Near-Optimal Course Scheduling at the Technion*

Ofer Strichman

Information Systems Engineering, IE, Technion, Haifa, Israel ofers@ie.technion.ac.il

Abstract. The focus of this article is the automation of course, classroom, and exam scheduling for the faculty of Industrial Engineering (IE) at the Technion in Haifa, Israel. The system, called the Technion Industrial Engineering Scheduler (TieSched), has been operational since 2012. It is based on a distributed collection of constraints and multiple engines running in parallel, including SAT, pseudo-Boolean, CSP, and weighted-Max-SAT solvers. A sophisticated decision support subsystem accommodates manual edits to the schedule. This article describes the manual process used previously and the TieSched system architecture, and provides details about the model formulation and solving engines. It also presents the new process that TieSched enables and the path to stakeholder acceptance. The benefits of TieSched include improved efficiency of the scheduling process (i.e., a reduction from 9-10 weeks to 3-4 weeks), better schedules, and enhanced levels of service to lecturers, assistants, and students.

Introduction

Each of the 18 faculties (departments) at the Technion in Haifa, Israel, has a dedicated administrative assistant (the ‘assistant’ in the remainder of this paper) to handle administrative tasks related to teaching issues. Most of an assistant’s time is dedicated to scheduling courses, classrooms, and exams. The assistant is the only person within a faculty who has an overall view of the faculty’s goals and their relative importance; however, if a conflict that the assistant cannot solve arises, the dean or vice dean might intervene. The scheduling process involves a significant number of hidden priorities. In practice, the assistant determines these priorities in an ad-hoc manner. For example, an assistant must determine if granting a lecturer’s request to teach in a specific slot (i.e., a specific day and hour) outweighs scheduling that class such that it overlaps with an elective course; or the assistant might have to decide if preventing a gap in the schedule outweighs allowing a teaching assistant (TA) to take a graduate student course. Similar problems exist with scheduling exams; for example, how should the days allocated to exams be split among the various courses? These are important decisions that must be made at a higher level than the administrative assistant and applied as uniformly and as transparently as possible. Furthermore, even if the priorities are explicit, the number of considerations and possible combinations is so large, that using a manual process to find a near-optimum schedule is virtually impossible.

The industrial engineering (IE) faculty at the Technion is relatively large in that it comprises four undergraduate tracks and nine graduate tracks and has complex teaching requirements. The programs in each track change every few years; as a result, the scheduling requirements are determined not only by the track, but also by the year and semester in which the students were admitted. The complexity of the scheduling process led the Technion to attempt to automate it five times between the late 1970s and the 1990s. All these attempts, which focused solely on an automated scheduling engine, failed. The schedules generated were inadequate because many constraints were not considered, and the constraints of the lecturers and TAs were never entered completely. The assistant apparently found that asking a lecturer when to schedule his (her) course was easier than entering his (her) constraints.

Our automated scheduler TieSched and its peripheral subsystems became operational in 2012 and have since evolved. TieSched’s automated scheduler module attempts to cope with the computational complexity of the problem by running multiple engines in parallel, most of which are based on propositional satisfiability solver (SAT) and constraint solver CSP technologies.

The Manual Process Prior to TieSched

The manual course-scheduling process has four phases. The first phase—scheduling the lectures—is done synchronously among all faculties. One week of the scheduling process is dedicated to each of the six semesters of studies. Although most programs comprise eight semesters, the last two semesters include only electives, which do not require coordination and are relatively easy to schedule. For example, in the first week of the scheduling period, all assistants schedule the first-semester-level courses taught in their faculties; in the second week, they schedule the second-semester-level courses. This synchronous approach ensures the cross-faculty coordination that is required for scheduling joint courses (i.e., courses that can be taken in two or more faculties). Generally an assistant attempts to start with the previous year’s schedule and uses it as a basis for making changes. The scheduling is based on an estimation of the demand; for example, the assistant estimates how many recitation groups will be needed based on the default upper bound of 35 students per class (the number of students per lecture group is more flexible). Estimating enrollment to elective courses is typically easy; because the number of students in these classes is always small, one recitation group is usually sufficient. Demand for mandatory courses is also typically easy to estimate. In most cases, the students who can be expected to take a given course are currently registered in another mandatory course. Therefore, the numbers are expected to be similar. For some courses, however, the enrollment is difficult to predict because the courses are open to multiple faculties as electives. The number of students who will repeat a course is also difficult to predict.

The second phase—scheduling the recitation groups—takes two weeks. This phase requires cross-faculty coordination. Phase 2 is much more difficult to schedule than Phase 1 because (1) the schedule is already constrained by Phase 1, and (2) the TAs take courses as graduate students. The combination of these two factors can cause a situation in which a TA cannot take a desired or necessary course; worse, the TA may need to confirm a slot before the recitations of the desired (necessary) course are fixed. The automated scheduler we discuss in this paper solves these problems.

During Phase 1 and Phase 2, which comprise eight to nine weeks, the assistant must personally contact all teachers (in this paper, we use the terms lecturer and teacher interchangeably) and TAs to coordinate their slots. As of this writing (spring 2016), the IE department has 133 such teachers and TAs. During this period (i.e., Phase 1 and Phase 2), the assistant also schedules the exams for the Technion’s two exam periods, which correspond to first- and second-chance exams. If a student fails an exam the first time that he (she) takes it, that student may sit for the exam a second time; we refer to these exams as first-chance and second-chance exams. The process includes many hard and soft constraints that must be satisfied; these are described in detail below.

In Phase 3, the schedule of courses and exams is sent to the students’ representatives (i.e., students elected to represent their peer students), who check the schedule and may request changes. The most common reason is that they occasionally have information that the assistant does not have, such as knowledge that a large number of students intend to retake a course and therefore will not follow the program. The assistant attempts to accommodate these requests, while coordinating changes with the relevant teachers and TAs.

In Phase 4, the schedule is entered into the Technion central computer, the students are informed about the schedule, and are allowed to register for courses. The registration software includes the maximum capacity of each class (i.e., a group of students and a teacher) and closes it for registration once that capacity is reached. If the number of students who want to register for a course exceeds the number allowed, given the groups that were opened for registration, then the assistant is notified and must make a decision, in conjunction with the vice dean for teaching, whether to open a new group, enlarge an existing one, or block additional students from registering. Based on the number of registrants, the assistant assigns classrooms, while considering the special requirements of courses; for example, a course might require a specific laboratory.

TieSched: The System Architecture

TieSched has three main components: A Web-based system for collecting constraints, a control system, and an automated scheduler.

1. **Web-based system for collecting constraints.** Tables 1 and 2 show the constraints that the lecturers and TAs control. The assistant can control all the constraints through the control
component, which we describe in Item 2 below. A large subset of the constraints can be mapped to constraints in the curriculum-based course timetabling problem. Its complete description can be found in Gaspero et al. (2007) and in Achá and Nieuwenhuis (2014). Below, in the Related Work section, we point out the main differences between our model and the model in Gaspero et al. (2007). Wherever possible, the terminology in our tables is consistent with that in Achá and Nieuwenhuis (2014). Constraints 1–3 in Table 1 are the desired, undesired, and impossible teaching hours. In the Web interface, they are marked using colors—green, orange, and red, respectively. Each lecturer and TA has a ‘budget’ of labels of each color; for example, he (she) can mark only a specific number of slots \( (x) \) with red. The administrative assistant is not restricted by those budgets. Figure 1 shows a sample Web page.

