AgentPlanner – Agent-based Timetabling System

Rafał Tkaczyk
Systems Research Institute of the Polish Academy of Sciences, Warsaw, Poland
IT Systems Department of the Vemco Co. Ltd., Sopot, Poland
E-mail: rafal.tkaczyk88@gmail.com

Maria Ganzha
Systems Research Institute of the Polish Academy of Sciences, Warsaw, Poland
Institute of Informatics, University of Gdańsk, Gdańsk, Poland
E-mail: maria.ganzha@ibspan.waw.pl

Marcin Paprzycki
Systems Research Institute of the Polish Academy of Sciences
Warsaw Management Academy, Warsaw, Poland
E-mail: marcin.paprzycki@ibspan.waw.pl

Keywords: software agent, multi-agent system, timetabling, negotiations, agent-based timetabling system

Received: July 24, 2015

The aim of the paper is to describe the AgentPlanner, an agent-based timetabling system. After its initial implementation (described in [1]), based on results of experiments, we have modified the design (to eliminate discovered shortcomings). Here, we describe the improved AgentPlanner and compare its performance with the state-of-the-art, Free Timetabling Software (FET).

Povzetek: Opisan je AgentPlanner, agentni sistem za urnike.

1 Introduction

Creating a timetable is a challenging problem. On the one hand, timetables are widely used in multiple application areas. On the other, timetabling is an NP-hard problem. As a result, many methods that solve this problem have been proposed (see, section 5).

Recently, we have developed an agent-based timetabling system and reported preliminary experimental results concerning its performance (in, [1]). After the publication, we have been contacted by the developers of the FET program [2]. Discussions that ensued, combined with our own assessment of the shortcomings of the initial version of the AgentPlanner, resulted in improvements in its design. Furthermore, we have made changes in the experimental setup, to make the comparison more fair. Therefore, in the current contribution, we present a completely new set of experimental results.

We proceed as follows. In the next section we summarize the state-of-the-art in timetabling. Next, we outline the reasons that shaped the specific design of our AgentPlanner system. We, then, describe details of its implementation. In the penultimate section, we present the results of performed experiments. Finally, we discuss issues related to flexibility of the AgentPlanner design and possible future research directions / improvements.

2 Timetabling – related work

Timetabling is a common problem, which is applied in many domains (e.g. business, industry, science, private applications, etc.). Therefore, many scientists have considered it, and many solution methods have been proposed. Below, we summarize few most common and effective methods for solving the timetabling problem. Let us note, that our work concerns scheduling courses in a “college,” and this provides the context for what follows. Furthermore, due to the lack of space, details of described methods are omitted. Interested readers should consult references.

2.1 Heuristic methods

2.1.1 Genetic algorithms (GA)

Typical approach, used when applying genetic algorithms to solve the timetabling problem, is as follows. A gene is regarded as an activity, and it is associated with a group of students and their teacher. The gene is obtained as a result of the assignment of an activity to the teacher (who leads the activity) and to a group of students (who participate in the activity). A chromosome (schedule) consists of genes (activities). The idea of scheduling is to allocate activities in a plan (genes in the chromosome), i.e. the problem to be solved is treated as a problem of assignment of entities within available slots. Chromosomes prepared in this way are evaluated against constrains and standard techniques for evolving improving solutions are applied. More specific description can be found in [5].

2.1.2 Artificial Immune Systems (AIS)

Artificial Immune Systems (AIS) are based on the metaphor of the natural immune system, and work by "focusing" on anomaly detection [15]. The main idea of AIS is to operate on a population of antibodies (feasible timetables) and using proper method (immune algorithm (IA), or a hybrid, e.g. combining IA with GA) to find and replace an "anomalous entity" (bad timetable) with a better solution. In [16], application of IAS to the university timetabling has been described. Authors presented three kinds of algorithms using AIS: clonal selection algorithm, immune network algorithm and a negative selection algorithm.

2.1.3 Graph coloring

To solve the timetabling problem, edge and vertex coloring can be applied. (i) Vertex coloring. In this approach, all activities are vertices. Edges indicate pairs of vertices (activities) that cannot be scheduled at the same time (e.g. when the same teacher leads them). The core of the method is to perform legal coloring of the graph representing conflicts, where colors indicate time slots. (ii) Edge coloring. Here, a very simple example of a possible approach is a bipartite graph coloring, where the first set are vertices that present teachers and the second set presents activities. The idea is to color this graph and (similarly as in the case of vertex coloring) colors indicate time slots [3].

2.1.4 Simplex method

An example of using this method in scheduling we can find in [4]. The constraints are transformed into a system of linear equations. There are 3 main steps of solving the problem: (i) Generate acceptable (non-negative) solution baseline (initial). (ii) Check the optimality of the obtained solution. (iii) If it is not optimal, generate a new basic feasible solution that is not worse than the one previously obtained, and check if the obtained solution is optimal. If it is optimal, the process is completed (because better solution cannot be found). In other words, the last obtained solution is considered optimal.

2.1.5 Tabu Search

The basic paradigm of this heuristic method (examples of which we can be seen in [17]) is to use the search history (distribution of activities in the timetable) to guide the local search approaches to overcome the problem of solver being stuck in local optimum (repeating suboptimal results). It is possible to combine this method with other algorithms, e.g. with graph coloring.

Regardless of how successful are these methods, **all** of them have a major disadvantage. Namely, it is practically impossible to change an already existing schedule. Observe that, when scheduling courses at a university (which is our application area) it is necessary not only to generate a "high quality schedule" (where quality is judged against one or more criteria, see below), but also to provide mechanisms that would allow to shift an individual class, add a new one into an existing schedule, (ex)change rooms, etc. Since the above described methods treat the schedule from a "holistic" perspective, re-scheduling a single class, e.g. for a teacher that got sick during the semester, is a relatively complex task. Simply said, this is not what these methods were created for. Obviously, such changes can be accomplished manually, or by using additional (separate) software, but this means that multiple approaches have to be combined. One, to generate the "initial" schedule, and one to manage it during the course of the semester. Moreover, the larger the input data set (and the more links between items in this set) the more complex is the problem. As a result, the algorithms that can solve the timetabling problem need more computational power and take more time to complete. Note that, due to the holistic approach, in each "step," these algorithms treat the complete problem at once.

Interestingly, it can be stipulated that software agents can handle *both* the schedule preparation and its management, as they are characterized by autonomy, reactiveness, and ability to communicate / negotiate (see, [6, 12], for discussion of application of agents in timetabling).

Furthermore, as will be shown, agents allow to "divide" the problem into smaller subproblems that are solved in each step; thus reducing its overall complexity. Therefore, we have developed a prototype of an agent-based timetabling system (AgentPlanner), which uses agent negotiations to create and maintain (modify) the schedule. In what follows, we describe the AgentPlanner and discuss results of its experimental evaluation, when applied to scheduling university courses.

2.2 Agent based methods

Before proceeding with description of the AgentPlanner, let us summarize the state-of-the-art in using agents in timetabling. We found a few agent based systems (some of them are described in [12]) that are used for planning in logistics, production, defense and insurance sectors, e.g. a scheduling system for taxi companies ([13]) or hospitals ([14]). Obviously, some of them could be reorganized for school timetabling, but it would be difficult because they are designed to solve a specific problem. However, there are also agent-based systems, designed specially for timetabling.

Authors of [9] use a divide-and-conquer approach, combined with software agent technology. Timetabling Agents generate initial solutions, where each agent is responsible for the solution of a specific subproblem. Every agent uses a different heuristic. It is a big advantage, because this approach can apply proper heuristic (appropriate to the specific problem). Moreover there is a Mediator Agent, guarding that all plans are arranged, while satisfying predefined criteria and constraints. Test data is divided into three categories: small, medium and large, and the large dataset(s) are actually big enough to be considered realistic (matching the actual situation at a university). Unfortunately, it is not described how complex are the links between the items. For example, in our test data, students can belong to multiple different groups (obligatory, elective, language group, etc.), thus avoiding a collisions of students' activities is important (but relatively difficult).

