AgentPlanner – agent-based timetabling system – preliminary design and evaluation

Rafał Tkaczyk
is with Systems Research Institute
of the Polish Academy of Sciences,
Warsaw, Poland
and Institute of Informatics,
University of Gdańsk, Gdańsk, Poland
rafal.tkaczyk88@gmail.com

Maria Ganzha
is with Systems Research Institute
of the Polish Academy of Sciences,
Warsaw, Poland
and Institute of Informatics,
University of Gdańsk, Gdańsk, Poland
maria.ganzha@ibspan.waw.pl

Marcin Paprzycki
is with Systems Research Institute
of the Polish Academy of Sciences
and Warsaw Management Academy,
Warsaw, Poland
marcin.paprzycki@ibspan.waw.pl

***Streszczenie*—The aim of this note is to present the initial design and preliminary evaluation of the AgentPlanner, an agent-based timetabling system. The primary advantage of the agent approach is the relative ease of schedule modification. This is particularly important in the proposed application area: scheduling university courses. Experimental results, comparing the performance of the AgentPlanner with a state-of-the-art genetic algorithms based software, obtained for a single department and a single building, are presented and analyzed.**

I. Introduction

Creating a timetable is an interesting and challenging problem. On the one hand, timetables are widely used in many application areas. On the other, timetabling is an NP-hard problem. As a result, many methods that solve this problem have been proposed. For example: graph coloring [1], simplex method [2], and genetic algorithms (GA; [3]). Since we will compare the proposed approach with the one using the GA, let us recognize that application of GA's in scheduling was discussed in detail in [4].

Regardless of how successful they are, all these methods have a major disadvantage. Namely, it is practically impossible to change an existing schedule. Observe that, when scheduling courses at a university (which is our application area) it is necessary to provide mechanisms that would allow one to to shift an individual class, add a new one into an existing schedule, (ex)change rooms, etc. Since methods like the AG treat the schedule from a "holistic" perspective, re-scheduling a single class for a teacher got sick during the semester, is not what the GA was created for. Obviously, such changes can be accomplished manually, or by using extra software, but this means that multiple approaches have to be combined. One to generate the "initial" schedule, and one to manage it. 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" they treat the complete problem at once.

Interestingly, it can be stipulated that software agents can handle *both* schedule preparation and management; as they are characterized by autonomy, reactiveness, and ability to communicate / negotiate (see, [5], [6], for discussion of application of agents in timetabling). Furthermore, as will be shown, they allow to "divide" the problem into smaller subproblems that are solved in each step; thus reducing its complexity. Therefore, we have developed a prototype of the AgentPlanner; an agent-based timetabling system, which uses agent negotiations to create and maintain (modify) the schedule. In what follows, we describe the AgentPlanner and discuss results of its initial experimental evaluation, when applied to scheduling university courses.

978-1-4799-2228-4/13/$31.00

II. Agent Planner – preliminary considerations

Let us start from discussing the requirements that have driven the development of the AgentPlanner system, which was created: (1) to develop a timetable, in accordance with specified restrictions, and (2) to manage it (be capable of making requested changes / modifications in the existing schedule). Note that the selected application area is scheduling of courses at a university, and this selection guided us in specifying functional and non-functional requirements. 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 analyzing the actual scheduling process that takes place at the University of Gdańsk, it was decided that only the *planner* (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 schedule maintenance phase).

In the initial version of the AgentPlanner we have introduced some restrictions on the implemented functions. In this way we were able to focus on core functionality and complete preliminary evaluation of our approach. Therefore, 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 our 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. It 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 possibility of implementing the AgentPlanner on mobile devices (which may be a very useful functionality for an actual system).