2. Control system. The control system is a client, which is installed on an assistant’s computer, for helping with the editing of the raw data and the schedule. This module (a) is a decision support system for high-quality manual scheduling, (b) provides information management, and (c) enables the user to invoke the automated scheduler.

(a) As a decision support system, it:
Table 1. **TieSched** Uses 17 Constraints For Scheduling Courses; The Two Columns On The Right Indicate If The Constraints Are Controlled By The Lecturers And TAs, Respectively, Using The Web Interface

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Hard/Soft</th>
<th>Lec.</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  Desired hours</td>
<td>S</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>2  Undesired hours</td>
<td>S</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3  Time availability (impossible hours)</td>
<td>H</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>4  Split lecture (same day or different days)</td>
<td>H</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>5  Recitation should follow the lecture</td>
<td>H</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>6  No overlap with a list of up to three courses, which TAs wish to take as students</td>
<td>H</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>7  Max and min number of hours of teaching</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8  Force desired hours(^a)</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9  Curriculum clashes (mandatory courses): no overlap of class periods taken by the same population.(^b)</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 Teacher clashes: no overlap of class periods of the same lecturer or TA</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 External scheduling constraints (dictated by other departments)</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 General constraints (e.g., no classes on Thursday afternoon)</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 Distribution of class periods (i.e., in some courses, we specify that not more than half of a course’s recitation groups will overlap).</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 Curriculum clashes (mandatory and/or electives)</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 Curriculum clashes (electives)</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 Late hours</td>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 Minimize gaps in the schedule and balance the teaching load throughout the week</td>
<td>S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^a\) The assistant can mark the teacher’s desired hours as hard constraints. This is applicable for several external teachers who have no flexibility.

\(^b\) Exceptions to this rule are supported. Specifically, this is not enforced for class periods of the same course or type (e.g., two recitations of the same course), general university-wide courses, and situations in which the catalogue specifies a set of alternative courses.

i. indicates which slots cannot be used for scheduling a course because of schedule conflicts with other populations (i.e., groups of students in the same track who were admitted in the same year and semester) who are taking the same course;

ii. indicates whether the correct number of lectures, recitations, and/or labs were scheduled;

iii. shows teacher’s constraints;

iv. generates a report, including checks, such as unscheduled class periods, or a schedule that violates a constraint.

(b) As an information management system, it provides:

i. access to the relevant tables for updating (e.g., tables of teachers, courses, semesters, tracks, classroom information, exam information, parameter-scheduling tuning, and general options);

ii. access to semester-specific information, such as the courses to be taught, the lecturers who will teach these courses, the number of groups, and class capacity;

iii. an export facility that supports exporting the schedule to the Technion central computer and to Excel files;

iv. a communication module that supports sending the schedule to the teachers and TAs by email in a calendar format (e.g., Google calendar).

(c) As an automated scheduler, it enables the user to tune and invoke the automated scheduler module, which we discuss below.

3. **Automated scheduler.** The automated scheduler supports scheduling courses, classrooms, and exams. It is based on solving a minimization problem with hard and soft constraints (Tables 1 and 2). Appendix A includes the mathematical formulation of these constraints. Appendix C shows data that are characteristic of the scheduling problem (e.g., the number of slots).

**Design Decisions**

The literature, for example, the curriculum-based course timetabling problem (Gaspero et al. 2007),
Table 2. TieSched Uses 11 Constraints For Scheduling Classrooms (Constraints 1 to 7) And Exams (Constraints 8 to 11)

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Hard/Soft</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Lecture or TA’s desired classroom</td>
<td>S</td>
</tr>
<tr>
<td>2 Room clashes: class periods scheduled at the same hour are scheduled in different classes</td>
<td>H</td>
</tr>
<tr>
<td>3 Room capacity</td>
<td>H</td>
</tr>
<tr>
<td>4 Designated rooms (e.g., labs)</td>
<td>H</td>
</tr>
<tr>
<td>5 Adjustment to class size</td>
<td>S</td>
</tr>
<tr>
<td>6 Room stability: class periods of the same course stay in the same class</td>
<td>S</td>
</tr>
<tr>
<td>7 Preference for teaching in the IE building</td>
<td>S</td>
</tr>
<tr>
<td>8 No exams on Saturdays and holidays</td>
<td>H</td>
</tr>
<tr>
<td>9 Number of preparation days ≥ recommended number</td>
<td>S</td>
</tr>
<tr>
<td>10 Requested dates for the exam</td>
<td>H</td>
</tr>
<tr>
<td>11 Distance between the first- and second-chance exams ≥ recommended number, based on class size</td>
<td>S</td>
</tr>
</tbody>
</table>

* A set of soft constraints enforcing the constraint that the best score is given when the number of students is in the range of 60 to 85 percent of the class capacity.

# Modeling and Solving

The literature mentions several scheduling capabilities that we chose not to support currently in TieSched, such as finding an optimal assignment of courses to teachers, scheduling exams to the resolution of hours (rather than just days), and assigning invigilators to exams. These problems are either irrelevant in our case or considered easy to solve in practice.

**Modeling and Solving**

**The Solvers**

The system is based on translating the constraints into one of several formats, and using multiple tools, as we show in Table 3. These tools are in the public domain and free for academic use. TieSched supports running multiple engines in parallel. When executed on a personal computer, it evenly distributes them among the computer’s cores. The engines collaborate in the sense that the first to find a solution notifies the central thread, which then stops all other engines and restarts them with a new constraint on the value of the objective, forcing it to be smaller than the one found.

The set of activated engines can be controlled via the control system’s graphical user interface (GUI); however, in practice, the assistant activates a default combination of three tools that we chose
based on results of benchmarking tests that we conducted. Figure 2 shows the overall architecture of the system.

In the past semester, the pseudo-Boolean formula we generated for scheduling courses had 467,000 variables and 1.78 million clauses (the corresponding file size is about 50 MB). The formulas for scheduling the classrooms and exams are \( \approx 20\% \) and \( \approx 30\% \) of this size, respectively. The constraints files are publicly available in Strichman (2016).

### Table 3. Users Of TIEsched Can Choose Which Of The Engines Listed To Operate In Parallel

<table>
<thead>
<tr>
<th>Engine type</th>
<th>Tool</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudo-Boolean</td>
<td>MINISATP</td>
<td>Eén and Sörensson (2006)</td>
</tr>
<tr>
<td></td>
<td>GLUCOSE</td>
<td>Audemard and Simon (2009)</td>
</tr>
<tr>
<td></td>
<td>HSAT</td>
<td>Gershman and Strichman (2009)</td>
</tr>
<tr>
<td></td>
<td>LINGELIN</td>
<td>Biere (2014)</td>
</tr>
<tr>
<td></td>
<td>TREENGELIN</td>
<td>Biere (2014)</td>
</tr>
<tr>
<td>CSP</td>
<td>HCSP</td>
<td>Veksler and Strichman (2015)</td>
</tr>
<tr>
<td>SMT</td>
<td>MS-Z3</td>
<td>de Moura and Bjørner (2008)</td>
</tr>
<tr>
<td>Weighted Max-SAT</td>
<td>MsUNCORE</td>
<td>Morgado et al. (2012)</td>
</tr>
<tr>
<td>ILP</td>
<td>OPL</td>
<td>IBM</td>
</tr>
</tbody>
</table>

![Fig. 2. The Graphic Depicts The Architecture Of TIEsched](image)

Note. Table 3 shows the list of engines. The ‘Central DB’ at the top left is the university’s central database. The ‘Room mng.’ database, also at the top left, is a separate system for managing the rooms in the building. The bottom elements are discussed below in the subsection titled *When the Hard Constraints Cannot be Satisfied*.

