Computing and Informatics, Vol. 34, 2015, 277–304

METRO-NG: COMPUTER-AIDED SCHEDULING AND COLLISION DETECTION

David Bednárek, Jakub Yaghob, Filip Zavoral

Charles University in Prague
Faculty of Mathematics and Physics
Malostranské nám. 25
118 00 Prague, Czech Republic
e-mail: {bednarek, yaghob, zavoral}@ksi.mff.cuni.cz

Abstract. In this paper, we propose a formal model of the objects involved in a class of scheduling problems, namely in the classroom scheduling in universities which allow a certain degree of liberty in their curricula. Using the formal model, we present efficient algorithms for the detection of collisions of the involved objects and for the inference of a tree-like navigational structure in an interactive scheduling software allowing a selection of the most descriptive view of the scheduling objects. These algorithms were used in a real-world application called MetroNG; a visual interactive tool that is based on more than 10 years of experience we have in the field. It is currently used by the largest universities and colleges in the Czech Republic. The efficiency and usability of MetroNG suggests that our approach may be applied in many areas where multi-dimensionally structured data are presented in an interactive application.

Keywords: Scheduling tools, collision detection, visualization

1 INTRODUCTION

Every university is faced with the problem of creating an acceptable schedule of their educational events within available resources of time, space, and personnel.

The main problem when creating a schedule of university lectures and seminars lies in checking and preventing collisions – one teacher should not teach two lectures at the same time, students should not visit more than one lesson at the same time, etc. Moreover, there is a great number of soft or hard constraints, e.g. a teacher cannot teach in a particular time period or students are not happy to have lessons continually 12 hours in one day. Therefore, creating ‘good’ schedule is a difficult task which requires some IT support.

The problem could be described as a constraint satisfaction problem (CSP) [14]. The formal constraints and the quality criteria or the objective functions should be specified in order to use an appropriate solver [10].

Each CSP solver requires the exact formal specification of all constraints. Besides individual constraints, the members of academia have their individual ideas of ‘good’ schedules; therefore, the cost or objective function shall be adjusted individually per each scheduling object. These constraints and objective functions are fuzzy and unclear by nature and ordinary users of the system are not able to specify them in the necessary formal manner. Thus, the use of the CSP solver requires skilled personnel to formalize all requirements of the users into constraints and objective functions.

An alternative approach to a complete formalization of the problem for a scheduling algorithm is creating the schedule by humans, using a software application to display and efficiently customize the (partial) schedule.

Although the manual schedule creation itself requires more human effort than an algorithmic solution, the difference may be balanced or even outweighed by the cost of formalization of the constraints. In addition, human-based scheduling offers the following advantages:

- humans can effectively work with incompletely defined constraints
- humans can intuitively recognize erroneously entered constraints
- humans can communicate individually to the subjects in case of ambiguity or to negotiate a relaxation of constraints
- in case of inevitably conflicted schedule, humans can find the ‘least bad’ solution instead of giving-up completely
- in case of complaints, humans can explain the rationale behind the schedule.

The importance of individual advantages and disadvantages of automated and manual approaches depends on many factors, including the organization structure of the university, the work flow of the scheduling, the level of the academic liberty, etc. According to our experience, the manual approach is still viable and favorable in some universities, despite the recent progress in the CSP and other algorithmic scheduling techniques.

A mixed approach is also possible: using a CSP solver with a weaker and smaller set of constraints as the first step and then the manual adjustment for the final tuning of the schedule.

Human-based scheduling requires strong software support capable at least of these tasks:

- display several views of the schedule (e.g., a schedule of the lecture rooms, the schedule of particular student groups, etc.)

- automatically or manually choose an appropriate level of detail of particular views; in other words, the application must present some *navigation structures* to allow efficient exploration of the scheduling space
- display all possible collisions (e.g., more lectures intended for the same student group in overlaping time slots, a lecture of a particular teacher conflicting with his constraints, etc.).

The article presents a formal model for the complex collision detection system which is used to identify situations where a group of students should attend two classes scheduled for the same time. Next, we present an incremental algorithm used to extract the navigation structures from the formal model. Using this model, we developed an application called MetroNG[1] for supporting the whole process of creating a complex university and curricula schedule. Besides the sophisticated student collision detection, the application has other unique features such as the combination of several views, handling of transfer collisions and various types of scheduled events besides regular 'once-a-week' classes.

The rest of the article is organized as follows: Next section contains related work, Section 3 describes the issues addressed in the article. The most crucial part of the article is contained in Sections 4 and 5 that describe a theoretical model of the entities and their collisions and the respective algorithms. Section 6 describes the MetroNG application that is based on the previously proposed formal methods. The final section summarizes and concludes the article.

2 RELATED WORK

There is a lot of research on classroom or curriculum scheduling (also called timetabling in some sources). The proposed solutions range from simple applications to large automatic or semi-automatic CSP solvers.

The scheduling problems are NP-complete in general [42], as far as their computational complexity is concerned. Therefore, various optimization methods have been proposed for solving the scheduling problem [17, 18, 19, 20]. Other methods are heuristic orderings [22], case-based reasoning [21], genetic/evolutionary algorithms [25, 26, 27], ant systems [30], local search techniques [23, 24], particle swarm optimization [28, 29], tabu search [31, 32], metaheuristics [33] and hyperheuristics [34, 35].

Recent definitions of the course scheduling problem can be found in [33, 36]. Universities have increasingly relied upon the automation of this task to produce efficient timetables that satisfy these constraints [36]. Many recent papers have been published on specific techniques [37, 38, 39] dealing with the university course scheduling problem.

[1] `https://www.erudio.cz/?stranka=sw.metrong`

High school timetabling involves weekly scheduling for all lecturers of the high school, where the schedule is regular, the number of events is quite small, there are no collisions, the constraints are simple, etc. The problem consists in assigning lectures to timeslots while other constraints of several different kinds are satisfied. These constraints may include both hard constraints that must be respected and soft constraints used to evaluate the solution's quality [40].

Some recent papers have been published on these specific techniques [41, 42, 43, 11, 12, 13] and corresponding software solutions [6, 7, 8] oriented to high-school timetabling were described. These tools are not sufficient for most university environments, mostly because they are not capable to capture the more complex structure of the groups of students.

The system [9] uses individual students as the subjects of scheduling. Of course, this approach is viable only in environments where the set of individual students for each lecture is known prior to the scheduling phase. In addition, this fine-grained approach is extremely demanding in terms of computing resources.

Systems combining automatic scheduling with human interaction described as interactive or semiautomatic scheduling were described in [15, 16].

The solution proposed in this paper is based on a strong theoretical background defined in Section 4. The problem of modeling events and their visitors is similar to some problems in the area of *concept analysis* – for instance, the role of test context in the Contextual Attribute Logic [3, 4] is similar to our group base (see Section 4.2). Nevertheless, our intent is visualization and not reasoning; therefore, our formal model is different.

3 PROBLEM DESCRIPTION

The scheduling problem essentially consists of a set of *events.* Each event is associated to a set of participants – teachers and students. While the teachers are easily identified by their enumeration, the description of the associated set of students is difficult: The enumeration of individual students is usually impossible because they are not known at the moment of scheduling. Instead of the enumeration, the student participants of an event are described in terms of student *groups.*

The schedule consists of the mapping of events to space and time, i.e. assigning a room and time slot to each event. The time slots are usually but not necessarily recurrent. The schedule shall satisfy a number of hard and soft constraints; most importantly it shall avoid *collisions.* However, since finding a collision-free schedule is not necessarily feasible, the system must be able to manipulate schedules containing collisions.

The main purpose of the scheduling application is to display the schedule and allow the users to modify it. The visualization part must cope with the huge amount of information present in the schedule. An average schedule of a college or university contains hundreds or thousands of events, spans several buildings with tens or hundreds of lecture rooms, and handles hundreds of teachers and many thousands of students. Since it is too much information to fit in one view, the visualization must be highly interactive and provide the user with only that part of information that he or she is interested in at the moment.

Although the modification of the schedule is basically a simple drag-and-drop action, it is a part of a larger, very complex problem. While the whole decision making is left to the user (the user picks the time and room for each event), it is not sufficient to give the users the ability to schedule the events. To do the task efficiently, they need a lot of information beyond "what, when and where". The most important piece of information is whether they are creating a collision-free schedule.

3.1 Student Groups

In the physical world, student groups consist of individual students who, for whatever reason, visit a particular event. In our setting, we do not know the individual students in the moment of scheduling; instead, the group must be described by using the properties of the expected participants.

The set of students visiting a particular lecture may be characterized using *attributes* like study program, specialization, year, etc. Some of them may be assigned to the students by an authority, others (like proficiency level) may represent a fiction created by the scheduling personnel.

