Multidiszciplináris tudományok, 12. kötet. (2022) 3 sz. pp. 45-55 **https://doi.org/10.35925/j.multi.2022.3.5**

MODELLING DIFFERENT ASPECTS OF FINAL EXAM SCHEDULING PROBLEMS FOR A GRAPH-BASED STATE SPACE REDUCTION METHOD

László Kálmán Trautsch
MSc student, Budapest University of Technology and Economics,
Department of Automation and Applied Informatics
1111 Budapest, Műegyetem rakpart 3., e-mail: trautschlaci@gmail.com

Szilvia Erdős
PhD student, Budapest University of Technology and Economics,
Department of Automation and Applied Informatics
1111 Budapest, Műegyetem rakpart 3., e-mail: erdos.szilvia@aut.bme.hu

Abstract

Final exam scheduling is a subtopic of scheduling where various special requirements with different levels of strictness should be managed. The formalisation of these requirements can be a challenge in itself. The number of possible schedules that should be examined is usually large, so the state space is vast. However, many of these schedules can be excluded completely at an early stage based on their inconsistency with the hard requirements, thus reducing the state space. The modelling approach for different aspects of final exam scheduling problems is presented in this paper through the modelling of an actual, complex problem. The created model was used as the basis for a state space reduction method. The results show that the method could handle all the hard and soft constraints of the problem and reduce the state space significantly based on them. The modelling approach can be used for formalising the requirements of various final exam scheduling and similar problems.

Keywords: *Operations research, Scheduling, Final exam scheduling, State space, Modelling*

1. Introduction

Most university courses end with a final examination. This is also the case at the Budapest University of Technology and Economics, Department of Automation and Applied Informatics, whose final oral examination is in the focus of this paper. Scheduling these exams is currently a manual, lengthy process, which involves taking a number of requirements into account and finding the best possible schedule. As it is done manually, there is a lot of room for error, and it is difficult to see how well a schedule has been created.

This is why there have been many attempts for automation, but the sheer number of possible schedules, the huge size of the state space and the varied and often conflicting requirements still make it a challenge today.

Final exam scheduling is a subset of the scheduling problems. Scheduling is a highly researched topic in the literature due to its NP-complete nature, but no general solution has been developed for this specific sub-task, which is final exam scheduling, due to its complexity.

Another difficulty is the precise formalisation of all the criteria. There are a number of requirements that are hard to formalise, including the management of lunch breaks for instructors and rooms, ensuring continuity and parallelism between different sessions, and minimising the number of changes in training level within a block of exams.

Also, there could be more than one acceptable schedule, but the best should be chosen from them. For this reason, in addition to the hard requirements that must be met, there are soft requirements that should be penalised when breached. The aim of scheduling is to find the valid schedule with the lowest total penalty score.

This paper seeks a solution to address the diverse requirements and to model the state space of final exam scheduling problems. Modelling the state space is useful because it can be used to obtain a simpler problem by directly reducing it, or to derive an estimate of the currently achievable penalty scores, which can be the basis for various solving heuristics and methods.

The organisation of this paper is as follows. Section 2 provides an overview of the relevant research on the problem. Section 3 introduces the background of our research, which is the idea and the main elements of our graph-based model representing the state space of scheduling problems. Section 4 describes how certain aspects of final exam scheduling problems can be formalised using our model, through the modelling of an actual problem. Section 5 shows how the modelling was evaluated using a real-world dataset. Section 6 provides concluding remarks and future plans.

2. Related work

Scheduling is a topic that has been researched for many years, as it can be used to describe a number of problems. There is a wide variety of requirements (Panwalkar et al., 1977), and the topic is still challenging even to this day, as most of the problems are proven to be NP-complete (Lenstra et al., 1977).