Paper [10] describes a similar approach as [9], but there are three types of guarding agents. The first is making sure that the generated sample solutions comply with the main requirements (no collision for trainers, only one class in one room, etc.). The second type agents is guarding "hard" constrains. The third type of agents guards the "soft" constraints (good to have, but not necessary). The number of second and third type of agents depends on the specific course timetabling problem. System can work in two modes: (i) where second and third type of agents evaluate proposals of the first type, and (ii) where second and third type of agents try to improve proposals put forward by the first agent. Unfortunately, in the paper the test data is not well described; just the grid of 5 days and 9 time units are specified, that gives much more space where the solution can be found than in our research (5 days x 6 time units). According to the author "the results are promising" but it is not possible to verify them because there are no actual results in the paper.

Author of [11], is firstly looking for any matching solution and then optimizes it. In the proposed approach, a larger number of agents types (than in the above described papers) is proposed. Almost every element in the plan has an agent representing it: CourseAgent, TeacherAgent, StudentAgent, RoomAgent. A potential disadvantage of this approach is that the more data is to be passed around, and the larger the number of agents in the system, the larger is the number of messages that are to be exchanged. This causes two possible problems: (i) an increased chance of a bottleneck, and (ii) problem of synchronization of communication and actions of the system. In the paper, author showed results when 40 agents were used. However, this is rather a small problem, and does not show any conclusive results concerning scalability of the proposed approach.

Overall, papers [9, 10, 11] describe very interesting approaches to application of agents to timetabling, but their are not well tested. Number of "elements" in the test data is not large and not complex enough to mimick real-world situations. Furthermore, during the our research we stumbled upon many papers of this kind; interesting approaches tested on non-realistic data sets, so we will omit them here.

There exist approaches similar to the AgentPlanner. Negotiation involving teachers' time preferences have been used in [19]. Here, authors use four classes of agents: (1) Teacher Agents, (2) Classroom Agents, (3) History Agent, and (4) two Interface Agents. The role of the Interface Agents is to initialize other agents based on the user setup. The core of the algorithm is negotiations between Teacher Agents who send propositions (prepared on the basis of Teacher preferences) to proper Classroom Agents. They consider the proposals, with help of the History Agent that contains information about all timetables and allows detection of collisions. The size of the dataset is impressive (but there is no description of its complexity) and results are very encouraging. Unfortunately it is not possible to compare effectively these results with our approach (described below) because authors adopted a different evaluation criteria. They focus on number of sent and analysed messages, which guarantee the speed of the system. Teachers' time preferences are used in scheduling but authors did not check if they have been actually fulfilled (to what extent).

Similar approach to result evaluation present authors of [20]. In their work, every weekday is a different platform, where Course Agents are run. They negotiate with each other (via a SignboardAgent – a coordinator that helps find a free time slot). Primary results found in the paper are that using a distributed architecture is better than a centralized one, because of reduction of run time.

Finally, work reported in [21] shows that many agent based systems do not deal with re-scheduling of an existing schedule (e.g. system presented in [20]). To deal with the problem, authors use the Probability Collective theory. Their experimental results are promising, but they cannot be naturally compared with our approach as they (again) have different criteria of evaluation (e.g. time of running having various sets of data or evolution of the probability collective).

Summarizing, results of our research into the state-of-the-art in timetabling have revealed three groups of results. First, large number of "global" approaches to finding the optimal schedule. Here, their main disadvantage is a difficulty to modifying the schedule in response to the changes that occur during its realization. This latter feature is particularly important when dealing with real-world schedule that has to run during a semester at a university / school / college. Second group involves agent-based solutions that were not properly tested, or tested on data sets that were "not complex enough" to represent real-world situations. Finally, agent-based approaches that were somewhat similar to our approach, but in their design and experiment focused on different aspects of timetabling than what was our main goal.

3 AgentPlanner – preliminary considerations

The most important attribute of our approach is take into account teachers' time preferences. In this context, we have started our work by analysing the real needs of faculty members of the Mathematics, Physics and Informatics Department of the University of Gdańsk.

The results of completed analysis allowed us to specify the requirements for the development of our agent-based course scheduling system (the AgentPlanner). First, the AgentPlanner has to deal with both scenarios: (1) to develop a timetable of academic courses in accordance with specified restrictions (creation of a new timetable), and (2) to manage it; i.e. be capable of making requested changes / modifications in the existing class schedule (timetable maintenance). It is important to note that the selected application area: scheduling of courses at a university, has guided formulation of functional and non-functional requirements for the developed system. University course scheduling means that, in addition to creation of an initial course schedule for a given semester, the AgentPlanner has to be able to deal, among others, with: change of location(s) of selected laboratory groups / lectures, sickness of a teacher (i.e. rescheduling missed classes for a later date), adding new activities (e.g. an unscheduled examination caused by multiple students failing the first attempt), etc.

After analysing the actual scheduling process that takes place at the University of Gdańsk, it was decided that only the *Planner* (human system administrator) will be able to run the AgentPlanner to create the timetable. In addition, the *Planner* is going to be the only person who will be authorized to make schedule changes in the database (in particular, during the timetable maintenance phase).

In the AgentPlanner we have introduced some restrictions on the implemented functions. In this way we were able to focus on core functionality and complete experimental evaluation of our approach. In this way, after the initial course schedule is created, both the *Planner* and the *teacher* can send two types of requests: (a) to insert a new activity (group exercises, laboratory, lecture), requiring reorganization of the plan, and (b) to change location of, already scheduled, activity(ies). Observe that both types of requests may impact other teachers. Hence, the proposed rescheduling (resulting from the work of the AgentPlanner) has to be negotiated with those teachers that are affected by the changes. In the current version of the AgentPlanner, to complete a change of the existing timetable, all affected teachers have to agree. Here, for the time being, we do not take into account the fact that the teacher may be forced to accept a change (e.g. by the Dean), and assume benevolence of teachers.

Analysis of the actual course scheduling process lead to the following extra requirements for any system similar to the AgentPlanner. (1) Scheduling should be completed in a reasonable time. (2) Used algorithms must be designed so that the system can be used on computers with limited power (i.e. personal computers). (3) The timetabling system should be easy to install (use well-known and well-documented software). (4) Ease of use (simplicity of the interface) is very important. (5) Timetable requires visualization both in the printed form, as well as in a form that can be sent to the website (to be displayed). Therefore, the system should have various data converters; from the database representation of the schedule, to the appropriate file formats. (6) The scheduling system should be reliable and resilient to possible errors. (7) For obvious reasons, data security is extremely important. Finally, (8) the timetabling system should be portable between various operating systems. This context let us stress, again, that the aim of our current work was not to develop a full-blown system. Therefore, the above "extra requirements" have been mostly omitted. For similar reasons, we have not considered the requirements involved in implementing the AgentPlanner on mobile devices (which may be a very useful – or ever required – functionality for an actual system).

Based on conversations with actual faculty members of the University of Gdańsk, we have formulated the initial "scheduling goals" for the AgentPlanner. As a result, the system aims at: (i) minimizing the number of days of teaching, and (2) locating activities as close as possible to each other (i.e. no big gaps between activities, resulting from some classes taking place in the morning and the remaining ones in the evening). However, it is also possible to control this process by incorporating teachers' preferences (for both: teaching days, and selected time-slots). Specifically, the teacher can rank her preferences concerning days of the week by assigning natural numbers from the interval $[0, 4]$, where 0 is considered to be "unacceptable" and 4 represents "the best option". Similar approach applies to ranking time-slots (each of them can be ranked individually; in this way we can capture preferences such as: I like to teach in the morning vs. I hate to wake up early). In this case, the interval depends on the number of time units per day (we consider $[0, 5]$). It has to be noted that, in the current design, there is no restriction on the number of teaching activities during a single day. Therefore it is possible for a teacher to have classes "all day long" (e.g. 5 courses at a given day; and no classes for the rest of the week). While seemingly unreasonable, this does reflect the actual preferences of faculty members. It is worth to mention that, in Polish universities, single lesson last 45 minutes while time unit usually consists of two lessons, i.e. 90 minutes.

It is very important to adopt some constraints that prevent input data that makes it impossible to create a plan, or that causes a negative, unreliable results of the evaluation function. Here, we have identified key steps of proper representation of time preferences (we have also utilized them when preparing the test data).

(1) All default (undefined by teacher) time units in the schedule are set to the highest possible rank (e.g. 5 for a day consisting of 6 time intervals).

(2) Teacher should consider, which time units are "the best option" for her/him and leave them without changing rank. Minimum number of highest ranked time units depends on the number of the teacher's activities.

