Ackermann, Christian; Hahne, Felix; Rieck, Julia

Article — Published Version

Matching and Scheduling of Student-Company-Talks for a University IT-Speed Dating Event

Operations Research Forum

Provided in Cooperation with:
Springer Nature

Suggested Citation: Ackermann, Christian; Hahne, Felix; Rieck, Julia (2022) : Matching and Scheduling of Student-Company-Talks for a University IT-Speed Dating Event, Operations Research Forum, ISSN 2662-2556, Springer International Publishing, Cham, Vol. 3, Iss. 3, https://doi.org/10.1007/s43069-022-00144-w

Operations Research Forum (2022) 3: 42
https://doi.org/10.1007/s43069-022-00144-w

ORIGINAL RESEARCH

Matching and Scheduling of Student-Company-Talks for a University IT-Speed Dating Event

Christian Ackermann[1] · **Felix Hahne**[1] · **Julia Rieck**[1]

Received: 19 January 2021 / Accepted: 19 May 2022 /Published online: 18 August 2022

Abstract
In this paper, the IT-speed dating of a German university is considered, where students have talks with different companies in order to find a suitable internship. The goal is to create a good and fair matching of students and companies for these talks, based on student preferences, and to schedule the resulting talks in order to maintain the given time horizon and minimize the necessary room changes for the students. We solved the problem in two steps. First, we modeled the matching problem as an extended version of the capacitated transportation problem and solved it using a modified stepping stone method. Second, we present two approaches to solve the scheduling problem. A Monte Carlo tree search procedure generates time-constrained schedules with minimal duration, while a genetic algorithm generates longer schedules with individual pauses and fewer room changes. The approaches led to significantly more talks with valuable content, a shorter duration, and greater satisfaction of all participants.

Keywords Matching and scheduling · Modified stepping stone method · Mixed-integer linear programming · Monte Carlo tree search · Genetic algorithm

1 Introduction

University teaching is characterized by the fact that knowledge is mainly imparted on a theoretical basis. It is therefore essential to apply the acquired basic knowledge in real economic situations at an early stage, i.e., during studies. Particularly,

✉ Christian Ackermann
ackermann@bwl.uni-hildesheim.de

Felix Hahne
hahne@bwl.uni-hildesheim.de

Julia Rieck
rieck@bwl.uni-hildesheim.de

[1] Institute for Business Administration and Information Systems, Operations Research Group, University of Hildesheim, Samelsonplatz 1, 31141 Hildesheim, Germany

students of *computer science* benefit from a short-term work experience in practice. For this reason, a 10-week internship is a fixed part of the two IT bachelor's degree programs in Information Systems and Information Management & Technology (IMIT) at the University of Hildesheim. In order to help students to find an internship company suiting their interests, the "IT-speed dating" was introduced in 2013. Following the idea of well-known speed dating events to find a (life) partner, the then 29 participating students conducted 5-minute one-to-one talks ("dates") with all 15 participating companies according to a round robin procedure. During these talks, it was possible to find out whether the students' qualifications, expectations, and preferences matched the offers and requirements of the companies. The speed dating was followed by a classic application phase, in which the students sent a written application to the companies they found suitable. Questioning of the company representatives showed that positive impressions from the speed dating greatly increased the chances of the application being accepted. This approach proved to be very successful over the years: On average more than 80% of the students carried out an internship in one of the companies.

With increasing numbers of students and companies, the duration of the event became longer and longer, with about 3–4 hours to plan. Moreover, feedback surveys from both students and companies revealed that a certain percentage of dates were unsuccessful ("empty date"). With these empty dates, it became clear within the first 30 seconds that the type of internship offered did not meet the wishes of the respective student. For example, a company only offered internships in the field of ERP systems, while the student was interested in improving the software development skills.

As a consequence for the planning for the IT-speed dating 2020, for which about 40 students and 40 companies were expected, the organizers decided to stop the round robin procedure. A new concept was required that could meet the time limit of 3–4 hours. In addition, a minimum talk duration of at least 5 minutes should be ensured and the students should be offered a talk with as many companies they are interested in as possible. We implemented a procedure that aims to reduce the total number of talks by preventing the occurrence of "empty dates" as far as possible. For that, students state preferences regarding the participating companies which are used in a *matching* algorithm to determine the final dates. Afterward, an acceptable time *schedule* has to be established to provide the students with an individual, conflict-free sequence of dates. Please note that the final decision on which companies students apply to and which companies offer internships to which students will still not be made by the organizers, but will be left to the participants.

The aim of the paper is to provide an interesting real-life application of a many-to-many matching problem with the consideration of fairness in the area of university education. Moreover, the solution of the matching problem is used as input for a scheduling problem in order to determine a concrete timetable for all participants. The focus of the paper is on the implemented solution procedures, which are rather conventional methods. We show how to incorporate fairness considerations in the matching procedure and how to find good solutions for the scheduling problem, for which even feasible solutions are hard to find. Section 2 explains the details of the new concept and the circumstances at the University of Hildesheim. Section 3

provides an overview of related work regarding matching and scheduling. Section 4 describes the first step of our proposed decomposition approach, where the *matching* problem is used to assign every student to a subset of attending companies. The problem is modeled as a transportation problem and solved with a modification of the stepping stone method to incorporate fairness. Section 5 describes the second step of the approach. Here, a *scheduling* problem is considered, where a scheme is determined which student, when, and in which room meets a company. We propose two different heuristic methods, one is a Monte Carlo tree search procedure and one is a genetic algorithm. Section 6 compares the results of both heuristics with the results of an exact solver for different problem sizes. The corresponding case study of the IT-speed dating event 2020 is then described further and the results are discussed. Finally, conclusions are given in Sect. 7.

