Noname manuscript No.
(will be inserted by the editor)

A Constraint Language For University Timetabling Problems

Vincent Barichard · Corentin Behuet · David Genest · Marc Legeay · David Lesaint

Received: date / Accepted: date

Abstract We present a domain-specific modeling language for a class of university timetabling problems (UTP) that involve course scheduling, resource allocation and student sectioning. The UTP language combines a formal domain model and a rules formalism to state constraints. The model is based on a multi-scale schedule horizon (i.e., weeks, weekdays and daily slots), a hierarchical course structure (i.e., course parts, part classes and class sessions), and an extended set of resources (i.e., rooms, lecturers, students and student groups). Student groups must be formed to populate classes and class sessions are to be scheduled individually and allocated single or multiple rooms and lecturers. The model encodes sectioning constraints on classes, core scheduling constraints on sessions as well as compatibility, capacity and cardinality constraints on resource allocation. Rules allow to state conjunctions of constraints on selected sets of entities and sessions using a catalog of timetabling predicates and a syntax to group, filter and bind entities and sessions. As for implementation, the UTP language is based on XML and comes with a tool chain that flattens rules into constraints and converts instances to solver-compatible formats. We present here the abstract syntax of the UTP language and alternative constraint programming models developed in MiniZinc and CHR together with preliminary experiments on a real case study.

Keywords University Timetabling · Domain-Specific Modeling Language · Constraint Programming · Resource Scheduling

This work has been funded by a research grant from Université d'Angers.

Vincent Barichard, Corentin Behuet, David Genest, Marc Legeay, David Lesaint
Univ Angers, LERIA, SFR MATHSTIC, F-49000 Angers, France
Tel.: +33 241-735-420
E-mail: vincent.barichard@univ-angers.fr
E-mail: corentin.behuet@univ-angers.fr
E-mail: david.genest@univ-angers.fr
E-mail: marc.legeay@univ-angers.fr
E-mail: david.lesaint@univ-angers.fr

1 Introduction

Course and exam organization in universities involves strategic, tactical and operational decisions relating to curriculum design, student sectioning, course staffing, room planning, class scheduling and resource allocation [28]. These computational tasks and their overall coordination vary between countries and educational institutions as does the level of process automation and decision tool support [35]. In French universities for instance (see Figure 1), curricula are conventionally revisited every 5 years and students enroll in courses prior to each teaching period in the course of the academic year. Demand is matched by sectioning courses into classes, partitioning students into fixed groups, and populating classes with groups. Eligible groups, lecturers, rooms and equipment are then identified for each course before class sessions get scheduled and allocated the necessary resources. Each stage involves different stakeholders with their own requirements (faculty departments, administrative units, course owners, lecturers, tutors, etc.) and the workflow naturally allows for deviations and contingencies (marginal amendments to curricula on a yearly basis, late student registrations, staff absences, etc.).

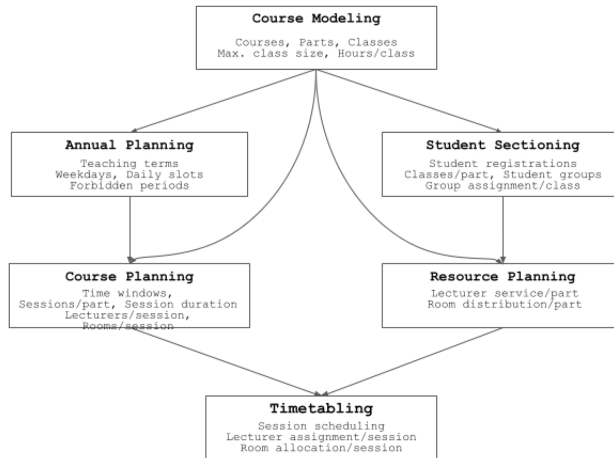


Fig. 1 Conventional workflow for course organization in French universities.

Various problem formulations together with data formats and algorithms have been proposed in the literature to tackle specific aspects of university timetabling including curriculum balancing [15, 17, 33], student sectioning [31, 34], examination timetabling [13, 8, 29], curriculum-based or post-enrolment-based course timetabling [29, 10, 27, 12, 22, 16], tutor allocation [14], and minimal timetabling perturbation [28, 26]. Modeling languages have also been developed, notably the `XML` language used in the 2019 international timetabling competition [30, 24] (which we refer to as the `ITC-2019` language) which pro-

vides a catalog of constraints and supports model variability. We adopt a similar approach in this paper and introduce a class of university timetabling problems called UTP that involve course scheduling, resource allocation and student sectioning. We present a domain-specific language to model UTP instances (UTP language) which is designed around a formal domain model and a rules language to state constraints. Each instance is decomposed into a model of entities, a rule set and a solution component. Rules express collections of timetabling constraints on model entities and the solution component lists assignment decisions. The latter may be void, partial or inconsistent to accommodate different contexts (e.g., a solution for student sectioning to turn into a complete timetable, an outdated solution that must be revised or repaired).

Similarly to the ITC-2019 language, the UTP language adopts a multi-scale schedule horizon (i.e., weeks, weekdays and daily slots), a mixed set of resources (i.e., students, student groups, rooms and lecturers), and a hierarchical course structure (i.e., course parts, part classes and class sessions). In our approach however, class sessions (a.k.a., class meetings) are considered as first-class objects that must be scheduled individually alongside resources. The model supports single-resource sessions (e.g., single lecturer) as well as multi-resource sessions (e.g., hybrid teaching), and encodes core constraints relating to student sectioning, session scheduling and resource allocation. All resources are assumed cumulative (i.e., rooms, lecturers and students may host, teach and attend overlapping sessions) but this policy may be overridden with disjunctive scheduling rules. The rules language effectively allows to enforce additional constraints on selected sets of sessions and entities (i.e., resources and course elements). Rules are expressed using a catalog of timetabling predicates and a comprehension syntax to group, filter and bind sessions and entities. Specifically, each rule denotes a conjunction of UTP constraints sharing the same predicate (e.g., periodicity of all lecture classes of a course) and constraints are technically generated through a rule flattening process.

Note that all constraints are handled as hard constraints and each UTP instance is reduced to a hard constraint satisfaction problem (CSP). The ability to model preferences and multi-criteria objectives by the means of soft constraints is paramount in course timetabling and will be the subject of future extensions. Likewise, the catalog of UTP predicates still lacks important constraints (e.g., gap, distribution and pattern constraints - see e.g. [6,16]) which will be gradually added in future versions.

As for implementation, the UTP language is based on XML and embedded in two constraint modeling languages, namely, MiniZinc [32,3] and CHR [18]. We developed a tool chain consisting of a XML parser, a rule processor to flatten rules into constraints, and an encoder to convert the resulting instances to solver-compatible formats (see Figure 2). Beyond MiniZinc and CHR, constraint-based UTP instances may be used as inputs to any solver implementing the model and predicates of the UTP language. We do not discuss here the XML syntax of the language (the reader is referred to [1] which provides access to the detailed specification, the MiniZinc and CHR models, the

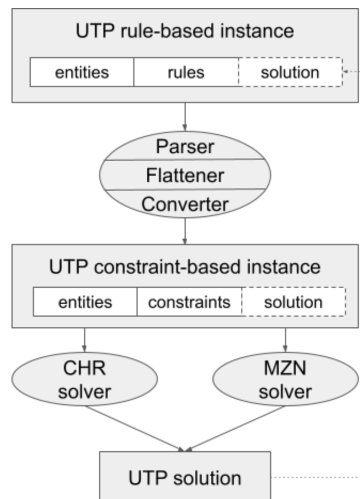


Fig. 2 The UTP toolchain.

tool suite, and a benchmark of instances). Rather, we present the abstract syntax of the UTP language and provide semantics for the key components.

The remainder of the paper is organized as follows. Section 2 introduces the UTP language and draws a comparison with the ITC-2019 schema. Section 3 presents a generic constraint-based UTP model. Section 4 discusses its implementation using MiniZinc and CHR and the cross-validation of the models on a real instance. Section 5 concludes and discusses extensions of this work.

2 University Timetabling Problem

A UTP instance is defined by an entity model and a rules set. A solution to a UTP instance is a list of choices made for all the decisions at stake that satisfies the core constraints of the entity model and the constraints expressed by the rules. We provide in this section an informal description and set-theoretic semantics for the UTP language components, namely the entity model (Section 2.1), constraints (Section 2.2), rules (Section 2.3) and solution (Section 2.4). Section 2.5 draws a comparison between the UTP language and the ITC-2019 schema.

2.1 Entity model

The entity model of a UTP instance defines its schedule horizon, course structure and resources, as well as properties of entities and relational maps (see Figure 3 for a sketch of the meta-model and Figure 4 for a toy example). First, the entity model uses a time grid that decomposes into weeks, weekdays and daily slots. Weeks share the same weekdays and weekdays the same daily slots. The latter make up 24 hours and have the same duration. Note that neither

successive weeks nor successive weekdays are assumed to be consecutive. The schedule horizon is implicitly defined by the series of time slots mapping to week, weekday and daily slot combinations. Slots hence serve as time points to represent start and end times of course sessions and to measure session duration, travel time and any gap between sessions.