For the collision checking, the system must be able to determine whether two student groups intersect. This test conceptually corresponds to testing intersection of two sets; however, these sets are not described by enumerating their contents. Instead, the intersection test shall be based on the attributes of the student groups.

Moreover, the set of real students does not completely fill the space of all attribute combinations. For instance, the group of 'second-year' students may or may not intersect with the group of 'beginners', depending on the current (or estimated) state of the student population. In our system, the student population is modeled by a *group base* – a set of student *prototypes.*

3.2 Visualization of Groups

Student attributes form a multi-dimensional space which cannot be visualized directly. Some dimensions may be mutually dependent (e.g., study program and specialization), other pairs are considered independent (e.g., language skills vs. specialization).

When creating a schedule, a portion of this multi-dimensional space must be accessible in an appropriate level of detail. A straightforward approach to selecting a portion of the space would be using a form to enter the filtering rules for each attribute. However, to control such a form is tedious; moreover, the user is often unable to predict whether a particular filter leads to a human-readable presentation.

To allow exploration of the attribute space, we propose using a *navigational tree* to perform the navigation and level-of-detail setting by expanding and collapsing a virtual tree. Besides avoiding annoying forms, this approach also allows to show several portions of the space at once, using different levels of detail in different parts of the space. In this sense, the tree replaces a filter in the form of disjunction of conjunctive clauses.

While this idea is simple, creating the tree is not easy. First, the degree of individual nodes of the tree shall be kept as low as possible in order to make the navigation easy for the user. In other words, while the user must be able to reach any desired view in the tree, the number of paths to the view shall be minimized. Second, the size of the completely expanded tree is exponential with respect to the size of group base; therefore, the tree shall be constructed incrementally.

3.3 Collisions

A collision is a state when two events are scheduled in such a way, that someone who is supposed to attend both events would be unable to do so. The two classes of participants – teachers and students – form two classes of collisions.

The collision may occur either directly, when two events intersect in time, or by transfer, when the time distance between the two events is too short to allow transfer between the two locations. In addition, two events may collide if scheduled to the same room at the same time.

While room and teacher-related collisions are relatively easy to detect, the detection of student-related collision involves a complex determination of the intersection of the groups, as described in Subsection 3.1. Also the visualization of the space of students must be arranged so that the user can easily determine the danger of collisions.

4 FORMAL MODEL

Instead of modeling individual students, the system works with *prototypes*; each prototype represents a group of students assumed to have the same preferences and/or prescriptions with respect to the lectures/events visited. The set of prototypes, called *group base*, is defined by human operators of the scheduling system, as accurate as possible and/or necessary.

The group base concept is similar to the test context in the Contextual Attribute Logic [3, 4] since the problem of modeling events and their visitors is similar to some problems in the area of *concept analysis* [2].

Given a group base, the visitors of an event could be defined by using a set of prototypes. However, it would be impractical for a human operator because such sets may be quite large. Instead of enumerating prototypes, student groups may be defined by using tuples of attributes as described in the following paragraphs.

Prototypes are distinguished by *attributes* carrying values (*tags*) from finite *domains*. Some attributes correspond to properties defined in the real world and assigned to real students, like the field of study, the year, or an administratively assigned group number. Others are defined solely for the purpose of scheduling, representing for instance assumed future specialization.

4.1 Attributes and Tuples

Each instance of the scheduling system defines a finite set $\mathcal{A}$ of attributes, divided into single-valued $\mathcal{A}_{\mathsf{s}}$ and multi-valued $\mathcal{A}_{\mathsf{m}}$ attributes. Each attribute $a \in \mathcal{A}$ is associated to a finite domain $\mathcal{D}_a$; members of these domains are called *tags*. These domains are specific for a particular school and they develop slowly over time to reflect the changes in the curricula. For the simplicity of notation, we assume that the attribute domains are pairwise disjoint; the union of domains is marked $\mathcal{D} = \bigcup\{\mathcal{D}_a \mid a \in \mathcal{A}\}$.

Example 1. Throughout this paper, we will show several examples based on the following set of attributes:

$$\begin{aligned} \mathcal{A}_{\mathsf{s}} &= \{D, P, R\}, \\ \mathcal{A}_{\mathsf{m}} &= \{S\}. \end{aligned}$$

The corresponding domains are:

$$\begin{aligned} \mathcal{D}_D &= \{B, N\}, \\ \mathcal{D}_P &= \{I, M\}, \\ \mathcal{D}_R &= \{1, 2, 4, 5\}, \\ \mathcal{D}_S &= \{IPR, IOI, ISS\}. \end{aligned}$$

Although our formalism distinguishes single-valued and multi-valued attributes, many definitions become simplified when the values of both kinds of attributes are viewed as sets. In this approach, the value of an attribute is a set of tags, i.e., a subset of the associated attribute domain. For a single-valued attribute, the cardinality of the attribute value is restricted to at most 1. The ability to assign the empty-set value to an attribute loosely corresponds to the concept of null value in relational databases; however, the treatment of empty values in predicates is different from the treatment of null values in databases.

All tuples in our system have the same schema, i.e., all tuples contain all attributes defined in the system (nevertheless, the value of an attribute may be empty in a tuple).

Definition 1 (Tuple)**.** A *tuple* is a mapping $t : \mathcal{A} \to \mathbf{P}(\mathcal{D})$ such that $t(a) \subseteq \mathcal{D}_a$ and $a \in \mathcal{A}_{\mathsf{s}} \Rightarrow \mathbf{card}(t(a)) \leq 1$. The universe of all such tuples is denoted $\mathcal{T}$.

Each tuple associates a value (a set of tags) to each attribute. In most cases, a tuple serves as a means to define a set of certain individuals by declaring which tags shall be present in these individuals. The more tags a tuple contains, the less individuals fit to the tuple; if an attribute of a tuple has the empty value, it does not restrict the set of individuals anyhow. This approach, reflected in the definitions below (and different from traditional null-value handling in relational systems), is motivated by the required conservative-approximation rule: “Possible collision is a collision.”

Definition 2 (Tuple meet)**.** The *meet* (denoted $t_1 \sqcap t_2$) of a pair of tuples $t_1, t_2 \in \mathcal{T}$ is the mapping t_3 such that $t_3(a) = t_1(a) \cup t_2(a)$ for each $a \in \mathcal{A}$, provided t_3 is a correct tuple, i.e. $t_3 \in \mathcal{T}$.

The name “meet” was borrowed from the lattice theory; however, it should be stressed that the tuple meet is not a semilattice because the meet may not exist. Also note that the sense of the operation is reverted with respect to the lattice of tag sets; i.e. tuple meet corresponds to union of tag sets. This reversion corresponds to the fact that adding attributes diminishes the set of individuals conforming to these attributes. Similarly to lattice theory, the tuple meet operation is associated with a partial ordering, named *tuple inclusion.*

Definition 3 (Tuple inclusion)**.** A tuple $t_1 \in \mathcal{T}$ is considered *included* in a tuple $t_2 \in \mathcal{T}$ (denoted $t_1 \sqsubseteq t_2$), if for each attribute $a \in \mathcal{A}$, $t_2(a) \subseteq t_1(a)$.

The rationale behind this definition is that a tuple defines a set of individuals by defining a set of required tags of these individuals. The tuple inclusion relation corresponds to the set inclusion relation on the associated sets of individuals. If t_1 requires more tags than t_2, then any individual satisfying t_1 also satisfies t_2. Note that tuple inclusion is a partial order on $\mathcal{T}$.

Example 2. The following mapping

$$t_1 = \{D \mapsto \{B\}, P \mapsto \emptyset, R \mapsto \{1\}, S \mapsto \{IPR, IOI\}\}$$

is a correct tuple with respect to the attribute set defined in the previous example. In the following examples, we will use the traditional positional notation; in addition, we will omit braces containing a single tag. Thus, t_1 will be written as $(B, \emptyset, 1, \{IPR, IOI\})$. For example,

$$t_1 \sqcap (B, I, \emptyset, ISS) = (B, I, 1, \{IPR, IOI, ISS\})$$

while $t_1 \sqcap (B, \emptyset, 2, ISS)$ is not defined because the attribute R is single-valued. Furthermore,

$$(B, I, 1, \{IPR, IOI, ISS\}) \sqsubseteq t_1$$

while $(B, I, \emptyset, ISS)$ and t_1 are incomparable.

Definition 4 (Tuple collision)**.** A pair of tuples $t_1, t_2 \in \mathcal{T}$ are said to *collide* (denoted $t_1 \triangle t_2$), if there exists a tuple $t_3 \in \mathcal{T}$ such that $t_3 \sqsubseteq t_1$ and $t_3 \sqsubseteq t_2$.