2 Concept and Circumstances

In order to offer the students dates with companies that have a good chance of being of interest to them, information about companies' offerings and students' preferences had to be collected in advance. The companies were asked to briefly describe the thematic orientation of their internship. The descriptions received were then handed out to the registered students. We removed all references to the associated company from the descriptions so that the preferences were given purely on the basis of the content offered and not on the basis of the company's reputation. Our aim was to bring less known companies that offer interesting tasks into the focus of the students. Motivation for that can be found in feedback surveys of former IT-speed dating events, which showed that more than 90% of the students agreed to the following statement: "Some companies I was not interested in beforehand have offered me interesting internships." Based on this feedback, the students were invited to classify the descriptions into "$\mathcal{A}$ – very interesting," "$\mathcal{B}$ – medium interesting," and "$\mathcal{C}$ – not interesting."

With this information, the task was to compute a *matching* of students to companies with the following *objectives*:

- The matching should be done in such a way that each student meets as many of the $\mathcal{A}$-ranked companies as possible and afterward as many $\mathcal{B}$-ranked companies as possible.
- Even though the speed dating event is an optional offer to the students to increase their chances of a successful internship, fairness regarding the final matchings should be considered in order to provide equal opportunities. Students who receive just a few talks with companies in their $\mathcal{A}$-category could feel disadvantaged when there are other students with significantly more $\mathcal{A}$-talks. To ensure objective comparability regarding the quality of the matching among all students, we decided to fix the number of companies per category. With that, the matching should be conducted such that each student obtains approximately the same number of $\mathcal{A}$- and $\mathcal{B}$-assignments as everyone else, as long as this does not significantly deteriorate the overall solution quality. Other approaches based on a

ranking list created by each student were also considered but finally disregarded because of the increased effort for creating such a ranking compared to the simple classification.

After the matching, the *scheduling* of the identified talks has to be performed. In the original round robin procedure, there was a fixed sequence of the tables and after each talk, every student just moved to the neighboring table to continue with the next talk. This allowed minimum transition times between talk rounds thus maximizing the productive utilization of the total time horizon. This simple method is no longer applicable if every student has an individual schedule. In order to still utilize as much time as possible for interesting talks, the transitions to the next talk should be optimized. Due to its size, the event has to take place in four neighboring seminar rooms. Even with prepared overviews of the room layouts, it takes some time to find the right company when students have to change rooms. Therefore, the minimization of necessary room changes is a major objective in the scheduling algorithm. Other potential and additional objectives are discussed in Sect. 5.

Please note that we consider maximizing the quality of the matching as our primary goal, since the event should bring students and companies together in the best possible way. *Based* on the best possible matching, the best possible schedule should *then* be determined.

3 Related Work

The problem at hand can be divided into a matching problem and a scheduling problem. In the following Sects. 3.1 and 3.2, we present related work to both subproblems.

3.1 Matching

The matching problem under consideration is characterized by two groups: students and companies, and each person of a group may perform talks with many persons from the other group. In our case, the number of talks is predefined. Moreover, only the group of students (one side) is allowed to express preferences. In general, the problem is known as a *one-sided many-to-many matching* problem.

There are a lot of papers in the literature that consider the one-sided *many-to-one* matching problem. In [1], students with preferences are distributed to high schools through a lottery. Each student can go to one school at most and each school can only offer a limited number of places. In order to solve the problem, two mechanisms are used: "partitioned random priority" and "partitioned random endowment." Kesten et al. [2] provide possibilities for efficient lottery design on the basis of "random serial dictatorship" (RSD) and "probabilistic serial" (PS). In the RSD mechanism, a random sequence of agents is selected and in this sequence they are allowed to take an item [3]. In the PS mechanism, a probability is first determined on the basis of preferences, which is then used to give each agent a certain item [4].

Kesten et al. [2] use RSD and PS to ensure fairness through stochastic components. Furthermore, Duan and Pettie [5] study the many-to-one matching problem from a graph-theoretical point of view by searching for maximum and minimum matches in bipartite graphs.

In the *many-to-many* matching problem, each item may be assigned multiple times and each agent may receive multiple items. In [6], the problem is considered by designing a dictatorship mechanism for course assignment in Harvard so that students can no longer benefit from giving wrong preferences. Sotomayor [7] investigates the existence of stable many-to-many matchings. Baïou and Balinski [8] study the transferability of properties (e.g., fairness, truthfulness) of one-to-one and many-to-one problems to many-to-many matching problems. They analyze the polygamous variant of the stable marriage problem and define explicitly the number of matches per participant. Unlike in our case, however, preferences are considered from both sides.

Often, matching mechanisms are examined to determine whether they have certain characteristics such as *fairness*, *truthfulness* (also known as strategy-proofness), and *Pareto efficiency*. Since we seek a fair matching result that cannot be heavily influenced by the students through strategic decisions, these features also play a role in our application. RSD, for example, is ex-ante fair with respect to equal treatment of equals (though agents far down in the random order feel disadvantaged), but it is only truthful and ex-post Pareto efficient if there are at most as many items as agents [9]. PS is also ex-ante fair, nobody prefers the chances of another over his own (stochastic dominance envy-free) and Pareto efficient, but not truthful either. Zhou [10] proved that there is no mechanism for the one-sided one-to-one matching problem, which is both symmetrical (equal treatment of equals) and ex-ante Pareto efficient as well as truthful. For more detailed studies on truthfulness and incentives in matching mechanisms, see [11].