### Modeling

For each combination of course type (i.e., lecture, recitation, lab), teaching group, and class period (recall that multiple-hour lectures can be split to two class periods at the request of the lecturer), we define two types of variables: \( h \in [8..19] \) (hour), and \( d \in [1..5] \) (day). Based on these variables, all the
TieSched reflects a policy. However, prior to the implementation of TieSched, no such policy existed, let alone a formalized and written policy. The manual schedule was based on ad-hoc considerations of the assistant. Although the author, as vice dean of teaching at the time, had the authority to determine a schedule in which the mandatory courses are concentrated near the central hour and the electives and many electives and multiple recitation and lecture groups.

To explain this, we provide two examples. In Example 1, suppose we schedule, for a given population, three consecutive hours of mandatory, elective, and again mandatory courses. Whether this schedule has a gap depends on each student’s decision on whether to take the elective course. In Example 2, the middle hour is a recitation; however, this recitation is one of several possible recitations for each student. Our solution to this problem is to define a central hour (1 pm in our case) and then increase the fine (defined as the weight of the corresponding soft constraint, which we add to the value of the objective function if it is violated) as the class period is scheduled further from that hour. For example, starting at 3 pm or ending at 11 am incurs the same fine, but starting at 4 pm incurs a higher fine. Furthermore, elective courses and recitations with multiple groups incur lower fines, hence encouraging a schedule in which the mandatory courses are concentrated near the central hour and the electives and recitations with multiple groups are scheduled in the early and late hours. This statistically reduces gaps in the students’ schedules, and also balances the schedule throughout the week. This solution is novel. To the best of our knowledge, previous attempts to solve this problem, such as in Gaspero et al. (2007), used a fine for nonadjacent lectures, regardless of the size of the gap between them. In contrast, in our model, the fine is proportional to that gap. Our modeling also balances the schedule throughout the week, because it assigns a higher fine to courses that are distant from the central hour.

The objective function in our model has the sole purpose of minimizing the overall weight of the violated soft constraints, as we discuss next.

Soft Constraints

As is evident in Tables 1 and 2, we have many soft constraints, and determining their relative weight reflects a policy. However, prior to the implementation of TieSched, no such policy existed, let alone a formalized and written policy. The manual schedule was based on ad-hoc considerations of the assistant. Although the author, as vice dean of teaching at the time, had the authority to determine this policy, it was in practice based on a feedback loop in which, in conjunction with the assistant, the results of TieSched were studied and the weights were adapted until the results were satisfactory. That the department was forced to formalize a policy is one of the benefits of such a system, because it ensures that the process is uniform and objective, rather than ad-hoc and personal. The weights can be viewed and changed via the user interface; however, in each of the three modules (i.e., courses, classes, exams), they stabilized quickly and have not needed modification in the past few years.

For each soft constraint \( \text{con} \) with weight \( w_{\text{con}} \) (\( w_{\text{con}} \) should be interpreted as a fine, that is, the cost of not satisfying it), we introduce an auxiliary Boolean variable \( a_{\text{con}} \). We then add the disjunctive constraint \( (a_{\text{con}} \lor \text{con}) \) to the set of constraints, and the term \( w_{\text{con}} \cdot a_{\text{con}} \) to the minimization objective. This means that the solver aspires to satisfy \( \text{con} \); otherwise, \( a_{\text{con}} \) must be assigned 1 and consequently the objective is increased by \( w_{\text{con}} \).

Some of the soft constraints, such as the distance from the central hour that we discuss in the Modeling section above, are not subject only to the extremes of being satisfied or not satisfied; rather, there are levels of satisfaction (e.g., the later the course begins after 1 pm, the higher the fine). The

\[
(x - y) \cdot c \quad \text{where } c \in \{\geq, \leq, =\},
\]

and where \( x, y \) are the \( d \) and \( h \) variables, and \( c \) is a constant. For example, suppose two class periods have length \( len_1 \) and \( len_2 \), respectively, and we wish to prevent their overlap. Let \( d_1, d_2, h_1, h_2 \) be the day and hour variables associated with these two class periods. Then the constraint is \( (d_1 \neq d_2) \lor (h_1 - h_2 \geq len_2) \lor (h_2 - h_1 \geq len_1) \); that is, either these two class periods are scheduled on different days, the first class period begins after the second one ends, or the second class period begins after the first one ends. Appendix B shows the conversion of such constraints to propositional logic, which most of our solvers require. Using separate variables for the day and hour has an advantage in modeling constraints that refer to the day (e.g., ‘two class periods should be scheduled on the same day with a window of two hours between them’). This is in contrast to modeling with a single variable, as Abdennadher and Marte (2000) discuss, with a domain equal to the number of hours in a week.

The constraints to enforce minimum gaps (i.e., time windows) in the schedule and balancing the days are particularly challenging, because defining a gap in the schedule is difficult in the presence of some electives and multiple recitation and lecture groups.
following example demonstrates how such constraints are modeled for a simple case of a one-hour course. Suppose that the starting hour of this course is \( h \). Then we add a sequence of soft constraints \( h < t \) for \( t \in [14..20] \) and \( h > t \) for \( t \in [8..12] \) (8 and 20 are the earliest and latest hours in the system), each with a weight 1. If, for example, the course is scheduled to begin at Hour 17, then the constraints \( h < t \) for \( t \in [14..16] \) are violated; accordingly, the accumulated fine is 3. In practice, these constraints are more complicated because class periods can be longer than one hour. Hence, if a two-hour course begins at 3 pm, we penalize the first hour with 2 and the second hour with 3. Appendix A includes details about these constraints.

**Solving Time-outs**

Despite the large amount of input, some tools (e.g., MINISATP, GLUCOSEP, and LINGELING) give an initial nonoptimal solution after a few minutes. These tools solve the optimization problem *indirectly*. They are primarily satisfaction (feasibility) engines: they solve the optimization problem in several iterations, by iteratively adding a constraint that forces the value of the objective to be smaller than in the best solution thus far. When given an input formula with an objective, MINISATP and GLUCOSEP *bias* the search toward values that improve the objective. Hence, the initial solution is not an arbitrary one; rather, it corresponds to a relatively low value of the objective if it is a minimization problem.

In our experience with these tools and their parallel composition, when solving the course scheduling problem they typically reach the best solution that they are ever going to reach (i.e., within the 24-hour time limit) within approximately 90 minutes.