(3) Next step is to set ranks less than the highest but higher than 0, represented as natural numbers from the interval $[1, \mathrm{HIGHEST\_RANK} - 1]$, where 1 means that her/his presence is possible but inconvenient. Proceeding in this way teacher can affect allocation of his/her activ-

ities. As a result there is higher probability to achieve a plan that is better than when one does not make such precise specifications.

(4) Teacher should carefully consider, which time units are "unacceptable" for him/her and rank them as 0. However, it is obvious that the more zeros, the more difficult the problem becomes. Therefore, the teacher should use it only if presence is really impossible at that time.

As far as the representation of interests of students is concerned, the prototype takes into account (what we believe to be) the key aspects of a plan: minimization of collisions of courses, number of days of instruction, and gaps during the day. However, we have to admit that the current version of the AgentPlanner has been implemented with primary focus on teacher satisfaction.

Finally, in the current version of the AgentPlanner system, the timetable is created for a single department, located in a single building.

3.1 AgentPlanner as an agent-based system

Recall that the AgentPlanner has been conceptualized as an agent system. On the basis of the requirements analysis, we have envisioned it as depicted in Figure 1.

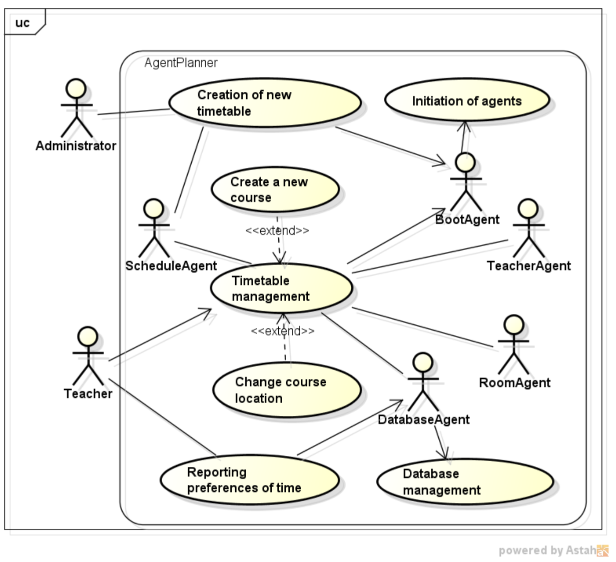

Figure 1: AgentPlanner use case diagram.

Here, we recognize the two main functions of the system, the *Creation of a new timetable*, and *Timetable management*, as well as a number of additional functions needed to complete the two main ones. The current design of the system has only two "external" actors: the *Planner* and the *teacher*. In the future, one may need to include in the design also the *student* actor, but this would lead to a system that is out of scope of our current work. Analysing functional and non-functional requirements of the AgentPlanner system, we have came to the conclusion that it should consist of the following agents:

– *BootAgent*, with the only task to create and start other agents that are required in the AgentPlanner system.

– *DatabaseAgent*, responsible for connection the system with database. All agents have access to data via the *DatabaseAgent*. This agent was found to be required to streamline and organize access to the data stored in the database.

– *RoomsAgent* represents rooms in the scheduling process. It downloads (from the database), filter and store data about rooms in the system (e.g. type of every room, seating capacity, etc.).

– *TeacherAgent* acts on behalf of a teacher (both during creation and management of the timetable). It stores: (i) information about the teacher (including personal data that has to be protected), (ii) list of activities (courses / groups) taught by the teacher, (iii) list of rooms (meeting the requirements of each group and course, e.g. laboratory group has to be scheduled in a laboratory); obtained from the *RoomsAgent*), (iv) results of the location evaluation function (described in Subsection 4.1), and (v) teacher's current timetable.

– *ScheduleAgent* is the central agent of the negotiation algorithm. It "knows" teachers involved in current negotiations (*TeacherAgents* that represent them). It also has access (read and write) to the timetable database (via the *DatabaseAgent*). Note that, all data concerning the currently considered timetable, is (after each change) saved in the database. This allows the *ScheduleAgent* to effectively issue verdicts, which room should be assigned to which requesting teacher (as it knows which rooms are already occupied and which are still available). Note that we are aware of the fact that, in a very large scheduling problem, the *ScheduleAgent* may become a bottleneck. However, solving this problem is out of scope of the current contribution. This is especially the case since the time to calculate the schedule for the (realistic) size of data used in our experiments was acceptable (it took about 1 minute to complete the scheduling task; after all agents were started and provided with their input data).

The negotiation process is between *ScheduleAgent* (a judge) and *TeacherAgents* (representing teachers) but not between *TeacherAgents*. Obviously, it is centralized model, where it is possible to run into a bottleneck (caused by limited processing capability of the *ScheduleAgent*; see, above). However, observe that very complex, time-consuming, negotiations take place only once – during early phases of creation of the initial schedule. This is "acceptable" as the time-pressure is, usually, not too-serious. At the same time, adaptations to the existing schedule, which take place during the semester do not take long time, as they involve only small number of agents (representing teachers affected by the required change(s)).

It is easy to note that the *DatabaseAgent* and the *RoomsAgent* did not have to be implemented as full-fledged agents. For instance, they could have been designed as FIPA-style services [23]. However, we have decided (for the simplicity and uniformity of implementation) to use agents "across the board". Acknowledging that this decision may seem somewhat controversial, we believe that our choice of an implementation method (i) has merit, and (ii) does not influence the experimental results supporting our approach (quality of the obtained solution).

4 Implementation of the AgentPlanner

Based on the above considerations, we have decided that the AgentPlanner should be implemented as a client-server-type system, where all operations concerning generation and maintenance of the timetable are going to be executed as an agent-based server application, while the client component will be responsible only for sending requests and reviewing / accessing results. It is important to keep in mind that access to the database is allowed only on the server side of the application, so every request of the client, or any other agent in system, has to be handled by the *DatabaseAgent*. This decision was based on the fact that, our software of choice (the JADE agent platform), does not provide a robust GUI for user interfaces. Therefore, following advice found in [8] we have decided to clearly separate the agent and non-agent functionality. Furthermore, in the current version of the prototype, the client application is simplified to a "line interface," while the server application has only functionalities needed for the two timetabling operations (schedule creation and maintenance). All data needed for the tests was inserted manually to the database via SQL scripts, or other scripts written for this purpose.

4.1 Evaluation algorithm

The core of the timetabling mechanism is the evaluation algorithm. Here, the *TeacherAgent*(s) evaluate the locations (room information received from the *RoomsAgent*) that best match the need of the teachers. The evaluation algorithm must takes into account: priority of course and lecture, links of students with other groups, teacher preferences, and the current state of the timetable. Overall, every activity has an assigned priority, which describes how important it is for the teacher (e.g. a lecture may have higher scheduling priority than a laboratory). Furthermore, some courses are "more important" than others, e.g. a core course may have a higher rank than an elective (all students have to take the core course, while they may sometimes be "forced" to take a different elective – to avoid course collision(s)). Moreover, courses related to the major (e.g. in our case, CS courses) have higher rank than non-major ones (e.g. psychology courses). Separately, when considering the current timetable, priority is given to activities that can be assigned in the time-vicinity of the already scheduled ones. In this way, the total number of gaps in the schedule of the teacher (and possibly students) can be minimized. The evaluation algorithm works as follows (1). The current

```
Data: S = priority_of_course * 10 + links_number
if day_priority or time_slot_priority is equal 0 then
    do not add room from this time slot to the list
else
    S := S * day_priority * time_slot_priority;
    if there are other lessons in this day then
        S:= S+5
    end
    if there are other lessons around time_slot then
        S:= S+5
    end
end
Result: S
```

Algorithm 1: Evaluation Algorithm.

version of the evaluation algorithm is quite different from the one reported in [1]. The initial value is the sum of the activity priority (multiplied by 10, because in this way, in the experiments, we have received better results) and the number of all groups, to which students from this activity belong. This should be understood as follows: the more links / dependencies between data elements, the harder it is to put the activity in the plan, because the algorithm has to avoid collisions between connected groups. Therefore, the most complex situations should be resolved in the first place. The next step is to consider the most important factor, the teacher's time preferences, i.e. rank of day and time unit. If one of them is equal 0, that means that the teacher cannot lead activity at that time, and function does not add this location to the list. The last element is the evaluation of the "vicinity". It is important to reduce the "time gaps" in the plan. Therefore, to place an activity between two others is the highest ranked situation.