Fairness can be judged by different criteria. For example, a single individual can consider an assignment as fair on the basis of maximin-share, proportional fair-share, min-max fair-share, envy-freeness, and competitive equilibrium from equal incomes [12, 13]. Besides individual fairness criteria, there are also global fairness criteria such as the Nash Social Welfare [14]. The Nash Social Welfare can be calculated from the product of all individual scores (see Eq. (3)). If this is maximized, a Pareto efficient solution is automatically created. In addition to criteria that are either fulfilled or not fulfilled, there are also key measures that indicate how fair (balanced) a certain allocation is. Here, the Jain index [15] is often used, which brings the coefficient of variation (ratio of standard deviation to arithmetic mean) to an easily interpretable scale from 0 to 1. A value of 0 (although impossible) would represent a completely unfair, and a value of 1 a completely fair (balanced) allocation. If the items to be distributed are divided equally among k% of all participants, while the remaining participants receive nothing, the Jain index is k%. Concrete consideration of fairness in optimization problems can be found, for example, in [16], where fairness is taken into account when creating timetables for academic courses and both the max-min fairness (maximize the currently worst individual score) and the Jain index are maximized. In [17], the Jain index is also used to generate an examination schedule for students that is as fair as possible.

Besides the lack of research in the area of one-sided many-to-many matching problems with a fixed number of matches per person, a strict preference order is generally assumed. We do not have this order of preference, because we only have 3 different preference values (for classes $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$). Please note that a random transfer of the classes into a strict ranking could lead to undesirable results. For this reason, we have developed our own matching approach. We will first present our approach and then discuss the actual characteristics. It was particularly important to us to achieve a solution that is globally as good as possible, but in which individual students are not seriously disadvantaged by chance.

3.2 Scheduling

Related problems to our scheduling task can be found in the area of sports scheduling, where match schedules for sports leagues have to be created. Many sports scheduling problems have in common that the schedule is based on a round robin-like matching. In European sports leagues, a double round robin procedure is often applied. Some American leagues implement this procedure as well or have a combination of multiple smaller round robin match plans possibly extended by a few special matches (e.g., the American MLB and NHL). Nemhauser and Trick [18], for example, consider the scheduling of a college basketball league. Relevant constraints include sequences of home and away matches, combinations of location (home/away) and match day (weekend/during the week), stadium availability, home match preferences on certain match days, and the avoidance of consecutive strong opponents. While the match plan follows some kind of round robin procedure, schedules can be created in a two-step procedure, where first feasible combinations of specific patterns (e.g., sequences of home and away matches) are created and afterward each team is matched with one of those patterns. The first step ensures feasibility regarding the general restrictions like conflict freedom while the second step ensures feasibility and/or improves the objective function regarding individual preferences of the teams. Examples for round robin based scheduling problems can be found, for example, in [19–24] as well as in [25, 26]. For a general overview of sports scheduling, we refer to [27]. In the problem at hand, finding an initial feasible solution is harder, since the creation and combination of patterns have to be made together with the assignment of the participants, as not each pattern is applicable for each participant. This is caused by the fact that each participant has its own subset of "opponents" which is not rule-based as it would be in a (multiple) round robin-based procedure. Please note that there is a difference between time-constrained and time-relaxed schedules. While in time-constrained schedules there are exactly as many match days as matches per team, in time-relaxed schedules there are more match days than matches per team. This leads to additional rest days for some of the teams and could result in additional constraints regarding these rest days. An application of a time-relaxed scheduling problem with further restrictions can be found in [25].

Another related problem is conference scheduling, where the talks at conferences are assigned to time slots and rooms. Usually the talks are grouped into sessions, which consist of consecutive talks in the same room. A typical goal is to maximize

the attendance at all talks or to maximize the preferences given by the participants in advance regarding the talks to be attended. Since the individual sessions are partly held in parallel, talks that are jointly preferred by many participants should be assigned to the same session. Vangerven et al. [28] present an integer programming model for the problem. In subsequent steps, the availability of presenters is taken into account and the so-called session-hopping (changing the session, i.e., changing the room) is minimized. Other approaches are introduced in [29, 30], and [31]. Conference scheduling problems differ from our problem in that we have one-to-one conversations, where attendance is mandatory and not the primary optimization goal.

4 Matching Problem

Let S be the set of students and C be the set of companies. Matching is based on the assumption that $s = |S|$ students and $c = |C|$ companies meet. Each participant (students as well as companies) should conduct exactly m talks; m is a predetermined parameter which depends on the duration of each talk and the time limit for the duration of the IT-speed dating event. As a subset of companies should be determined for each student, we set m smaller than the number of companies involved ($m < |C|$). The goal in the model formulation and the exact solution method is to maximize the sum of all students' preferences for their m dates taking place (see Sect. 4.1). Please note that the preferences are specified based on the three categories $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ that the students have used for the companies. We explain our strategy for specifying the concrete preference values in the case study section (Sect. 6). To generate solutions that are fair with respect to the assignment of students to companies, an adopted version of the solution algorithm is implemented. In Sect. 4.2 both methods are compared.

