INFORMS holds **copyright** to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

that 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 (enumerating unsatisfiable cores is a hard problem in its own right, as shown in Liffiton and Sakallah 2008), 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 that can still be simultaneously satisfied. 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 previous example, each of $c_1, \ldots, c_{100}$ 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) illustrate 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 previously addressed in the literature in the context of course scheduling (Guéret et al. 1996): 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 removed constraints 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 useful, because whereas in their solution the result does not necessarily satisfy the originally hard constraints, all of the intermediate solutions in our approach *do* satisfy the hard constraints, and we are typically able to find such solutions relatively fast.

The New Process and the Limits of Automation

Whereas the manual scheduling process, which the other departments still use, takes 9–10 weeks, the new process takes 3–4 weeks. Furthermore, the manual process requires almost 100 percent of the assistant's time, because he (she) must personally coordinate the schedule with nearly 150 people (all the teachers and TAs); in the new process, the assistant has relatively little work to do. Next, 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 key-in 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. (For example, "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.
- Student representatives make requests, typically because they have information that was unknown to the assistant a priori, as we discuss in section *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.

INFORMS holds **copyright** to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

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

• Our faculty has 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 in itself 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.

Having the necessity to combine automation with 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 automatic 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 student representatives, and solicit requests for changes to be received within the next seven days.

10. Send the final schedule to each teacher and TA; send a calendar entry (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 done late or not completed, 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

INFORMS holds **copyright** to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

"how do we benefit from this"? This reflects the main organizational problem with the manual approach: the professors, who control the department, mostly care 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 because they can request things they never knew they could (e.g., a favorite classroom, or splitting a course on the same day), and other advantages related to classrooms and exams scheduling that were only developed later, convinced them rather quickly. Initially, some did not enter their constraints; however, a single round in which the schedule was automated made them realize that they were 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 by TieSched until they are changed. After using the system for a year, teachers and TAs came to 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 incomplete or inconsistent data, which can abort the automatic 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 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 here.

- 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 teacher's course is split, if requested, is never scheduled on hours that are blocked by the teacher, 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 semiautomatic 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 100 percent 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 others (such as the dean) to communicate directly with everyone who actively teaches, rather than to all staff by using the general staff mailing list; teachers 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 teacher 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. First, when the assistant manually schedules exams, he (she) is not fully aware of the time required to prepare for each exam, relative to other exams that the students 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 of large classes. For a large class, the teacher and (or) TA requires more time to check and return the exams. By

INFORMS holds copyright to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

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 Özcan et al. (2016).

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, we model and solve the course- and class-scheduling problems separately. The major differences in our modeling (Tables 1 and 2) are the hard constraints 4–6, and 13, and the soft 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 Rudová and Murray (2002). We are not aware of solutions in the literature to the problems discussed in the *When Solving Times Out* section (i.e., previous solutions focused on heuristics and approximations, without guarantees of proximity to the optimal solution), and the *When the Hard Constraints Cannot Be Satisfied* section. 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-TIMETABLING. 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 University's School of Hotel Administration (Hinkin and Thompson 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

INFORMS holds **copyright** to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

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 TieSched 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 TieSched (e.g., because it fixes the connection between the class periods of the course, and the classroom), and is less stable between semesters because if a teacher's course changes, then those patterns must also change.

Unlike TieSched, 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 to make it solvable.

Conclusion

We presented TieSched, 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 needed to prepare for exams, and 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: CNF (conjunctive normal form for SAT solvers, solving the feasibility problem for a given value of the objective), WCNF (weighted CNF, for Max-SAT solvers), SMT2 (for SMT solvers such as MS-Z3), OPB (optimization pseudo-Boolean), MiniZinc (a constraints satisfaction standard format), and OPL (for IBM's CPLEX tool). Hopefully, this will lead to improved engines that will solve them to completion in a reasonable amount of time.