4.2 Timetable planning algorithm

Let us now consider creation of a new timetable. Recall, that the approach is based on a single "judge" (the *ScheduleAgent*), having access to the current timetable (which initially is empty). The *ScheduleAgent* negotiates the timetable with the *TeacherAgent(s)*, using information obtained from the *RoomsAgent*. Negotiations are divided into rounds (since in each round at least one activity is places in the schedule, their number is not larger than the total number of all "activities" – courses / exercise groups / laboratories – of all teachers). Due to the lack of space, we omit the pseudo-code (it is about 4 pages long and can be found in [22]). The general idea of actions that are performed in a single round is depicted in figure 2. Each round begins with the start signal (message) from the *ScheduleAgent* to the *TeacherAgent(s)* (that still have activities to allocate). During a single round, every *TeacherAgent* considers an activity from the list of all teacher's activ-

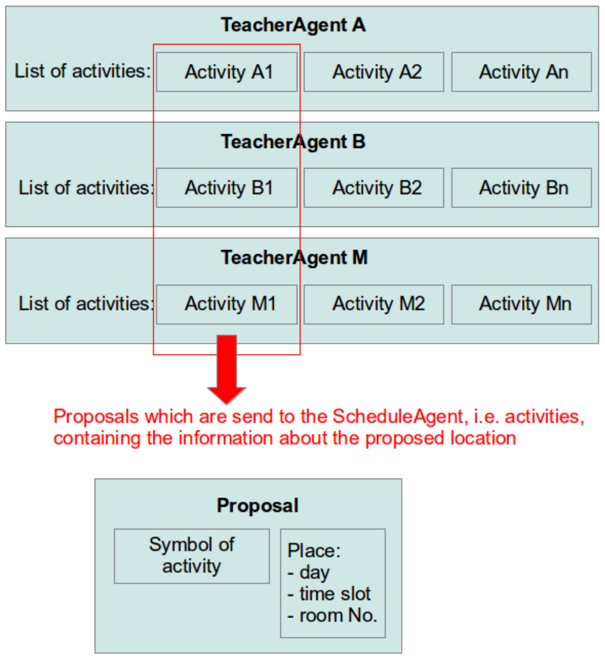

Figure 2: General schema of single round (sending proposals).

ities (initially sorted according to the priority of the activity type) and selects the "most important one". If activities list is empty then the *TeacherAgent* sends to the *ScheduleAgent* a message that it resigns from further negotiations. If not, the next step is to prepare a list of rooms acceptable for the considered activity and sort them according to the results of the evaluation algorithm (1). The *TeacherAgent* selects the most desirable location and sends the proposal to the *ScheduleAgent*. The proposal consists of: (i) symbol of the activity, (ii) number of the week day, (iii) number of the time slot, (iv) result of the evaluation algorithm. When all *TeacherAgents* send their proposals, the *ScheduleAgent* considers them and accepts the best (placing these activities into the current timetable), while rejecting others. Note that, in the current version, we omit the case when one (or more) *TeacherAgent(s)* do not send their proposals. While, in general, this is an important issue for the design of agent-based distributed systems, handling such anomaly is out of scope of our current work. The decision to accept a request depends on two factor: (1) is the requested location already occupied by another activity, and (2) does a given request involve course collisions. Obviously, it is possible that multiple *TeacherAgents* may ask for the same location. In this case, the *ScheduleAgent* selects the one that delivers the best value of the evaluation algorithm. It is also possible that proposals from multiple *TeacherAgents* "have the same value". In this case, the *ScheduleAgent* draws a winner (randomly). Next, the *ScheduleAgent* sends messages to the *TeacherAgents*, about rejected proposals. Then, the *TeacherAgents* select the next best location from the list, and create proposals for the *ScheduleAgent*. If all of its proposals are rejected, then *TeacherAgent* resigns from the given round of negotiations. The result of this decision is recorded in the database for the information of the *Planner*. The round ends when every *TeacherAgent* gets a place for its activity, or when some unscheduled requests cannot be satisfied. Note that, in a single round, the total number of evaluated requests is equal to the number of teachers with unscheduled activities and thus is relatively small and systematically decreasing (when at least some teachers have their schedules complete).

Observe that this approach is based on the assumption that all teachers have the same chances. This is because, in a single round, every *TeacherAgent* can reserve one permanent place for one of its activities. For example, if a professor has two seminar lectures, while an assistant has two exercise groups, then in the first round each one of them will "book" a room for one of their activities (regardless of their position in the academic hierarchy). However, it is not clear if such democratic approach would be sustainable in the real-life university course scheduling. If this was not the case, then the structure of academic dependencies (who, in a given moment, is more important than others) could be represented, as weights, in the evaluation function. However, exploring this possibility is out of scope of our interests.

Before the beginning of the next round, the *ScheduleAgent* receives messages from the *TeacherAgents* with resignations from the given round of negotiation. In response to these messages, it suspends the main thread of negotiations, and runs the timetable reorganization algorithm (see subsection 4.3) to deploy the rejected activities into the current schedule.

It could happen that the reorganization (adding new activity) is impossible, then the activity is added to list of rejected activities, for inspection by the human *Planner*. In this case, the *Planner* has to figure out how to improve the input data, e.g. to change the time preferences of teacher (e.g. by contacting her/him directly). After the schedule is reorganized and, previously rejected proposals are added, the *ScheduleAgent* returns to the main thread and starts the next round of negotiations. Thus, the timetabling algorithm continues from reception of the next group of proposals from those *TeacherAgents* that still have unscheduled activities. The sequence diagram of the timetable planning process is represented in figure 3. After an extended analysis of results reported in [1], we have made an important modification to this algorithm, aimed at eliminating collisions among student activities. Originally, during the negotiations, student collisions were checked against groups that were already inserted into the timetable, but not against the remaining groups that were involved in the given negotiation step. To deal with the collisions, we have decided to sort all proposals according to the results of the evaluation algorithm (see, section 4.1) and insert activities iteratively beginning from the top of the list. However, before inserting an activity into the plan, we now check for possible collisions between student activities and if there are any,

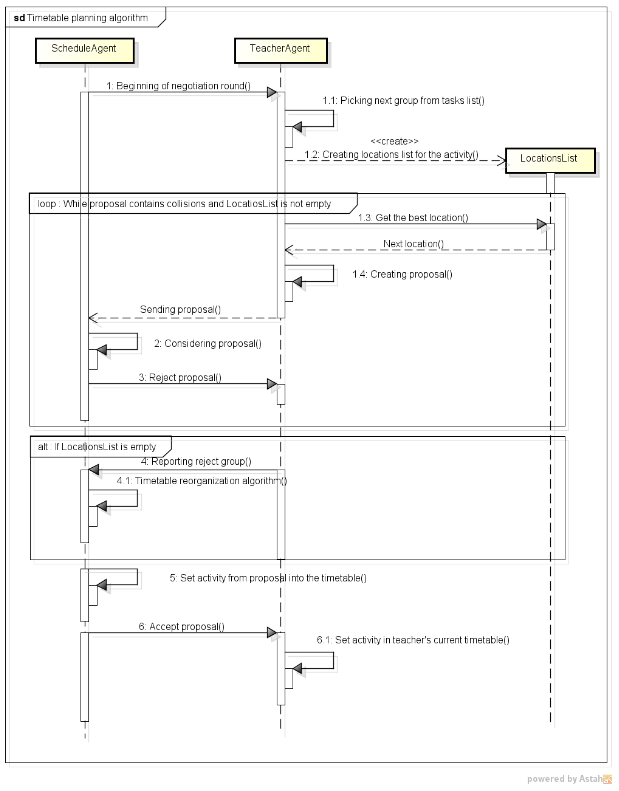
sd Timetable planning algorithm
ScheduleAgent
TeacherAgent
1: Beginning of negotiation round()
1.1: Picking next group from tasks list()
<<create>>
1.2: Creating locations list for the activity()
LocationsList
loop : While proposal contains collisions and LocatiosList is not empty
1.3: Get the best location()
Next location()
1.4: Creating proposal()
Sending proposal()
2: Considering proposal()
3: Reject proposal()
alt : If LocationsList is empty
4: Reporting reject group()
4.1: Timetable reorganization algorithm()
5: Set activity from proposal into the timetable()
6: Accept proposal()
6.1: Set activity in teacher's current timetable()
powered by Astah

Figure 3: Timetable planning algorithm sequence diagram.

we reject such proposal. This approach slows the progress of timetabling, but thanks to it we can generate timetable with no collisions between student activities.

