about 93 seconds and the D3 subsets on average for about 27 seconds.

4 Conclusion and future work

This paper presents an ongoing work about the design and implementation of an Iterated Local Search metaheuristic that can find optimal solutions for a subset of instances existing in the literature. We believe this approach is worth exploring further because of promising preliminary results and the ability to produce feasible solutions quickly, albeit not notably better than the existing ones for all instances in the test set.

Table 1. Comparison of the proposed approach against state-of-the-art methods.

Instance name	Simulated Annealing (SA)			ILS algorithm		
	Avg	Best	Time	Avg	Best	ILS vs. SA (%)
D2-1-18	427.77	426	94.7	630	570	25.26
D2-2-18	22.00	22	88.7	151	140	84.29
D2-3-18	22.00	22	95.0	169	97	77.32
D3-1-16	0.00	0	61.9	201	169	100
D3-1-17	0.00	0	83.5	472.5	401	100
D3-1-18	0.00	0	83.9	460	375	100
D3-2-16	0.00	0	0.8	1.7	0	0
D3-2-17	0.00	0	3.1	2.9	0	0
D3-2-18	0.00	0	3.5	0	0	0
D3-3-16	0.00	0	1.0	0	0	0
D3-3-17	0.00	0	2.3	3.1	0	0
D3-3-18	0.00	0	2.2	0	0	0

8

As part of future work, along with the development of new neighborhood operators, we aim to upgrade the existing operators with heuristic features that choose exam reallocations based on partial solution penalties and restrict the selection of periods and rooms to only those promising improvements. We will also explore applying our flat entity approach to address the practical challenges with *MiniZinc* modeling mentioned by Battistutta et. al [3]. Lastly, in addition to improving experimental benchmarking, we intend to modify our heuristics approach to also explore the infeasible part of the search space.

References

1. Qu, Rong, Edmund K. Burke, Barry McCollum, Liam TG Merlot, and Sau Y. Lee. "A survey of search methodologies and automated system development for examination timetabling." Journal of scheduling 12, no. 1 (2009): 55-89.
2. McCollum, Barry, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. "Setting the research agenda in automated timetabling: The second international timetabling competition." INFORMS Journal on Computing 22, no. 1 (2010): 120-130.
3. Battistutta, Michele, Sara Ceschia, Fabio De Cesco, Luca Di Gaspero, Andrea Schaerf, and Elena Topan. "Local Search and Constraint Programming for a Real-World Examination Timetabling Problem." In International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, Cham, (2020): 69-81.