Courses have a tree-structure wherein each course (e.g., Algorithms) decomposes into parts (e.g., Lecture and Lab), parts into classes (e.g., lecture classes A and B), and classes into sessions (e.g., sessions 1 to 10 for each lecture class). Class sessions are the elementary tasks to schedule when solving a `UTP` instance and the model fixes their number, duration and sequencing. First, the classes of a course part are decomposed into an identical number of sessions of equal duration, both constants being part-specific. Although this approach forbids classes using different session durations in a course part, it is paramount to capture requirements that rely on clear-cut sessions (e.g., starting lab classes after 2 lecture sessions, synchronizing the 5th sessions of the lab classes for a joint examination). Second, the sessions of a class are ranked in the model and must be sequenced accordingly in any solution (session 1 before session 2 . . .). Note that sessions are considered uninterruptible and, in particular, may not overlap two days.

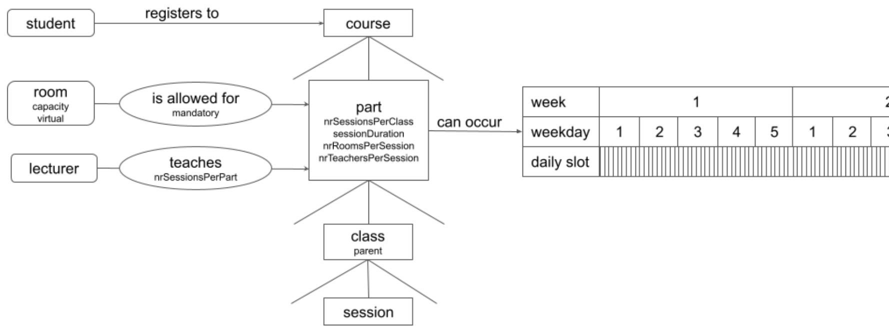


Fig. 3 Entity meta-model.

`UTP` resources fall into 4 types, namely, rooms, lecturers, students and (student) groups. All the resources of an instance, except groups (see Section 2.4), are declared and typed in the entity model. In practice, upstream processes and decisions determine the suitable rooms, eligible lecturers, candidate students and allowed times for the different courses (e.g., faculties prescribing degree-specific time grids, departments implementing room pooling policies and naming lecturers for courses, students registering to courses). These compatibility constraints are modeled by associating sets of possible start times, rooms and lecturers to each course part and a set of registered students to each course. Each session then inherits the sets of allowed resources from the course part and the course it belongs to.

The entity model also encodes flow constraints that govern the distribution of resources over courses based on student registrations and capacity planning decisions (e.g., workload distribution between lecturers). First, each lecturer is allocated a fixed number of sessions in each course part he is eligible for, leaving lecturer-to-session assignment decisions to solvers. Second, each room allowed in a course part may be freely allocated to any session of the part (possibly none) but the model provides the flexibility to mark a room as mandatory in which case it will host or co-host all the sessions. As for students, the sectioning policy is implicit and complies with the course structure, i.e., each student must be assigned to a single class in each part of a course he has registered to and attend all sessions of these classes. In addition, the model supports group nesting constraints between classes to implement course-specific policies (e.g., aggregating student groups bottom-up from labs to lectures) or cross-course sectioning (e.g., imposing the same groups between classes of different courses of a curriculum).

Resource utilization is naturally subject to demand and capacity constraints. Since modalities differ from one environment to the next, the language supports disjunctive and cumulative resources. The default policy is to consider all students, groups, lecturers and rooms as cumulative resources, i.e., they can attend, teach or host simultaneous sessions. Note though that rules may be stated to make some resources fully disjunctive or to prevent specific sessions from overlapping. Support for cumulative resources is paramount to address flexible attendance requirements (e.g., students assigned optional tutoring sessions that may overlap with compulsory courses) or to handle multi-class events (e.g., rooms hosting several classes for an exam or a conference). The model imposes no limits on the number of parallel sessions lecturers and students may attend. Rooms however may only host class sessions whose cumulated headcount is within their capacity. Upper bounds on room capacity and class size are encoded for all rooms and classes and the model also allows uncapacitated rooms to cater for the case of virtual rooms.

The language also supports sessions using multiple resources of the same type. The need for multiple rooms or lecturers arises in practical situations (e.g., multi-room sessions for hybrid teaching, joint supervision of practical work sessions, exams requiring several monitors). To this end, the model associates to each course part the number of lecturers required per session and indicates whether the sessions are single- or multi-rooms. Note that sessions without lecturers or rooms are allowed (e.g., unsupervised student project sessions). The model enforces specific constraints to handle multi-room sessions which override the default room allocation policy. Specifically, students attending the session may be freely dispatched in rooms irrespectively of the group structure, the cumulated capacity of the allocated rooms is taken into account for hosting, uncapacitated rooms cannot be allocated, and the allocated rooms are considered disjunctive for the time of the session.

Note finally that the language provides users with the ability to label resources and course elements to define their own classes of entities (e.g., teams

of lecturers, blocks of rooms). Labels together with built-in entity types and identifiers are used to filter entities and to scope rules appropriately.

We formalize below the entity model and introduce notations that will be used thereafter. Let E denote the set of entities and S the set of sessions. E is partitioned into a set of courses C, a set of course parts P, a set of classes K, a set of students U, a set of lecturers L, a set of rooms R, and the singleton domain of courses C^* ($C^* = \{C\}$). Let $\mathcal{E} = \{C^*, C, P, K, U, L, R\}$ denote the set of entity types ($E = \cup_{X \in \mathcal{E}} X$) and $\prec = \{(C^*, C), (C, P), (P, K), (K, S), (U, C), (L, P), (R, P)\}$ denote the relation over $\mathcal{E} \cup \{S\}$ that models the course hierarchy and the distribution of resource types over course components.

$\prec^*$ denotes the transitive closure of $\prec$ over $\mathcal{E} \cup \{S\}$ and $d^{X,Y} : X \rightarrow 2^Y$ denotes the function mapping each element of X to its set of compatible elements in Y for each pair $X \prec^* Y$. For instance, $d^{R,P}$ represents the distribution of rooms over course parts, $d^{P,K}$ the decomposition of course parts into classes, $d^{K,S}$ the decomposition of classes into sessions, and $d^{R,S}$ the inferred distribution of rooms over sessions. The functions corresponding to the pairs of $\prec$ are directly encoded in the entity model and the remaining functions are defined inductively using recursive aggregation.

We shall denote by $d_i^{X,Y}$ the image of entity i of type X over 2^Y and by $d^{Y,X}$ the inverse of $d^{X,Y}$. Equation (1) below models the hierarchical decomposition of course elements[1], Equation (2) is the closure rule over $\prec^*$, and Equation (3) models inverse maps.

$$\forall (X, Y) \in \{(C^*, C), (C, P), (P, K), (K, S)\} : Y = \sqcup_{i \in X} d_i^{X,Y} \quad (1)$$

$$\forall X, Y, Z \in \mathcal{E} \cup \{S\} : X \preceq^* Y \preceq^* Z \Rightarrow (\forall i \in X : d_i^{X,Z} = \sqcup_{j \in d_i^{X,Y}} d_j^{Y,Z}) \quad (2)$$

$$\forall X, Y \in \mathcal{E} : X \preceq^* Y \Rightarrow (\forall i \in X, j \in Y : j \in d_i^{X,Y} \Leftrightarrow i \in d_j^{Y,X}) \quad (3)$$

Table 1 provides the full list of constants, sets, properties and relational maps encoded in the entity model.[2]

2.2 Predicates and constraints

UTP constraints apply to pairs, called e-maps, which associate an entity with a non-empty subset of its compatible sessions. Constraints are built with predicates whose signature includes e-map variables, the number of which is referred to as the arity of the predicate. Note that some predicates may also accept parameters. Let $F = \cup_{X \in \mathcal{E}} \{(e, S') \mid e \in X, S' \subseteq d_e^{X,S} \wedge S' \neq \emptyset\}$ denote the set

---

[1] $\sqcup$ denotes the disjoint union operation, i.e. set union over pairwise disjoint sets.

[2] The following rules apply. $H = \{i.d.m + j.m + k \mid 0 \leq i < w, 0 \leq j < d, 1 \leq k \leq m\}$. For each class k in part p, $\{rank_s^S \mid s \in d_k^{K,S}\} = \{1, \ldots, |d_k^{K,S}|\}$, and $parents_k^{K,K} \not\subset d_p^{P,K}$. For each pair of sessions s, s', $(s, s') \in O$ iff $d_s^{S,K} = d_{s'}^{S,K}$ and $rank_{s'}^S = rank_s^S + 1$. For each course part p, $team_p^P.|d_p^{P,S}| = \sum_{l \in d_p^{P,L}} service_{l,p}^{L \times P}$.