Acknowledgments

Many students helped the author develop TieSched, mostly as part of annual undergraduate projects. In particular, the author thanks three students: Boris Milner for developing the web-based component for collecting constraints; Ola Rosenfeld, PhD student, for developing the initial automated scheduler, including the translation of the CSP model to SAT; *without her work, this system probably would not exist*; and finally, Michael Veksler for coming up with the idea of how to solve the problem of gaps in the schedule and balancing it.

Appendix A. Constraints Formulation

The objective in all three problems is

$$\min \sum_{con \in S} w_{con} \cdot a_{con}, \tag{A.1}$$

where S denotes the set of soft constraints, $con \in S$, w_{con} denotes the weight of the constraint, and a_{con} 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* section).

In the description of the constraints, we will 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_{len_u},

INFORMS holds copyright to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

which is the number of hours associated with that unit. The course-scheduling problem is defined in Table B.1. It relies on several types of auxiliary constraints that we show in Table B.2. 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 that holds 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 B.3.

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;

Table B.1. The Course-Scheduling Problem Is Defined via the Following Constraints

Name	Parameters	Constraint
Desired hours for teacher	G—The set of desired intervals $\langle c_d, c_{h_{\min}}, c_{h_{\max}}\rangle$. U—Units of the teacher.	$\bigwedge_{u\in U} \bigvee_G$ *timeBetween*$(d_u, h_u, c_{len_u}, c_d, c_{h_{\min}}, c_{h_{\max}})$
Undesired hours for teacher	G—The set of undesired intervals $\langle c_d, c_{h_{\min}}, c_{h_{\max}}\rangle$. U—Units of the teacher.	$\bigwedge_{u\in U} \bigwedge_G$ *noOverlap*$(d_u, h_u, c_{len_u}, c_d, c_{h_{\min}}, c_{h_{\max}} - c_{h_{\min}})$
Time availability (impossible hours)	Same as undesired hours, as a hard constraint.	
Split lecture: Same day	$d_{u_1}, h_{u_1}, c_{len_{u_1}}, d_{u_2}, h_{u_2}, c_{len_{u_2}}$. The gap is configured to be 2 or 3 hours.	$d_{u_1} = d_{u_2} \wedge (h_{u_1} \geq h_{u_2} + c_{len_{u_2}} + 2 \vee h_{u_2} \geq h_{u_1} + c_{len_{u_1}} + 2) \wedge (h_{u_1} \leq h_{u_2} + c_{len_{u_2}} + 3 \wedge h_{u_2} \leq h_{u_1} + c_{len_{u_1}} + 3)$
Split lecture: Different days	d_{u_1}, d_{u_2}	$d_{u_1} \neq d_{u_2}$
Recitation should follow the lecture	Lecture: $d_{u_1}, h_{u_1}, c_{len_{u_1}}$. Recitation: d_{u_2}, h_{u_2}. Applied only to the first unit of first lecture group, and first recitation group.	$d_{u_1} = d_{u_2} \wedge h_{u_2} = h_{u_1} + c_{len_{u_1}}$
No overlap of a TA's course with the courses he (she) takes as a student	U_1—Units taught by the TA, U_2—Units of courses that the TA takes.	*noOverlapUnitSets*(U_1, U_2).
Max and min teaching hours	(Hard constraints implicitly constrained by setting the domain of the h variables to the range [8..20].)	
Force desired hours	(Same as desired hours, only marked as hard constraints.)	
Curriculum clashes (mandatory courses): No overlap of units taken by the same population	U—List of units taken by the population.	$\bigwedge_{u,v\in U, u<v}$ *noOverlapUnits*(u,v)
Teacher clashes: No overlap of units of the same teacher	U—List of units taught by the teacher.	$\bigwedge_{u,v\in U, u<v}$ *noOverlapUnits*(u,v)
External scheduling constraints	Day and time dictated from outside.	*timeBetween* constraints
Distribution of units	Two lists of units U_1, U_2 of equal size.	For $i \in \lvert U_1\rvert$: $\bigwedge_{u_i\in U_1, v_i\in U_2}$ *noOverlapUnits*(u_i, v_i)
Curriculum clashes (mandatory, electives)	U_1, U_2—List of elective and mandatory units taken by the population.	$\bigwedge_{u\in U_1, v\in U_2}$ *noOverlapUnits*(u,v)
Curriculum clashes (electives)	U—List of elective units taken by the population.	$\bigwedge_{u,v\in U, u<v}$ *noOverlapUnits*(u,v)
Late hours	Not teaching later than 6 PM.	For each unit u: $h_u \leq 18 - c_{len_u}$
Minimize gaps	Center = 13 (central hour).	For each unit u: (for each hour $Center \leq c \leq 20 - len_u$: $h_u \leq c$, and for each hour $8 + len_u \leq c \leq Center$: $h_u \geq c - len + 1$)