4.1 Model Formulation and Solution Method

The matching problem at hand is modeled and solved as a variant of the *capacitated transportation problem* [32]. Each student has a supply and each company a demand of m talks (units). From the experience of the last years, it is known that the number of interested students usually exceeds the number of participating companies. Therefore, the sum of all supplies ($s \cdot m$) is greater than the sum of all demands ($c \cdot m$) and a fictitious company $|C| + 1$ must be introduced to cover the missing demand ($(s - c) \cdot m$). This makes the fictitious company the only "participant" with more than m talks. The (transport) connections $\langle i, j \rangle, i \in S, j \in C$, are valued with preference values p_{ij}, where p_{ij} indicates the preference of student $i \in S$ for company $j \in C$. Values for p_{ij} could be chosen, e.g., from the intervals [0, 1] or [0, 100] with $\mathcal{A}$-preferences having a higher value than $\mathcal{B}$- or $\mathcal{C}$-preferences. We set the minimum capacity to $\lambda_{ij} := 0$ and the maximum capacity to $\kappa_{ij} := 1$ for all arcs $\langle i, j \rangle$. Connections to the fictitious company $|C| + 1$ obtain a weight $p_{i,|C|+1} := 0$ as well as capacities $\lambda_{ij} := 0$ and $\kappa_{i,|C|+1} = \infty$. Using the decision variables

$$x_{ij} = \begin{cases} 1 \text{ if student } i \text{ meets company } j \\ 0 \text{ otherwise,} \end{cases} \tag{1}$$

we are able to formulate the objective function to be maximized, in which the preferences that need to be fulfilled are included:

$$\text{Max} \quad Z = \sum_{i \in S} \sum_{j \in C} p_{ij} x_{ij}. \tag{2}$$

Due to the modeling, a feasible solution consists of $m \cdot s$ student-company-combinations (matches). However, based on the assumption that $s > c$ holds, only $m \cdot c$ of these matches represent real conversations, while the other combinations consist of a student and the fictitious company. A scheduled meeting with the fictitious company can be seen as a break time for the student.

For the problem sizes we consider, the resulting mixed-integer linear program can easily be solved optimally with a standard solver like CPLEX. However, the current objective function (2) only maximizes the overall preferences across all students. A solution with many $\mathcal{A}$-talks for one student and no $\mathcal{A}$-talks for another student is as good as a solution with both students having the same amount of $\mathcal{A}$-talks as long as the sum of fulfilled $\mathcal{A}$-talks stays the same. Consequently, for a balanced and fair solution, it makes sense to also investigate the individual preference fulfillment score z_i of each student $i \in S$, which can be calculated according to:

$$z_i = \sum_{j \in C} p_{ij} x_{ij}. \tag{3}$$

As described above (cf. Sect. 3), there are different ways to measure the fairness of an allocation. The often used Jain index becomes larger (and therefore better) the more uniform the individual scores are and thus correlates with the variance. Hence, to ensure that no student is disadvantaged, a solution should be found, where the Jain index of the student scores $z_i, i \in S$, is as big as possible or rather the variance of the scores is as small as possible. For this purpose, one could determine all optimal solutions of the problem and search for the most balanced among them. In preliminary studies, when applying the method to practical problem instances, we found out that the number of optimal solutions is very large. In addition, the variances of the solutions with respect to the individual preference scores given in (3) were quite similar and not as small as desired. Thus, we have implemented an adapted variant of the *stepping stone* method – an exact procedure with polynomial runtime for solving the capacitated transportation problem – in order to actively reduce the variance between the student fulfillment scores during optimization of the overall objective function (2).

In order to be able to use the *stepping stone* method [33], a feasible start solution must first be generated. We implemented an adapted version of the *column maximum* method. Here, the companies are considered in a fixed order and in each iteration, one of the students with the highest preference is assigned to the current company. Afterward, the number of pending talks of students and companies is reduced by one. This procedure has some parallels with the RSD

mechanism. If a student is assigned to m companies, the corresponding student is deleted and not considered for the next iterations. Similarly, a company with already m assigned students is not considered any further. The process is repeated until m different students are assigned to each company. Students with less than m talks meet the fictitious company in their remaining conversations. Once a start solution is generated, the standard stepping stone method would search for exchange sequences for assigned conversations that improve the overall solution. If it is no longer possible to exchange student-company-matches without reducing the overall objective function value of (2), an optimal solution is found.

In our *modified method*, dynamic preference values $f_i \cdot p_{ij}, i \in S, j \in C$ are used instead of always the same preference values p_{ij}. In each iteration, the dynamic fulfillment factor f_i is recalculated. The goal of dynamic preferences is to avoid below-average and above-average fulfillment scores. Suppose two students i_1 and i_2 meet the companies j_1 and j_2, respectively, and both students have classified j_1 as $\mathcal{A}$ and j_2 as $\mathcal{B}$. Who meets which company has no influence on the objective function value in (2), since in both cases one $\mathcal{A}$- and one $\mathcal{B}$-wish is fulfilled. However, it makes sense to fulfill the $\mathcal{A}$-wish of the student who currently has a lower score (say i_1) in order to reduce the difference between the two scores. For this reason, the $\mathcal{A}$-wish of the student with the lower score receives a higher dynamic preference value ($f_{i_1} > 1$) and the $\mathcal{A}$ -wish of the other student receives a lower one ($f_{i_2} < 1$). In general, let $\bar{z}$ be the average student score and V_z the variance of student scores. Factor f_i can then be calculated as follows:

$$f_i = \begin{cases} \tilde{f}_i & \text{if } z_i < \bar{z} \\ \frac{1}{\tilde{f}_i} & \text{otherwise,} \end{cases} \tag{4}$$

where $\tilde{f}_i = 2\frac{\delta_i}{V_z}$ if $\delta_i > \alpha \cdot V_z$ and 1 otherwise; with $\delta_i = (z_i - \bar{z})^2$. Parameter α is a threshold factor, which increases starting from 1 in order to ensure that a *stable state* is reached with increasing run time.