4.3 Timetable reorganization algorithm

The timetable reorganization algorithm is used in two situations. First, when in a single round, all proposals of the *TeacherAgent* (concerning a given activity) were rejected by the *ScheduleAgent*. Then, the *ScheduleAgent*, has to find a place for such activity in the current timetable. Second, during the timetable maintenance phase, when the requested changes require schedule reorganization.

The list of activities that require adding, via the timetable reorganization algorithm, is based on messages received from the *TeacherAgents* and stored (by the *ScheduleAgent*) in the rejected activity list (and ordered according to their priority). The *ScheduleAgent* considers these messages one by one.

When the *TeacherAgent* T1 wants to take location that is occupied by the *TeacherAgent* T2, then the *ScheduleAgent* sends to the T2 a message with a proposal of release this location. Then, the *TeacherAgent* T2 requests a new place for its activity. If it succeeds, the *TeacherAgent* T2 accepts the proposal and the *TeacherAgent* T1 can book the requested room for its activity. If not, the *TeacherAgent* T1 has to find another place. Here, it is assumed that all *TeacherAgents* are cooperating and all have a chance to put all activities in the timetable, even in a "conflict situation". Furthermore, a simple mechanism that prevents this phase from reaching a deadlock (when all agents depend on others releasing their rooms, "in a loop") is applied by the *ScheduleAgent*. In the pessimistic situation (very difficult input dataset), activity may find no place. In that case, the algorithm cannot deal with it and the *Planner* (human being) is informed about the situation, and has to change the dataset (e.g. by convincing a teacher to change preferences) or to put the activity manually into the schedule.

On the other hand, when making changes in the existing plan (during the semester) owner of the *TeacherAgent* T2 would receive a request to accept the proposed change. Obviously, in this case, success of the schedule adjustment depends in large part on the benevolence of the involved instructors. However, let us recall that, for the time being, we assume such benevolence.

4.4 Technologies used in the implementation

The following technologies were used to implement the AgentPlanner:

– Agent platform: JADE (version 4.3.0) [7]

– MySQL database 5.1.69 [24]

– NetBeans 7.0.1 [25]

5 System testing and analysis of results

5.1 Test data

The test data used in our experiments was prepared on the basis of the actual organizational structure and room base of the Mathematics, Physics and Informatics Department of the University of Gdańsk (UG MFI). To evaluate the efficiency of the proposed method, the results obtained by the AgentPlanner were compared with these produced by the Free Timetabling Software (FET, version: 5.19.1) [2], which uses the GA. Each software solved the same timetabling problem.

The problem involved 5 days (Monday-Friday), each consisting of 6 time slots, and the building with 21 rooms, which results in a "grid" that contains 630 locations. There were 301 activity groups (62 courses, comprising total of 734 students). While it may seem that there is "a lot of space" to allocate activities, constraints imposed by the teachers and the student grouping limited this space considerably. Specifically, time preferences of 78 teachers were the main limiting factor, without it, both algorithms would find a solution without any problem (for the grid: 5 days x 6 time units x 21 rooms). Moreover, the problem is more difficult, because connections between students and groups are very complex, primarily due to the possibility of choosing elective courses. For example, a single student can have a few core courses (consisting of lectures for all students and exercises/laboratories for student groups) and (s)he has to choose a few elective courses (like facultatives, language(s), seminars, etc.). The most difficult situation involved an activity that consisted of 120 students, who belonged to 107 other activities. In this situation, the timetabling algorithms have to allocate these 107 activities in the schedule without collisions (for both teachers and students) and additionally take into account teachers time preferences.

Let us now stress that the input data and the setup of the FET system, reported in [1], was based on the *actual settings* used in the UG MFI department. Interestingly, the obtained results were not very impressive. However, after the publication, a co-author of the FET system contacted us and shared insights and advice how to setup the FET system better. Let us now make a few comments about the specific issues in experimental setup of the AgentPlanner and the FET.

First, it is important to explain, why we could not set 100% of constraints in the FET. In general, during preparation of the data for both systems (FET and AgentPlanner) we tried make them most similar. We have set up subjects, activities (with tags), subactivities, rooms (recall that we solve the problem for one department and one building only), teachers, students (assigned to activities). The AgentPlanner has been already designed to resolve the most important (hard) constraints like elimination of collisions, minimization number of days of instruction and gaps during the days. The only light constraint, we took into account, were time preferences, because they are the core of the AgentPlanner algorithms. The AgentPlanner takes into account teacher's time preferences by using ranks. In the FET we have similar option but we can only set the time units when teacher cannot lead any activity, and we have used this option. However, in the FET it is impossible to make a more specific ranking of teacher preferences.

It is worth mentioning that there is significant difference between the AgentPlanner and the FET rank function. In the AgentPlanner we can describe the time preference of teacher using sets of natural numbers: (1) $[0, d]$ where d is maximum number of days, and describe the best option for the teacher, while 0 means that the teacher absolutely cannot have an activity in that day; (2) $[0, t]$ where t is maximum number of time units during the day, and describe the best option for the teacher, and 0 means that teacher absolutely cannot have an activity in that time. In the FET, we can set the rank for only the time units, when the teacher cannot have an activity. Actually it is possible to describe the percentage value number that describes the time unit, but it is only one and we can use it to describe any time.

The next important issue is preparation of the student test data. In the AgentPlanner, we can describe students group as activity and link with it any single student that belongs to it. Due to this setup, it is very easy to detect the collision of an activity, which we try to insert into the schedule. It is possible to use similar solution in the FET. We are able to create 3 types of groups: years, groups, subgroups. In [1] we understood it literally, so set of students was divided into years (IT, Math), groups (as a lectures of subjects) and subgroups (as a parts of lectures from groups). This approach prevented linking students individually with groups and reduced the potential for effective detection of collisions. As a result, we have decided to prepare 2 types of groups, similar to the AgentPlanner: first contains activities of subjects, while in the second the students that are linked directly to these activities. As a result, the FET achieved much better results and effectively eliminated collisions between student activities. We plan to suggest this approach to the *Planner* at the Mathematics, Physics and Informatics Department of the University of Gdańsk.

5.2 Comparison metrics

Note that the AgentPlanner is being designed with focus on the "human factor" (convenience of teachers and students). In this way it differs from most approaches reported in work summarized in Section 2.1. Therefore, we have constructed a teacher and a student satisfaction functions that estimate satisfaction of this criterion. The teacher satisfaction is represented by the following formula:

$$S_T = \frac{s * 100}{a * n * m} \tag{1}$$

where: a is the number of activities of a teacher in a given semester, n is highest rank assigned to any day, m is the highest rank assigned to any time slot. Furthermore, s is the sum of evaluations of all time units of all activities of teacher, obtained by using formula 2.

$$s = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} D[i] * T[i][j] \tag{2}$$

Here, $D[i]$ represents the evaluation of each day of the schedule, while $T[i][j]$ (formula 3) represents evaluation of each of time slots assigned in the schedule of that day.

$$T[i][j] = \begin{cases} < 0, m > & \text{teacher's evaluation of time unit} \\ 0 & \text{when teacher has no activity} \end{cases} \tag{3}$$

In other words, in the numerator we represent the actual time slots assigned by the planing software, while in the denominator we represent the best potential schedule. In this way we introduce a measure that allows us to capture satisfaction of the teacher represented as a percent of the schedule that would be an ideal one.

The student satisfaction is assessed as follows. We start from 100% satisfaction and subtract: (1) 10% for one collision between the desired activities, (2) 10% for two collisions, (3) 20% for more than two collisions, (4) 10% for one extra gap between activities (we allow for one gap during a day), (5) 10% for two extra gaps, (6) 20% for more than two extra gaps, (7) 10% for one additional day (the situation when there is more days than necessary), (8) 10% for more than one additional day. Assessment of student satisfaction was conceptualized in this way, as it is impossible (at least in the current version of the AgentPlanner) to include in the process (and aggregate in some way) individual preferences of each student. While somewhat artificial, we believe that this function gives a reasonable way of assessing student satisfaction. Obviously, values 10%, 20% etc. are arbitrary ones, but they allow us to quantitatively capture the quality of schedule (seen from the student perspective).