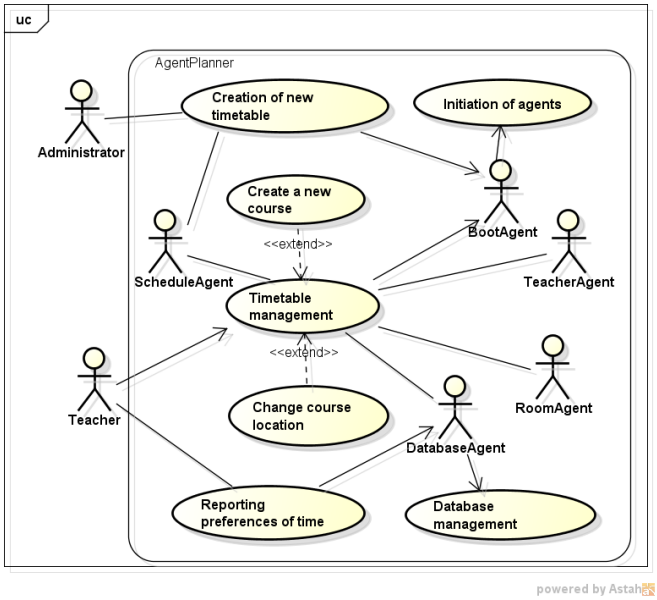
Based on conversations with faculty members of the University of Gdańsk, we have formulated the initial "scheduling goals" for the AgentPlanner. As a result, the system aims at minimizing the number of days of teaching, and at locating activities as close as possible to each other (i.e. no big gaps between activities, with some classes 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 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 have to wake up early). It has to be noted that in the current design there is no restrictions 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). While seemingly unreasonable, this reflects actual preferences of faculty members.

As far as representation of interests of students, 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 focus of teacher satisfaction.

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

A. *AgentPlanner as an agent-based system*

Recall that our decision was to develop the AgentPlanner as an agent system. On the basis of the requirements analysis, we have conceptualized it as depicted in Figure 1.



Rysunek 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 we may need to include in the design also the *student* actor, but this would lead to a problem much more complex than our current design. Analyzing functional and non-functional requirements of the AgentPlanner system, 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 communication between in the system and the timetable database.
- *RoomAgents* represent rooms in the scheduling process. They download (from the database), filter and store data about room(s) that they represent (e.g. type of the room, seating capacity, etc.). This is done to speed up the matching of activities to room(s) (data is available, instead of being continuously requested from the database).
- *TeacherAgent* acts on behalf of a teacher (both during creation and management of the timetable). It stores: information about the teacher (including personal data that has to be protected), list of activities (courses / groups taught by the teacher), list of rooms (meeting the requirements of each group; obtained from the *RoomAgent*s), results of the evaluation function (for each group), and teachers' current timetable.

- *ScheduleAgent* is the central agent of the negotiation algorithm. It "knows" teachers involved in current negotiations (*TeacherAgent*s that represent them). It has access to the timetable database (via the *DatabaseAgent*). Note that, all data concerning the currently considered timetable, is systematically 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 large scheduling problem, the *ScheduleAgent* may become a bottleneck. We have some ideas how to solve this problem. However, solving it is out of scope of the current contribution.

III. Implementation of the AgentPlanner

Based on the above considerations, we have decided that the AgentPlanner should be implemented as a client-server 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 handle only sending requests and reviewing / accessing results. This decision was based on the fact that, our software of choice, JADE agent platform does not provide a robust GUI for user interfaces. Therefore, following advice found in [9] we have decided to clearly separate the agent and non-agent functionality. Furthermore, in the initial prototype, the client application was simplified to a "line interface," while the server application has only basic functionality needed for the two timetabling operations (schedule creation and reorganization). All data needed for the tests was inserted manually to the database via SQL scripts, or other scripts written for this purpose.

A. Evaluation algorithm

The core of the timetabling mechanism is the evaluation algorithm. Here, *TeacherAgent*s evaluate locations (room information received from the *RoomAgents*) that best match the need of their teachers. The evaluation algorithm takes into account: priority of course and lecture, 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). Moreover, courses related to the major (e.g. in our case CS courses) have higher rank than non-major ones (e.g. psychology ones). 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 can be minimized. The evaluation algorithm works as follows:

B. Timetable planning algorithm

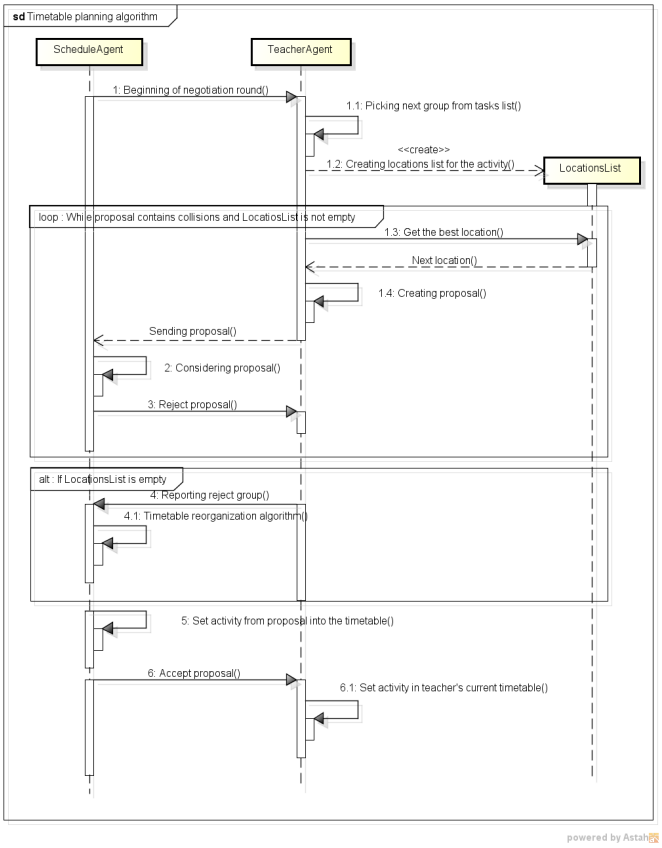
Let us now consider the scheduling of a new timetable. The approach is based on the existence of a single "judge" (the *ScheduleAgent*), having access to the current timetable

```
Data: S = priority of the course
if day_priority or time_slot_priority is equal 0 then
    do not add rooms from this time slot to the list
else
    S := S * day_priority * tiem_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 := S + random(5)
```

(which initially is empty). The *ScheduleAgent* negotiates the timetable with the *TeacherAgent(s)*, using information obtained from the *RoomAgent(s)*. The negotiations are divided into rounds (their number is not larger than the total number of all "activities" – courses / exercise groups / laboratories – of all teachers). During a single round, every *TeacherAgent* (with still unscheduled activities; note that, in a given semester, different teachers may have different number of activities), sends to the *ScheduleAgent* a message requesting a room for a selected activity. The *ScheduleAgent* considers all received requests and accepts some of them (placing these activities into the current timetable), while rejecting others (and awaiting new proposals). The decision depends on two aspects (1) is the requested location already occupied by another activity, (2) does a given request involve course collisions. Obviously, it is possible that multiple *TeacherAgents* ask for the same location. In this case, the *ScheduleAgent* selects the one with the best value returned by the evaluation algorithm. The round ends when the list of received requests is empty (all activities were scheduled), or when the unscheduled requests cannot be satisfied. At the end of a round, the *ScheduleAgent* informs involved *TeacherAgents* about its decisions. Note that, in a single round, the total number of evaluated requests is equal to the number of teachers with unscheduled activities and thus relatively small.

Observe that this approach guarantees that all teachers have the same chances, 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. Otherwise, it could be possible that all requests from professors would be processed before these of the assistants. However, it is not clear if such democratic approach will be sustainable in real-life course scheduling.

Before beginning of a next round, the *ScheduleAgent* receives messages from the *TeacherAgent*s with rejected proposals. In response it pauses the main thread of negotiations, and runs the timetable reorganization algorithm (see, the next section) to deploy the rejected activities in the current schedule. After the schedule is reorganized and, previously rejected, proposals added, the *ScheduleAgent* returns to the main thread. Thus, the timetabling algorithm continues from reception of the next group of proposals form those *TeacherAgent*s that still have unscheduled activities. The sequence diagram of the timetable planning process is depicted in figure 2.



Rysunek 2. Timetable planning algorithm sequence diagram

C. Timetable reorganization algorithm

The timetable reorganization algorithm is used in two situations. First, when the proposal of some *TeacherAgent* (submitted during a given schedule negotiation round) was rejected by the *ScheduleAgent*. In that situation, the *ScheduleAgent*, will find a place for such activity in the current timetable. The list of activities that require adding through application of the timetable reorganization algorithm is based on messages received from the *TeacherAgent*s and stored (by the *ScheduleAgent*) in the rejected activity list (and ordered according to their priority). Next, they are considered one by one. Second case for use of the timetable reorganization algorithm, is when an actual "runtime" timetable adjustment has to take place.

In both cases, the same timetable reorganization algorithm is applied, since it deals with an existing schedule, that has to be reorganized to find / adjust location / time slot of an activity. The only difference is that when a new timetable is created no human "intervention" is needed. Specifically, the timetable creation process involves agents only. For example, when *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*.

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 on the benevolence of the involved teachers.

D. 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
- NetBeans 7.0.1

IV. System testing and analysis of results

A. 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. 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) [8], which uses the GA. Each software solved the same timetabling problem. The problem involved five days, each consisting of 5 time slots, and the building with 21 rooms, which results in a "grid" that contains 525 locations. The task was to allocate 91 courses, consisting of 214 groups (comprising total of 628 students), taught by 67 teachers.