Please note that the dynamic preferences are not transferred to the next iteration. New values for f_i are determined in each sequence exchange step. Furthermore, it should be noted that the resulting solutions do not necessarily have the optimal objective function value, but are often nearly optimal, as practical studies have shown. Accordingly, approximately equivalent solutions with significantly lower variance are searched for in the direct neighborhood of an optimal solution. In order not to be limited to the neighborhood of the initial optimal solution, a multi-start procedure with a fixed number of runs using a random sequence of students and companies is applied. Each run ends in a stable state resulting in a near-optimal solution (regarding the total sum of preference fulfillment). After all runs of the multi-start procedure, the solution with the lowest variance is selected. This solution represents a very fair, balanced solution with nearly optimal sum of fulfillment scores.

4.2 Comparison of the Exact and Modified Method

An analysis of the relationship between the improvement of the variance and the decrease of the objective function value was made. On average, the variance of the solutions found by the modified method was 75% (73%) lower than the mean (smallest) variance of optimal solutions (an optimal solution). For all tested instances the resulting Jain index was well above 0.99, indicating a very fair distribution. In addition, as part of the variance minimization, the Nash Social Welfare was also increased compared to an optimal solver solution. Depending on the concrete fulfilment of the students' $\mathcal{A}$-, $\mathcal{B}$-, and $\mathcal{C}$-wishes, the objective function value of the heuristic was on average between 0.05% and 0.35% worse than in the optimum. Based on the evaluation, this corresponds to the fulfilment of a $\mathcal{B}$- instead of an $\mathcal{A}$-wish for one in 80 students or the fulfilment of three $\mathcal{C}$-wishes instead of three $\mathcal{B}$-wishes for one in 42 students. In our opinion, this reduction in the assignment quality is acceptable, as it allows for much fairer student-company-matches. It must also be taken into account that due to the different number of students and companies ($s > c$), the number of talks with real companies per student can vary. This aspect also increases the need for a balanced, fair allocation.

Our main goal was to find the best possible global solution. This is supported by our approach, as it basically maximizes the sum of all fulfilled preferences. If the maximum possible sum is reached, the resulting assignment is automatically *Pareto efficient*. A solution is only not Pareto efficient if some students could swap companies and thus none is worse off, but at least one is better off. However, this would necessarily increase the sum of the fulfilled preferences, which contradicts the assumption that the sum is already maximum. In order to achieve an even and *fair* distribution of the student scores, we try to maximize the Jain index without deviating significantly from the maximum sum (and thus from a Pareto efficient solution). The above–mentioned results of our research show that this has been successful. In general, our approach is *not truthful*, as it can theoretically allow students to gain an advantage by wrongly stating priorities. Particularly, this is possible if a student prefers a company that is not preferred by the majority of other students. In this case, it is not necessary to waste a high priority on this company, because it would be assigned to him/her even if the priority is lower. However, we believe that due to the small number of preference classes (only $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$), a strategic devaluation of a company by one level is already too risky for the students. Additionally, the students receive no reliable information about the preferences of their fellow students before submitting their own preferences, thus further reducing the potential advantage.

5 Scheduling Problem

The solution to the matching problem specifies which students will meet which companies. Based on this assignment, a *schedule* must now be generated that shows students when they will visit a company. In addition, the schedule has to determine which company will be placed in which of the available rooms. Furthermore, a feasible solution to the scheduling problem must ensure that all scheduled meetings take place exactly once, that no company or student has more than one talk during a time slot, and that

the given room capacities are not exceeded. As mentioned before, the minimization of room changes will maximize the utilization of the available time because unnecessary walks, searches, and confusion can be reduced. Therefore, the primary objective function used in the model (see Sect. 5.1) and in the solution methods (see Sects. 5.2 and 5.3) is the minimization of the total number of room changes. Alternate or additional objectives are discussed in Sect. 5.4.

5.1 Notation and Model Formulation

The scheduling problem assigns the predetermined $m \cdot c$ (real) conversations of students and companies to n time slots. Moreover, the companies are placed in the existing $|R|$ rooms. The associated model can be formulated as a mixed-integer linear program considering the following sets, parameters, and variables (each sorted alphabetically). Please note that the decision variables $x_{ij}, i \in S, j \in C$, of the matching problem are input parameters (indicators) to the scheduling problem.