Such a tuple t_3 must satisfy the condition $t_1(a) \cup t_2(a) \subseteq t_3(a)$ for each $a \in \mathcal{A}$. The existence of such a tuple may be prevented by the condition $a \in \mathcal{A}_{\mathsf{s}} \Rightarrow \mathbf{card}(t(a)) \leq 1$ from the definition of tuple. If such tuples exists, the tuple meet $t_1 \sqcap t_2$ is one of them. Therefore, the following lemma holds:

Lemma 1 (Tuple collision)**.** A pair of tuples $t_1, t_2 \in \mathcal{T}$ *collides* ($t_1 \triangle t_2$) if and only if

$$t_1(a) \neq \emptyset \wedge t_2(a) \neq \emptyset \Rightarrow t_1(a) = t_2(a)$$

for each single-valued attribute $a \in \mathcal{A}_{\mathsf{s}}$.

Example 3. The tuples $(B, I, \emptyset, \emptyset)$ and $(\emptyset, I, 4, \emptyset)$ collide while $(B, M, \emptyset, \emptyset)$ and $(\emptyset, I, 4, \emptyset)$ do not.

4.2 Group Base

Group base is the device used to specify which individuals exist by the means of prototypes. Each prototype is modeled by a tuple; a *group base* is simply a set of tuples, as shown in the following definition.

Definition 5 (Group base)**.** A *group base* is a set $\mathcal{G} \subseteq \mathcal{T}$. A group base $\mathcal{G}$ is *well-formed* if $t_1 \triangle t_2 \Rightarrow t_1 = t_2$ for each $t_1, t_2 \in \mathcal{G}$.

Each scheduling problem defines a group base and all (student-based) scheduling constraints are related to this group base. We assume that there are no other individuals than those corresponding to the prototypes enumerated in the group base. Consequently, any group of individuals is completely described by a set of prototypes from the group base. Instead of enumerating these prototypes directly, a tuple inclusion offers the ability to define a set of prototypes using a single tuple, as shown in the following definition of *trace*. Of course, not every set of prototypes is a trace of a tuple; however, every set of prototypes may be defined as a union of traces, i.e., any group of individuals may be described using a set of tuples. In real-life cases, these tuples offer a significantly shorter description than the enumeration of prototypes.

Definition 6 (Trace)**.** The set

$$\mathsf{T}_{\mathcal{G}}(t) = \{g \in \mathcal{G} \mid g \sqsubseteq t\}$$

is called the *trace* of a tuple t with respect to a group base $\mathcal{G}$.

The following lemma shows how the tuple meet operation corresponds to operations on prototype sets.

Lemma 2 (Trace of tuple meet)**.** For each pair of tuples $t_1, t_2 \in \mathcal{T}$ such that $t_1 \sqcap t_2$ exists, $\mathsf{T}_{\mathcal{G}}(t_1 \sqcap t_2) = \mathsf{T}_{\mathcal{G}}(t_1) \cap \mathsf{T}_{\mathcal{G}}(t_2)$.

The following important notions are defined with respect to a given group base $\mathcal{G}$: $\mathcal{G}$-relative inclusion and collision. In these definitions, a group base functions as a restriction on the set of individuals existing in the system; thus, the inclusion relation is weakened and the collision relation strengthened by the $\mathcal{G}$-relativization, as shown in Lemma 3.

Definition 7 (Relative inclusion)**.** For a pair of tuples $t_1, t_2 \in \mathcal{T}$, t_1 is considered *$\mathcal{G}$-included in* t_2 with respect to a group base $\mathcal{G}$ (denoted $t_1 \sqsubseteq_{\mathcal{G}} t_2$), if $\mathsf{T}_{\mathcal{G}}(t_1) \subseteq \mathsf{T}_{\mathcal{G}}(t_2)$.

Definition 8 (Relative collision)**.** Tuples $t_1, t_2 \in \mathcal{T}$ are said to *$\mathcal{G}$-collide* with respect to a group base $\mathcal{G}$ (denoted $t_1 \triangle_{\mathcal{G}} t_2$), if $\mathsf{T}_{\mathcal{G}}(t_1) \cap \mathsf{T}_{\mathcal{G}}(t_2) \neq \emptyset$.

Lemma 3. For each pair of tuples $t_1, t_2 \in \mathcal{T}$

$$t_1 \sqsubseteq t_2 \Rightarrow t_1 \sqsubseteq_{\mathcal{G}} t_2, \qquad t_1 \triangle_{\mathcal{G}} t_2 \Rightarrow t_1 \triangle t_2.$$

D	P	R	S
B	I	1	IPR
B	I	1	IOI
B	I	1	ISS
B	I	2	IPR
B	I	2	IOI
B	M	1	
B	M	2	
N	I	4	
N	I	5	

Table 1. Example: A group base

Example 4. Let $\mathcal{G}$ be the group base shown in Table 1. Let $t_3 = (N, \emptyset, 4, \emptyset)$, $t_4 = (N, I, \emptyset, \emptyset)$. Their traces with respect to $\mathcal{G}$ are

$$\mathsf{T}_{\mathcal{G}}(t_3) = \{(N, I, 4, \emptyset)\}; \mathsf{T}_{\mathcal{G}}(t_4) = \{(N, I, 4, \emptyset), (N, I, 5, \emptyset)\}$$

and they are, by definition, ordered by $\mathcal{G}$-inclusion ($t_3 \sqsubseteq_{\mathcal{G}} t_4$) although they are incomparable by inclusion ($t_3 \not\sqsubseteq t_4$). Furthermore, the tuples $(B, I, \emptyset, \emptyset)$ and $(\emptyset, I, 4, \emptyset)$ do not $\mathcal{G}$-collide although they do collide (see Example 3).

4.3 Normalization

The relativization with respect to a group base may cause that two different tuples correspond to the same group of individuals, which means that the two tuples are equivalent with respect to the group base.

Definition 9 (Relative equivalence)**.** Tuples $t_1, t_2 \in \mathcal{T}$ are considered *$\mathcal{G}$-equivalent* (denoted $t_1 \approx_{\mathcal{G}} t_2$), if $t_1 \sqsubseteq_{\mathcal{G}} t_2 \wedge t_2 \sqsubseteq_{\mathcal{G}} t_1$.

The $\mathcal{G}$-inclusion is induced by set-inclusion on traces; therefore, it is a partial order on $\mathcal{T}$. Consequently, the $\mathcal{G}$-equivalence is an equivalence.

We will show that $\mathcal{G}$-equivalence classes on tuples can be uniquely represented by *normalized* tuples.

Definition 10 (Implied attribute)**.** An attribute $a \in \mathcal{A}$ is called *implied* with respect to a tuple $t \in \mathcal{T}$ and a group base $\mathcal{G}$ (denoted $t \vdash_{\mathcal{G}} a$), if there exists a tag $v \in \mathcal{D}_a$ such that $t \approx_{\mathcal{G}} (t \sqcap \{a \mapsto \{v\}\})$. The attribute is called *non-trivially implied* if $v \notin t(a)$.

Definition 11 (Normalized tuple)**.** A tuple $t \in \mathcal{T}$ is called *normalized* with respect to a group base $\mathcal{G}$, if $\mathsf{T}_{\mathcal{G}}(t) \neq \emptyset$ and there exists no attribute being non-trivially implied with respect to t.

Lemma 4 (Tuple equivalence)**.** If tuples $t_1, t_2 \in \mathcal{T}$ are normalized with respect to a group base $\mathcal{G}$ then

$$t_1 \approx_{\mathcal{G}} t_2 \Rightarrow t_1 = t_2.$$

Each tuple whose trace is non-empty can be normalized by adding implied attributes and their values until a normalized tuple is reached. In addition, the lemma essentially states that normalized tuples are unique representatives of equivalence classes induced by the $\mathcal{G}$-equivalence, with the exception of the class of tuples whose trace is empty.

Normalized tuples are also representatives of their traces. Not every subset of a group base $\mathcal{G}$ is a trace of a tuple; however, every subset of $\mathcal{G}$ can be represented by a union of traces of some set of tuples. Thus, sets of normalized tuples can represent subsets of $\mathcal{G}$.

4.4 Events and Visitors

In our scheduling problem, a subset of $\mathcal{G}$ represents the visitors of an event. However, this subset may be quite large for human perception; therefore, an alternative representation using normalized tuples is understood more easily because it can be usually significantly smaller. On the other hand, for processing in software, the traces can be stored in bitmaps and handled using Boolean operators.