The problem of timetable-type scheduling has been divided into three different subtypes in the literature (Stichting et al., 1996). These are School Timetabling, which is the creation of timetables for primary or secondary schools, Course Timetabling, which is the scheduling of courses in higher education, and Examination Timetabling, which is the scheduling of examinations in higher education. Among these, Examination Timetabling is the closest to the problem of final exam scheduling, but there are still significant differences in the basic requirements, as it aims to optimise the distribution of individual exams for each student.

Since many completely different final exam scheduling problems can be formulated, the algorithms used to solve them may differ significantly and cannot be used for other problems. The problems studied by Erdős are the closest to the task explored in this paper. In her paper (Erdos, 2019), she presented simplified requirements for the problem described here and used an algorithm based on the Hungarian method for the solution. In (Erdos, 2020), she explained the difficulties of handling the time requirements of the complete final exam scheduling problem in integer programming based automatic scheduling. The state space grew to an unmanageable size in that case, and even with several weeks of runtime, it was not possible to produce a schedule that met the hard requirements of the extended problem.

In (Varela et al., 2002), a state space reduction algorithm for the job shop scheduling problem is presented. However, that is a specific subproblem of scheduling, thus their solution cannot handle the many requirements of final exam scheduling, nor the soft constraints.

In the literature, graph-based approaches have been used as a representation of scheduling problems, such as in (Cheng et al., 1996), or (Wang et al., 2014). However, in those cases, the graph is not used to model the state space.

3. Background

In our previous work (Trautsch et al., 2021), we developed the concept of reducing the state space of various scheduling tasks using graph-based models. In this paper, we use that modelling technique to create a model for the requirements of an actual, complex final exam scheduling problem, demonstrating how certain aspects of final exam scheduling can be modelled. The model is based on the methodology presented there, which is briefly summarised in this section for comprehensibility.

Scheduling problems involve associating different entities with each other according to the requirements given and finding the best possible acceptable schedule. The concept of our method is to represent in an efficient way which associations are possible based on what is known so far, and which states can be excluded completely. The possible assignments can be iterated over continuously, using various algorithms to narrow down the space of them, in theory even until the acceptable schedules are obtained. Or it can be detected if a valid schedule cannot be produced due to inconsistencies in the defined requirements and input data. In addition, other algorithms and methods can be based on the modelled state space.

3.1. The graph-based model

The possible associations that arise in the scheduling problem can be represented by a simple undirected graph, assigning a vertex to each entity, and if the current knowledge suggests that the entities may be related in some way, then by connecting their corresponding vertices with edges. Thus, the possibilities of associations are stored in the graph. In the default state of the graph, all vertices are connected to all other vertices, all associations are allowed, and the complete state space is represented.

Each vertex can be classified into different types and sets, and rules can be defined at the level of these. Such a rule might be, for example, that if no edge from a vertex of a type runs to a vertex of another type, the vertex can be deleted. Thus, the vertices of the graph are essentially subject to conditions one by one. Based on the violated requirements, operations can be performed on the graph, after which the rules of the relevant vertices can be evaluated again. This can be repeated until a reduced state space is obtained where none of the requirements are violated.

The most important operations that can be performed on the graph are the deletion of vertices or edges. By performing these, a record can be kept of whether an association cannot be made, or an entity cannot be used for scheduling. In addition, there are some associations and entities that are known to be certain to be part of the finished schedule. It is therefore possible to select certain vertices and edges of the graph, indicating that they are needed. Only selected edges and vertices are left in the graph of a fully completed schedule.

The model is also able to handle weak requirements, which specify which of several acceptable schedules is preferable. Minimum and maximum penalty scores can be assigned to each vertex, as well as rules that specify how much the minimum penalty score should be increased and how much the maximum should be decreased based on the current state of the graph. The penalty score of each vertex is used to calculate the minimum and maximum total penalty score available in the current state.

To easily define rules for multiple types at the same instance, the relations between them can be modelled as vertices of the graph. If a relationship between two vertices is to be defined, the edge between them is replaced by a new vertex, which is connected to the other two vertices. This new vertex also has a type and requirements can be specified regarding it. *Figure 1* shows how a vertex can be used to represent a relation.