5.3 Analysis timetable creation

Because the FET uses genetic algorithms, every generated schedule is different. Therefore, to get a reliable results of the test, we decided to run it 100 times and, in what follows, we report the best results for both teachers and students. In other words, each time we present two outcomes obtained by FET. This has to be done in this way because (in all reported cases) the best schedule for teachers is **not** the best one for students. In the case of the AgentPlanner, there is only one result (for the given set of test data). This is different than in the case of the results reported in [1], since we have resigned from the random factor used there.

In Figure 4 and Figure 5, we depict the schedule satisfaction results, for all 78 teachers, originating from the AgentPlanner and two results for the FET (the best result for the teachers and for the students). We can see that the teachers achieved higher satisfaction in the case of the AgentPlanner results. We compared the results with the average, the best, and the worst result of all test runs of the FET. The more specific conclusions follow. In Table 1 we can see

Runs	Average	Max.	Min.
AgentPlanner	98.03%	100.00%	66.67%
FET (all)	93.16%	100.00%	30.00%
FET (the best)	94.71%	100.00%	66.67%
FET (the worst)	91.73%	100.00%	50.00%
FET	93.95%	100.00%	55.00%

Table 1: Comparison of teachers satisfaction function results.

in a row: (1) results for all teachers for the AgentPlanner, (2) the average result for the FET from all runs, (3) results of the best FET run for the teachers, (4) results of the worst FET run for the teachers, (5) results of the best FET run for the students. The columns describe: (1) “Average” the number of all results of all test runs, (2) “Max.” the best result of all test runs, (3) “Min.” the worst result of all test runs. We can see that the AgentPlanner achieves better result than the best one obtained by the FET. Note that every case has 100% of maximum satisfaction, because (in the test data) there were teachers who did not define their time preferences, so they were “happy” with what they got.

When we compare Figure 6 with Figure 7 and Figure 8 we can see that in the AgentPlanner the set of satisfied teachers was 14.1% higher than the same set obtained using the FET (for the best result) and 17.95% higher than in the case of the best result (obtained by the FET) for the students.

Next, in the figures 9 and 10, we represent student satisfaction.

The diagram in Figure 10 shows the average student schedule satisfaction for the best result for the teachers. Here, the the largest group of students belongs to the interval (70%, 80%] for the AgentPlanner (27.11% of all students), and (30%, 40%] for the FET (31.88% of all students). Observe also that, in the case of the AgentPlanner, 9.95% of students belong to the interval (90%, 100%], while in the case of the FET none student was satisfied to this extent.

In the diagram in Figure 9 we can see the average student schedule satisfaction, considering the best FET run for students. Here, the largest group of students belongs to the interval (70%, 80%] for the both systems (27.11% for the AgentPlanner and 27.80% for the FET). Observe also that, in this case the FET (only) 0.14% of students belong to the interval (90%, 100%]. In Table 2 we can see in subse-

Runs	Average	Max.	Min.
AgentPlanner	73.27%	100.00%	40.00%
FET (all)	59.37%	100.00%	40.00%
FET (the best)	66.28%	100.00%	40.00%
FET (the worst)	52.21%	90.00%	40.00%
FET	60.41%	100.00%	40.00%

Table 2: Comparison of students satisfaction function results.

quent rows: (1) results of all students for the AgentPlanner, (2) average result of the FET for all runs, (3) results of the FET run that was the best for the students, (4) results of the worst run for the students, (5) results of the best run for the teachers. We can see that the AgentPlanner achieved 6.99% better result than the best case of the FET. Moreover, the difference between the best and the worst FET result is 7.16%.

In the Table 3 we depict number of gaps in students' timetables. Specifically, we depict the percentage of students who have: (1) no gaps in their schedule (the perfect situation), (2) 1 gap, (3) 2 gaps, and (4) more than 2 gaps. We present results obtained by the AgentPlanner and (as previously) two versions of the FET results (best from the point of students and teachers). In the case of the Agent-

Gaps	AgentPlanner	FET (the best)	FET
0	41.28%	33.79%	18.26%
1	28.34%	26.02%	27.79%
2	17.57%	14.17%	20.71%
> 2	12.81%	26.02%	33.24%

Table 3: Average number of gaps of students.

Planner, the biggest set of students (41.28%) has no gaps,

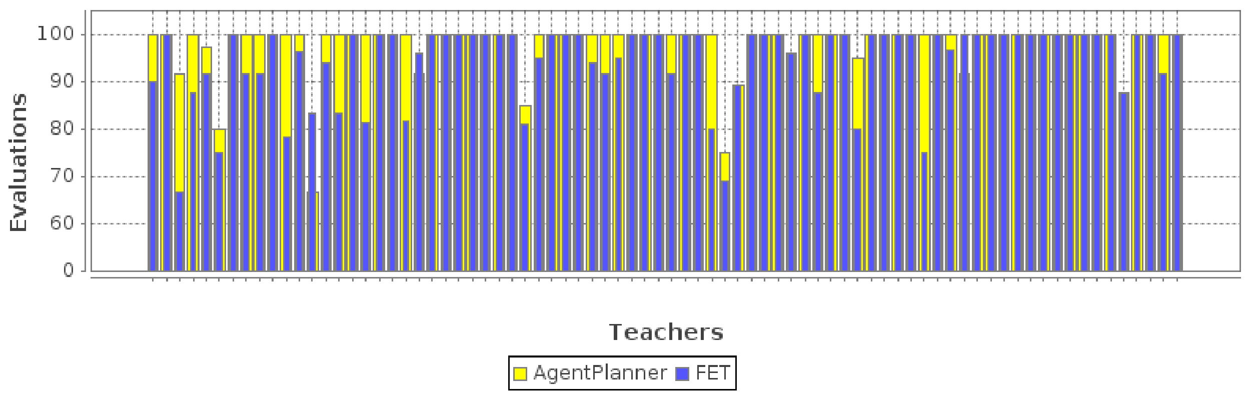

Figure 4: Satisfaction evaluations of all teachers (the best result for teachers).

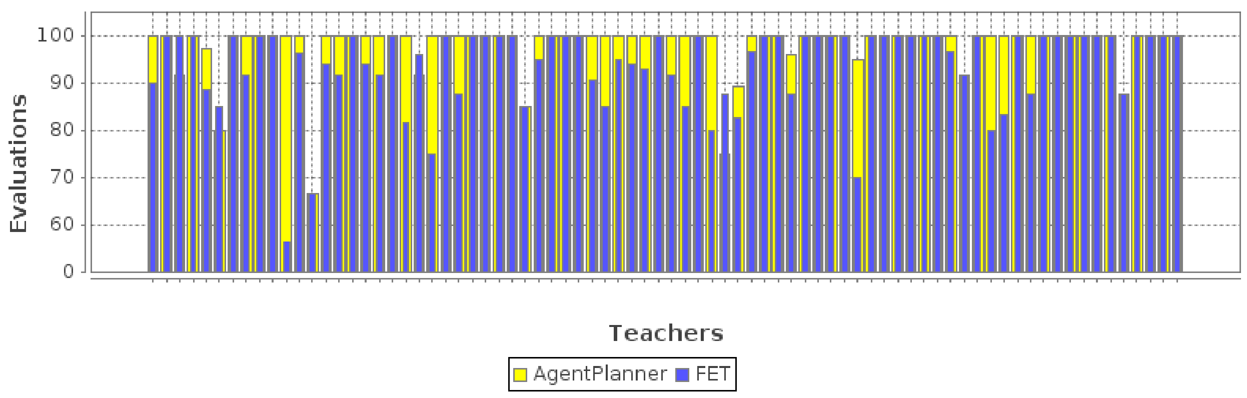

Figure 5: Satisfaction evaluations of all teachers (the best result for students).

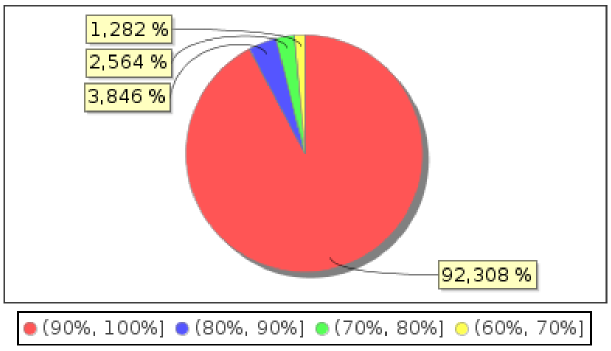

Figure 6: AgentPlanner: sets of satisfaction evaluations of teachers.