In addition, representation by tuples usually works in better accordance with reality when the group base $\mathcal{G}$ is changed due to evolution of curricula. Therefore, tuples are more suitable for persistent representation than subsets of $\mathcal{G}$.

These observations led to the following design principles:

- When presented to users, representation using normalized tuples is always used.
- In the database and in SQL-based applications (web interface), normalized tuples are used.
- In C++ applications, tuples are converted to bitmaps representing traces and handled using vectorized bit operations.

Definition 12 (Events)**.** Let $\mathcal{G}$ be a group base; $\mathcal{E}$ be a set of events. A mapping $\beta : \mathcal{E} \to \mathbf{P}(\mathcal{T})$ is called *event binding.* An event binding is called *normalized* if, for each event $e \in \mathcal{E}$, each tuple in $\beta(e)$ is normalized with respect to $\mathcal{G}$ and $t_1 \sqsubseteq_{\mathcal{G}} t_2 \Rightarrow t_1 = t_2$ for each $t_1, t_2 \in \beta(e)$.

The visualization algorithm uses a precomputed relation $\mathsf{M}_{\mathcal{G}} \subseteq \mathcal{E} \times \mathcal{G}$ stored as a binary matrix according to the following definition.

Definition 13 (Concern matrix)**.**

$$\mathsf{M}_{\mathcal{G}} = \{\langle e, t\rangle \mid (\exists t_e \in \beta(e))\ t \in \mathsf{T}_{\mathcal{G}}(t_e)\}.$$

Definition 14 (Event collision)**.** Events $e_1, e_2 \in \mathcal{E}$ are said to *$\mathcal{G}$-collide* if there are tuples $t_1 \in \beta(e_1)$ and $t_2 \in \beta(e_2)$ such that $t_1 \triangle_{\mathcal{G}} t_2$.

4.5 Visualization of the Tuple Hierarchy

As mentioned in Section 3.1, the partially-ordered set of groups is displayed using a tree. In this section, we will define the tree more formally.

Definition 15 (Attribute tree)**.** *Attribute tree* is a labeled unranked rooted tree

$$\mathcal{U} = (V_{\mathsf{m}}, V_{\mathsf{p}}, r, \pi, L_{\mathsf{m}}, L_{\mathsf{p}}, \tau)$$

whose nodes are of three kinds – *meta*-nodes V_{m}, *plain* nodes V_{p} and the root node r. The mapping

$$\pi : (V_{\mathsf{m}} \to (V_{\mathsf{p}} \cup \{r\})) \cup (V_{\mathsf{p}} \to V_{\mathsf{m}})$$

defines the structure of the tree by defining the parent $\pi(n)$ of every node $n \neq r$. The mapping

$$L_{\mathsf{m}} : V_{\mathsf{m}} \to \mathcal{A}$$

assigns attributes to *meta*-nodes and the mapping

$$L_{\mathsf{p}} : V_{\mathsf{p}} \to \mathcal{D}$$

assigns tags to plain nodes. Each node n is also assigned a tuple $\tau(n)$ defined recursively as

$$\begin{aligned}
\tau(r) &= \emptyset,\\
(\forall n \in V_{\mathsf{m}})\ \tau(n) &= \tau(\pi(n)),\\
(\forall n \in V_{\mathsf{p}})\ \tau(n) &= \tau(\pi(n)) \sqcap \{L_{\mathsf{m}}(\pi(n)) \mapsto \{L_{\mathsf{p}}(n)\}\}.
\end{aligned}$$

Definition 16 (Distinctive attribute tree)**.** An attribute tree is called *distinctive* with respect to a group base $\mathcal{G}$ if

$$(\forall n \in V_{\mathsf{p}})\ \mathsf{T}_{\mathcal{G}}(\tau(n)) \neq \emptyset \wedge \tau(n) \not\approx_{\mathcal{G}} \tau(\pi(n)).$$

Definition 17 (Functional dependency)**.** An attribute $b \in \mathcal{A}$ is called *functionally dependent* on an attribute $a \in \mathcal{A}$ with respect to a tuple $t \in \mathcal{T}$ and a group base $\mathcal{G}$ (denoted $t \vdash_{\mathcal{G}} a \rightsquigarrow b$), if, for each tag $v \in \mathcal{D}_a$, the attribute b is implied with respect to the tuple $(t \sqcap \{a \mapsto \{v\}\})$ and the group base $\mathcal{G}$. The functional dependency is called *non-trivial* if $b \neq a$ and $t(b) = \emptyset$.

Definition 18 (Non-skipping attribute tree)**.** An attribute tree is called *non-skipping* with respect to a group base $\mathcal{G}$ if

$$(\forall n \in V_{\mathsf{m}})\ \tau(\pi(n)) \vdash_{\mathcal{G}} a \rightsquigarrow L_{\mathsf{m}}(n) \Rightarrow a = L_{\mathsf{m}}(n).$$

An attribute tree is used to display (a part of) the partial order defined by the $\mathcal{G}$-inclusion on the set of normalized tuples. In this sense, the role of the attribute tree is similar to Hasse diagrams [5]; however, the interactive environment allows to incrementally unroll the lattice DAG to a tree. Note that the completely unrolled tree may have an exponential number of nodes with respect to the original DAG; therefore, the interactive incremental approach is crucial.

Attribute trees are constructed incrementally from the group base $\mathcal{G}$, whenever the user expands a node n. The construction algorithm scans the trace $\mathsf{T}_{\mathcal{G}}(\tau(n))$ to determine which attributes are functionally dependent on $\tau(n)$. These attributes are excluded; in addition, the application may exclude some attributes based on site-specific configuration. The remaining attributes, if any, generate a meta-node children of n. When expanding a meta-node, all corresponding attribute values in $\mathsf{T}_{\mathcal{G}}(\tau(n))$ are used.

An attribute tree contains interleaved layers of meta-nodes and value nodes; each pair of layers corresponds to adding an attribute/value pair to the corresponding tuple. The two layers allow the user to select the required attribute first; then, values from the corresponding domain are selected. In a distinctive attribute tree, the children of a meta-node n_{m} correspond to a disjoint (in the sense of $\mathcal{G}$-collision) cover of the parent value node $\pi(n_{\mathsf{m}})$. Consequently, a sibling of n_{m} contains the same set of groups; arranged, however, in a different manner.

4.6 Visualization of Event Binding

When displaying the schedule relevant to a tuple t, *concern relations* are used to determine the associated set of events. *Covering concern* collects all events which are bound to all individuals from t (i.e. all basic tuples included in t) while *intersecting concern* contains events bound to some individuals from t. Naturally, the covering concern is a subset of the intersecting concern.

Definition 19 (Concern relations)**.** The mappings $\mathsf{cover}_{\mathcal{G}}, \mathsf{intersect}_{\mathcal{G}} : \mathcal{T} \to \mathbf{P}(\mathcal{E})$ are called *covering concern* and *intersecting concern* and defined as

$$\begin{aligned} \mathsf{cover}_{\mathcal{G}}(t) &= \{e \in \mathcal{E} \mid (\exists t_e \in \beta(e))\ t \sqsubseteq_{\mathcal{G}} t_e\}, \\ \mathsf{intersect}_{\mathcal{G}}(t) &= \{e \in \mathcal{E} \mid (\exists t_e \in \beta(e))\ t \triangle_{\mathcal{G}} t_e\}. \end{aligned}$$

For each node n of the attribute tree, the application displays all events from the covering concern which are not in the covering concern of the parent, i.e., the event set $\mathsf{cover}_{\mathcal{G}}(\tau(n)) \setminus \mathsf{cover}_{\mathcal{G}}(\tau(\pi(n)))$, horizontally positioned at their scheduled time. Note that for an internal node a covering event is displayed using a rectangle whose height corresponds to the visual height of the node. In addition, events from $\mathsf{intersect}_{\mathcal{G}}(\tau(n)) \setminus \mathsf{cover}_{\mathcal{G}}(\tau(n))$ are displayed as grayed areas to indicate their presence.

When two covering events $e_1, e_2 \in \mathsf{cover}_{\mathcal{G}}(\tau(n))$ intersect at the time axis, it is obvious that there is a collision and the intersecting area of their rectangles is painted in red to indicate the problem. Similarly, if a covering event intersects with an intersecting event, it also means a collision; however, this kind of collision is weaker because it affects only a part of individuals from $\tau(n)$. On the other hand, if two intersecting events $e_1, e_2 \in \mathsf{intersect}_{\mathcal{G}}(\tau(n))$ intersect at the time axis, it may or may not indicate a problem – in this case, the pair of events must be examined using the definition of event collision (see Section 4.4).