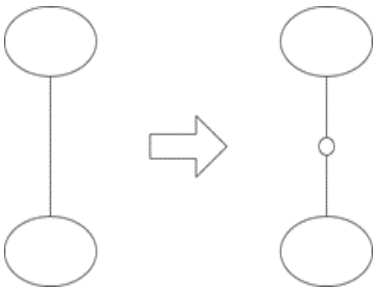

Figure 1. *A vertex representing a relation*

The relationship between three vertices can also be modelled by a vertex as shown in *Figure 2*. This can be done by modelling the relationship between each possible pair of vertices with a single vertex, adding a new vertex, and connecting it to each of the three newly modelled vertices.

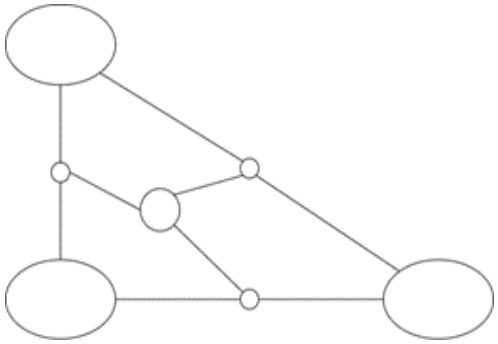

Figure 2. *Triangular relationship*

There are several types of rules that can be applied by the method, the one used in this paper is regarding the number of outgoing edges from a vertex towards the vertices of a union of given types. From a given vertex the minimum and maximum number of edges that are allowed to run to another vertex of that type can be specified. Furthermore, it can be specified how much penalty score should be given below or above a certain number of edges. The rule can be expressed in the following form: "{Minimum Edge Number Requirements} ; {Maximum Edge Number Requirements}". If there are no minimum or maximum edge number requirements, that part should be left blank. In each part, the edge numbers to be checked are listed, and it can be specified whether to delete the vertex or just to add the given penalty score in case of less/more edge numbers. "{Edge Number}:DEL" is used to indicate when a deletion is to be made, and "{Edge Number}:{Score}p" is used to indicate when the specified penalty score is to be applied for violating the rule.

Figure 3 shows a rule based on the number of edges. From each vertex of type A towards the vertices between types A and B and the vertices between types A and C there must be a total of at least 2 and not more than 6 edges. If there are less than 3 edges, the rule results in 10 penalty points, and if there are more than 5, the rule results in 20 penalty points.

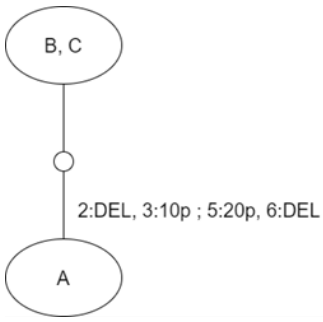

Figure 3. *An example rule based on the number of edges*

4. Formalisation of different aspects of an actual problem

This paper focuses on the final oral examination of the Faculty of Electrical Engineering and Informatics of Budapest University of Technology and Economics, which is described in the University's Study and Examination Regulations (Rektori Kabinet Oktatási Igazgatóság, 2016) and its Faculty Supplement (Sujbert, 2017). Erdős and Kővári on their GitHub page (Erdos et al., 2018) explain in detail the simplified requirements for this final exam scheduling problem, and assign penalty scores to each soft constraint. These penalty scores and requirements were used in the model, supplemented with intuitively allocated scores based on the requirements and the methodologies used in practice. There are some exponential penalty scores, ensuring that states that are further and further away from optimal are penalised increasingly.

The objective of final exam scheduling is to assign students to examination periods. Exams require instructors in different roles. It is important to make sure that each role is filled by instructors who are allowed by the rules to actually perform that role, furthermore, there are some roles that can be combined.