Interestingly, we can calculate an upper bound on the distance of this solution from the optimum, and it is remarkably small: typically our solution is no more than 5 percent larger than the optimum value. The calculation is based on the observation that all the variables are of finite domain (i.e., they are Boolean); hence, we can calculate the theoretical lower and upper bounds of the objective. Furthermore, in parallel to solving the optimization problem, we run another solver that runs ‘from the other side’ (i.e., it attempts to prove that no solution exists, which is a given percentage from the lower bound as that percentage increases (Fig. 3). Hence, after approximately 90 minutes, we typically have a solution which is, for example, 14 percent from the lower bound; however, we already proved that there is no solution within 9 percent of the lower bound. Thus, the optimal solution is somewhere within this range. We can only guess that the solution we find is optimal or very close to it, based on manually analyzing the solution; indeed, we can never improve it with respect to the constraints.

![Fig. 3](image)

**Fig. 3.** In Parallel With Attempting To Minimize The Objective (From Right To Left), We Also Try to Prove That There Is No Solution With Low Values Of The Objective (From Left To Right)

*Note.* After approximately 90 minutes, the gap between these two engines is typically about 5 percent.

Using a lower bound as a reference is not a new concept: some commercial integer linear programming (ILP) solvers (e.g., the solver in MatLab) that are based on branch and bound, generates the ratio between the best solution found so far and the solution to the relaxed problem (i.e., without integrality constraints), which is much easier to find. However, to the best of our knowledge, none solves the problem from the other side in parallel in an attempt to improve the estimation of the distance of the current solution from the optimal one.

**When the Hard Constraints Cannot be Satisfied**

When the constraints cannot be satisfied, assistance in solving the problem is imperative, given the large number of constraints. Below, we describe two features of TIESCHED that we developed to address this problem: minimal unsatisfiable cores and minimal correcting sets.
A **minimal unsatisfiable core** is a subset of the original constraints that cannot be satisfied and from which no constraint can be removed without making the remaining set satisfiable. This is a well-known problem in the $\Sigma_2$ complexity class (it is harder than problems in NP). In linear programming, it is better known by the name *irreducible inconsistent set*; for example, IBM's CPLEX has a utility for finding such sets. Our solution differs in that we attempt to minimize the set of original high-level constraints (e.g., the class periods of Course A and Course B cannot overlap), rather than the low-level mathematical constraints that model them. Furthermore, we associate with each such high-level constraint a description in natural language, thus making the problem easier to address. Indeed, in the few times that the unsatisfiability condition occurred, our assistant was able to solve the problem based on the output of these descriptions. The solution typically involves one TA or lecturer whose request was not satisfied. In one situation, the constraints were satisfied only after three iterations (i.e., there were three separate unsatisfiable cores).

When the constraints are inconsistent, we automatically invoke a tool called Haifa’s high-level minimal unsatisfiable core (HHL-MUC) [Ryvchin and Strichman 2011], which finds a minimal high-level core; based on its output, we print the text associated with the constraints in that core. Despite the theoretical complexity, in practice, this process never takes more than a minute. One reason is that when the formula is unsatisfiable, it is typically because of a combination of a very small number of high-level constraints (typically not more than three or four), and the underlying solvers tend to prove unsatisfiability with little redundancy. Hence, the initial core entered into HHL-MUC is already small.

Briefly, HHL-MUC works as follows: in each iteration, it attempts to remove the entire set of constraints that emanate from a single high-level constraint. If the formula is still unsatisfiable, it discards this set of constraints and repeats the process. Otherwise, it marks the high-level constraint as necessary for the minimal core and reintroduces the associated set of mathematical constraints into the formula.

In case of unsatisfiability, TIESCHED also finds the **weighted high-level minimum correcting set** (WH-MCS). To the best of our knowledge, this problem has not been covered previously in the literature. First, we associate with each high-level hard constraint, a secondary weight indicating its importance relative to the other hard constraints. If all hard constraints cannot be satisfied simultaneously, this number helps us to determine which constraint to remove or relax. Based on these numbers, given an unsatisfiable set of constraints, the WH-MCS problem is to find a set of hard constraints with a minimal total weight, such that removing them makes the rest of the constraints satisfiable. For example, suppose there are a hundred high-level constraints $c_1, \ldots, c_{100}$ which cannot be mutually satisfied and, to simplify the example, that we gave them all the same secondary weight. Suppose further that there are three high-level minimal unsatisfiable cores: $\{c_1, c_2, c_3, c_4\}$, $\{c_3, c_8, c_9\}$ and $\{c_{20}, c_{21}\}$. Then a WH-MCS is, for example, $c_3, c_{20}$. We emphasize that the unsatisfiable cores are not given to us, because enumerating unsatisfiable cores is a hard problem in its own right, as [Liffton and Sakallah 2008] show, and that nonuniform weights can change the answer.

We solve this problem by reducing it to a **weighted high-level Max-SAT problem**. Let us first recall the standard Max-SAT problem. Given a propositional formula in conjunctive normal form (CNF), the goal of Max-SAT is to find the maximum number of clauses that can be satisfied simultaneously. The **weighted high-level** version of Max-SAT requires the set of clauses to be partitioned to groups, and each group must be associated with a weight. It then finds a set of groups, with the largest total weight, which can still be satisfied simultaneously. The connection of this problem to WH-MCS is evident: the constraints emanating from a high-level constraint form a group, which we associate with the secondary weight. Solving the group Max-SAT problem with this input gives us a set of high-level constraints that can be satisfied: the complement of this set is our desired outcome. Continuing the example above, each $c_1, \ldots, c_{100}$ is represented by a group of clauses, and each such group is associated with a weight representing the relative importance of that constraint. Then solving the group Max-SAT problem may result in all constraints other than $c_3, c_{20}$.

Although we are not aware of a tool that solves the weighted-high-level Max-SAT problem, [Heras et al. 2015] illustrates how to reduce this problem to a **weighted Max-SAT problem**. TIESCHED uses the encoding in [Heras et al. 2015], and invokes the MsUNCORE tool [Morgado et al. 2012] to solve it. The assistant receives a list of high-level constraints, with a minimum total secondary weight, whose removal solves all the inconsistencies in the constraints.

We note that the problem of relaxing hard constraints when the formula is unsatisfiable has been addressed previously in the literature in the context of course scheduling (Guéret et al. 1995); the
authors remove one of the constraints that is causing the inconsistency; they then reiterate until they achieve consistency. In contrast to our method, this process does not guarantee that the weight of the constraints removed is minimal. Another alternative is the approach used in [Miranda et al.] (2012). The authors model hard constraints as soft with a large weight; hence, an optimal solution satisfies as many of those constraints as possible. In the realm of a restricted time budget for solving the problem, our approach is likely to be more effective, because every solution it finds respects the hard constraints.

**The New Process and the Limits of Automation**

Whereas the manual scheduling process, which the other departments still use, takes 9 to 10 weeks, the new process takes 3 to 4 weeks. Furthermore, the manual process requires almost 100 percent of the assistant’s time, because he (she) must personally coordinate the schedule with nearly 200 people (i.e., all the teachers and TAs); using the new process, the assistant has relatively little work to do. Below, we describe the reasons that the process is not 100 percent automated, and hence requires manual changes.