INFORMS holds copyright to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

Table B.2. The Modeling of the Course-Scheduling Problem in Table B.1 Relies on the Following Auxiliary Constraints

Name	Parameters	Constraint
noOverlap	$d_1, h_1, c_{len1}\ d_2, h_2, c_{len2}$	$(d_1 \neq d_2) \vee (h_1 - h_2 \geq c_{len2}) \vee (h_2 - h_1 \geq c_{len1})$
noOverlapUnits	Units u_1, u_2	$noOverlap(d_{u_1}, h_{u_1}, c_{len_{u_1}}, d_{u_2}, h_{u_2}, c_{len_{u_2}})$
noOverlapUnitSets	Sets of units U_1, U_2	$\bigwedge_{u_1 \in U_1, u_2 \in U_2} noOverlapUnits(u_1, u_2)$
timeBetween	d, h, c_{len}, required interval: $c_d, c_{h_{\min}}, c_{h_{\max}}$	$d = c_d \wedge c_{h_{\min}} \leq h \wedge h + c_{len} \leq c_{h_{\max}}$

Table B.3. The Class-Scheduling Problem Is Defined via the Following Constraints

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

[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 percent 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 $c' \neq c$.

hence, it includes Saturdays and holidays (if they are within the scheduling period). Table B.4 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 *minDiff*, which is a constant that represents the global minimum distance between the two exams, and c_{ABDiff}, which is a constant representing the difference between the starting dates of the two exam periods.

Appendix 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

Table B.4. 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

Name	Parameters	Constraint
No exams on Saturdays or holidays	All courses[a] S, forbidden dates in the exam periods D	$\bigwedge_{s \in S, d \in D} a_s \neq d$
Min no. of preparation days	A mandatory course s_1 and its requested prep. days c_{s_1}; a set S of mandatory courses taken by the same population	For $s \in S$: For $minDays < i \leq c_{s-1}$: $(a_{s-1} - a_s \geq i \vee a_s > a_{s-1})$ (soft) For $i = minDays$: the same, but as a hard constraint
Requested date	Course s, requested date d[b]	$a_s = d$
First day[c]	All courses S, and for each $s \in S$, its requested prep days c_s	$a_s \geq c_s - 4$
Distance between the first- and second-chance exams	All courses S, and for each $s \in S$, its recommended minimum distance in days c_{sd}, based on class size; *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	For $minDiff < i \leq c_{sd}$: $b_s - a_s \geq i - c_{ABDiff}$ (soft) For $i = minDiff$: the same, but as a hard constraint

[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.)

INFORMS holds **copyright** to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

use *order encoding* (Petke and Jeavons 2011). For each variable $v \in [\min, \max]$, we introduce max–min Boolean variables $b_{\min}, b_{\min+1}, \ldots$ such that b_i represents the fact that $v \le i$. We then add $\max - \min - 1$ transitivity constraints to enforce order: $b_i \Rightarrow b_{i+1}$, for each of $i \in [\min, \max - 1]$. Finally, we add constraints to enforce the original difference constraint. This is best illustrated with examples. Next, we use the standard Boolean connectives: "$\wedge$" for conjunction, "$\Longrightarrow$" for implication, and "$\neg$" for negation.