Exams should also be scheduled in time, which is assessed within an accuracy of 5 minutes. Examinations can be held in several rooms at the same time, but the number of these should be kept to a minimum. The exams form blocks, and the blocks form full-day sessions. The length of each of these can vary, with requirements determining how optimal these lengths are. Lunch breaks should also be included in the timetable.

The students form a heterogeneous group, which means that their major and level of education must be considered. For example, the number of changes in the level of training within a block of exams should be minimised.

There is an hourly breakdown of the times at which each instructor can be assigned to exams. Another aim is to balance the workload of instructors within each role and minimise the overall workload. There are also requirements for the continuous allocation of examination sections and instructors.

For the various acceptable but not optimal schedules, penalty scores can be used to indicate the importance of complying with each requirement. The aim is to produce a valid schedule with a minimum total penalty score.

All aforementioned requirements were formalised using the model. Some of the types required to model the problem followed directly from the task description, but in many cases, it was necessary to introduce additional types to allow more complex requirements to be specified.

The modelling approach for certain aspects of the problem is presented in this section. Each final exam scheduling problem has completely unique requirements, but these aspects often appear and need to be modelled in a similar way.

4.1. Allocation

The simplest and most general rules relate to allocations. These can be used to model how instructors participate in exams, in blocks, or in full-day sessions. Exams can be allocated to blocks, and blocks can be allocated to full-day sessions. These rules can be used to model whether participants or other sections are assigned to different sections. Penalty scores may also be given for assigning too many or too few entities to a given section. For example, the workload for trainers can be minimised this way.

The 5-minute timeslots can also be allocated to the different sections. This allows us to model the length of the sections, examine their start and end times, and assign penalty scores according to these.

Figure 4 shows an example of allocation. The rules specify that a student must take exactly one block, but the number of exams within a block can vary. There must be a minimum of 2 and a maximum of 6 students per block, and less than 3 or more than 5 students will result in 40 penalty points per block.

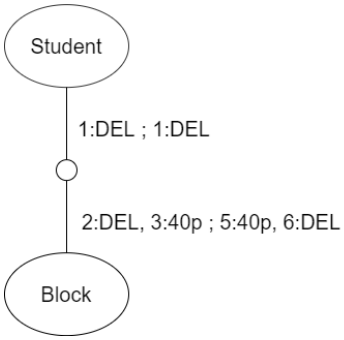

Figure 4. *Rule regarding the students within a block*

4.2. Roles

Requirements can also be formulated for different roles. These are complex rules about the sometimes abstract roles that entities play within a given unit.

These roles include the different roles of instructors, which may be, for example, president, examiner of a course, or supervisor. Instructors also have roles in blocks and full-day sessions.

Heterogeneous examinations are also handled by using roles, and requirements can be specified for the major and level of education of exams within different sections. This includes which exams belonging to a specific major can be held by each instructor.

The theme of roles even includes whether an instructor holds exams all day, and therefore needs a lunch break. This is modelled by considering the time of day as a kind of role, and based on this, if an instructor holds exams playing both the morning and afternoon roles, then he/she needs a lunch break on that day.

It can also be considered a role whether a given minute is the edge of a section. Thus, common edges can be examined for different types of sections.

When optimizing the workload of instructors, the penalty score is computed based on each role. An average is calculated along selected entities, and the deviation from this average is examined.

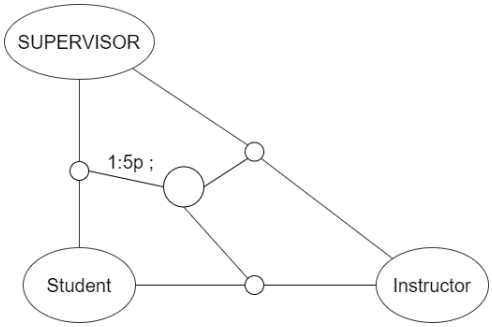

Figure 5. *Rule regarding the supervisor of a student*

Figure 5 provides an example of a rule for a role. It states that the exams of students should be preferably attended by their supervisor, and that 5 penalty points should be given if the role of the supervisor is not played by any instructor, meaning that the supervisor of the student does not attend the exam.