- Teachers and (or) TAs request changes to their schedules to address constraints they did not enter through the Web interface in response to the first request.
- The system does not support some constraints. A teacher or TA might enter a special request that the system cannot currently model (e.g., Tuesday afternoon is good only if I teach three hours consecutively; however, if I teach on Wednesday, I want to split it into two sessions with a gap of two hours.). The assistant reviews these comments and attempts to satisfy them by making manual changes.
- Students representatives make requests, typically because they have information that was unknown to the assistant a priori, as we discuss above in the The Manual Process Prior to TieSched. For example, a large number of students may wish to repeat a course; hence, the schedule and the recommended program are not synchronized.

In addition, nine courses are scheduled manually before running the automated scheduler for two reasons:

- Our faculty runs a joint program with another faculty (CS), which still uses manual scheduling. This means that CS must coordinate with us the schedule of several courses about five weeks before we run our scheduler (which is not aware of the constraints in CS).
- Our faculty offers a graduate program for working students that is concentrated in a single day of the week. We realized that manually selecting the three courses from that program and manually scheduling them with the proper breaks is easier than trying to automate their scheduling.

For these reasons, we achieved 89.5 percent automation in the previous semester. This number is based on comparing three figures: \(s\) – the total number of slots, \(m\) – the number of manual changes that were made to the automated schedule, and \(p\) – the number of slots that were prescheduled as hard constraints. The 89.5 percent figure is the result of \((s - p - m)/s\). For comparison, in the first year, this number was approximately 70 percent. It reached its current level of 89.5 percent in the second year when we determined the missing constraints that were creating most of the problems.

With an objective of combining automation and manual work in mind, we now describe the new process. This description is based on a day-by-day action list that we wrote in a Web-based spreadsheet. The assistant marks off the completed items on that list each semester.

1. Prepare data. This step takes two to three days to complete.
   - (a) Copy all data from the previous year to the new semester;
   - (b) Make the necessary changes to the list of courses, the course teachers and TAs, and the number of groups to be scheduled;
   - (c) Adapt, if necessary, the hard constraints from the previous year; these constraints are typically related to courses given by other faculties, or courses given by external teachers who can teach only in one particular slot;
   - (d) Review the textual comments of the teachers to determine if any can be translated to a constraint; typically, some can be translated;
(e) Update, if necessary, the academic programs; this is done by changing the list of populations that is associated with each course;
(f) Generate a report of inconsistent and (or) incomplete data and correct the problems specified in the report.

2. Update mailing lists to reflect changes to the teachers and TAs in the next semester.
3. Send a message through the mailing lists to all teachers and TAs to request that they update their constraints within the next three days.
4. Activate the automated scheduling of courses.
5. (Manually) check and edit the schedule in accordance with the textual requests entered by the teachers and TAs.
6. Share the schedule with teachers and solicit requests for changes within the next three days.
7. Activate the automated scheduling of exams.
8. Share the schedule with the TAs and solicit requests for changes to be received within the next two days.
9. Share the schedule with the students representatives and solicit requests for changes to be received within the next seven days.
10. Send the final schedule to each teacher and TA; in addition, send a calendar entry (e.g., for a Google or Outlook calendar) to those who indicated that they want it.
11. Export the schedule to the Technion central computer.
12. After the students have completed registering, activate the automated scheduling of the classrooms.
13. Export the classroom schedule to the Technion central computer.

With the exception of Steps 1(b)–1(f) and Step 5, all these steps are done by pressing a button in the control system. Overall the assistant has less than 30 percent of the work that her peers have. As a result of this extreme reduction in work, the faculty has transferred work to her; this work has traditionally been allocated to another assistant who was overloaded and sometimes paid overtime. Unfortunately, the faculty cannot show a reduction in manpower; however, it can show that various tasks that were completed, or not completed at all, or completed on overtime pay, are now fulfilled as part of the routine.

Problems and Acceptance

An initial version of the system was presented to the faculty council, and the initial question asked was “how do we benefit from this”? This reflects the main organizational problem the faculty encountered when it used the manual approach: the professors who control the department care mostly about the schedules for their own courses rather than the overall schedule, and they usually get the schedules they want. Hence, they were not highly motivated to cooperate. The convenience and the ability to request features or resources they did not know were possible (e.g., a favorite classroom, or splitting a course during the same day), and other advantages related to classroom and exam scheduling, which we developed later, eventually convinced them. Initially, many did not enter their constraints; however, a single round in which the schedule was automated made them realize that they are hurting themselves by not cooperating. We note here that the amount of cooperation needed from the teachers is minuscule; they must enter their constraints into the Web-based page. These constraints are considered as permanent by TieSched until a teacher changes them. After using the system for a year, teachers began to appreciate and like it.

TieSched’s acceptance by the assistant, who is its operator, was crucial. The system has a fairly simple windows-based graphical user interface that provides access to all the necessary tables and parameters. Although the automated scheduler is invoked by a key stroke (assuming that the default engines are used), our biggest challenge in reaching a state in which the assistant is almost autonomous was resulted from incomplete or inconsistent data, which can abort the automated scheduler module or make its results meaningless. We now prevent most potential inconsistencies by performing checks at the time of data entry. In addition, TieSched produces a report in plain language, based on dozens of SQL queries that search for such inconsistencies. Observe that in Step 1(f) in the detailed procedure that we describe above requires the assistant to invoke this feature and address the problems in the report. The assistant now is proficient enough to handle such problems without requiring help.
Benefits

Our new process offers several advantages, as we describe below.

- All hard constraints are satisfied (if the hard constraints do not conflict), and the solution is near optimal relative to the soft constraints. A review of the lists of constraints in Tables 1 and 2 suggests the implications of this. For example, a lecturer’s course is split, if requested, considering impossible hours (i.e., the hours the lecturer cannot teach), and the recitation is scheduled to be adjacent to the lecture, if requested. In addition, each TA can now take the courses that he (she) wants without conflicting with his (her) own recitation. This is a hard constraint (Table 1), which implies that preventing TAs from taking their desired courses, a situation that occurred commonly over many years, was avoidable.

- The control system supports manual and semi-automated scheduling, by visually showing some of the constraints (e.g., conflicting schedules of other populations), and performs dozens of checks upon request. As we discuss above, we do not believe that such a complex system can provide a fully automated solution. Hence, the decision support that this module offers is necessary for achieving high-quality schedules.

- It automates the creation of a per-semester mailing list. This allows the assistant and other stakeholders (e.g., the dean) to communicate directly with everyone who teaches actively, rather than to all staff, by using the general staff mailing list; lecturers and TAs have the option of having their schedules entered directly into their calendars.

- The classroom-scheduling module attempts to adhere to the requests of each lecturer and (or) TA, guarantees the correct class size, and minimizes the number of times a class is given outside our faculty building.

- Automated exam scheduling solves major problems that exist with the manual schedule process. First, when the assistant manually schedules exams, he (she) is not fully aware of the time the students require to prepare for each exam, relative to other exams that they must take in the same semester. Our system solves this problem. The student representatives gave questionnaires to students from all tracks and years, requesting them to split a given number of days between the various exams. These numbers are now the basis for the optimization problem that TIEsched solves. Second, TIESCHEd attempts to assign a greater number of days between the first- and second-chance exams for large classes. For a large class, the lecturer and (or) TA requires more time to review and return the exams. By lengthening the time between these exams, students have sufficient time before the second-chance exam to decide whether to take it.