(w, d, m)	the number of weeks w, weekdays d and daily slots m
H	the time slots
E	the entities
$C^* \subseteq E$	the course domain
$C \subseteq E$	the courses
$P \subseteq E$	the course parts
$K \subseteq E$	the classes
$R \subseteq E$	the rooms
$L \subseteq E$	the lecturers
$U \subseteq E$	the students
$d_i^{X,Y} \subseteq Y$	the entities of type Y associated with entity i of type X
$\mathcal{L} \subseteq 2^E$	the labels
$G \subseteq 2^U$	the groups of students
S	the sessions
$d_i^{X,S} \subseteq S$	the sessions compatible with entity i of type X
$d_s^{S,X} \subseteq X$	the entities of type X compatible with session s
$d_s^{S,H} \subseteq H$	the start times allowed for session s
$length_s^S \in H$	the duration of session s
$rank_s^S \in \mathbb{N}^*$	the rank of session s in its class
$O \subseteq S \times S$	the pairs of sessions with consecutive ranks in a class
$parents_k^{K,K} \subseteq K$	the parent classes of class k if any
$maxsize_k^K \in \mathbb{N}$	the maximum size of class k
$capacity_r^R \in \mathbb{N}$	the capacity of room r
$virtual_r^R \in \mathbb{B}$	whether room r is virtual or not
$V \subseteq R$	the virtual rooms
$multi_p^P \in \mathbb{B}$	whether course part p is multi-room or not
$M \subseteq P$	the multi-room parts
$mandatory_p^P \subseteq R$	the mandatory rooms of part p
$team_p^P \in \mathbb{N}$	the number of lecturers required by every session of part p
$service_{l,p}^{L \times P} \in \mathbb{N}$	the number of sessions required by lecturer l in part p

Table 1 Entity model: constants, sets, maps and relations.

of e-maps, a UTP constraint has the form

$$c((e_1, S_1), \ldots, (e_m, S_m), p_1, \ldots, p_n) \tag{4}$$

where c is a predicate symbol of arity m, $(e_1, S_1), \ldots, (e_m, S_m)$ are e-maps ($(e_i, S_i) \in F$, $i = 1 \ldots m$) and $p_1, \ldots, p_n$ are values for the parameters of c ($n \geq 0$). Three constraints (C1, C2, C3) are illustrated in Figure 4.

Every predicate may be used indistinctly with e-maps defined on course elements or on resources. E-maps defined on resources are interpreted as conditional session-to-resource assignments when checking constraints whereas e-maps defined on course elements are unconditional assignments since they model constitutive sessions. In other words, a constraint is only evaluated on the sessions for which its e-map arguments and the considered solution propose the same entity assignment.[3]

[3] Formally, let $x_e^{E,S}$ be the variable denoting the set of sessions assigned to entity e and $S'_1, \ldots, S'_m$ be sets of sessions, the conditionality of a constraint c is stated as follows: $(x_{e_1}^{E,S} = S'_1 \wedge \ldots \wedge x_{e_m}^{E,S} = S'_m) \Rightarrow (c((e_1, S_1), \ldots, (e_m, S_m), p_1, \ldots, p_n) \Leftrightarrow c((e_1, S_1 \cap S'_1), \ldots, (e_m, S_m \cap S'_m), p_1, \ldots, p_n))$.

It follows that a constraint is evaluated on every session that is mapped to a course element by one of its e-map arguments. Constraints that apply exclusively to course elements are therefore unconditional. Note also that the use of e-maps that model the whole set of sessions compatible with an entity will necessarily constrain any session that may be assigned to this entity.

Name	Arity	Parametric	Semantics
same_daily_slot	1	no	Sessions start on the same daily slot
same_weekday	1	no	Sessions start on the same weekday
same_weekly_slot	1	no	Sessions start on the same weekly slot
same_week	1	no	Sessions start the same week
same_day	1	no	Sessions start the same day
same_slot	1	no	Sessions start at the same time
forbidden_period	1	yes	Sessions cannot start in the given time period
at_most_daily	1	yes	The number of sessions scheduled in the daily period is upper-bounded
at_most_weekly	1	yes	The number of sessions scheduled in the weekly period is upper-bounded
sequenced	≥ 2	no	Sessions are sequenced
weekly	1	no	Sessions are weekly
no_overlap	1	no	Sessions cannot overlap
travel	1	yes	Travel time is factored in if sessions hosted in the given rooms
same_rooms	1	no	Sessions are hosted in the same room(s)
same_students	1	no	Sessions are attended by the same student(s)
same_lecturers	1	no	Sessions are taught by the same lecturer(s)
adjacent_rooms	1	yes	Sessions are hosted in the given adjacent rooms
lecturer_distribution	≥ 2	yes	Distributes lecturer workload over classes

Table 2 Catalog of `UTP` predicates.

Table 2 lists the predicates of the language and indicates which are variadic or parametric. The first predicates `same_daily_slot`, ..., `same_slot` enforce common restrictions on the start times of the targeted sessions (e.g., sessions starting the same day). Additionally, any start time interval may be forbidden by passing its start and end points as parameters to predicate `forbidden_period`. Predicates `at_most_daily` and `at_most_weekly` upper-bound the number of sessions scheduled daily or weekly within the given time interval. `sequenced` is a n-ary predicate ($n \geq 2$) which constrains the latest session of the i-th e-map to end before the earliest session of $i+1$-th e-map ($i = 1..n-1$). Predicate `weekly` ensures sessions are scheduled weekly without presuming any particular sequencing. Predicate `no_overlap` ensures sessions do not overlap in time and is typically used to model disjunctive resources. Predicate `travel` factors in any travel time incurred between consecutive sessions hosted in distant rooms. The travel time matrix is a parameter of the predicate. `same_rooms`, `same_students` and `same_lecturers` require that sessions be assigned to the same set of rooms, students or lecturers. Predicate `adjacent_rooms` require that sessions be hosted in adjacent rooms based on an adjacency graph passed as a parameter. Lastly, predicate `lecturer_distribution` distributes the volumes of sessions represented by the different e-map arguments among different lecturers. Lecturers and session volumes are parameters of the predicate.

2.3 Rules

Rules are used to state conjunctions of constraints and in particular single constraints. Each rule is defined by a universally quantified formula which bounds the domains of the e-map variables of a given predicate. The collection of constraints hence represented is derived by instantiating the predicate with each tuple of e-maps belonging to the cross-product of the prescribed domains. E-map domains are not given in extension but represented using a language of selectors allowing to generate and filter e-maps. Let $\mathcal{F}$ denote the language of e-map domain selectors, a `UTP` rule has the form

$$c(F_1, \ldots, F_m, p_1, \ldots, p_n) \tag{5}$$

and is interpreted by the formula

$$\forall (e_1, S_1) \in [\![F_1]\!], \ldots, (e_m, S_m) \in [\![F_m]\!] : c((e_1, S_1), \ldots, (e_m, S_m), p_1, \ldots, p_n) \tag{6}$$

where c is a predicate symbol of arity m, $F_1, \ldots, F_m$ are selectors ($F_i \in \mathcal{F}$, $i = 1 \ldots m$), $[\![F_i]\!]$ denotes the domain of e-maps represented by selector $F_i \in \mathcal{F}$, and $p_1, \ldots p_n$ are values for the parameters of c ($n \geq 0$), .

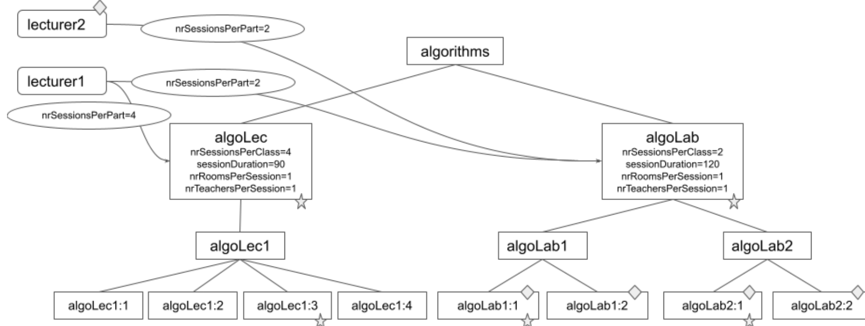
The language of selectors allows to target entities based on type, label or identifier and to filter their sets of sessions based on session rank and mutual compatibility with other entities. It is complete in the sense that it allows to construct any domain of e-maps whose entities share the same type. For instance, one may construct the e-maps which associate any of the rooms labeled `Building-A` with the compatible sessions of rank 2 or 4 that are also constitutive of course `course-1` or class `class-3`. A selector combines a generator and an optional list of filters. Generators and filters are triples (T_i, L_i, O_i) consisting of an entity type T_i, an entity label or identifier L_i and a subset of session ranks O_i (a.k.a., session mask), the latter two elements being optional. A selector matches any e-map whose entity satisfies the type, label and identifier constraints of the generator and whose image includes any compatible session satisfying the mask of the generator and one of the filters. Note that rules featuring null selectors are discarded during the flattening stage.