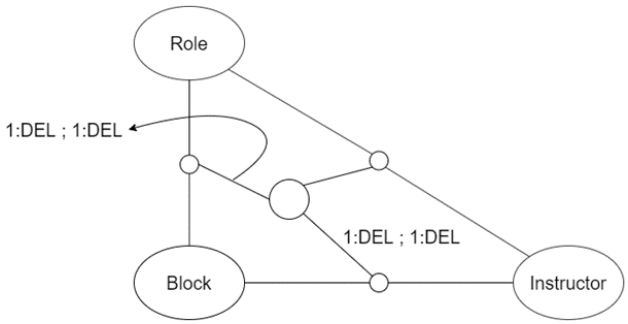

Figure 6. *Rule regarding the roles within a block*

An example of a common use case is shown in *Figure 6*. It states that an instructor should only be associated with a block if he/she fills exactly one of the roles in that block. Furthermore, a role for a block can only be kept if each of the roles associated with it is filled by an instructor.

4.3. Intervals

Requirements can also be formulated in relation to different intervals. Such intervals are exams, blocks, full-day sessions, and breaks. Also, the scheduling of instructors can also be considered as non-continuous intervals on a given day.

It can be examined whether the roles are the same throughout the intervals or whether they differ. Thus, for example, it can be inspected how much variation there is in the training level for exams within a block.

Figure 7 shows that a minute of a time interval is either between two minutes of the interval or at the edge of the interval. The "EDGE X Day" type contains entities representing the edge belonging to a particular day, and the "Day X Room" type contains the full-day examination sessions. The shown rule is important for the verification of the continuity of time intervals. For using it, it is necessary that each minute in the model is connected to all its neighbouring minutes and only to those minutes.

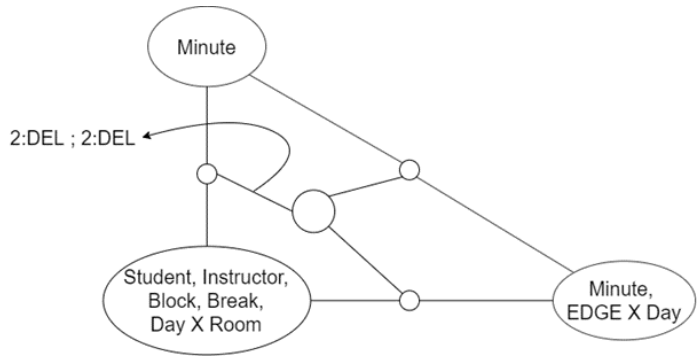

Figure 7. *The rule defining the edges of different time intervals*

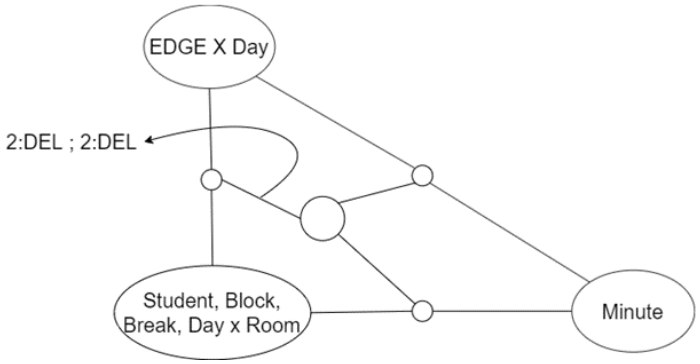

Figure 8. *The rule ensuring the continuity of different time intervals*

Figure 8 shows the rule that full-day sessions, breaks, blocks, and exams have exactly two minutes that are at the edges, ensuring that the durations of these intervals are continuous. There may be interruptions within a day in the schedule of instructors, so the type representing them is not specified in the rule.