Given the constraint $x \le y$, where $x, y \in [1..3]$, we introduce the Boolean variables b_1^x, b_2^x for x and b_1^y, b_2^y for y (we do not need b_3^x and b_3^y because they are always true; that is, they represent tautologies), and add the transitivity constraints

$$b_1^x \implies b_2^x \wedge b_1^y \implies b_2^y \tag{B.1}$$

(that is, if $x \le 1$ then $x \le 2$, and the same for y), and constraints enforcing $x \le y$:

$$b_2^y \implies b_2^x \wedge b_1^y \implies b_1^x. \tag{B.2}$$

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

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

$$b_i^x \implies b_{i+1}^x \wedge b_i^y \implies b_{i+1}^y \quad \text{for } i \in [1..3], \tag{B.3}$$

and the constraints enforcing the equivalence $x - y = 2$ are

$$\begin{aligned} b_3^x \iff b_1^y \wedge b_4^x \iff b_2^y \\ \neg b_2^x \wedge b_3^y\,, \end{aligned} \tag{B.4}$$

which is perhaps easier to understand when written as the inequalities that it represents: $x \le 3 \iff y \le 1, x \le 4 \iff y \le 2, x \ge 3, y \le 3$.

Appendix C. TieSched Parameter Data for 2016

During the 2016 spring semester, scheduling for the IE Department involved the following: 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 teacher.) The timetable has five days, from 8:30 AM to 8:30 PM. There are 15 classrooms in the building, and many classrooms 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 one-week break in between.

References

Abdennadher S, Marte M (2000) University course timetabling using constraint handling rules. *Appl. Artificial Intelligence* 14(4):311–325.

Achá RJA, Nieuwenhuis R (2014) Curriculum-based course timetabling with SAT and MaxSAT. *Ann. Oper. Res.* 218(1):71–91.

Audemard G, Simon L (2009) Predicting learnt clauses quality in modern SAT solvers. Boutilier C, ed. *Proc. 21st Internat. Joint Conf. Artificial Intelligence, Pasadena, CA*, 399–404.

Biere A (2014) Yet another local search solver and lingeling and friends entering the SAT competition 2014. Accessed May 10, 2017, http://fmv.jku.at/papers/Biere-SAT-Competition-2014.pdf.

Chin-A-Fat KK (2004) School timetabling using satisfiability solvers. Master's thesis, Delft University of Technology, Delft, Netherlands.

de Moura LM, Bjørner N (2008) Z3: An efficient SMT solver. Ramakrishnan CR, Rehof J, eds. *Tools and Algorithms for the Construction and Analysis of Systems* (*TACAS*), Lecture Notes in Computer Science, Vol. 4963 (Springer-Verlag, Berlin), 337–340.

De-Werra D (1985) An introduction to timetabling. *Eur. J. Oper. Res.* 19(2):151–162.

Eén N, Sörensson N (2006) Translating pseudo-Boolean constraints into SAT. *JSAT* 2(1–4):1–26.

Gaspero L, McCollum B, Schaerf A (2007) The Second International Timetabling Competition (ITC): Curriculum-based course timetabling. Accessed May 10, 2017, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.211.3344&rep=rep1&type=pdf.

Gershman R, Strichman O (2009) HaifaSat: A SAT solver based on an abstraction/refinement model. *J. Satisfiability Boolean Model. Comput.* 6(1–3):33–51.

Guéret C, Jussien N, Boizumault P, Prins C (1996) Building university timetables using constraint logic programming. Burke E, Ross P, eds. *Practice and Theory of Automated Timetabling. PATAT 1995*, Lecture Notes in Computer Science, Vol. 1153 (Springer, Berlin), 130–145.