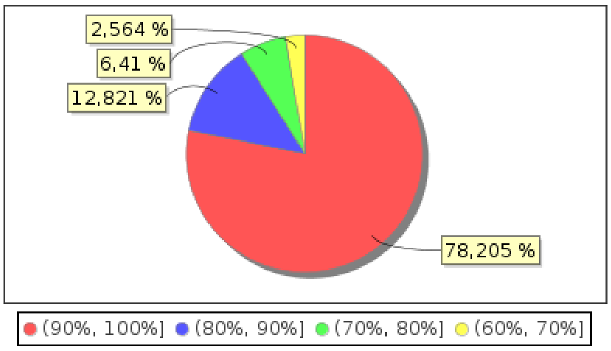

Figure 7: FET: sets of satisfaction evaluations of teachers (the best result for teachers).

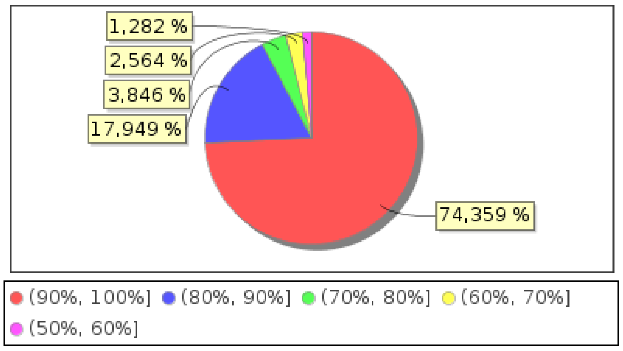

Figure 8: FET: sets of satisfaction evaluations of teachers (the best result for students).

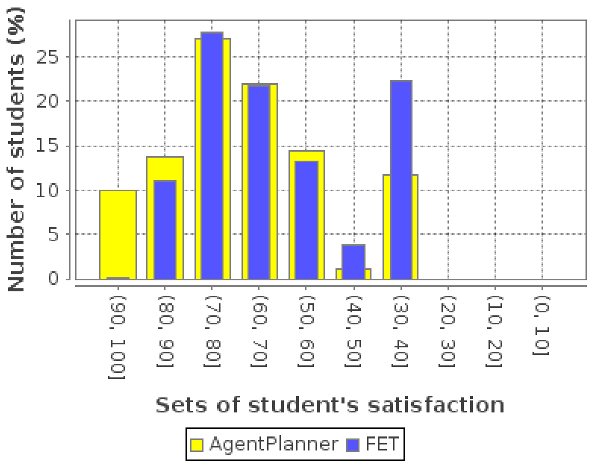

Figure 9: Average satisfaction evaluations of students (the best result for students).

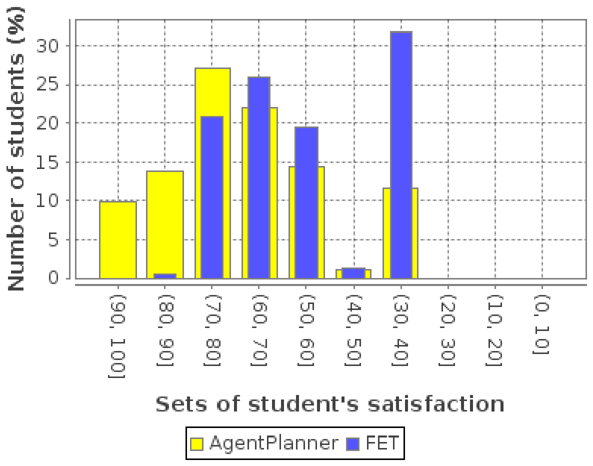

Figure 10: Average satisfaction evaluations of students (the best result for teachers).

while in the case of the "best FET" only 33.79% students reach this result, and only 18.26% for the best result for the teachers. It is very important to notice that in the case of the AgentPlanner, the number of students who have more than 2 gaps is only 12.81%, while for FET it is equal 26.02% (for the students' best result) and 33.24% (for the teachers' best result).

Table 4 captures the number of additional days (from the students' point of view). Note that multiple days of instruction are not desired (actual students of University of Gdańsk prefer to have minimal number of days of instruction as most of them are already full-time employees). Additional day is a result of a timetable "unfavorable" for the student. For example, if someone has just 4 activities during a week, the best result for her/him is a plan with all activities scheduled within a single day. But very often it is a very hard (or even impossible) task to have such schedule, because of complex data dependencies. Then, the AgentPlanner must split student's activities into more days, e.g. two (i.e. one additional day) or three (i.e. two additional days). We again consider two versions of FET results, best for the students and for the teachers.

Days	AgentPlanner	FET (the best)	FET
1	10.22%	0.14%	0.00%
2	26.02%	20.98%	6.27%
> 2	63.76%	78.88%	93.73%

Table 4: Average number of additional days of students.

Taking into account this criterion, the AgentPlanner is quite "student friendly". Specifically, 10.21% of students have schedule with just one additional day of instruction, while the same time, in the FET, practically all students have more than one additional day. The situation is particularly "bad" in the case of FET-generated schedule which is the best for the teachers. Here, more than 93% of students have schedule with more than two additional days. Because of very complex dataset, none of the student obtained the most favorable timetable.

Let us note that the situation can be even more complex in situations, which are very natural at most universities. Consider, for instance, elective courses, which can be chosen by students from various specialties, years, and even different majors. The AgentPlanner approaches this situation in a flexible way. Since the timetable is stored in the database, there is possibility of an easy (and fast) way of checking course collisions for each student (using SQL requests). Furthermore, in our approach, we can introduce the notion of collision threshold (it is an option in the prototype application, which we have explored to a certain extent, but which is out of scope of current contribution). In other words, we can specify what percentage of collisions is acceptable. Specifically, when the tolerance threshold is exceeded, a proposal to take a given slot could be reject by the *ScheduleAgent*. This notion could be very useful, particularly in the case of very complicated and difficult to schedule timetables (e.g. involving multiple departments).

Finally, it is worthy noting that we have further checked robustness of both approaches. To make the problem more complicated we have reduced the schedule grid by one time unit (5 days x 5 time units x 21 rooms), which results in a grid that contains only 525 locations. In this case, both systems produced initial schedule when they did not consider teachers time preferences. When they were considered, the FET could not find any solution. The AgentPlanner could, but the delivered timetable contained collisions in student courses. The reason is that the links within the data set were too complex and it was not possible to create an "optimal" solution (where optimal would mean that at least there were no such collisions).

5.4 Testing modification of an existing plan

To test the AgentPlanner's ability to modify an existing timetable, we have experimented with insertion of an extra teacher, who is leading three activities (a single lecture, for 60 students, and two laboratories, with 30 students in each one). Note that the timetable reorganization using the FET would involve creation of a completely new schedule. Obviously, this would be "impossible" in the real–world – as it, most likely, would destroy the whole schedule (special techniques would have to be used to minimize the propagation of changes). Henceforth, the FET was omitted in this experiment.

In general, the AgentPlanner worked well. Specifically, the timetable reorganization, caused by the insertion of a new activity, marginally affected the average satisfaction of all teachers. The average satisfaction after the reorganization was 97.72% (compared to the original 98.03%; the difference of 0.31%). The average students satisfaction, after the reorganization, was 73.27% (there was no change). It is worthy to mention that no additional collisions between the activities were generated.

6 Flexibility of the AgentPlanner

The big advantage of the AgentPlanner is the possibility of its easy modification to use in other cases of planning, e.g. business meeting, booking of meeting rooms in company, etc. The most laborious aspect would be creation of a new database that would describe the new "reality" that the timetabling is to work with (however, its overall structure will be quite similar). Furthermore, it is very likely that the evaluation algorithm would have to be adjusted (to match the nature of the problem). After such modifications, it can be postulated that "*IndividualAgents*" (instead of *TeacherAgents*) would negotiate locations in the timetable. Note that, due to the nature of the design of the AgentPlanner, use of a different database would involve only modifications in a single agent, the *DatabaseAgent* that provides the interface to the database. Let us also stress that the current (agent-based) design and implementation of the system, which is highly modular, allows relatively easy modifications (e.g. modifications mentioned in this section).