Let $\mathcal{O}$ denote the range of session ranks, $d^{\mathcal{O},S} : \mathcal{O} \rightarrow 2^S$ the rank-based partitioning of sessions ($s \in d^{\mathcal{O},S}_o$ iff $rank^S_s = o$), and $\mathcal{L}^* = \mathcal{L} \cup \{E\} \cup \{\{e\} \mid e \in E\}$ the set of labels completed with the whole set of entities to mock label optionality and singleton entities to support identity-based selection, the language of selectors is the set $\mathcal{F} = \cup_{n \geq 1} (\mathcal{E} \times \mathcal{L}^* \times 2^{\mathcal{O}})^n$. Each selector $d = ((T_1, L_1, O_1), \ldots, (T_k, L_k, O_k))$ decomposes into a generator (T_1, L_1, O_1) and a possibly empty list of filters $((T_2, L_2, O_2), \ldots, (T_k, L_k, O_k))$. d matches any e-map whose entity has type T_1 and label L_1 and whose image includes any compatible session satisfying mask O_1 and any of the filters. The set of e-maps

$\llbracket d \rrbracket$ matched by d is defined by

$$\llbracket d \rrbracket = \bigcup_{e \in T_1 \cap L_1} \Big\{ (e, S') \mid S' = d_e^{T_1,S} \bigcap \bigcup_{i=2\dots k} \Big(d^{T_i,S}[L_i] \bigcap d^{\mathcal{O},S}[O_1 \cap O_i] \Big) \wedge S' \neq \emptyset \Big\}$$

where $d^{X,Y}[X'] = \bigcup_{i \in X'} d_i^{X,Y}$ with $X' \subseteq X$.




```
forbidden_period((<(L,lecturer2,_)>,9120,9240)                          (R1)
sequenced(<(K,_,{3}),(P,algoLec,_)>, <(K,_,{1}),(P,algoLab,_)>)         (R2)

forbidden_period((lecturer2,{algoLab1:1,algoLab1:2,algoLab2:1,algoLab2:2}),
9120,9240)                                                              (C1)
sequenced((algoLec1,{algoLec1:3}), (algoLab1,{algoLab1:1}))             (C2)
sequenced((algoLec1,{algoLec1:3}), (algoLab2,{algoLab2:1})              (C3)
```

Fig. 4 Rules flattening and corresponding constraints on a toy example.

Figure 4 illustrates the rules flattening process on a toy example. Course `algorithms` is split into a lecture part `algoLec` and a lab part `algoLab`. The lecture part has a single class of 4 sessions taught by `lecturer1` and the lab part has 2 classes of 2 sessions each taught by `lecturer1` or `lecturer2`. Rule R1 requires that `lecturer2` has no session between slots 9120 and 9240, corresponding for instance to 8am and 10am on Tuesday of week 2. The selector includes no mask and no filter hence matches with all possible sessions of `lecturer2` as indicated with diamonds on Figure 4. The resulting domain of e-maps is the singleton $\{(lecturer2, d_{lecturer2}^{L,S})\}$ and the rule is flattened into a single `forbidden_period` constraint (C1). Rule R2 requires that the first sessions of the labs start after the third lecture. The two selectors include a filter. The first selector matches with all class sessions of rank 3 in part `algoLec`, and the second matches with all class sessions of rank 1 in part `algoLab` as indicated with stars on the figure. The rule is flattened into 2 `sequenced` constraints (C2 and C3) corresponding to the cross product of the e-map domains

$\{(algoLec1, \{s \in S \mid rank_s^S = 3\} \cap d_{algoLec}^{P,S})\}$ and $\{(algoLab1, \{s \in S \mid rank_s^S = 1\} \cap d_{algoLab}^{P,S}), (algoLab2, \{s \in S \mid rank_s^S = 1\} \cap d_{algoLab}^{P,S})\}$.

2.4 Solution

The solution component includes assignment decisions relating to the choice of slots and resources for sessions, the placement of students in groups and the assignment of groups to classes. The solution hence represented may be partial, even empty, and does not have to be consistent with the constraints built in the entity model or entailed by the rules. The support for partial solutions allows to tackle subproblems using separate `UTP` instances and solution seeds. For instance, a scheduling instance may be defined on the basis of partial and consistent solutions pre-generated for the student sectioning and resource allocation subproblems. Likewise, the support for inconsistent solutions is paramount to repair solutions that have become inconsistent due to unforeseen changes.

Student groups are considered a by-product of student sectioning. For this reason, groups may only be listed in the solution component, not in the entity model, and defined both by the students they include and the classes they are assigned to. This sectioning process is subject to different constraints. First, students are partitionned into groups and students are inextricably bound to their group. Second, a group may only include students with identical course registrations. Third, group-to-class assignments must comply with any subgroup inclusion constraint stated in the entity model.

2.5 Related work

We highlight here the main differences between the `UTP` language and the `ITC-2019` language (`ITC-2019` for short).

A first difference between the two frameworks lies in the representation of the possible times a class can meet. In `UTP`, a class is defined by a single sequence of sessions of equal duration and the problem is to schedule each session. In `ITC-2019`, a class is given alternative fixed session schedules (`times` elements in the `XML` schema) and the problem is to choose one of the schedules for the class. A schedule is the repetition over a set of weeks of one or more sessions that have the same duration and start on specific days of the week at the same predefined time (daily slot). The two representations are not reducible to one another. For instance, alternative schedules using different session durations cannot be modeled in `UTP`. Conversely, class schedules where sessions do not necessarily start on the same daily slot cannot be modeled in `ITC-2019`. Nevertheless, basic class schedules may be represented in either approach by stating `ITC-2019` constraints or `UTP` rules on classes. For instance, a class meeting every week on the same day and the same daily slot, both being subject to time restrictions, may be modeled using `same_daily_slot`,

weekly and forbidden_period constraints. The implementation of a more comprehensive reduction method will be the subject of future work.

Second, ITC-2019 represents alternative course configurations by introducing an intermediate layer in the course hierarchy that sits between courses and parts. The configurations of a course typically differ in their number of (sub)parts and are mutually exclusive from a student sectioning standpoint, that is, a registered student must be assigned a single configuration and attend all of its parts. This feature is not currently supported in UTP. As for resources, UTP explicitly represents lecturers on par with rooms whereas ITC-2019 only models rooms. UTP also provides the flexibility to allocate different resources within a class (and specify lecturer workload in particular) whereas the same room must be allocated in ITC-2019. Additionally, UTP supports multi-resource sessions whereas ITC-2019 is restricted to single-room sessions.

Lastly, the two constraint languages present important differences. While ITC-2019 constraint predicates apply to classes, UTP predicates apply to any set(s) of sessions and may be used in particular on individual sessions, hence granting finer-grained control. Besides, UTP rules and the selector language allows to constrain any class of resources or course elements in a concise way.

Lastly, the ITC-2019 schema addresses the timetabling problem as a combinatorial optimization problem. It includes a cost function weighting 4 criteria which respectively penalize the choice of sessions and rooms for the classes, the violations of constraints and the overlapping of sessions per student. In its current version, the UTP language addresses the problem as a hard constraint satisfaction problem. The integration of soft constraints and the possibility of aggregating penalties or preferences, either in solution generation or repair contexts, is under investigation.

3 A Constraint-Based Model for UTP

We introduce in this section a constraint-based model for UTP instances. The model of an instance combines the constraints associated to the entity model and the constraints resulting from the flattening of the rules, if any. The former are decomposed into 4 fragments relating to student sectioning, resource distribution, session scheduling and resource allocation. We present each fragment in turn by reusing notations of Table 1, illustrate the modeling of some predicates before discussing opportunities for model reformulations on a per-instance basis. Note that some constraints are given a naive formulation to clarify semantics and more efficient implementations using MiniZinc and CHR will be discussed in Section 4.

Table 3 lists the decision variables of the model. All, except time slot variables, are set variables.

$x_g^{G,U} \subseteq U$	the set of students assigned to group g
$x_k^{K,G} \subseteq G$	the set of groups assigned to class k
$x_s^{S,R} \subseteq R$	the set of rooms assigned to session s
$x_s^{S,L} \subseteq L$	the set of lecturers assigned to session s
$x_r^{R,S} \subseteq S$	the set of sessions assigned to room r
$x_s^{S,H} \in H$	the start slot assigned to session s

Table 3 The decision variables.