Harwood GB, Lawless RW (1975) Optimizing organizational goals in assigning faculty teaching schedules. *Decision Sci.* 6(3):513–524.

Heras F, Morgado A, Marques-Silva J (2015) MaxSAT-based encodings for group MaxSAT. *J. AI Commun.* 28(2):195–214.

Hinkin TR, Thompson GM (2002) SchedulExpert: Scheduling courses in the Cornell University School of Hotel Administration. *Interfaces* 32(6):45–57.

IBM (2014) CPLEX optimizer. Accessed August 14, 2017, https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

Kassa BA (2015) Implementing a class-scheduling system at the College of Business and Economics of Bahir Dar University, Ethiopia. *Interfaces* 45(3):203–215.

Kroening D, Strichman O (2008) *Decision Procedures—An Algorithmic Point of View* (Springer-Verlag, Berlin).

Liffiton MH, Sakallah KA (2008) Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Automated Reasoning* 40(1):1–33.

Miranda J, Rey PA, Robles JM (2012) Udpskeduler: A Web architecture based decision support system for course and classroom scheduling. *Decision Support Systems* 52(2):505–513.

Morgado A, Heras F, Marques-Silva J (2012) Improvements to core-guided binary search for MaxSAT. Cimatti A, Sebastiani R, eds. *Theory and Applications of Satisfiability Testing—SAT*, Lecture Notes in Computer Science, Vol. 7317 (Springer-Verlag, Berlin), 284–297.

INFORMS holds **copyright** to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.

Nethercote N, Stuckey PJ, Becket R, Brand S, Duck GJ, Tack G (2007) MiniZinc: Towards a standard CP modelling language. Accessed August 14, 2017, https://www.comp.nus.edu.sg/~gregory/papers/cp07.pdf.

Özcan E, Burke EK, McCollum B, Kjenstad D, Riise A (2016) The practice and theory of automated timetabling (2012). *Ann. Oper. Res.* 218(1):1–2.

Petke J, Jeavons P (2011) The order encoding: From tractable CSP to tractable SAT. Sakallah KA, Simon L, eds. *Theory and Applications of Satisfiability Testing—SAT 2011* (Springer, Berlin), 371–372.

Rudová H, Murray KS (2002) University course timetabling with soft constraints. Burke EK, Causmaecker PD, eds. *Practice and Theory of Automated Timetabling IV, Selected Revised Papers*, Lecture Notes in Computer Science, Vol. 2740 (Springer, Berlin), 310–328.

Ryvchin V, Strichman O (2011) Faster extraction of high-level minimal unsatisfiable cores. Sakallah KA, Simon L, eds. *Theory and Applications of Satisfiability Testing—SAT*, Lecture Notes in Computer Science, Vol. 6695 (Springer, Berlin), 174–187.

Strichman O (2016) Scheduling benchmarks page. Accessed May 10, 2017, http://ie.technion.ac.il/~ofers/TieSched/.

Thompson JM, Dowsland KA (1996) Variants of simulated annealing for the examination timetabling problem. *Ann. Oper. Res.* 63(1–4):105–128.

Veksler M, Strichman O (2016) Learning general constraints in CSP. *Artificial Intelligence* 238(September):135–153.

Verification Letter

Professor Avishai Mandelbaum, Dean, and Professor Aharon Ben-Tal, former Dean, Faculty of Industrial Engineering and Management, Technion Israel Institute of Technology, Haifa, Israel, write:

"It is our honor and pleasure to comply with the requirement of *Interfaces*, and testify on the application and great value of the Automatic Scheduling System, developed by Professor Ofer Strichman. Indeed, the system has been used routinely, since 2012, in our Faculty of Industrial Engineering and Management (IEM) at the Technion.