Related Work

The amount of literature on automated course scheduling is immense. A biennial conference on the practice and theory of automated timetabling (PATAT) is largely dedicated to this field; see

The constraints in our model have a great deal in common with the curriculum-based course timetabling problem (Gaspero et al. 2007); however, as we discuss above, we define the class-scheduling and course-scheduling problems in a single model in our work. The major differences in our modeling (Tables 1 and 2) are the hard constraints, Constraints 4, 5, 6, and 13, and the soft constraints, Constraints 1, 2, 14, 15, and 17. We differ in how we treat the minimization of gaps in the schedule and the balancing of the days, as we explain in the Modeling section.

To the best of our knowledge, the first article dedicated to this subject is Harwood and Lawless (1975). De-Werra (1985) includes a survey of early works (up to 1985) with an emphasis on graph-theoretical models, and the work in Thompson and Dowsland (1996) is dedicated to scheduling exams using simulated annealing. TIESCHEd is not the first scheduler to rely on constraints over discrete finite-domain variables (in contrast to ILP solutions). Abdennadher and Marte (2000) is the earliest work of which we are aware that considers the course-scheduling problem as a CSP problem with both soft and hard constraints; however, their work includes no discussion of the case in which the system times out or the constraints are unsatisfiable. This is also true for the course-scheduling and class-scheduling solutions in Rudova and Murray (2002). We are not aware of an example in the literature of solutions to the problems discussed above in Solving Time-outs (i.e., that is, previous solutions focused on heuristics and approximations, without guarantees of proximity to the optimal solution), and When the Hard Constraints Cannot be Satisfied. Regarding the former, making all constraints soft
does not solve the problem because in the case of a time-out, the solution does not necessarily respect the hard constraints, even if such a solution exists.

Chin-A-Fat (2004) reports on SAT-based school scheduling. A recent article by Achá and Nieuwenhuis (2014) describes SAT and Max-SAT algorithms for university scheduling, based on hard and soft constraints; hence, it is the closest to our solution relative to the underlying solving engines. It solves existing scheduling benchmarks; it is not concerned with modeling.

Scheduling-software packages that were developed by commercial companies are also relevant to our discussion. The commercial system that most closely resembles ours is BEST-TIMETABLEING. It supports numerous types of constraints; however, its solution is heuristic in nature and gives no guarantee of near optimality. Its constraints-collection Web interface collects information only on preferred teaching hours, in contrast to our more elaborate interface, as we describe in Tables 1 and 2. The tool UNITIME has no distributed constraints-collection facility, and is based on stochastic local search; that is, it is logically incomplete: when there is no solution owing to conflicting constraints, it cannot detect this situation and iterates forever. The MIMOSA tool, which is principally used in schools rather than universities, is the most widespread scheduling-software package; however, its optimization features are weak. For example, after generating an initial manual schedule, it can move classes in the same day to attempt to close gaps. Hence, its main focus is on assisting manual scheduling.

Kassa (2015) describes an effort to implement course scheduling in the college of business and economics of Bahir Dar university in Ethiopia. It covers room assignments, timetable scheduling, and scheduling teachers to classes.

The SCHEDULEEXPERT system from Cornell’s school of hotel administration (Hinkin and Thomp-son 2002), which was later developed as a commercial product focusing on hospitality staffing agencies, does course and class scheduling, and also assigns teachers to subjects. The constraints are all soft, and the objective is to minimize their violation. The modeling differs from ours; for example, it does not have constraints for minimizing the gaps in the schedule; however, it has constraints to minimize the time between the first and last lecture of each faculty member in a given day. It is based on a heuristic search, with no guarantee of the closeness of the result to an optimum. The user examines the schedules generated by the system at run time, and determines if they are good enough or if the system should run for more time to improve the solution. A similar approach can be found in udpSKEDULER (Miranda et al. 2012), a tool that is based on IBM’s CPlex and was developed for scheduling courses in the faculty of engineering of the Universidad Diego Portales (UDP) in Santiago. They solve the course and classroom scheduling as part of a single model. This difference emanates from a different scheduling process: they publish an initial draft of the schedule based on estimating the course registration; after the students register, they determine the final schedule. Another difference between udpSKEDULER and TIE_SCHED is that in the latter, teachers and TAs mark desired, undesired, and impossible hours; in the former, each teacher is obliged to mark a minimum number of patterns, where a pattern is a precise schedule of the course and the classroom in which it will be given. Therefore, if two teachers mark their courses at the same hour and at the same class, one of these patterns will not be satisfied, and the system will not offer the alternative of scheduling during the same hours but in a different room.

It is also significantly less flexible than the input model of TIE_SCHED (e.g., because it fixes the relationship between the class periods of the course, and the classroom). It is also less stable between semesters, because if a lecturer’s course changes, then those patterns must also change. Unlike TIE_SCHED, it does not consider the objective of minimizing gaps in the schedule or balancing the days. Finally, we note that its solution is not necessarily optimal, even with respect to those restrictions, because of capacity restrictions. Once the number of pattern combinations becomes too large, udpSKEDULER manually adds filters, which exclude many patterns that could make it solvable.

Conclusion

We presented TIE_SCHED, a system that was developed and deployed at the Industrial Engineering department in the Technion, Haifa, Israel. This system has several components: a Web-based interface to collect constraints, an optimization engine to solve the scheduling problem for courses, classes, and exams, and a management system with which the assistant can edit the schedule and perform many other tasks. The system was used first in 2012, and numerous improvements and extensions have since been incorporated. It is currently used routinely by the assistant with minimal support. It eliminated
most of her work, allowing her to assume new duties. As we discuss in this paper, the system provides many benefits to the faculty; these include better schedules for all stakeholders, a uniform scheduling policy, academic benefits such as better allocation of the days necessary to prepare for the exams, and an avoidance by TAs of having their recitations overlap with courses they want to take.

Each year, we incorporate the most current engines—the engines that win the various competitions; this is easy to do given the standard constraints language that these competitions impose. Various constraint files are now available for benchmarking purposes (Strichman 2016), and in several formats: conjunctive normal form (CNF) for SAT solvers, which solves the feasibility problem for a given value of the objective), weighted cnf (WCNF) for Max-SAT solvers, SMT2 for SMT solvers, such as MS-Z3, and optimization pseudo-Boolean (OPB). Hopefully, our refinements will lead to improved engines, which will solve scheduling problems to completion in a reasonable amount of time.

A Constraints Formulation

The objective in all three problems is

\[ \text{min} \sum_{c \in S} w_c \cdot a_c, \]

where \( S \) denotes the set of soft constraints, \( c \in S \), \( w_c \) denotes the weight of the constraint, and \( a_c \) denotes the auxiliary Boolean variable that controls this constraint: it is set to true if and only if the constraint is violated; see the Soft Constraints subsection above.

Below, in the description of the constraints, we denote constants with \( c \) and a subscript. For example, \( c_d \) and \( c_h \) are constants representing a day and an hour, respectively. A time interval is defined by three constants: a day \( c_d \), and time bounds \( c_{h_{\min}} \) and \( c_{h_{\max}} \). Other symbols represent decision variables. Specifically, for each class period \( u \) (henceforth, a teaching unit), we define decision variables \( d_u, h_u \), which are the day and starting hour of that unit, and a constant \( c_{\text{len}_u} \), which is the number of hours associated with that unit. We define the course-scheduling problem in Table 4. It relies on several types of auxiliary constraints, which we show in Table 5. For two elements \( u_1, u_2 \) in a list of units, we write \( u_1 < u_2 \) to show that \( u_1 \) appears before \( u_2 \) in the list.