3.1 Student Sectioning

The sectioning constraints partition students into groups and assign groups to classes while satisfying sectioning rules and class size upper-bounds. Constraint (1) ensures the groups partition the set of students. Constraint (2) prevents the clustering of students who register to different courses. Constraint (3) ensures that classes of a course part have no shared groups and Constraint (4) that the group of a student attends each part of a course he is registered to. Constraint (5) implements the parent relationships between classes. Constraint (6) ensures maximum class size is never exceeded by the number of students in its groups. Note that expressions $(g \in x_k^{K,G})$ in this contraint denote pseudo-boolean variables. The same notation is used for convenience in other constraints.

$$U = \bigsqcup_{g \in G} x_g^{G,U} \quad (1)$$

$$\forall u, u' \in U, d_u^{U,C} \neq d_{u'}^{U,C}, g \in G: \quad \{u, u'\} \nsubseteq x_g^{G,U} \quad (2)$$

$$\forall p \in P, k, k' \in d_p^{P,K}, k \neq k': \quad x_k^{K,G} \bigcap x_{k'}^{K,G} = \emptyset \quad (3)$$

$$\forall u \in U, g \in G: \quad (u \in x_g^{G,U}) \rightarrow \bigwedge_{p \in d_u^{U,P}} \bigvee_{k \in d_p^{P,K}} (g \in x_k^{K,G}) \quad (4)$$

$$\forall k \in K, k' \in parents_k^{K,K}: \quad x_k^{K,G} \subseteq x_{k'}^{K,G} \quad (5)$$

$$\forall k \in K: \quad maxsize_k^K \geq \sum_{g \in G} |x_g^{G,U}|.(g \in x_k^{K,G}) \quad (6)$$

3.2 Resource Distribution

Resource distribution involves domain, cardinality and basic summation constraints. Constraint (7) defines the allowed rooms and allowed lecturers per session. Constraint (8) models single-room sessions and Constraint (9) models mandatory rooms of course parts. Constraint (10) ensures sessions get assigned the right number of lecturers (possibly none) as defined in each course part and Constraint (11) ensures each lecturer is assigned the expected number of sessions.

$$\forall W \in \{R, L\}, s \in S: \quad x_s^{S,W} \subseteq d_s^{S,W} \tag{7}$$

$$\forall p \in P \setminus M, s \in d_p^{P,S}: \quad |x_s^{S,R}| = 1 \tag{8}$$

$$\forall p \in P, s \in d_p^{P,S}: \quad mandatory_p^P \subseteq x_s^{S,R} \tag{9}$$

$$\forall p \in P, s \in d_p^{P,S}: \quad team_p^P = |x_s^{S,L}| \tag{10}$$

$$\forall l \in L, p \in P: \quad service_{l,p}^{L \times P} = \sum_{s \in d_p^{P,S}} (l \in x_s^{S,L}) \tag{11}$$

3.3 Session Scheduling and Resource Allocation

Session scheduling and resource allocation involve positioning, sequencing, non-overlapping and capacity constraints. Constraint (12) defines the allowed slots per session and Constraint (13) ensures sessions do not span over two days. Constraint (14) sequences sessions if they are ranked consecutively in a class. Constraint (15) models multi-room class sessions and enforces exclusive access to their rooms. This constraint is formulated using auxiliary predicate $split(w, S_1, S_2)$ (18) which ensures no session of S_1 overlaps with a session of S_2 if both are assigned to resource w. We provide a naive decomposition of this predicate using Predicate (19). Constraints (16) and (17) model room utilization and capacity limits and use auxiliary variables $y_{r,k,s,h}$. $y_{r,k,s,h}$ models the number of students attending session s of class k in room r at time h and is defined using auxiliary constraint (20). Constraint (16) is the default cumulative constraint which applies to non-virtual rooms when allocated to single-room sessions. Constraint (17) handles the specific case of multi-room sessions and ensures the cumulated capacity of the rooms used by a multi-room session exceeds the number of students attending the session. Note that the constraint is purely quantitative and allows each individual group to be distributed over different rooms.

$$\forall s \in S: \quad x_s^{S,H} \in d_s^{S,H} \tag{12}$$

$$\forall s \in S: \quad x_s^{S,H}/m = (x_s^{S,H} + length_s^S)/m \tag{13}$$

$$\forall (s, s') \in O: \quad x_s^{S,H} + length_s^S \leq x_{s'}^{S,H} \tag{14}$$

$$\forall k \in d^{P,K}[M]: \quad \bigwedge_{r \in d_k^{K,R}} split(r, d_k^{K,S}, d_r^{R,S} \setminus d_k^{K,S}) \tag{15}$$

$$\forall r \in R \setminus V: \quad \bigwedge_{h \in H} capacity_r^R \geq \sum_{\substack{p \in d_r^{R,P} \setminus M \\ k \in d_p^{P,K} \\ s \in d_k^{K,S}}} y_{r,k,s,h} \tag{16}$$

$$\forall p \in M: \quad \bigwedge_{\substack{h \in H \\ k \in d_p^{P,K} \\ s \in d_k^{K,S}}} \sum_{r \in d_p^{P,R}} (r \in x_s^{S,R}).capacity_r^R \geq \max_{r \in d_p^{P,R}} y_{r,k,s,h} \tag{17}$$

Let $W \in \{R, L, U\}, w \in W, S_1, S_2 \subseteq S$:

$$split(w, S_1, S_2) \leftrightarrow \bigwedge_{\substack{s_1 \in S_1, s_2 \in S_2 \\ s_1 \neq s_2}} (w \in x_{s_1}^{S,W} \cap x_{s_2}^{S,W} \rightarrow split(s_1, s_2)) \tag{18}$$

Let $s, s' \in S$:

$$split(s, s') \leftrightarrow (x_s^{S,H} + length_s^S \leq x_{s'}^{S,H} \vee x_{s'}^{S,H} + length_{s'}^S \leq x_s^{S,H}) \tag{19}$$

Let $r \in R, k \in K, s \in S, h \in H$:

$$\begin{aligned} y_{r,k,s,h} = (r \in x_s^{S,R}).(x_s^{S,H} \leq h \wedge h \leq x_s^{S,H} + length_s^S) \\ .\sum_{g \in G} |x_g^{G,U}|.(g \in x_k^{K,G}) \end{aligned} \tag{20}$$

3.4 UTP Predicates

We present a subset of UTP constraint predicates, namely, `forbidden_period` (21), `same_weekday` (22), `same_rooms` (23), `no_overlap` (24) and `sequenced` (25). Note that `forbidden_period` accepts start and end point parameters. Predicate `no_overlap` uses auxiliary predicate *split* for resources (18) and a variant for course elements (26).
Let $X \in \mathcal{E}, e \in X, S' \subseteq d_e^{X,S}, h, h' \in H \ (h < h')$.

$$\begin{aligned} &forbidden_period((e, S'), h, h') \\ &\qquad \leftrightarrow \bigwedge_{s \in S'} (x_s^{S,H} + length_s^S \leq h \vee h' < x_s^{S,H}) \end{aligned} \tag{21}$$

$$same_weekday((e, S')) \leftrightarrow \bigwedge_{s \in S'} x_s^{S,H}/d = (x_s^{S,H} + length_s^S)/d \tag{22}$$

$$same_rooms((e, S')) \leftrightarrow \bigwedge_{s, s' \in S'} (x_s^{S,R} = x_{s'}^{S,R}) \tag{23}$$

$$no_overlap((e, S')) \leftrightarrow split(e, S', S') \tag{24}$$

Let $i \in \{1, \ldots, n\}, X_i \in \mathcal{E}, e_i \in X_i, S_i \subseteq d_{e_i}^{X_i,S}$:

$$\begin{aligned} &sequenced((e_1, S_1), \ldots, (e_n, S_n)) \\ &\qquad \leftrightarrow \bigwedge_{j=1 \ldots n-1} \max_{s \in S_j} (x_s^{S,H} + length_s^S) \leq \min_{s \in S_{j+1}} x_s^{S,H} \end{aligned} \tag{25}$$

Let $X \in \{C^*, C, P, K\}, e \in X, S_1, S_2 \subseteq S$:

$$split(e, S_1, S_2) \leftrightarrow \bigwedge_{\substack{s_1 \in S_1, s_2 \in S_2 \\ s_1 \neq s_2}} (e \in d_{s_1}^{S,X} \cap d_{s_2}^{S,X} \rightarrow split(s_1, s_2)) \tag{26}$$

3.5 Reformulation

The model presented above is generic and may be adapted on a per instance basis depending on the features and rules at stake. We discuss here a few variants of the UTP problem which provide opportunities for model reformulation.