Sets and indices	
$i \in S$	set of students, $S = \{1, \ldots, s\}$
$j \in C$	set of companies, $C = \{1, \ldots, c\}$
$r \in R$	set of rooms, $R = \{1, \ldots, \|R\|\}$
$t \in T$	set of time slots, $T = \{1, \ldots, n\}$
Parameters	
κ^r	capacity of room r
x_{ij}	indicator, showing if student i meets company j (value 1) or not (value 0)
Decision variables	
Π_{it}^r	$= \begin{cases} 1 \text{ if student } i \text{ is at time } t \text{ in room } r \\ 0 \text{ otherwise} \end{cases}$
s_{ijt}	$= \begin{cases} 1 \text{ if student } i \text{ meets company } j \text{ at time } t \\ 0 \text{ otherwise} \end{cases}$
Θ_{it}	$= \begin{cases} 1 \text{ if student } i \text{ has to switch rooms between time } t \text{ and } t+1 \\ 0 \text{ otherwise} \end{cases}$
y_j^r	$= \begin{cases} 1 \text{ if company } j \text{ is assigned to room } r \\ 0 \text{ otherwise} \end{cases}$

$$Min. \quad \sum_{i \in S} \sum_{t \in T \setminus \{n\}} \Theta_{it} \tag{5}$$

$$w.r.t. \quad \sum_{j \in C} s_{ijt} \leq 1 \qquad \forall i \in S, t \in T \tag{6}$$

$$\sum_{i \in S} s_{ijt} \leq 1 \qquad \forall j \in C,\ t \in T \tag{7}$$

$$\sum_{t \in T} s_{ijt} = x_{ij} \qquad \forall i \in S,\ j \in C \tag{8}$$

$$\sum_{r \in R} y_j^r = 1 \qquad \forall j \in C \tag{9}$$

$$\sum_{j \in C} y_j^r \leq \kappa^r \qquad \forall r \in R \tag{10}$$

$$\sum_{r \in R} \Pi_{it}^r = 1 \qquad \forall i \in S,\ t \in T \tag{11}$$

$$\Pi_{it}^r \geq s_{ijt} + y_j^r - 1 \qquad \forall i \in S,\ j \in C,\ r \in R,\ t \in T \tag{12}$$

$$\Theta_{it} \geq \Pi_{it}^{r'} + \sum_{\substack{r \in R \\ r \neq r'}} \Pi_{i,t+1}^r - 1 \qquad \forall i \in S,\ r \in R,\ t \in T \setminus \{n\} \tag{13}$$

$$\Pi_{it}^r \in \{0, 1\} \qquad \forall i \in S,\ r \in R,\ t \in T \tag{14}$$

$$s_{ijt} \in \{0, 1\} \qquad \forall i \in S,\ j \in C,\ t \in T \tag{15}$$

$$\Theta_{it} \in \{0, 1\} \qquad \forall i \in S,\ t \in T \setminus \{n\} \tag{16}$$

$$y_j^r \in \{0, 1\} \qquad \forall j \in C,\ r \in R \tag{17}$$

Objective function (5) minimizes the sum of the room changes for all students. Constraints (6) and (7) ensure that no student or company has more than one conversation at a time. Equalities (8) guarantee that all planned conversations take place exactly once. Moreover, constraints (9) indicate that each company is assigned to exactly one room and constraints (10) satisfy that room capacities are maintained. In order to ensure the correct assignment of the students' variables to the rooms, each student must be in exactly one room at all times (11). If a student has a talk at time t, the student must be in the same room as the company with which the talk is being held at that time (12). Constraints (13) guarantee the correct counting of the room changes that occur when a student in time slot $t + 1$ is in a different room than in time slot t. Finally, constraints (14) to (17) impose the domains of decision variables.

In what follows, we present two heuristic approaches in order to solve the underlying problem. We will compare both approaches with each other as well as with the usage of a standard solver regarding quality and runtime. With the first approach, schedules can be generated that require exactly m time slots, so that each company has a conversation at any time. In the following, we will refer to these schedules as *time-constrained* schedules (see Sect. 3.2). Please note that due to the assumed oversupply of students, students will have individual break times if a conversation with the fictitious company is scheduled for a time slot. Our second approach generates schedules with n ($n > m$) time slots, as does the presented model (5)–(17). In addition to the breaks due to the fictitious company, there are now $n - m$ individual break times for all participants, in which past conversations may be reflected upon and coming conversations may be prepared. The more time slots can be used, i.e., the longer the underlying time horizon is, the fewer room changes are necessary, as the conversations can be better coordinated. Please note that at least $\lfloor m / \max_{r \in R} \kappa^r \rfloor$ room changes occur in a feasible schedule. In the following, we will refer to these schedules as *time-relaxed* schedules. Due to the design of the second approach, it is practically impossible to determine a schedule with a duration of $n = m$ time slots.

5.2 Approach for a Time-Constrained Schedule

The first approach is able to create time-constrained schedules directly and quickly. Thereby, the determination of a time-constrained schedule results from the proof of its existence. For the sake of simplicity, we divide the previously introduced fictitious company $|C| + 1$ into several fictitious companies $c + 1, c + 2, \ldots, s$, so that there are as many companies as students. The conversations of the fictitious company $|C| + 1$ are distributed among the new fictitious companies in such a way that each fictitious company has exactly m conversations with different students (each real company also has m conversations). The question is whether, for s students and s companies, each assignment to m different talk partners can be made in such a way that for each student there is a possible sequence of conversations in which everyone has exactly one conversation at any given time.

In order to answer the question and construct a corresponding solution, the problem is modeled as a bipartite graph analogous to a matching problem, with two sets of nodes representing the s students as well as the s real and fictitious companies. For each time slot $t = 1, \ldots, m$, perfect matchings in the graph-theoretical meaning are sought, whereby no two edges may have a common node and exactly one edge of each node is included in the matching. The edges identify the student-company-combinations for the respective time slot.