B. Comparison metrics

Note that the AgentPlanner is being designed with focus on the "human factor" (convenience of teachers and students). Therefore, we have constructed a teacher and a student satisfaction functions. Teacher satisfaction function has the form:

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

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 the following formula:

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

Here, $D[i]$ represents the evaluation of each day of the schedule, while $T[i][j]$ represents evaluation of each of time slots assigned in the schedule of that day:

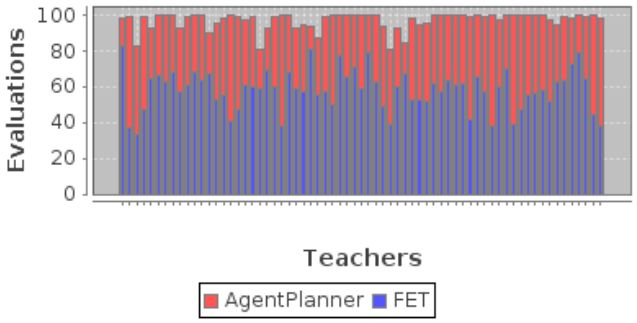
$$T[i][j] = \begin{cases} < 0, m > & \text{teacher's evaluation of time unit} \\ 0 & \text{when teacher has no activity} \end{cases}$$

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. Obtained results are the percent of satisfaction of the most desired timetable.

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 way of assessing student satisfaction.

C. Testing timetable creation

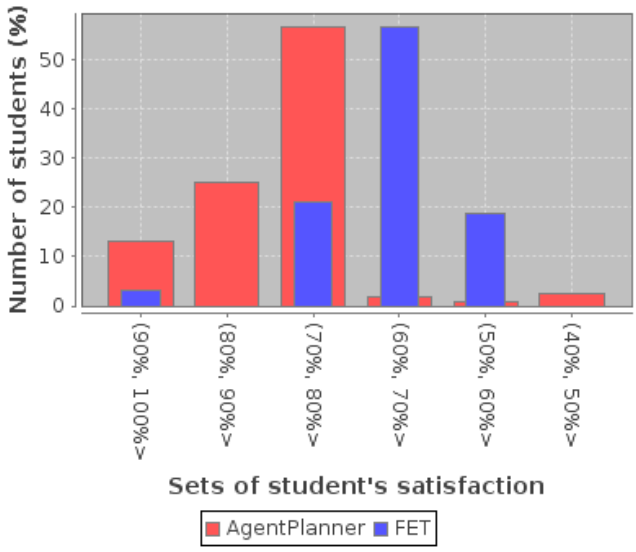
To test the AgentPlanner, and to compare it with the FET software, a total of 100 runs was completed, and in what follows we report averages for both teachers and students.



Rysunek 3. Average satisfaction evaluations of all teachers

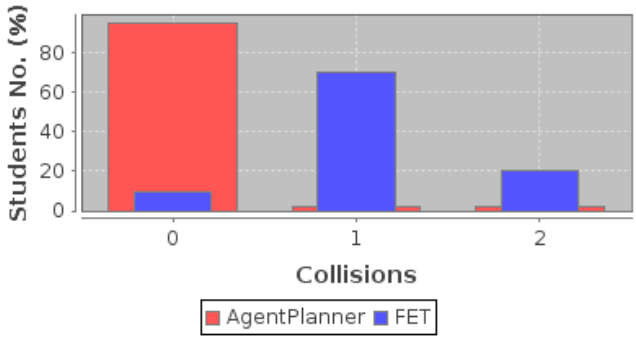
In Figure 3 we depict the average schedule satisfaction results of the AgentPlanner and the FET for all 67 teachers. Here, the schedule satisfaction obtained using the AgentPlanner was as high as 97.06%, while that obtained by the FET reached 57.32%. The smallest average value of satisfaction of a teacher was 80.6% for the AgentPlanner and 33.33% for the FET. The smallest score (the lowest satisfaction) from all trials of all teachers was 6.25% for both systems. This was caused by the schedule of the same teacher (for both scheduling systems), who had only a single activity, and had a badly assigned schedule in a specific run. Overall, an average satisfaction measure for this teacher was 99.06% for the AgentPlanner, and 60.63% for the FET.