Covering events may be computed using bit-vector operations on columns $\mathsf{M}^{\mathsf{T}}_{\mathcal{G}}(t)$ of the concern matrix, according to the following lemma.

Lemma 5 (Concern relations).

$$\begin{aligned} \mathsf{cover}_{\mathcal{G}}(t_n) &= \bigcap \left\{ \mathsf{M}^{\mathsf{T}}_{\mathcal{G}}(t) \mid t \in \mathsf{T}_{\mathcal{G}}(t_n) \right\}, \\ \mathsf{intersect}_{\mathcal{G}}(t_n) &= \bigcup \left\{ \mathsf{M}^{\mathsf{T}}_{\mathcal{G}}(t) \mid t \in \mathsf{T}_{\mathcal{G}}(t_n) \right\}. \end{aligned}$$

Events found using the bit-vector arithmetics are then ordered by their position on the time axis and their mutual collisions are checked during a single scan along the time axis, as shown in the following section.

5 ALGORITHMS

In order to use the formal model practically, we have developed three algorithms that compute the useful knowledge from the underlying data. The most important algorithm is the detection of collisions among events in a given context. Other two tightly coupled algorithms, the extraction algorithm and the generator of functional dependencies, are used for user navigation in the attribute trees. The algorithms were then used in the application MetroNG [1], see Section 6 for its description.

5.1 Collision Detection

The basic task solved by the collision detector is the enumeration of all collisions among all events in a given context, i.e., all events in the intersecting concern $\mathsf{intersect}_{\mathcal{G}}(t_C)$ (see Definition 19) of a given tuple $t_C \in \mathcal{T}$. The corresponding algorithm is shown in Algorithm 1.

The collision detection algorithm starts by the enumeration of all events in the intersecting concern (line 2) – based on the Lemma 5, this can be done in $O(|\mathcal{E}| \cdot |\mathcal{G}|)$

Algorithm 1 Collision Detection Algorithm

```
Input: $t_C \in \mathcal{T}$
Output: $C \subset \mathbf{P}(\mathcal{E})$ – the set of collisions
1: $A := \emptyset$
2: for all $e \in \mathsf{intersect}_{\mathcal{G}}(t_C)$ do
3:   $A := A \cup \{\langle e.T_{\mathsf{begin}}, 0, e\rangle, \langle e.T_{\mathsf{end}}, 1, e\rangle\}$
4: end for
5: sort $A$ using lexicographical order on triplets
6: $E := \emptyset$
7: for all $\langle T, f, e\rangle \in A$ do
8:   if f $= 0$ then
9:     for all $e_1 \in E$ do
10:      if $\mathsf{M}_{\mathcal{G}}(e) \cap \mathsf{M}_{\mathcal{G}}(e_1) \neq \emptyset$ then
11:        $C := C \cup \{\{e, e_1\}\}$
12:      end if
13:    end for
14:    $E := E \cup \{e\}$
15:  else
16:    $E := E \setminus \{e\}$
17:  end if
18: end for
```

time. For every event, its begin and end times $e.T_{\mathsf{begin}}$, $e.T_{\mathsf{end}}$ are added to the time axis A and later sorted (line 3 and 5).

The core part of the algorithm examines all intervals on the time axis (line 7), keeping track of active events E at each moment of time. Whenever a new event e starts, it is compared to every previous event e_1 (line 10), using bit operations on rows of the concern to detect event collision according to Definition 14. The worst-case time complexity of the core part is $O(|\mathcal{E}| \cdot |\mathcal{E}| \cdot |\mathcal{G}|)$ because $|A| \leq 2|\mathcal{E}|$ and $|E| \leq |\mathcal{E}|$.

Although the time complexity of the algorithm is, in principle, cubic, its real performance cost is negligible even for an interactive application, because the above-mentioned worst-case limits for the sizes $|A|$ and $|E|$ are highly overestimated with respect to real data. Furthermore, the bit operations may be computed extremely quickly in current computer architectures.

5.2 Extraction Algorithm

The non-skipping attribute trees defined in Definition 18) are used for navigation in structures of event visitors, namely student groups and teachers. When the user selects the appropriate level-of-detail, he/she incrementally expands the branches from the root of the tree until the desired nodes are reached.

Unfortunately, the trees may be very large for a common user (hundreds or thousands of nodes); their presentation in a fully expanded tree is unacceptable. In order to expand only the proper parts of the tree, the extraction algorithm is used.

When a leaf of a non-skipping attribute tree is expanded (by the user), the extraction algorithm computes the set of children of the selected node. If the set is non-empty, the node is no longer a leaf but an internal node – from the theoretical point of view, the previously displayed non-skipping attribute tree is replaced by a new, larger one. In each step, the extraction algorithm computes all feasible nodes, i.e., all possible expansions of the trace.

The algorithm takes the whole tuple set and the expanded tree branch as an input; it incrementally computes a set of successors (attribute types) for the branch. Two principal data structures are used within this algorithm:

- $\mathcal{G}$: a group base defined by an instance administrator.
- f_{D}: boolean matrix indexed by attribute types containing functional dependencies between pairs of attributes with respect to the tree branch t_{B} being currently unrolled. $f_{\mathsf{D}}(x, y) = \mathit{true} \Leftrightarrow t_{\mathsf{B}} \vdash y \rightsquigarrow x$. This matrix is computed on each run of the algorithm when the next tree level is incrementally unrolled.

The algorithm is divided into two main parts – computing next level and generating functional dependencies.

5.2.1 Next Level

The Algorithm 2 contains the main *NextLevel* function. The idea is that based on a functional dependency matrix computed from the tree branch all dependent attribute types are excluded from the set of candidates.

Algorithm 2 NextLevel function

```
Input: $t_{\mathsf{B}} \in \mathcal{T}$
Output: $A_R \subseteq \mathcal{A}$
 1: $A_C := \{a \in \mathcal{A} \mid t_{\mathsf{B}}(a) = \emptyset\}$
 2: $f_{\mathsf{D}}$ := GenFunDep($t_{\mathsf{B}}$,$A_C$)
 3: $A_R := A_C$
 4: for all $x \in A_C$ do
 5:   if $f_{\mathsf{D}}(x, x)$ then
 6:     $A_R := A_R \setminus \{x\}$
 7:   end if
 8:   for all $y \in A_C \mid f_{\mathsf{D}}(x, y)$ do
 9:     $A_R := A_R \setminus \{y\}$
10:   end for
11: end for
```

Detailed description:

- 1: A_C is a set containing all possible candidates for the next tree level, initialized by all unused attributes, i.e., the attributes that are not contained in the tree branch.
- 2: The functional dependency of all attribute pairs is computed.
- 4–11: All unused functionally dependent attributes are removed.
- 5–7: Empty and constant attributes effectively behave like functionally self-dependent, they are removed from A_C.
- 8–10: Each attribute y that is functionally dependent on any unused attribute x is removed since x must be unrolled prior to y.

The remaining attributes in A_C define the set of possible attributes for the next level.

5.2.2 Functional Dependencies

The key operation of the *NextLevel* function is the computation of functional dependencies. The corresponding Algorithm 3 works as follows:

- 1–5: The *lv* and *ch* maps indexed by $\mathcal{A}$ are used for constantness detection, their elements contain last attribute values and number of distinct values respectively. Initially, no attribute values exist, no attribute shall be displayed (it will be changed later).
- 3–4: First, the dependency matrix is filled. The diagonal true values represent self dependence, i.e., the absence of values of such attribute.
- 6–21: Each matching tuple in the tuple set is detected for constantness, non-emptiness and potential functional dependency.
- 6–8: Within each tuple in the tuple set matching the tree branch, all attributes that were not used in the tree branch are tested.
- 9–10: Number of distinct values of such attribute (or, more exactly, value changes) within the matching tuple set is detected.
- 12–13: If the unused attribute is contained in the tuple, the functional self-dependency is cleared.
- 14–18: Now we have one particular attribute in one tuple; each distinct non-empty unused (not contained in the tree branch) attribute is set to be a functionally dependent candidate since there may be a relation between these attributes. These relations will be checked later. The dependency cannot be detected in one pass since there may be independent attributes (x_1y_1, x_1y_2, x_2y_1, x_2y_2); in this case all possible combinations should be generated.
- 22–26: Each constant attribute (its value was never changed in the set of matching tuples) is excluded from candidates – there is nothing to select from.