For the class-scheduling problem, for each unit \( u \), we have a single decision variable \( r_u \), which represents the classroom in which this unit will be taught (recall that the class-scheduling problem is solved only after the schedule has been determined; hence, that schedule is part of the input to the class-scheduling problem). The constraints for this problem are listed in Table 6.

For the exam-scheduling problem, each course \( s \) has two decision variables, \( a_s \) and \( b_s \), which correspond to first- and second-chance exams. Their domains correspond to the length of the respective exam periods. This domain is a range; hence, it includes Saturdays and holidays (if they are within the scheduling period). Table 7 shows the constraints for the first-chance exams; the constraints for the second-chance exam are similar; therefore, we do not show them in this appendix. The constraints refer to \( \text{minDiff} \), which is a constant representing the global minimum distance between the two exams, and \( \text{cABDiff} \), which is a constant that represents the difference between the starting dates of the two exam periods.

B Translating Difference Constraints to Propositional Logic

To translate each of the difference constraints, which are over finite domains, to constraints over Boolean variables, we use order encoding (Petke and Jeavons 2011). For each variable \( v \in [\text{min}, \text{max}] \), we introduce \( \text{max} - \text{min} \) Boolean variables \( b_{\text{min}}, b_{\text{min}+1}, \ldots \), such that \( b_j \) represents \( v \leq i \). We then add \( \text{max} - \text{min} - 1 \) transitivity constraints to enforce order: \( b_i \Rightarrow b_{i+1} \), for each of \( i \in [\text{min}, \text{max} - 1] \).

Finally, we add constraints to enforce the original difference constraint. This is best illustrated with examples. Below, we use the standard Boolean connectives: ‘\&’ for conjunction, ‘\( \Rightarrow \)’ for implication, and ‘\( \neg \)’ for negation:

Given the constraint \( x \leq y \), where \( x, y \in [1..3] \), we introduce the Boolean variables \( b_x^1, b_x^2 \) for \( x \) and \( b_y^1, b_y^2 \) for \( y \) (we do not need \( b_x^3 \) and \( b_y^3 \) because they are always true, that is, they represent tautologies), and add the transitivity constraints

\[ b_x^1 \Rightarrow b_y^2 \land b_x^2 \Rightarrow b_y^3. \]
### Table 4. The Course-Scheduling Problem Is Defined Via The Following Constraints

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desired hours</td>
<td>for $G$ - the set of desired intervals $(c_d, c_{min}, c_{max})$. $U$ - units of the teacher.</td>
<td>$\bigwedge_{u \in U} \bigvee_{G} timeBetween(d_u, h_u, c_{len_u}, c_d, c_{min}, c_{max})$</td>
</tr>
<tr>
<td>Undesired hours</td>
<td>for $G$ - the set of undesired intervals $(c_d, c_{min}, c_{max})$. $U$ - units of the teacher.</td>
<td>$\bigwedge_{u \in U} \bigwedge_{G} noOverlap(d_u, h_u, c_{len_u}, c_d, c_{min}, c_{max})$</td>
</tr>
<tr>
<td>Time availability</td>
<td>Same as undesired hours, as a hard constraint</td>
<td>$d_{u_1} = d_{u_2} \land (h_{u_1} \geq h_{u_2} + c_{len_{u_2}} + 2 \lor h_{u_2} \geq h_{u_1} + c_{len_{u_1}} + 2) \land (h_{u_1} \leq h_{u_2} + c_{len_{u_2}} + 3 \land h_{u_2} \leq h_{u_1} + c_{len_{u_1}} + 3)$</td>
</tr>
<tr>
<td>Split lecture: same</td>
<td>same $d_{u_1}, h_{u_1}, c_{len_{u_1}}$, $d_{u_2}, h_{u_2}, c_{len_{u_2}}$. Day The gap is configured to be 2 or 3 hours.</td>
<td>$d_{u_1} = d_{u_2} \land h_{u_2} = h_{u_1} + c_{len_{u_1}}$</td>
</tr>
<tr>
<td>Split lecture: different</td>
<td>$d_{u_1}, d_{u_2}$</td>
<td>$d_{u_1} \neq d_{u_2}$</td>
</tr>
<tr>
<td>Recitation should follow</td>
<td>$d_{u_1}, h_{u_1}, c_{len_{u_1}}$ Recitation: $d_{u_2}, h_{u_2}$. Applied only to the first unit of first lecture group, and first recitation group.</td>
<td></td>
</tr>
<tr>
<td>No overlap of $U_1$ - units taught by the TA, $U_2$</td>
<td>noOverlapUnitSets($U_1, U_2$). TA’s course with - units of courses that the TA the courses he (she) takes takes as a student.</td>
<td></td>
</tr>
<tr>
<td>Max and min teaching hours</td>
<td>(Hard constraints implicitly constrained by setting the domain of the $h$ variables to the range 8, 20)</td>
<td></td>
</tr>
<tr>
<td>Force desired hours</td>
<td>(Same as desired hours; only marked as hard constraints)</td>
<td></td>
</tr>
<tr>
<td>Curriculum clashes $U$</td>
<td>$U$ - List of units taken by the $U$ (mandatory courses): population no overlap of units taken by the same population</td>
<td>$\bigwedge_{u, v \in U, u &lt; v} noOverlapUnits(u, v)$</td>
</tr>
<tr>
<td>Teacher clashes: no</td>
<td>$U$ - List of units taught by the $U$ overlap of units of the teacher same teacher</td>
<td>$\bigwedge_{u, v \in U, u &lt; v} noOverlapUnits(u, v)$</td>
</tr>
<tr>
<td>External scheduling Day and time dictated from outside constraints</td>
<td>$timeBetween$ constraints</td>
<td></td>
</tr>
<tr>
<td>Distribution of units $U_1, U_2$ of equal size</td>
<td>For $i \in</td>
<td>U_1</td>
</tr>
<tr>
<td>Curriculum clashes $U_1, U_2$</td>
<td>$U_1, U_2$ - List of elective and (mandatory, elective) mandatory units taken by the population</td>
<td>$\bigwedge_{u \in U_1, v \in U_2} noOverlapUnits(u, v)$</td>
</tr>
<tr>
<td>Curriculum clashes $U$</td>
<td>$U$ - List of elective units taken by the population</td>
<td>$\bigwedge_{u, v \in U, u &lt; v} noOverlapUnits(u, v)$</td>
</tr>
<tr>
<td>Late hours</td>
<td>Not teaching later than 6 pm</td>
<td>For each unit $u$: $h_u \leq 18 - c_{len_u}$</td>
</tr>
<tr>
<td>Minimize gaps</td>
<td>Center = 13 (central hour)</td>
<td>For each unit $u$: (for each hour $Center \leq c \leq 20 - len_u$: $h_u \leq c$, and for each hour $8 + len_u \leq c \leq Center$: $h_u \geq c - len + 1$)</td>
</tr>
</tbody>
</table>
Table 5. The Modeling Of The Course-Scheduling Problem In Table 4 Relies On The Following Auxiliary Constraints