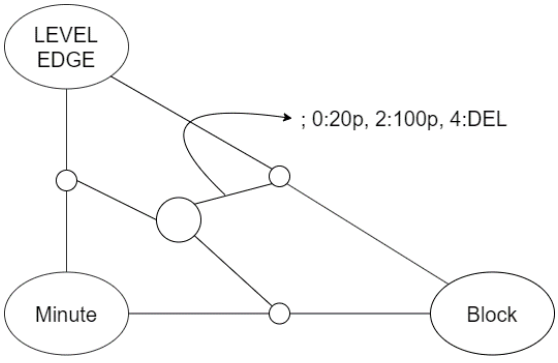

Figure 9. *Rule penalising the number of level changes within a block*

If it is not specified that an interval must be continuous, a penalty score may be given for different numbers of violations. *Figure 9* presents that the number of training level changes within a block should

be minimised. 1 change will result in 20 penalty points, 2 changes will result in 100 penalty points and 3 changes are strictly forbidden. This requirement is evaluated by examining the number of minutes within a block that lie on the edge of a continuous examination sequence of the same training level. The role of such an edge is represented by the "LEVEL EDGE" type.

5. Case Study

The created model was used as a basis of a simple state space reducing algorithm. The dataset used for testing contained information about an actual problem of final exam scheduling for a heterogeneous group consisting of 100 students of Budapest University of Technology and Economics. The timeframe of the scheduling problem was divided into 5-minute intervals and the scheduling had to be done based on these slots. The length of the exams varied, and they could be scheduled in parallel for different rooms. Thus, an exam could start in any of the 840 timeslots. 84 instructors could be assigned to the exams, with 3-6 instructors per exam. Based on that data the number of possible schedules is approximately 10^{1156}, according to *Equation (1)*.

$$840^{100} * (\binom{84}{6} + \binom{84}{5} + \binom{84}{4} + \binom{84}{3})^{100} \approx 10^{1156} \quad (1)$$

For a basic state space reduction, a C++ program was implemented. All the hard and soft constraints could be represented with our model. The program applied the constraints continuously while reading the input data, for faster execution. Running on a computer with an i5 @ 3.30 GHz processor and 16 GB of RAM, the total run time was 192 seconds, which included the data reading and the reduction of the state space.

A good measure of the size of the state space is the number of vertices, since vertices represent the different possible entities and their possible relations. For a given problem, the total number of possible vertices, which is the total size of the state space, can be calculated from the given types, their relationships, and the number of entities in the types. The number of vertices remaining after the state space reduction can be compared to this, and thus the extent to which the state space has been reduced can be examined. The test program reduced the number of vertices significantly, by 83.17% compared to the total possible number. The results are shown in *Table 1*.

Table 1. *Results of the state space reduction*

	Initial problem	Reduced state space
Number of vertices	11,311,686	1,903,694
Minimum penalty score	–	25
Maximum penalty score	–	86,502

The program also calculated a minimum and maximum penalty score for the graph. The minimum penalty score is the most important, as it indicates if certain soft requirements cannot be met with the data provided. In this case, there were 5 supervisors who were certainly unable to take the exam of their students due to the availability given.

The program also detected a discrepancy in the input data. One instructor belonged to a different major than one of his courses, thus he could never sit the exams for that course.

Similarly to *Equation (1)*, we can estimate the number of possible schedules in the reduced state space. Based on the remaining number of timeslots that can be selected from each exam, there are approximately 10^{262} ways to divide the exams into timeslots. The number of instructors that can be assigned to each exam gives that a total of approximately 10^{523} assignments are possible. Thus, the number of possible schedules after the state space reduction is approximately 10^{785} as opposed to the initial 10^{1156}.