When instances only involve single-room sessions ($M = \emptyset$), one may adopt integer or enumerated room allocation variables instead of set variables $x_s^{S,R}$ and rewrite constraints accordingly. In the same way, lecturer assignment variables and constraints may be adapted when a single lecturer is required per session. Note that hybrid models mixing single or multi-resource session variables may be considered too. The temporal model may also be simplified when the time grid is coarse-grain and guarantees no session can span over consecutive start times ($\forall s \in S, length_s^S \leq \min(\{h' - h \mid h, h' \in d^{P,H}[P] \wedge h < h'\})$. This situation occurs in institutions that impose a common time grid to ensure sessions (with any travel time incurred) necessarily fit in each time slot. If so, sessions may be handled as time points rather than time intervals and temporal predicates and constraints may be adapted. Capacity constraints may also be simplified for disjunctive rooms. A room is disjunctive if a `no_overlap` constraint is stated on the whole set of its compatible sessions ($r \in D \leftrightarrow no_overlap(r, d_r^{R,S})$ where $D \subseteq R$ denotes the set of disjunctive rooms). If so, the default cumulative constraint (16) may be overridden by Constraint (27).

$$\forall r \in D : \bigwedge_{\substack{h \in H \\ k \in d_r^{R,K}}} capacity_r^R \geq \max_{s \in d_k^{K,S}} y_{r,k,s,h} \tag{27}$$

4 Constraint Programming Implementation

In this section, we present two constraint-based models for UTP instances developed in MiniZinc and CHR. The two models use the same arrays, functions and constants for representing input data. We do not list them here but they are easily understandable such as `part_sessions` which gives the set of sessions constitutive of a part, `session_rooms` which gives the set of allowed rooms for a session, `week` which gives the week of a slot, and `nr_weekly_slots` which is the number of slots in a week.

4.1 MiniZinc model

MiniZinc is a high-level language to model constrained optimization problems [32,3]. MiniZinc models are translated into Flatzinc [4] which allows to interface different types of solvers including solvers on finite domain CSPs such as Gecode [2]. The MiniZinc model for UTP is presented in Table 5 and based on the decisions variables listed in Table 4. The model uses some of the

global constraints supported in MiniZinc which are dedicated to scheduling problems.

array[U] of var G:	x_group	group assigned to a student
array[K] of var set of G:	x_groups	set of groups assigned to a class
array[S] of var set of R:	x_rooms	set of rooms allocated to a session
array[S] of var set of L:	x_lecturers	set of lecturers allocated to a session
array[S] of var H:	x_slot	starting slot of a session

Table 4 Decision variables (MiniZinc).

Sectioning constraints partition students into groups and assign each group to a class according to sectioning rules and class size thresholds. Constraint (1) allows students to be part of the same group only if they are registered to the same courses. (2) imposes that every student attends all the part of the courses to which he is registered. (3) ensures that classes from the same part do not have any common group. (4) implements the parent-child relation between classes. Lastly, (5) checks that the groups fit in the class they have been assigned to.

Resource distribution relies on domain, cardinality and sum constraints. Constraints (6) and (7) define available rooms and lecturers for each session. (8) forces the number of rooms allocated to a session according to the specific requirements of the course part (i.e., no room, single-room or multi-room). (9) allocates the required number of lecturers to a session and (10) checks that every lecturer has the right number of sessions in a part.

Session scheduling and resource allocation involves positioning, sequencing, non-overlaping and capacity constraints. Constraint (11) defines the allowed slots for each session. (12) forbids a session to be on two days. (13) sequences the sessions of a class according to their rank. Constraints (14) and (15) model multi-room sessions and the exclusive access to their rooms. (14) makes disjunctive any resource that is allocated to a multi-room session while it is hosting the session. (15) ensures that the number of students attending a multi-room session do not exceed the cumulated capacity of the allocated rooms. (16) models the mandatory rooms to be allocated. (17) models the default cumulative capacity constraint controlling the allocation of non-virtual rooms to single-room sessions. This constraint uses the `cumulative` global constraint of MiniZinc (see [9] for the Gecode implementation) which MiniZinc also reuses to rewrite the global `disjunctive` constraint.

Table 5 also presents some UTP predicates when the targeted resources are rooms. (18) implements the `forbidden_period` predicate that takes the start and end time slots of the period as parameters. (19), (20) and (21) model `same_weekday`, `same_rooms` and `sequenced` predicates, respectively. (22) implements the `no_overlap` predicate that relies on the `disjunctive` global constraint.

```
forall(u, v in U where u<v)
        (student_courses[u]!=student_courses[v] -> x_group[u]!=x_group[v])          (1)
forall(u in U, p in student_parts[u])
        (exists(k in part_classes[p])(x_group[u] in x_groups[k]))                    (2)
forall(p in P, k1, k2 in part_classes[p] where k1<k2)
        (x_groups[k1] intersect x_groups[k2] = {})                                    (3)
forall(k1 in K, k2 in class_parents(k1))(x_groups[k1] subset x_groups[k2])           (4)
forall(k in K)(maxsize[k]<=sum(g in G)
        (bool2int(g in x_groups[k]) * sum(u in U)(bool2int(x_group[u] = g)))          (5)
-----------------------------------------------------------------------------------------
forall(s in S)(x_rooms[s] subset part_rooms[session_part[s]])                        (6)
forall(s in S)(x_lecturers[s] subset part_lecturers[session_part[s]])                (7)
forall(s in S, p in P where p = session_part[s])(
        (part_room_use[p] = none -> x_rooms[s] = {})
    /\  (part_room_use[p] = single -> card(x_rooms[s]) = 1)
    /\  (part_room_use[p] = multiple -> card(x_rooms[s])>=1))                         (8)
forall(s in S)(card(x_lecturers[s]) = team[session_part[s]])                         (9)
forall(p in P, l in part_lecturers[p])
        (sum(s in part_sessions(p))(bool2int(l in x_lecturers[s]) = service[l, p]))  (10)
-----------------------------------------------------------------------------------------
forall(p in P, s in part_sessions(p))
        (week(x_slot[s]) in weeks[p]
    /\  weekday(x_slot[s]) in weekdays[p]
    /\  dailyslot(x_slot[s]) in dailyslots[p])                                        (11)
forall(s in S)
        ((x_slot[s] − 1) div nr_slots_per_day =
        (x_slot[s] + length[s] − 1) div nr_slots_per_day)                            (12)
forall(k in K, s1, s2 in class_sessions[k] where rank(s1)<rank(s2))
        (x_slot[s1] + length[s]>=x_slot[s2])                                          (13)
forall(p in P, s1 in part_sessions[p], r in part_rooms[p], s2 in room_sessions[r]
         where is_multi_rooms[p] /\ s1!=s2)
        (disjunctive([x_slot[s1], x_slot[s2]],
         [bool2int(r in x_rooms[s1]) * length[s1], bool2int(r in x_rooms[s2]) *      (14)
length[s2]]))
forall(p in P, s in part_sessions[p] where is_multi_rooms[p])
        (sum(r in part_rooms[p])(bool2int(r in x_rooms[s]) * capacity[r])
        <=sum(g in class_groups[session_class[s]])(card(group_students[g])))         (15)
forall(p in P, s in part_sessions[p])(mandatory_rooms[p] subset x_rooms[s])          (16)
forall(r in R where not(virtual[r]))(
        let {set of S: RS= room_sessions[r] intersect single_room_sessions;} in
        (cumulative([x_slot[s]|s in RS],
        [bool2int(r in x_rooms[s]) * length[s]|s in RS],
         [sum(g in G)(bool2int(g in x_groups[session_class[s]])) * sum(u in U)(
bool2int(g = x_group[u]))|s in RS], capacity[r]))                                    (17)
-----------------------------------------------------------------------------------------
forbidden_period((r, S′), h1, h2) = forall(i in S′)(
        r in x_rooms[i] -> (x_slot[i] + length[i]<=h1 \/x_slot[i]>h2))               (18)
same_weekday((r, S′)) = forall(i, j in S′ where i<j)(
        (r in x_rooms[i] intersect x_rooms[j]) ->
        (x_slot[i] div nr_weekly_slots = x_slot[j] div nr_weekly_slots))             (19)
same_rooms((r, S′)) = forall(i, j in S′ where i<j)((
        r in x_rooms[i] intersect x_rooms[j]) -> x_rooms[i] = x_rooms[j])            (20)
sequenced((r1, S1), (r2, S2)) = forall(i in S1, j in S2)(
         (r1 in x_rooms[i] /\ r2 in x_rooms[j]) -> x_slot[i]+length[i]<=x_slot[j])   (21)
no_overlap((r, S′)) =
        disjunctive([x_slot[i]|i in S′], [length[i]*bool2int(r in x_rooms[i])|i in S′])  (22)
```

Table 5 Constraints and predicates of the `MiniZinc` model.

4.2 CHR model

CHR (for *Constraint Handling Rules*) [18,21,19,20] are a committed-choice language consisting of multiple-heads guarded rules that replace constraints by more simple constraints until they are solved. CHR are a special-purpose language concerned with defining declarative constraints in the sense of *Constraint logic programming* [23,25]. CHR are a language extension that allows to introduce *user-defined* constraints, i.e. first-order predicates, into a given host language as Prolog, Lisp, Java, or C/C++. CHR have been extended to $\text{CHR}^{\vee}$ [5] that introduces the *don't know* nondeterminism in CHR [11]. This nondeterminism is freely offered when the host language is Prolog and allows to specify easily problems from the NP complexity class.