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>noOverlap</td>
<td>$d_1, d_1, c_{len1}, d_2, h_2, c_{len2}$</td>
<td>$(d_1 \neq d_2) \lor (h_1 - h_2 \geq c_{len2}) \lor (h_2 - h_1 \geq c_{len1})$</td>
</tr>
<tr>
<td>noOverlapUnits</td>
<td>units $u_1, u_2$</td>
<td>$\neg \text{noOverlap}(d_{a_1}, h_{u_1}, c_{len_{u_1}}, d_{a_2}, h_{u_2}, c_{len_{u_2}})$</td>
</tr>
<tr>
<td>noOverlapUnitSets</td>
<td>Sets of units $U_1, U_2$</td>
<td>$\bigwedge_{u_1 \in U_1, u_2 \in U_2} \neg \text{noOverlapUnits}(u_1, u_2)$</td>
</tr>
<tr>
<td>timeBetween</td>
<td>$d, h, c_{len}$, required interval: $d = c_d \land c_{h_{\min}} \leq h \land h + c_{len} \leq c_{h_{\max}}$</td>
<td></td>
</tr>
</tbody>
</table>

Table 6. The Class-Scheduling Problem Is Defined Via The Following Constraints

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Room clashes$^a$</td>
<td>List of units $U$ taught at a particular hour$^a$</td>
<td>$\bigwedge_{u, v \in U, u \neq v} r_u \neq r_v$</td>
</tr>
<tr>
<td>Unit in class$^b$</td>
<td>All units $U$, all classrooms $C$</td>
<td>$\bigwedge_{u \in U, c \in C} r_u \neq c$</td>
</tr>
<tr>
<td>Designated rooms</td>
<td>(Via the domains, for example, normal units do not have the lab rooms in their domain)</td>
<td></td>
</tr>
<tr>
<td>Room stability</td>
<td>List of units $U$</td>
<td>$\bigwedge_{u, v \in U, u \neq v} r_u = r_v$</td>
</tr>
</tbody>
</table>

$^a$ Note that this includes units that were scheduled in previous hours but have length larger than 1.

$^b$ This constraint encapsulates three type of constraints via its weight: the teacher’s preferred classroom, the preference to teaching in the IE building, and the relative capacity (the fine is 0 if the number of registrants is 60–85% of the class’s capacity, and higher if it is higher or lower). For example, if a teacher prefers class $c$ for his (her) unit $u$, then a fine will be associated with the constraint $r_u \neq c'$ for each $c' \neq c$.

Table 7. The Exam-Scheduling Problem Is Defined Via The Following Constraints; The Constraints Shown Refer To The First-Chance Exam Only, And The Gap Between Them Refers To The Second-Chance Exam

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>No exams on Satur-</td>
<td>All courses$^a$ S, forbidden dates in the exam periods $D$</td>
<td>$\bigwedge_{s \in S, d \in D} a_s \neq d$</td>
</tr>
<tr>
<td>Min # of preparation</td>
<td>A mandatory course $s_1$ and its requested prep. days $c_{s_1}$; a set $S$ of mandatory courses taken by the same population</td>
<td>For $s \in S$: For $\text{minDays} &lt; i \leq c_{s_1}$: \begin{align*} (a_{s_1} - a_s) \geq i \lor a_s \geq a_{s_1} \end{align*} (soft) \begin{align*} a_s \geq c_s - 4 \end{align*} (hard constraint)</td>
</tr>
<tr>
<td>First day$^c$</td>
<td>All courses $S$, and for each $s \in S$, its requested prep. days $c_s$</td>
<td></td>
</tr>
<tr>
<td>Requested date</td>
<td>Course $s$, requested date $d$</td>
<td>$a_s = d$</td>
</tr>
<tr>
<td>Distance</td>
<td>All courses $S$, and for each $s \in S$, its recom- the first- and second- mended minimum distance in days $c_{s_{cd}}$, based on class size. $\text{minDiff}$ is a global minimum distance between the two exams, and $c_{ABDiff}$ is the difference between the starting dates of the two exam periods.</td>
<td>For $\text{minDiff} &lt; i \leq c_{s_{cd}}$: \begin{align*} b_s - a_s \geq i \lor a_s \geq a_{s_1} - c_{ABDiff} \end{align*} (soft) \begin{align*} a_s \geq c_s - 4 \end{align*} (hard constraint)</td>
</tr>
</tbody>
</table>

$^a$ This is the set of all courses for which the department is responsible.

$^b$ Mostly used for forcing the exam schedule of external courses, as dictated by other departments.

$^c$ The days before the first day of the examination period also count for preparation. We count them as four days; hence if an exam requires five days, scheduling it on the first day incurs a fine.
That is, if \( x \leq 1 \) then \( x \leq 2 \), and the same for \( y \), and constraints enforcing \( x \leq y \):

\[
b_2^y \Rightarrow b_2^x \land b_1^y \Rightarrow b_1^x .
\]  

That is, if \( y \leq 2 \) then \( x \leq 2 \), and if \( y \leq 1 \), then \( x \leq 1 \).

As another example, consider the constraint \( x - y = 2 \) for \( x, y \in [1..5] \). The Boolean variables are \( b_i^x, b_i^y \) for \( i \in [1..4] \); the transitivity constraints are

\[
b_i^x \Rightarrow b_{i+1}^x \land b_i^y \Rightarrow b_{i+1}^y \quad \text{for} \quad i \in [1..3],
\]

and the constraints enforcing the equivalence \( x - y = 2 \) are

\[
\begin{align*}
b_3^x &\iff b_1^y \land b_3^x \iff b_2^y, \\
\lnot b_2^x &\land b_3^y,
\end{align*}
\]

which is perhaps easier to understand when written as the inequalities that it represents: \( x \leq 3 \iff y \leq 1, x \leq 4 \iff y \leq 2, x \geq 3, y \leq 3 \).

C TieSched Parameter Data for 2016

During the 2016 spring semester, scheduling for the IE department involved 133 faculty and teaching assistants; 127 courses, of which 30 have a fixed schedule determined by other faculties; 722 scheduled slots, of which 265 have a fixed schedule determined by other faculties (the gap between this number and the number of courses occurs because each course can have multiple teaching and recitation groups. In addition, lectures longer than two hours can be split into two class periods at the request of a lecturer). The timetable has five days, from 8:30 am to 8:30 pm. There are 15 classrooms in the building, and many classrooms that are outside the building and therefore less preferable. The schedule of all first-chance exams must fit within three weeks, and the schedule of all the second-chance exam must fit within two-and-a-half weeks, with a week’s break between them.

Acknowledgments

Many students helped me to develop TieSched, mostly as part of annual undergraduate projects. In particular, I thank three students: Boris Milner developed the Web-based component for collecting constraints; Ola Rosenfeld, a PhD student at the time, developed the initial automated scheduler, including the translation of the CSP model to SAT; \textit{without her work, this system probably would not exist}; finally, Michael Veksler developed the concept that allowed us to solve the problem of gaps in the schedule and balance it.
Bibliography


IBM (????) Cplex optimizer.