From the condition of Hall in the context of the marriage theorem, it can be deduced that a bipartite graph with two disjunctive node sets S and C with $|S| \geq |C|$ has a perfect matching exactly when the following condition is fulfilled [34]:

$$\forall H \subseteq S: \quad |N(H)| \geq |H| \tag{18}$$

In (18), $N(H)$ is the neighborhood of the nodes in H.

A subset H consisting of h students has $h \cdot m$ not pairwise disjunctive companies in the neighborhood (every node has a degree of m). Each company may appear a maximum of m times in a feasible schedule, i.e., there must be at least h different companies in the neighborhood of h students. Therefore, the condition (18) is fulfilled for the student-company-combinations and a perfect matching can be determined. Once such a perfect matching is found, all contained edges are removed from the graph, since we assume that the corresponding conservations have taken place at $t = 1$. What remains is a bipartite graph in which each node now has a degree of $m - 1$ instead of degree m, thus guaranteeing the existence of another perfect matching. By induction, it can be concluded that all conversations can take place in m rounds without violating the condition.

For the generation of a complete schedule, a perfect matching must be identified in the bipartite graphs of the different time slots. Galil [35] describes several possibilities for determining a perfect matching, including the variant of adding a super-source and a super-sink and solving a *maximum flow problem*. To do this, all edges (both the existing ones and the newly introduced ones) must have a maximum capacity of 1. Then, a maximum flow can be calculated with the algorithm of Ford & Fulkerson [36]. The (original) edges used in a flow are the conversations to take place in the current time slot. It can be shown that the number of perfect matchings in a graph with node degree m is at least $(m!)$. Thus, the number of possible complete, time-constrained schedules is at least $((m!)!)$.

Since the time slots are planned independently of each other in our time-constrained method, it is difficult to optimize the room changes. However, in order to obtain a good schedule with regard to room changes, a procedure based on *Monte Carlo tree search* (MCTS, [37, 38]) was implemented. MCTS estimates the "win probability" by simulations for each position (room allocation), in order to identify the best moves at the root node and all child nodes. The estimated probability should be close to the "true" probability to ensure the moves selected are good moves toward winning the "game." The tree to be considered has m stages, whereas on the kth stage each node represents a partial schedule, with the first k slots already scheduled. Child nodes (at the $(k + 1)$th stage) differ from their parent nodes (at the kth stage) in that a further time slot has been added and a schedule for $t = 1, \ldots, k + 1$ student-company-combinations has been created. To ensure that the tree contains all possible schedules, parent nodes at stage k would have to consider all possibilities for the time slot $t = k + 1$. Since k conversations per participant have already been determined, $m - k$ more must be regarded, so that there are at least $(m - k)!$ possibilities for the time slot $t = k + 1$ (see above). For each possibility, a child node (at stage $k + 1$) would need to be created. In order to reduce the computational complexity, only a random subset of b possible child nodes is generated (filtered beam search). Figure 1 illustrates the concrete procedure.

In Step 1, an initial allocation of the companies to rooms is made. The aim of Step 1 is to accommodate companies that appear together in schedules of many students in the same room. This should enable students to meet several companies in one room before they have to visit another room. For implementation, a similarity matrix is constructed, where the similarity between two companies indicates how many students both companies visit. Based on this similarity matrix, the companies are then divided into $|R|$ clusters, where cluster i has size κ^i. In

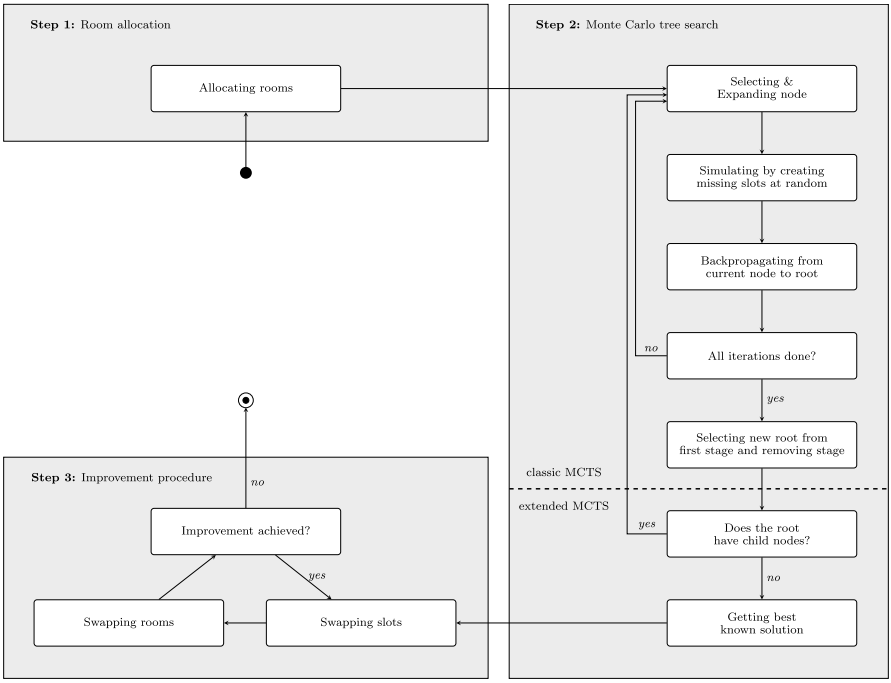


Fig. 1 Monte Carlo tree search-based procedure for creating time-constrained schedules

each step, the company with the lowest average similarity to all others is determined and assigned to an empty cluster. Then, the most similar $(\kappa^i - 1)$ companies are assigned to the same cluster. This procedure is repeated until all companies are positioned in clusters. Since different room sizes are available, it is relevant in which order the rooms are filled. Therefore, the clustering is performed for all possible sequences of rooms. At the end, the allocation with the highest average similarity within a cluster is chosen. This allocation remains identical during the Step 2 of the MCTS.