To model and solve UTP instances with the CHR language, we use the CHR++ solver [7] (for Constraint Handling Rules in C++), which is an efficient integration of CHR in the programming language C++.

The full model for CHR++ is too long to be detailed here[4]. We give in Table 7 the list of constraints taken into account by the solver. The decision variables to be instantiated are given in Table 6. They are similar to those of the MiniZinc model, only the end-of-session variables are added.

$\forall s \in S$: x_rooms[s] $\subseteq R$	set of rooms allocated to a session
$\forall s \in S$: x_lecturers[s] $\subseteq L$	set of lecturers allocated to a session
$\forall s \in S$: x_slot_start[s] $\in H$	starting slot allocated to a session
$\forall s \in S$: x_slot_end[s] $\in H$	ending slot allocated to a session

Table 6 Decision variables (CHR).

To simplify its implementation, the model is partly non-cumulative and some resources such as lecturers cannot be shared. It also considers that the sectioning and allocation of students to groups is done beforehand. Thus, computing a solution amounts to finding a consistent resource allocation while placing the schedules for all sessions.

Several constraints can be set at the instance analysis stage. This is the case for constraints (1) to (9) of Table 7. Constraints (2), (3) and (4) filter the domains by removing the rooms, lecturers or time slots which are impossible by construction of the instance. Constraint (5) ensures that a session starts and ends on the same day by removing from the domain values that contradict it.

Other constraints are set and managed by rules which monitor modifications to the domains of variables. This is the case for Constraint (1) which ensures the integrity of the start and end of session variables. The same is true for (6) which ensures that the number of lecturers teaching a session is

[4] The interested reader can download the sources of the model [1].

Integrity constraint :	
$\forall s \in S$: x_slot_end$[s]$ = x_slot_start$[s]$ + length(s)	(1)
Static constraints (instance input filtering)) :	
$\forall s \in S$: x_rooms$[s] \subseteq$ part_rooms[session_part(s)]	(2)
$\forall s \in S$: x_lecturers$[s] \subseteq$ part_lecturers[session_part(s)]	(3)
$\forall p \in P, \forall s \in$ part_sessions(p) : $\big($week(x_slot_start$[s]$) $\in$ weeks$[p]\big)$ $\wedge\big($weekday(x_slot_start$[s]$) $\in$ days$[p]\big)$ $\wedge\big($dailyslot(x_slot_start$[s]$) $\in$ dailyslots$[p]\big)$	(4)
$\forall s \in S$: x_slot_start$[s]/\mathit{nr_slots_per_day}$ = x_slot_end$[s]/\mathit{nr_slots_per_day}$	(5)
$\forall s \in S$: card(x_lecturers$[s]$) = team$[\mathit{session_part}[s]]$	(6)
$\forall k \in K, \forall s \in$ class_sessions$[k]$: If $\big($part_room_use[class_part(k)] = none$\big)$ then card(x_rooms$[s]$) = 0; If $\big($part_room_use[class_part(k)] = single$\big)$ then card(x_rooms$[s]$) = 1; If $\big($part_room_use[class_part(k)] = multiple$\big)$ then card(x_rooms$[s]$) ≥ 1	(7)
$\forall k \in K, \forall s, s' \in$ class_sessions$[k]$, $s.t.$ rank$(s) <$ rank(s') : $\texttt{before}(s, s')$	(8)
$\forall k_1, k_2 \in K, s.t.\ \exists g_1 \in$ class_groups$[k_1]$, $\exists g_2 \in$ class_groups$[k_2]$, avec $g_1 = g_2$: $\forall s_1 \in$ class_sessions(k_1), $s_2 \in$ class_sessions(k_2) : $\texttt{disjunct}(s_1, s_2)$	(9)
Static predicates :	
$\mathit{forbidden_period}((e, S'), h, h') = \forall i \in S'$: (x_slot_start$[i]$ + length$(i) \leq h) \vee$ (x_slot_start$[i] > h'$)	(10)
$\mathit{sequenced}((e_1, S_1), (e_2, S_2)) = \forall i_1 \in S_1, \forall i_2 \in S_2$: $\texttt{before}(i_1, i_2)$	(11)
$\mathit{same_rooms}((e, S')) = \forall s_1, s_2 \in S', s.t.\ s_1 < s_2$: x_rooms$[s_1] \sim$ x_rooms$[s_2]$	(12)
Dynamic constraints :	
$\forall p \in P, \forall l \in$ part_lecturers$[p]$: $\big\|\{x \mid x \in$ part_sessions$(p), l \in$ x_lecturers$[x]\}\big\|$ = service$[l, p]$	(13)
$\forall s \in S, \forall r \in$ session_rooms(s) : $\sum\{$group_students$[g] \mid g \in$ session_room_group$(s, r), r \in$ x_rooms$[s]\} \leq$ capacity$[r]$	(14)
$\forall s \in S, s.t.\ \mathit{has_mandatory_room}(s)$: session_mandatory$[s] \subseteq$ x_rooms$[s]$	(15)
Dynamic predicate :	
$\mathit{same_weekday}((e, S'))$ = $\forall s_1, s_2 \in S', s.t.\ s_1 < s_2$: x_slot_start$[s_1]/\mathit{nr_weekly_slots}$ = x_slot_start$[s_2]/\mathit{nr_weekly_slots}$	(16)
Introspective constraints :	
$\forall k_1, k_2 \in K, \forall s_1 \in$ class_sessions$[k_1]$, $\forall s_2 \in$ class_sessions$[k_2]$, s.t. $s_1 \neq s_2$: x_lecturers$[s_1] \cap$ x_lecturers$[s_2] \neq \emptyset \Rightarrow \texttt{disjunct}(s_1, s_2)$	(17)
$\forall k_1, k_2 \in K, \forall s_1 \in$ class_sessions$[k_1]$, $\forall s_2 \in$ class_sessions$[k_2]$ s.t. $s_1 \neq s_2$: x_rooms$[s_1] \cap$ x_rooms$[s_2] \neq \emptyset \Rightarrow \texttt{disjunct}(s_1, s_2)$	(18)

Table 7 Constraints and predicates of the `CHR` model.

valid and (7) which checks that the number of rooms allocated to a session corresponds to what is required in the instance.

We give as an example the `CHR++` rule which checks the integrity of the variables of beginning and end of session. The rule uses a `plus` propagator to ensure consistency of the constraint. This is triggered as soon as a domain of a variable is updated:

```
session_slot(_, S_Start, S_End, S_Length)
  =>> CP::Int::plus(S_Start, (*S_Length)-1, S_End);;
```

We use `CHR++` which allows us to manipulate values associated with logical variables and to wake up the corresponding rules as soon as a modification of the value occurs. This mechanism combined with the forward chaining of

CHR allows us to implement an efficient rule wake-up and domain propagation mechanism in the manner of a CSP solver.

Constraints (8) and (9) add new CHR constraints to the model. Indeed, constraints `before` and `disjunct` are constraints ensuring the precedence and non-overlapping of two sessions. They are accompanied by rules verifying the coherence of the disjunctive graph created implicitly by the addition of all these constraints. The static predicates correspond to those read from the instance. They are processed and some new constraints (filtering constraints, CHR constraints or unification of variables) are added.

Dynamic constraints ranging from (13) to (18) are only triggered under certain conditions. CHR guarded rules are used for this purpose. (13) checks that a lecturer teaches the expected number of sessions in each course part. (14) ensures that the capacity of the rooms is respected and (15) verifies that the rooms marked as mandatory are indeed found in the solution. Predicate (16) ensures that sessions subject to the same constraint *same_weekday* are set on the same day of the week.

Constraints (17) and (18) add constraints when certain conditions are verified. Thus, (17) adds a `disjunct` between two sessions when the same lecturer participates. (18) adds a constraint between two sessions if they take place in the same room. These constraints enrich the disjunctive graph representing the sequencing of all the sessions.

It should be noted that the CHR model performs domain filtering but also analyses the disjunctive graph in order to eliminate non-solutions. The edges of the disjunctive graph are oriented as the resolution progresses and the decision variables are instantiated.

4.3 Experimentations

We carried out preliminary experiments on a real-life instance modeling the second semester of the last year of Bachelor in Computer Sciences at Université d'Angers (available at [1]). The main objective was to validate the solvers and assess their ability to generate solutions in a reasonable time.

The instance contains 5 mandatory courses and 2 courses to choose among 4 additional courses. The instance thus consists of 9 courses decomposed into 24 parts, 45 classes and 241 sessions. Courses are taught during 12 weeks, 5 days a week (Monday to Friday), where each day is divided into 1440 slots. At the Faculty of Sciences of Université d'Angers, course sessions last 1h and 20 minutes or 2 hours and start at regular intervals every 90 minutes starting at 8h00 and finishing at 19h50. The 90 minutes interval includes a 10 minutes break allowing students and lecturers to change rooms.