7 Concluding remarks

The aim of this paper was to discuss development and experimental evaluation of an agent-based timetabling system (AgentPlanner). The proposed system was based on assumptions originating from the *actual academic settings* (class scheduling at a department at the University of Gdańsk). The results are quite encouraging. First, the AgentPlanner outperformed the state-of-the-art timetabling software based on the genetic algorithms. Second, it is capable of satisfactorily solving the problem of schedule adjustment. Finally, it is worthy to note that even though our application area was precisely defined, we believe that our systems may be successfully applied to other timetabling problems. To achieve this status, a several actions must be done.

First of all, it is necessary to design and implement: (1) GUI for the administrator (the *Planner*), to make her/him able to manage the system (i.e. to input the data, to create/change a timetable, etc.), (2) to design and implement the GUI for for teachers (to allow them to input the data, i.e. the time preferences and, possibly, other data to be specified in the future); moreover, teachers should have access to: (i) the proposed timetable, (ii) function related to a request to change the timetable, and (iii) communication with the system, e.g. to request change of the place of the activity, (3) GUI for students as a module that providing the visualisation of timetable, (4) optional, but very helpful, would be a mobile module for the teachers. The latter one could facilitate the process of *Timetabling reorganisation* (see Section 4.3).

Second, very important issues that are needed to improve the functionality of the system are: (1) introduction of further “scheduling goals” (see Section 3), i.e. constraints for both teachers and students, (2) possibility of adding more departments and buildings (considering the location and time for change a place).

The last but not least, ways of improving the AgentPlanner’s core algorithms should be explored. At the moment, the most pressing problems are as follows. (1) To eliminate or reduce the bottleneck in the *Timetable planning algorithm* 4.2 in order to make it faster. (2) Consider ways of extending / modifying / improving the algorithm involved in asking the *TeacherAgents* to release the place in the *Timetabling reorganisation algorithm* (see Subsection 4.3). At the moment, the *TeacherAgent* T1 is asking the *TeacherAgent* T2 to release the location. Next, the T2 searches for a new one and accepts the proposal (to release the current location) or rejects it. This takes place in a single iteration and completes the process. However, it is easy to envision that if T2 would not be able to find a new place, it could ask T3 for an analogous operation. Obviously, this process could be repeated (with proper care taken to avoid an infinite loom of requests). This improvement would create more possibilities for adjusting the timetable.

References

[1] Rafał Tkaczyk, Maria Ganzha, Marcin Paprzycki (2013) AgentPlanner – agent-based timetabling system – preliminary design and evaluation, *2013 17th International Conference on System Theory, Control and Computings*, Emil Petre, Marius Brezovan, Sinaia, Romania, pp. 795–800.

[2] Liviu Lalescu, Volker Dirr, FET Free Timetabling Software, `http://www.lalescu.ro/liviu/fet/`.

[3] Timothy A. Redl, On Using Graph Coloring to Create University Timetables with Essential and Preferential Conditions, `http://cms.uhd.edu/faculty/redlt/iccis09proc.pdf`.

[4] Karl Nachtigall, Jens Opitz (2007) A Modulo Network Simplex Method for Solving Periodic Timetable Optimisation Problems, *Operations Research Proceedings*, pp. 461–466.

[5] Paweł Myszkowski, Maciej Norberciak (2003) Evolutionary algorithms for timetable problems, *Annales UMCS Informatica AI 1*, pp. 115–125.

[6] Marcin Paprzycki (2003) Agenci programowi jako metodologia tworzenia oprogramowania, `http://www.e-informatyka.pl/wiki/Agenci_programowi_jako_metodologia_tworzenia_oprogramowania`.

[7] Fabio Bellifemine, Giovanni Caire, Giovanni Rimassa, Agostino Poggi, Tiziana Trucco, Elisabetta Cortese, Filippo Quarta, Giosue Vitaglione, Nicolas Lhuillier, Jereme Picault, Java Agent Development Framework, `http://jade.tilab.com/`.

[8] Maciej Gawinecki, Minor Gordon, Pawel Kaczmarek, Marcin Paprzycki (2005) The Problem of Agent-Client Communication on the Internet, Scalable Computing Practice and Experience, 6(1), pp. 111–123

[9] Joe Henry Obit, Dario Landa-Silva, Djamila Ouelhadj, Teong Khan Vun, Rayner Alfred (2011) Designing a Multi-agent Approach System for Distributed Course Timetabling, *Proceedings of the 2011 IEEE Hybrid Intelligent Systems Conference (IEEE-HIS)*, IEEE Press, Melacca Malaysia, pp. 103–108.

`http://www.dcs.kcl.ac.uk/staff/mml/publications/assets/aamas06.pdf`.

[10] Yan Yang, Raman Paranjape, Luigi Benedicenti (2006) An Agent Based General Solution Model For

the Course Timetabling Problem, *AAMAS ’06 Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, ACM New York, New York, USA, pp. 1430–1432 .

[11] Curak Ivan (2008) Negotiatian–based multi-agent system for timetabling, *Annals of DAAAM & Proceedings*, DAAAM International Vienna, `http://www.freepatentsonline.com/article/Annals-DAAAM-Proceedings/225316139.html`.

[12] Roxana A. Belecheanu, Steve Munroe, Michael Luck, Terry Payne, Tim Miller, Peter McBurney, Michal Pechoucek (2006) Commercial Applications of Agents: Lessons, Experiences and Challenges, *AAMAS ’06 Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, ACM New York, NY, USA, pp. 1549–1555.

[13] Andrey Glaschenko, Anton Ivaschenko, George Rzevski, Petr Skobelev (2009) Multi–Agent Real Time Scheduling System for Taxi Companies, *The Eighth International Conference On Autonomous Agents And Multiagent Systems*, Budapest, Hungary, `http://www.ifaamas.org/Proceedings/aamas09/pdf/03_Industrial_Track/13_70_it.pdf`.

[14] Anja Zöller, Lars Braubach, Alexander Pokahr, Franz Rothlauf, Torsten O. Paulussen, Winfried Lamersdorf, Armin Heinzl (2006) Evaluation of a Multi–Agent System for Hospital Patient Scheduling, *International Transactions on Systems Science and Applications*, SpringerOpen, pp. 375–380.

[15] Carlos A. Coello Coello, Daniel Cort’s Rivera and Nareli Cruz Cort’s (2003) Use of an Artificial Immune System for Job Shop Scheduling, *Second International Conference on Artificial Immune Systems (ICARIS’2003)*, Springer-Verlag, pp. 1–10.

[16] Muhammad Rozi Malim, Ahamad Tajudin Khader, Adli Mustafa (2006) Artificial Immune Algorithms for University Timetabling, *6th International Conference on the Practice and Theory of Automated Timetabling*, Czech Republic.

[17] A. R. Mushi (2006) Tabu search heuristic for university course timetabling problem , *African Journal of Science and Technology (AJST) Science and Engineering Series Vol. 7, No. 1*, pp. 34–40.

[18] Houssem Eddine Nouri, Olfa Belkahla Driss (2013) Tabu search heuristic for university course timetabling problem , *African Journal of Science and Technology (AJST) Science and Engineering Series Vol. 7, No. 1*, pp. 34–40.

[19] Houssem Eddine Nouri, Olfa Belkahla Driss (2013) Distributed model for university course timetabling problem, *International Conference on Computer Applications Technology (ICCAT)*, Tunisia.

[20] Yan Yang, Raman Paranjape, Luigi Benedicenti, Nancy Reed (2005) A mobile agent system for university course timetabling, *Indian International Conference on Artificial Intelligence (IICAI-05) Pune*, India.

[21] Brian Autry, Kevin Squire (2008) University course timetabling with Probability Collectives, *The 7th International Conference on the Practice and Theory of Automated Timetabling*, Canada.

[22] Rafał Tkaczyk (2013) AgentPlanner – agentowy system zarzładzania planem zajeć, *University of Gdańsk, Gdańsk, Poland*, pp. 33–36.

[23] Foundation for Intelligent Physical Agents, FIPA Services Technical Committee, `http://www.fipa.org/activities/services.html`.

[24] Oracle Corporation, MySql Website, `http://downloads.mysql.com`.

[25] Oracle Corporation, NetBeans Website, `https://netbeans.org/downloads/7.0.1`.