Algorithm 3 Function GenFunDep

```
Input: $\mathcal{G} \subseteq \mathcal{T}; t_{\mathsf{B}} \in \mathcal{T}$
Output: $f_{\mathsf{D}} : \mathcal{A} \times \mathcal{A} \to$ Boolean
 1: $lv : \mathcal{A} \to \mathcal{D}$ ; $lv := \emptyset$
 2: $ch : \mathcal{A} \to \mathbf{N}$ ; $(\forall a)\ ch(a) := 0$
 3: for all $\langle x, y \rangle \in \mathcal{A} \times \mathcal{A}$ do
 4:   $f_{\mathsf{D}}(x, y) := (x = y)$
 5: end for
 6: for all $i \in \mathsf{T}_{\mathcal{G}}(t_{\mathsf{B}})$ do
 7:   for all $x \in \mathcal{A} \setminus \mathbf{dom}(t_{\mathsf{B}})$ do
 8:     if $i(ax) \neq lv(x)$ then
 9:       $ch(x) := ch(x) + 1$
10:       $lv(x) := i(x)$
11:     end if
12:     if $x \in \mathbf{dom}(i)$ then
13:       $f_{\mathsf{D}}(x, x) :=$ false
14:       for all $y \in \mathcal{A} \setminus \mathbf{dom}(t_{\mathsf{B}})$ do
15:         if $x \neq y \wedge y \in \mathbf{dom}(i)$ then
16:           $f_{\mathsf{D}}(x, y) :=$ true
17:         end if
18:       end for
19:     end if
20:   end for
21: end for
22: for all $x \in \mathcal{A}$ do
23:   if $ch(x) \leq 1$ then
24:     $f_{\mathsf{D}}(x, x) :=$ true
25:   end if
26: end for
27: for all $i \in \mathsf{T}_{\mathcal{G}}(t_{\mathsf{B}})$ do
28:   for all $j \in \mathsf{T}_{\mathcal{G}}(t_{\mathsf{B}}) \wedge j < i$ do
29:     for all $x \in \mathcal{A} \setminus \mathbf{dom}(t_{\mathsf{B}})$ do
30:       if $x \in \mathbf{dom}(i) \wedge x \in \mathbf{dom}(j) \wedge i(x) = j(x)$ then
31:         for all $y \in \mathcal{A} \setminus \mathbf{dom}(t_{\mathsf{B}})$ do
32:           if $x \neq y \wedge (y \notin \mathbf{dom}(i) \vee y \notin \mathbf{dom}(j) \vee i(y) \neq j(y))$ then
33:             $f_{\mathsf{D}}(y, x) :=$ false
34:           end if
35:         end for
36:       end if
37:     end for
38:   end for
39: end for
```

- 27–39: The main part of the functional dependency detection – dependency candidates are checked.
- 27–29: Each pair of the matching tuples is compared and functional dependency disablers are detected.
- 30: Each unused attribute x having equal value in both tuples is considered.
- 31–32: If another unused attribute y is either empty in one of the tuples or their values are different ...
- 33–...: then x is not functionally dependent on y since there exist distinct values of y for one value of x.

5.2.3 Examples

This section contains examples of particular data and their processing. The examples use the group base defined in Table 1. This data set is a very simplified real-world excerpt. Nevertheless, using this data the important properties of the extraction algorithm can be demonstrated.

There are 4 attributes used – 'D', 'P', 'R' and 'S'. Their real-world meaning is the education level (Bachelor/Master), the field of study (Informatics/Mathematics), the year of study and specialization (Programming/Computer Science/Software Systems), although it is irrelevant for the algorithm.

Example 5. Branch tree = $\{B, -, -, -\}$

	D	P	R	S
D	1	0	0	0
P	0	0	1	1
R	0	1	0	1
S	0	1	1	0

	D	P	R	S
D	1	0	0	0
P	0	0	0	1
R	0	0	0	0
S	0	0	0	0

Table 2. Dependency matrix for $\{B, -, -, -\}$

Table 2 contains the computed dependency matrix in two versions – the left matrix is produced by the initialization phase of the algorithm, the right matrix is a final output of the algorithm.

In this example the branch tree consists of one attribute D with value of 'B'. The first phase detects the following facts:

- D is excluded since it is already contained in the branch tree
- P, R and S attributes are not used and nonempty; they may have functional dependencies

The second phase erases those functional dependency candidates that have no data dependency in matching tuples. E.g., the tuple pair $i = \{B, I, 1, -\}$, $j = \{B, I, -, -\}$ and attribute pair $ax = P$, $ay = R$ induce that P is not functionally dependent on R since there exist distinct values of R for one value of P, so that

dp[P, R] is erased. Similarly the tuple pair $i = \{\mathrm{B, I, 1, IPR}\}$, $j = \{\mathrm{B, I, -, -}\}$ induces that P is not functionally dependent on S.

Finally, S $\rightsquigarrow$ P, D $\rightsquigarrow$ D, the D and S attributes are excluded from the next level; the resultset (set of allowed attributes) $= \{\mathrm{P, R}\}$.

Example 6. Branch tree $= \{\mathrm{N}, -, -, -\}$

Table 3 displays the result of using another value of the attrubute D. There are no 'M' values of the attribute P and no values of the attribute S at all in the matching tuple set. It excludes the S attribute from the resultset in the first phase and the dependecy R $\rightsquigarrow$ P in the second phase. Since the attribute P is constant within the matching tuple set, the final resultset $= \{\mathrm{R}\}$.

	D	P	R	S
D	1	0	0	0
P	0	0	1	0
R	0	1	0	0
S	0	1	1	1

	D	P	R	S
D	1	0	0	0
P	0	0	1	0
R	0	0	0	0
S	0	0	0	1

Table 3. Dependency matrix for $\{\mathrm{N}, -, -, -\}$

Example 7. Branch tree $= \{-, -, -, -\}$

If the branch tree is empty, the whole tuple set is processed for detection of the first level attributes. During the first phase, all diagonal values are erased since no attribute is used. All nondiagonal values are set since all pairs of attributes have a nonempty value in at least one tuple; all attributes are functional dependency candidates.

The second phase removes all dependency disablers. The resulting matrix shows three remaining dependencies: R $\rightsquigarrow$ D, S $\rightsquigarrow$ D, S $\rightsquigarrow$ P. The final resultset is $\{\mathrm{D, P}\}$; these attributes may be used at the first level.

	D	P	R	S
D	0	1	1	1
P	1	0	1	1
R	1	1	0	1
S	1	1	1	0

	D	P	R	S
D	0	0	1	1
P	0	0	0	1
R	0	0	0	0
S	0	0	0	0

Table 4. Dependency matrix for the empty tree

6 VISUALIZATION

The theoretical background and the algorithms described in the previous sections have been implemented in the MetroNG system, which consists of two complimentary applications. The web interface offers access to the schedule namely for students and teachers; this web application will not be discussed further in this article. The second application is the MetroNG client application intended for creators of the schedule.

6.1 Application Modes

MetroNG supports several application modes; each mode de facto displays several dimensions of the data and it is tailored for a specific class of events. The most common type of events is a regular event being held every week of the semester at the same time. The regular modes display these regular events using a days-of-the-week time dimension as the X axis. These modes are used most of the time and an example is displayed in Figure 1. Week-oriented modes are used for irregular events. These modes allow the users to display schedule for each week individually.

Another type of events are block-oriented lectures; all working days in one week are dedicated to a single lecture for a particular group of students. This type of events is commonly used for practices and clinical education. The block mode supports this type of events by using the weeks of the semester as the X axis; while each week is displayed as a single column.

Moreover, there are three special-purpose application modes used for maintenance work and a lot of other events in the system, e.g., room reservations not related to lectures, students' busy time, teachers' preferences, etc.

6.2 Display Areas and Axes

Figure 1 displays the principal areas of the main application window. The text-oriented part on the left contains a sorted and filtered list of events together with their most important data fields. It is possible to directly schedule or reschedule these events by dragging them to the graphical grid.

The graphical part displays events in a grid-like way. All views share the same horizontal axis, but each view has a different vertical axis, usually rooms, student groups, and teachers. Although these areas are used in all regular modes, their content is dependent on the horizontal axis bound to a particular mode. These grid areas are complemented by a detail area at the bottom that displays additional information on any object including overlapping or colliding events.

The areas are bound to vertical and horizontal axes; there are 10 different linear and tree-shaped axes in MetroNG and their combinations make possible to display the data in the most useful way for particular tasks.