"There are two signatories below: the present dean of IEM, Avishai Mandelbaum, and the former dean Aharon Ben-Tal, who was the dean when the system was implemented.

"In way of background, scheduling of academic activities is a notoriously difficult problem: It must satisfy system needs (create a schedule) while accommodating constraints of capacity (e.g., rooms), and preferences of individuals (faculty, students). This difficulty challenged us until 2012, as our faculty used a manual scheduling process that required the skills of a highly experienced secretary. Our experienced secretary was due to retire a few months after Prof. Ben-Tal became the dean, which provided the perfect timing for implementing Prof. Strichman's system.

"During the first year of its use, there were faculty members who were skeptical and expressed concerns that the system might not work. This was mainly due to the acknowledgement that academic scheduling is an inherently difficult problem. It might have also to do with the fact that, in parallel to adopting a new system, the secretary was new as well, without any prior experience with the manual system. We are happy to report that these initial hurdles were overcome within two years. Then both our faculty and students became very satisfied with the system, which is now running well and saving a great amount of time and effort for all parties involved in its application.

"The scheduling system enables faculty to specify their teaching constraints explicitly (which are generally met), their favorite classroom, how to split their teaching hours, etc. Teaching assistants now enjoy the fact that their recitations do not overlap with the courses that they take as students. The students themselves enjoy a better schedule, which balances the load along the week, minimizes gaps in the schedule, etc.

"The above benefits are consequences of a state-of-the-art scheduling system, which is based on the following scientific and practical principles. First, the teachers and their assistants are given a web-based friendly interface with which they can enter and edit their constraints (these remain from the previous semester, and hence practically teachers use it only if their constraints have changed). Second, the automatic scheduler combines a multitude of optimization engines that cooperate, and those are able to find optimal or close to optimal solutions despite the theoretical complexity of the problem. Third, the process does not end there: The system supports an easy-to-use client for the secretary, with which she can edit the schedule in a supported manner (the system marks impossible slots and then gives a detailed report about possible problems and violations of constraints). The overall scheduling process for a semester takes a few weeks, and includes time for teachers and their assistants to request manual changes after the automatic scheduler has emitted the schedule.

"Professor Strichman's system is dynamically evolving. For example, this year we have added a module for automatic exam scheduling, which is solving a major problem for us. Specifically, it used to be the case that the secretary alone prepared the schedule, effectively deciding on the number of days that are available to study for each of the exams. This has now changed with the help of the automated scheduler and input from our students. To this end, we asked three students from each semester to hypothetically allocate the days in the exam period, between the courses that they took in the previous year and based on the relative load of these courses. This information is of course known to no one but the students, and it provides the required input for the automatic exam scheduler.

"It is hard to put a price-tag on all of the above benefits, but the value of Prof. Strichman's Automatic Scheduling System is clear and very significant. Indeed, faculty and students are happy that their needs are well catered to. Next, we estimate that the system reduces scheduling efforts by no less than 70%. Also, as a last testimony for its success, the system is now in the process of being adopted by additional faculties at the Technion, as well by additional universities in Israel.

"In summary, our scheduling system is an excellent testimony for the success in the marriage of a long-existing important challenging need with first-rate cutting-edge research."

Ofer Strichman is a computer scientist at the Technion in Haifa, Israel. He has been active in the formal verification and computational logic communities since 1999. He has published two books: *Decision Procedures—An Algorithmic Point of View* with Daniel Kroening and *Efficient Decision Procedures for Validation*, edited a third one, and coauthored more than 90 peer-reviewed articles in these fields. He is also active in the SAT (Boolean satisfiability) community and cochaired the main conference for SAT in 2010. He is best known for his contributions to SAT (e.g., incremental satisfiability, phase saving), linear-time proof manipulation, and bounded model-checking.

INFORMS holds **copyright** to this article and distributed this copy as a courtesy to the author(s).
Additional information, including rights and permission policies, is available at https://pubsonline.informs.org/.