The diagram in Figure 4 shows the average student schedule satisfaction. Here, the largest group of students (56,69% of all students, in both systems) belongs to the interval (70%, 80%] for the AgentPlanner, and (60%, 70%] for the FET. Observe also that, in the case of the AgentPlanner, more than 13.06% of students belong to the interval (90%, 100%], while in the case of the FET only 3.19% students are satisfied to this extent. Furthermore, for the FET, no student belongs to the



Rysunek 4. Average satisfaction evaluations of students

interval (80%, 90%], while 25.16% of students are satisfied to this extent as a result of scheduling completed by the AgentPlanner. The biggest disadvantage of the AgentPlanner is in the fact that 2.39% of students belong to the interval (40%, 50%], while none of them are so dissatisfied in the case of the FET.



Rysunek 5. Collisions in student schedules

Let us now look into activity collisions generated by both systems. In Figure 5 we represent number of collisions generated by each software. It is clearly visible in Figure 5) that both the AgentPlanner and the FET generated some collisions in the students timetables. In the case of the FET, this is primarily due to the fact that each activity is assigned to a group that contains a set of students. In other words, there is no way to involve individual students and their plans in the process (when using the FET software). Note that not all students have to take the same courses (e.g. in the case of electives, groups of students taking them do not correspond to the "standard" groups). In the case of the AgentPlanner, students are individually assigned to groups and thus collisions can be recognized on the individual level. This shows that it is very important to judiciously create groups and assign them to activities, because one student can be in many groups.

Let us note that the situation can be even more complicated 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. From the get go, the AgentPlanner approaches this situation in a more 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. 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 should be very useful, particularly in the case of very complicated and difficult to schedule timetables. However, in the reported tests, this approach was not applied.

The experimental results show that the AgentPlanner copes better with collisions, but is not able to completely avoid them. This is because, during the negotiations, student collisions are checked against groups that were already inserted into the timetable, but not against the remaining groups that are involved in the given negotiation step. It is difficult to solve it in easy way, because it is hard to specify where each group will ultimately be located. Some of groups can immediately get a location, which they are applying for, but some of them can be rejected multiple times and sometimes it is necessary to reorganize the timetable for them.

This problem has to be solved in the future, and this will contribute to further increase in the efficiency of the system.

D. Testing modification of an existing plan

To test the AgentPlanner modifying an existing timetable, we have experimented with insertion of an extra teacher, who is leading five activities. 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.

In general, the AgentPlanner worked well. Specifically, the timetable reorganization, caused by the insertion of a new teacher, marginally affected the average satisfaction of all teachers. The average satisfaction after the reorganization was 96,56% (compared to the original 97.06%; the difference of 1.01%). The average students satisfaction after the reorganization, was 79.69% (compared with 80.01%; the difference of 0.32%). Note that this difference was caused by a 6% increase in the number of gaps in the schedule. It is worthy to stress that no additional collisions between activities were generated.

V. 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 describes the environment of system (however, its structure will be quite similar). Furthermore, it is very likely that the locations evaluation algorithm would have to be adjusted (according to the problem). After that modifications *IndividualAgents* (instead of *TeacherAgents*) would negotiate for locations in the timetable. It may be also possible that a different database could be used (but this would require only modification of the interface). Note that, the current (modular) implementation is would allow relatively easy modification of the system.

VI. Concluding remarks

The aim of this note was to discuss development and preliminary experimental evaluation of an agent-based timetabling system (AgentPlanner). The proposed system was based on assumptions originating from 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 genetic algorithms. Second, it is capable of satisfactorily solving the problem of schedule adjustment. Here, it is worthy to note that even though our application area was precisely defined, we believe that our encouraging results indicate that agent systems may be successfully applied to other timetabling problems. In the near future we plan to address research questions outlined above.

Literatura

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

[2] Karl Nachtigall, Jens Opitz, A Modulo Network Simplex Method for Solving Periodic Timetable Optimisation Problems, In: Operations Research Proceedings 2007, 2007, p. 461-466.

[3] Maciej Norberciak, Przegląd metod automatycznego planowania – przykład wykorzystania algorytmu genetycznego w rozwiązaniu prostego problemu planowania, In: , 2002, p. 45-67.

[4] Marek Jaszuk, Zastosowanie algorytmów genetycznych do układania planu zajęć, http://www.kmis.pwsz.chelm.pl/publikacje/III/Jaszuk.pdf.

[5] Marcin Paprzycki, Agenci programowi jako metodologia tworzenia oprogramowania, 2003.

[6] Roxana A. Belecheanu, Steve Munroe, Michael Luck, Terry Payne, Tim Miller, Peter McBurney, Michal Pechoucek, Commercial Applications of Agents: Lessons, Experiences and Challenges, http://www.dcs.kcl.ac.uk/staff/mml/publications/assets/aamas06.pdf.

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

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