The tree axes hierarchically organize the main scheduling entities – e.g. rooms, students, teachers, and courses. The extraction algorithm (Section 5.2) is useful especially in the students' axis since the structure is often very complex, irregular and the data are too extensive to be displayed at once in an unstructured or regular way. A small slice of the real-world students' axis is depicted in Figure 2. Nevertheless, the algorithms are used for other tree axes in the same way; there is no special adjustment for student groups. The most valuable data (e.g., a classroom for the

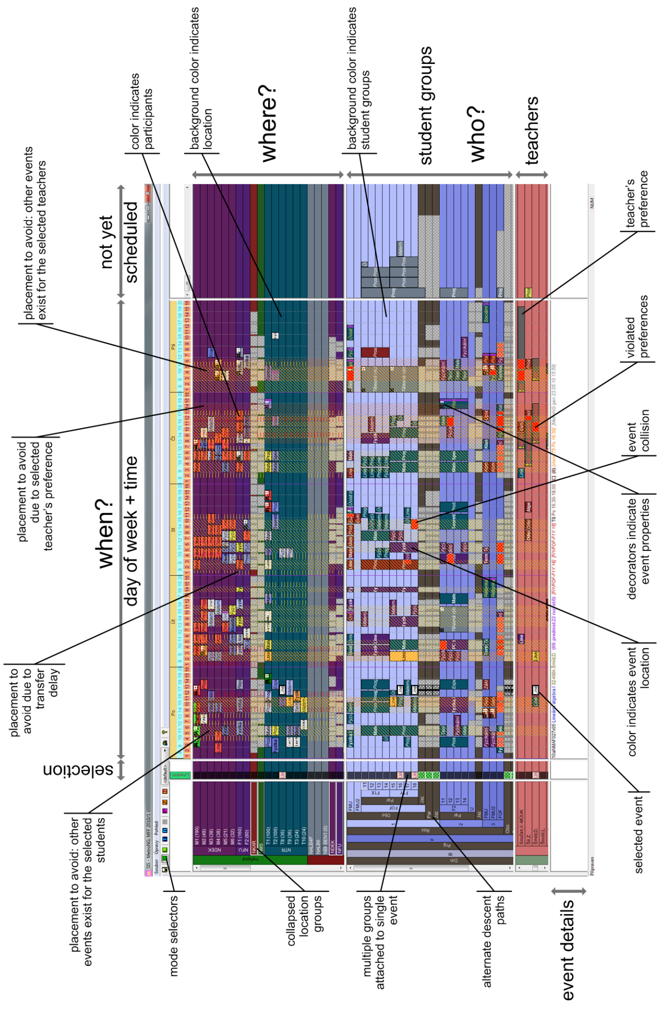
placement to avoid: other events exist for the selected teachers
placement to avoid due to selected teacher's preference
placement to avoid due to transfer delay
placement to avoid: other events exist for the selected students
selection
when?
day of week + time
not yet scheduled
color indicates participants
background color indicates location
where?
background color indicates student groups
student groups
who?
teachers
teacher's preference
violated preferences
event collision
decorators indicate event properties
color indicates event location
selected event
mode selectors
collapsed location groups
multiple groups attached to single event
alternate descent paths
event details

Figure 1. Principal MetroNG Areas

students in particular time, possible conflicts of location, etc.) during the whole scheduling process is then displayed in the intersections of these axes.

6.3 Collisions and Decorators

One of the most valuable feature of MetroNG is the detection of collisions and the prevention of collisions during the scheduling process. Both types of the collisions are implemented using the Collision Detection algorithm (Section 5.1). Existing collisions are displayed as hatched rectangles, so that the user can immediately

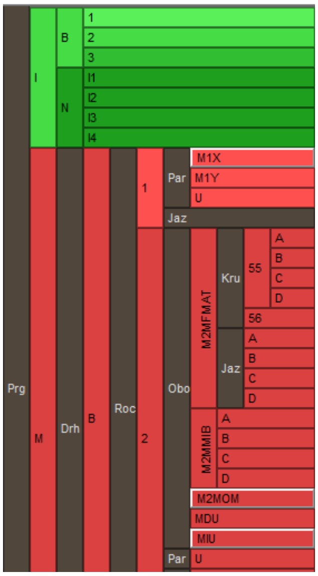

Figure 2. Students' axis

see an existing collisions (see Figure 3 and Figure 4). The figure also shows other *decorators* used to display important information about the events (e.g., status flags, event lock, etc).

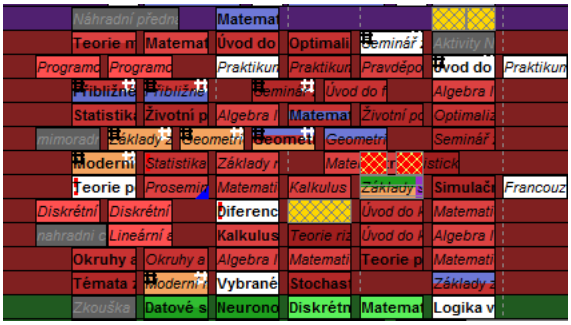

Figure 3. Event Decorators

Other important decorator is the color used as the background of the rectangle. Its meaning depends on the area where the rectangle is displayed. In the students area, the color of the rectangles is the same as the color of the building they are scheduled to. In the room area, there is the same color as the student groups that attend the event. Note that in Figure 1 in the top (room) section of the view the color of the background (color of the building) very often

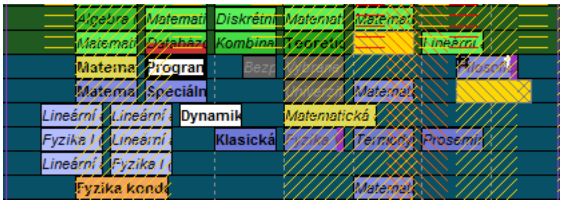

Figure 4. Collisions

matches the color of the rectangle. This is an intentional side effect of this principle.

So far, we described graphical elements that help the user see the current state of the schedule. However, there is another important group – features that allow the user to see possible effects of his or her actions (of changing the schedule). For an example see Figure 4. When an event is selected, some parts of the view are covered with hatching. It displays the areas where there is a potential collision between the selected event and other events – that is, if the event is scheduled to that time and room, it would create a particular collision.

7 CONCLUSIONS

The main contributions of this paper can be summed up as follows:

- formal model of events and visitors
- algorithms that efficiently detect event collisions and compute a structure of user navigation in such multidimensional data without explicit specification of the navigation structure
- the MetroNG application that implements the formal model and proposed algorithms; the application supports the whole process of creating a complex university and curricula schedule.

The formal model for the complex collision detection system is used to identify situations where a group of students should attend two classes scheduled for the same time. Using the model and algorithms, there is no need for the users to explicitly describe the navigation structure of the underlying data; the structure is computed automatically from the used data-set (group base) and its current state (tuple traces). The algorithms are successfully implemented in the MetroNG scheduling tool which is currently used in real-life by some of the largest universities in the Czech Republic for modeling complicated large-size schedules – e.g., at the Charles University in Prague, there are 53 000 students, 6 000 teachers, 650 study programs and 40 000 scheduling events. The real-life experience of the users shows that the presented methods are able to solve their task efficiently.

In our future work, we intend to formalize the user requirements and the quality of the resulting output and compare our solution to automatic schedulers.

Acknowledgment

This work was supported by the Grant Agency of the Czech Republic, grant number SVV-2013-267312, GACR 204/13/08195 and PRVOUK P46.

REFERENCES

[1] Bednarek, D.—Dokulil, J.—Yaghob, J.—Zavoral, F.: MetroNG: Multimodal Interactive Scheduling Interface. Proceedings of the International Conference on Advanced Visual Interfaces, ACM, 2010, pp. 317–320.

[2] Carpineto, C.—Romano, G.: Concept Data Analysis: Theory and Applications. Wiley, 2004, ISBN 978–0–470–85055–8.

[3] Ganter, B.—Wille, R.: Contextual Attribute Logic. In: Tepfenhart, W., Cyre, W. (Eds.): Conceptual Structures: Standards and Practices. LNAI, 1999, Vol. 1640, pp. 377–388.

[4] Ganter, B.: Contextual Attribute Logic of Many-Valued Attributes. In: Carbonell, J. G., Siekmann, J. (Eds.): Formal Concept Analysis. LNCS, 2005, Vol. 3626, pp. 101–103.

[5] Birkhoff, G.: Lattice Theory. American Mathematical Soc., 1984.

[6] Sehwan, Y.—Jongdae, J.—Dae Ryong, K.: Self Conflict Resolving Interactive Web-Based Class Scheduling System. Academy of Information and Management Sciences Journal, Vol. 8, 2005, No. 2, pp. 69–78.

[7] Visual Classroom Scheduler, Visual Scheduling Systems, 2001, `http://www.vss.com.au/index.asp`.

[8] Class Scheduler, Cyber Matrix, 2009, `http://www.cybermatrix.com/class_scheduler.html`.

[9] Lantiv Timetabler, Lantiv, 2009, `http://www.lantiv.com/`.