The instance contains 8 rooms, 12 lecturers and 67 students. In our case, student sectioning was performed in advance and prepartitioned the students into 4 groups. Lecturers are either course owners involved in all the parts of a course (lecture, tutorial and lab) or tutors that are involved in labs of different courses. There are 47 rules defined in the instance: 13 `weekly`, 17 `sequenced`,

2 `same_slot`, 5 `same_week`, 5 `same_rooms` and 5 `same_lecturers`. The 47 rules were flattened into 216 constraints and the order of 1000 decision variables.

The `MiniZinc` and `CHR` solvers presented in Section 3 were used to solve the instance with an Intel Core i7-10875H 2.30GHz. Both solvers generate a valid solution in less than 5 seconds. The solutions are different due to the two resolution strategies but compliant with each solver which shows the convergence of both models and solvers.

5 Conclusion and Perspectives

We introduced in this paper a domain-specific language for university course timetabling. The language allows to model a wide variety of course timetabling problems such as those encountered in French universities. It provides support for typical timetabling entities (students, sessions, lecturers, rooms, groups) and features (student sectioning, resource distribution, session scheduling, resource allocation) and includes a rules language to easily express constraints (sequencing, periodicity, etc.). Rules allow to target any subset of domain entities and sessions and enforce timetabling-specific predicates.

We used the language to encode a real instance (Bachelor courses of a French university) and implemented a tool chain to convert the `XML` instance files into solver-compatible formats. In order to validate our approach, we implemented a `CSP` model in `MiniZinc` and `CHR` and produced solutions for the considered instance.

We are currently working on different extensions of the language and the back-end solvers. First, we intend to represent preferences and priorities in order to support timetable optimization and repair tasks. Second, the current `CP` models may be improved using dedicated scheduling constraints, search strategies and heuristics and take advantage of model simplication and reformulation techniques. Another objective is to improve scalability by testing our solvers on large-scale instances aggregating different curriculae or converted from academic benchmarks. Lastly, we intend to investigate the revision of timetables to manage unexpected events (e.g. unavailability of a lecturer, late registration of students) or to support incremental solution construction.

References

1. University Service Planning. URL https://ua-usp.github.io/timetabling/
2. Generic Constraint Development Environment (2022). URL https://www.gecode.org/
3. Minizinc (2022). URL https://www.minizinc.org/
4. Specification of Flatzinc. Version 1.6 (2022). URL https://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf
5. Abdennadher, S., Schütz, H.: CHR: A Flexible Query Language. In: Proceedings of the 3rd International Conference on Flexible Query Answering Systems, pp. 1–14 (1998)
6. Aziz, N.L.A., Aizam, N.A.H.: University course timetabling and the requirements: Survey in several universities in the east-coast of Malaysia. p. 040013. Kuala Terengganu, Malaysia (2017). DOI 10.1063/1.4995845. URL http://aip.scitation.org/doi/abs/10.1063/1.4995845

7. Barichard, V., Stéphan, I.: Quantified constraint handling rules. In: ICLP 2019, vol. 306, pp. 210–223. Las Cruces (2019). DOI 10.4204/EPTCS.306.25. URL http://okina.univ-angers.fr/publications/ua20272
8. Battistutta, M., Ceschia, S., De Cesco, F., Di Gaspero, L., Schaerf, A., Topan, E.: Local search and constraint programming for a real-world examination timetabling problem. In: E. Hebrard, N. Musliu (eds.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 69–81. Springer International Publishing, Cham (2020)
9. Beldiceanu, N., Carlsson, M.: A new multi-resource cumulatives constraint with negative heights. In: CP 2002, pp. 63–79 (2002)
10. Bettinelli, A., Cacchiani, V., Roberti, R., toth, P.: An overview of curriculum-based course timetabling. TOP **23**, 313–349 (2015). DOI https://doi.org/10.1007/s11750-015-0366-z
11. Betz, H., Frühwirth, T.: Linear-logic based analysis of constraint handling rules with disjunction. ACM Transactions on Computational Logic **14**(1) (2013)
12. Cambazard, H., Hebrard, E., O'Sullivan, B., Papadopoulos, A.: Local search and constraint programming for the post enrolment-based course timetabling problem. Ann. Oper. Res. **194**(1), 111–135 (2012). DOI 10.1007/s10479-010-0737-7
13. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. The Journal of the Operational Research Society **47**(3), 373–383 (1996). URL http://www.jstor.org/stable/3010580
14. Caselli, G., Delorme, M., Iori, M.: Integer linear programming for the tutor allocation problem: A practical case in a british university. Expert Systems with Applications **187**, 115967 (2022). DOI https://doi.org/10.1016/j.eswa.2021.115967. URL https://www.sciencedirect.com/science/article/pii/S095741742101318X
15. Castro, C., Manzano, S.: Variable and Value Ordering When Solving Balanced Academic Curriculum Problems. ARXIV (2001)
16. Chen, M., Sze, S., Goh, S.L., Sabar, N., Kendall, G.: A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities. IEEE Access **PP**, 1–1 (2021). DOI 10.1109/ACCESS.2021.3100613
17. Chiarandini, M., Di Gaspero, L., Gualandi, S., Schaerf, A.: The balanced academic curriculum problem revisited. Journal of Heuristics **18**(1), 119–148 (2012). DOI 10.1007/s10732-011-9158-2. URL https://doi.org/10.1007/s10732-011-9158-2
18. Frühwirth, T.: Constraint Handling Rules. In: Constraint Programming: Basics and Trends, pp. 90–107 (1994)
19. Frühwirth, T.: Constraint Handling Rules. Cambridge University Press (2009)
20. Frühwirth, T., Raiser, F. (eds.): Constraint Handling Rules: Compilation, Execution, and Analysis. Cambridge University Press (2011)
21. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming **37**(1-3), 95–138 (1998)
22. Goh, S.L., Kendall, G., Sabar, N.R.: Improved local search approaches to solve the post enrolment course timetabling problem. European Journal of Operational Research **261**(1), 17–29 (2017). DOI https://doi.org/10.1016/j.ejor.2017.01.040. URL https://www.sciencedirect.com/science/article/pii/S0377221717300759
23. Hentenryck, P.V.: Constraint logic programming. Knowledge Engineering Review **6**(3), 151–194 (1991)
24. ITC19: International Timetabling Competition (2019). URL https://www.itc2019.org/
25. Jaffar, J., Maher, M.: Constraint logic programming: A survey. Journal of Logic Programming **19/20**, 503–581 (1994)
26. Lemos, A., Monteiro, P., Lynce, I.: Disruptions in timetables: A case study at universidade de lisboa. Journal of Scheduling (2021). DOI 10.1007/s10951-020-00666-3
27. Lewis, R., Paechter, B., Mccollum, B.: Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff University, Cardiff Business School, Accounting and Finance Section, Cardiff Accounting and Finance Working Papers (2007)
28. Lindahl, M., Stidsen, T., Sørensen, M.: Quality recovering of university timetables. European Journal of Operational Research **276**(2), 422 – 435 (2019). DOI https://doi.org/10.1016/j.ejor.2019.01.026. URL http://www.sciencedirect.com/science/article/pii/S0377221719300451

29. Mccollum, B., McMullan, P., Paechter, B., Lewis, R., Schaerf, A., Di Gaspero, L., Parkes, A., Qu, R., Burke, E.: Setting the research agenda in automated timetabling: The second international timetabling competition. INFORMS Journal on Computing **22**, 120–130 (2010). DOI 10.1287/ijoc.1090.0320
30. Müller, T., Rudová, H., Müllerová, Z.: University course timetabling and International Timetabling Competition 2019. In: E.K. Burke, L. Di Gaspero, B. McCollum, N. Musliu, E. Özcan (eds.) Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018), pp. 5–31 (2018)
31. Müller, T., Murray, K.: Comprehensive approach to student sectioning. Annals of Operations Research **181**, 249–269 (2010). DOI 10.1007/s10479-010-0735-9
32. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: C. Bessière (ed.) Principles and Practice of Constraint Programming – CP 2007, pp. 529–543. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
33. Rubio, J.M., Palma, W., Rodriguez, N., Soto, R., Crawford, B., Paredes, F., Cabrera, G.: Solving the Balanced Academic Curriculum Problem Using the ACO Metaheuristic. Mathematical Problems in Engineering **2013**, e793671 (2013). DOI 10.1155/2013/793671. URL https://www.hindawi.com/journals/mpe/2013/793671/. Publisher: Hindawi
34. Schindl, D.: Optimal student sectioning on mandatory courses with various sections numbers. Annals of Operations Research **275** (2019). DOI 10.1007/s10479-017-2621-1
35. Vrielink, R.A.O., Jansen, E.A., Hans, E.W., van Hillegersberg, J.: Practices in timetabling in higher education institutions: a systematic review. Ann. Oper. Res. **275**(1), 145–160 (2019). DOI 10.1007/s10479-017-2688-8