6. Conclusions and Future Work

In this paper, it was shown how certain aspects of final exam scheduling problems can be formalised using a graph-based method, through the modelling of the complete problem of scheduling a heterogeneous group of 100 students. The method can be used to model the state space of scheduling tasks in order to provide a basis for directly reducing the state space or calculating the minimum and maximum penalty scores that can be achieved within it.

A test program was also created, in which the method was tested on the actual final exam scheduling problem. Using the model, the program was able to reduce the state space significantly and also found contradictions in the combination of the initial input data and constraints.

In the future, the model can be extended to handle additional types of requirements, and even a domain-specific language can be created to be used for any kind of scheduling tasks. In addition, it is possible to define several algorithms to further narrow down the state space, or to completely construct the schedule using heuristics or neural networks. These algorithms also include a number of optimisation possibilities to solve a given problem as efficiently as possible.

7. Acknowledgements

The work presented in this paper has been carried out in the frame of project no. 2019- 1.1.1-PIACI-KFI-2019-00263, which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.1. funding scheme.

References

[1] Panwalkar, S. S., Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, 25 (1), pp. 45–61, **https://doi.org/10.1287/opre.25.1.45**

[2] Lenstra, J., Rinnooy Kan, A., Brucker, P. (1977). Complexity of machine scheduling problems. In: Hammer, P., Johnson, E., Korte, B. & Nemhauser, G. (eds.). *Studies in Integer Programming*. Vol. 1 of Annals of Discrete Mathematics, pp. 343–362, Elsevier. **https://doi.org/10.1016/S0167-5060(08)70743-X**

[3] Stichting, C., Centrum, M., Schaerf, A. (1996). A survey of automated timetabling. *Artificial Intelligence Review*, 13 (01), **http://dx.doi.org/10.1023/A:1006576209967**

[4] Erdos, S. (2019). Algorithm based on hungarian method for final exam scheduling. *Proceedings of the Automation and Applied Computer Science Workshop*, pp. 81–89. **https://doi.org/10.26649/musci.2019.021**

[5] Erdos, S. (2020). The problems of handling time in IP-based automatic scheduling. *Proceedings of the Automation and Applied Computer Science Workshop*, pp. 107–116.

[6] Varela, R., Soto, E. (2002). Scheduling as heuristic search with state space reduction. In: Garijo, F. J., Riquelme, J. C., & Toro, M. (eds.). *Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence -IBERAMIA 2002*, Berlin, Heidelberg, pp. 815–824. **https://doi.org/10.1007/3-540-36131-6_83**

[7] Cheng, R., Gen, M., Tsujimura, J. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms–I. representation. *Computers & Industrial Engineering*, 30 (4), pp. 983–997, **https://doi.org/10.1016/0360-8352(96)00047-2**

[8] Wang, J., Fan, X., Zhang, C., Wan, S. (2014). A graph-based ant colony optimization approach for integrated process planning and scheduling. *Chinese Journal of Chemical Engineering*, 22 (7), pp. 748–753, **http://dx.doi.org/10.1016/j.cjche.2014.05.011**

[9] Trautsch, L., Erdos, S. (2021). A graph-based model for final exam scheduling. In: *Abstract book for the 17th MIKLÓS IVÁNYI INTERNATIONAL PHD& DLA SYMPOSIUM*, pp. 153.

[10] Rektori Kabinet Oktatási Igazgatóság (2016). A Szenátus X./10./2015-2016. (2016. VII. 11.) számú határozata *A BME Tanulmányi és Vizsgaszabályzatáról*. https://kth.bme.hu/document/2061/original/BME_TVSZ_2016/

[11] Sujbert L. (2017). *BME VIK BSc szakdolgozat, záróvizsga, oklevél szabályzat a BME Tanulmányi és Vizsgaszabályzatába ágyazva*. https://www.vik.bme.hu/document/1343/original/BSc-ZV-170607.pdf

[12] Erdos, S., Kovari, B. (2018). *Algorithms for final exam scheduling*. https://github.com/Bence Kovari/Schedule/blob/master/Fitness.md.