[10] Pothitos, N.—Stamatopoulos, P.—Zervoudakis, K.: Course Scheduling in an Adjustable Constraint Propagation Schema. 2012 IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI), 2012, Vol. 1, pp. 335–343, DOI: 10.1109/ICTAI.2012.53.

[11] Dasgupta, P.—Khazanchi, D.: Adaptive Decision Support for Academic Course Scheduling Using Intelligent Software Agents. International Journal of Technology in Teaching and Learning, Vol. 1, 2005, No. 2, pp. 63–78.

[12] Saltzman, R.: An Optimization Model for Scheduling Classes in a Business School Department. Journal of Operations Management, Vol. 7, 2009, No. 1, pp. 84–92.

[13] Kingston, J. H.: The KTS High School Timetabling System. The 6th International Conference on Practice and Theory of Automated Timetabling (PATAT), 2006, pp. 181–195.

[14] Tsang, E.: A Glimpse of Constraint Satisfaction. Artificial Intelligence Review, Vol. 13, 1999, No. 3, pp. 215–227.

[15] Carter, M. W.: A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo. Practice and Theory of Automated Timetabling III. LNCS, 2001, Vol. 2079, pp. 64–82.

[16] Mathaisel, D. F. X.—Comm, C. L.: Course and Classroom Scheduling: An Interactive Computer Graphics Approach. Journal of Systems and Software, Vol. 15, 1991, Issue 2, pp. 149–157, ISSN 0164–1212.

[17] Abdennadher, S.—Marte, M.: University Course Timetabling Using Constraint Handling Rules. Applied Artificial Intelligence, Vol. 14, 2000, No. 4, pp. 311–325.

[18] Burke, E.—Bykov, Y.—Petrovic, S.: A Multicriteria Approach to Examination Timetabling. LNCS, 2001, Vol. 2079, pp. 118–131.

[19] Dimopoulou, M.—Miliotis, P.: Implementation of a University Course and Examination Timetabling System. European Journal of Operational Research, Vol. 130, 2001, No. 1, pp. 202–213.

[20] Rudová, H.—Murray, K.: University Course Timetabling with Soft Constraints. LNCS, 2003, Vol. 2740, pp. 310–328.

[21] Burke, E. K.—MacCarthy, B.—Petrovic, S.—Qu, R.: Case-Based Reasoning in Course Timetabling: An Attribute Graph Approach. LNCS, 2001, Vol. 2080, pp. 90–104.

[22] Burke, E. K.—Newall, J. P.: Solving Examination Timetabling Problems Through Adaption of Heuristic Orderings. Annals of Operations Research, Vol. 129, 2004, No. 1-4, pp. 107–134.

[23] Schaerf, A.—Meisels, A.: Solving Employee Timetabling Problems by Generalized Local Search. LNCS, 2000, Vol. 1792, pp. 380–389.

[24] Burke, E.—Bykov, Y.—Newall, J.—Petrovic, S.: A Time-Predefined Local Search Approach to Exam Timetabling Problems. IIE Transactions, Vol. 36, 2004, No. 6, pp. 509–528.

[25] Ross, P.—Hart, E.—Corne, D.: Genetic Algorithms and Timetabling. Natural Computing Series, Advances in Evolutionary Computing: Theory and Applications, 2003, pp. 755–777.

[26] Beligiannis, G. N.—Moschopoulos, C. N.—Kaperonis, G. P.—Likothanassis, S. D.: Applying Evolutionary Computation to the School Timetabling Problem: The Greek Case. Computers and Operations Research, Vol. 35, 2008, No. 4, pp. 1265–1280.

[27] Nedjah, N.—de Macedo Mourelle, L.: Evolutionary Time Scheduling. International Conference on Information Technology, Coding and Computing (ITCC '04), Vol. 2, 2004, pp. 357–361.

[28] Qarouni-Fard, D.—Najafi-Ardabili, A.—Moeinzadeh, M.-H.: Finding Feasible Timetables with Particle Swarm Optimization. 4th International Conference on Innovations in Information Technology, 2007, pp. 387–391.

[29] Chu, S.-C.—Chen, Y.-T.—Ho, J.-H.: Timetabling Scheduling Using Particle Swarm Optimization. 1st International Conference on Innovative Computing, Information and Control, 2006, pp. 324–327.

[30] Socha, K.—Knowles, J.—Sampels, M.: A MAX-MIN Ant System for the University Course Timetabling Problem. LNCS, 2002, Vol. 2463, pp. 1–13.

[31] di Gaspero, L.—Schaerf, A.: Tabu Search Techniques for Examination Timetabling. LNCS, 2001, Vol. 2079, pp. 104–117.

[32] Burke, E. K.—Kendall, G.—Soubeiga, E.: A Tabu-Search Hyperheuristic for Timetabling and Rostering. Journal of Heuristics, Vol. 9, 2003, No. 6, pp. 451–470.

[33] Rossi-Doria, O.—Sampels, M.—Birattari, M.—Chiarandini, M.—Dorigo, M.—Gambardella, L. M. et al.: A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem. LNCS, 2003, Vol. 2740, pp. 329–351.

[34] Bilgin, B.—Özcan, E.—Korkmaz, E. E.: An Experimental Study on Hyper-Heuristics and Exam Timetabling. 6th International Conference on the Practice and Theory of Automated Timetabling, 2006, pp. 123–140.

[35] Burke, E. K.—McCollum, B.—Meisels, A.—Petrovic, S.—Qu, R.: A Graph-Based Hyper Heuristic for Educational Timetabling Problems. European Journal of Operational Research, Vol. 176, 2007, pp. 177–192.

[36] Burke, E. K.—Petrovic, S.: Recent Research Directions in Automated Timetabling. European Journal of Operational Research, Vol. 140, 2002, pp. 266–280.

[37] Adriaen, M.—de Causmaecker, P.—Demeester, P.—Berghe, G. V.: Tackling the University Course Timetabling Problem with an Aggregation Approach. 6th International Conference on the Practice and Theory of Automated Timetabling, 2006, pp. 330–335.

[38] Malim, M. R.—Khader, A. T.—-Mustafa, A.: Artificial Immune Algorithms for University Timetabling. 6th International Conference on the Practice and Theory of Automated Timetabling, 2006, pp. 234–245.

[39] Perzina, R.: Solving the University Timetabling Problem with Optimized Enrolment of Students by a Parallel Self-Adaptive Genetic Algorithm. 6th International Conference on the Practice and Theory of Automated Timetabling, 2006, pp. 264–280.

[40] Ten Eikelder, H. M. M.—Willemen, R. J.: Some Complexity Aspects of Secondary School Timetabling Problems. LNCS, 2001, Vol. 2079, pp. 18–27.

[41] de Haan, P.—Landman, R.—Post, G.—Ruizenaar, H.: A Four-Phase Approach to a Timetabling Problem in Secondary Schools. 6th International Conference on the Practice and Theory of Automated Timetabling, 2006, pp. 423–425.

[42] Jacobsen, F.—Bortfeldt, A.—Gehring, H.: Timetabling at German Secondary Schools: Tabu Search Versus Constraint Programming. 6th International Conference on the Practice and Theory of Automated Timetabling, 2006, pp. 439–442.

[43] Kingston, J. H.: The KTS School Timetabling System. 6th International Conference on the Practice and Theory of Automated Timetabling, 2006, pp. 181–195.

[44] Cooper, T. B.—Kingston, J. H.: The Complexity of Timetable Construction Problems. TR No. 495, Basser Department of Computer Science, The University of Sidney, 1995.

David Bednárek received his Ph. D. in informatics from Charles University in Prague in 2009. His research interests include programming languages, compiler construction, parallel programming, and database systems. He was involved in many national and international research projects. He served as member of the program committee of several international conferences.

Jakub Yaghob is currently associated with the Charles University in Prague, Faculty of Mathematics and Physics. He graduated consequently at the Charles University in Prague (1991 – M.Sc., 2003 – Ph.D.). He is responsible supervisor for numerous development grants of the Ministry of Education and FRV grants (e.g. Environment for teaching parallel programming). He was also working on numerous GAR grants (Highly Scalable Parallel and Distributed Methods of Data Processing in e-Science amongst the others). He is a member of a few international program committees of various conferences and workshops taking the role of the program or organization committee chair several times.

Filip Zavoral is the vice-head of the Department of Software Engineering, Charles University in Prague, Czech Republic. His research interests include distributed and parallel technologies, cloud computing and efficient data processing. He was involved in many national and international research projects. He has co-authored more than 60 research publications such as: journal papers, conference proceedings papers, book chapters, and editorials of journal special issues. He is member of the editorial board of several international journals and served as member of the program or organizing committee of many international conferences.