In Step 2, the core of the MCTS is executed. This step consists of the following sub-steps (cf. Fig 1):

2.1 In each iteration, *upper confidence bounds applied to trees* (see [38]) are used to determine the currently most promising node, where the expected "win" is maximized with respect to a randomly drawn belief. If the most promising node has less than b child nodes, then it is *selected* for expanding. A child node is added randomly. This is done by scheduling a subsequent time slot based on the Ford & Fulkerson algorithm, which is implemented as a breadth-first search based on the Edmonds Karp variant [39, 40]. The trick here is to add the predecessor and successor of a node in *random* order to the queue, resulting in a different maximum flow and a different perfect matching (and thus a different child node).

2.2 Starting from the newly inserted child node up to a leaf of the tree (or up to the "end of the game"), a *simulation* is performed, adding more time slots randomly according to the same scheme until the schedule is complete (i.e., until the schedule contains student-company-combinations for m time slots). The resulting number of room changes of the complete schedule is transformed to the interval [0, 1] using a sigmoid function, whereby fewer room changes lead to values close to 1 (indicating a "win"). If the simulation results in a better solution (according to the objective function "minimization of room changes") than previously known, the solution is stored.

2.3 Finally, a *backpropagation* is performed and the determined value of the new child node is carried back to the root. The backpropagation changes the node values, which have an influence on the selection of the most promising node in the next iteration.

The whole process is executed for a certain number of iterations or for a certain time. Afterward, the most frequently visited node (usually "the best") is selected on the first stage of the current tree, supposing that time slot $k \in \{1, \dots, m\}$ was planned. In classic MCTS this "move" would now be played and the opponent would continue. In our extended MCTS, we consider the student-company-combinations specified for $t = 1, \dots, k$ as fixed and start planning the next time slot $k + 1$. Therefore, we set the selected first stage node as new root node. Starting from the new root node, again an enumeration tree is built, where the number of stages is reduced by 1. The entire MCTS is terminated when a node with m planned time slots is set as the root. In this case, the root has no child nodes at which the approach can be continued. It is important to note that this final schedule does not have to be the overall best (according to the objective function) schedule found during the search. Whenever the search finishes one stage and a new time slot has to be fixed, the MCTS selects the partial solution created until now with the highest probability of producing high-quality solutions later in the search. Afterward, the search will focus on this part of the solution space. Due to this sampling-based greedy technique, it is possible that previous solutions with other schedules per slot than the finally fixed ones have better objective values. Therefore, each new best solution found is stored and in the end, this overall best solution found is returned as the result of Step 2.

Step 3 is used in order to *improve* the solution obtained with a comprehensive 2- and 3-swapping method. First of all, it is checked whether *swapping* 2 or 3 *time slots* leads to an improvement. This is repeated until no further reduction of the room changes is possible. Then, the room allocation is checked for an improvement. The procedure is the same as for the exchange of 2 or 3 time slots. If the *room swapping* has led to an improvement, the swapping of time slots is checked again. In case neither of the two procedures results in an improvement, the current solution is taken as the final result.

Please note that the assignment of companies to rooms is made initially in Step 1, is kept fixed during Step 2, and is just checked for improvement in Step 3. The reason for this is that – according to preliminary experiments – finding a suitable schedule during Step 2 seems to have a far greater influence on the number of room

changes than the initial distribution of companies to rooms. This might be influenced by the still relatively high number of company visits per student and the dissimilar voting behavior of the students. Furthermore, the time slots to be fixed by the search are selected based on the *current* room assignment. Changing the room assignment during Step 2 would change the quality of the already fixed slots and would therefore be in conflict with the main idea of the MCTS.

5.3 Approach for a Time-Relaxed Schedule

Our second approach is based on the idea to interpret the problem as a modification of the open shop scheduling problem (OSSP). In the OSSP (see, e.g. [41]), a given set of jobs must be processed on each of the existing machines in an arbitrary order. The goal is to determine the time at which each job should be processed on each machine such that an objective function is minimized, e.g., the makespan. In order to apply the problem to our scheduling problem, we associate the students with jobs and the companies with machines.

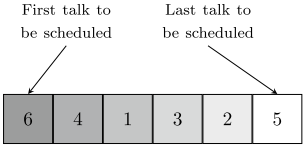
Due to the current problem characteristics, each job (student) can only be performed on a specific machine (can only meet a specific real company). The duration of each job (student) is one time unit (one time slot) and the capacity of the machines (companies) is 1 to ensure that each company has a maximum of one talk at a time. The aim is to find a schedule for jobs (students) on machines (at companies) so that the makespan is equal to the number n of possible time slots, while minimizing the number of room changes for the students.

The problem is solved using a *genetic algorithm*. The choice of the genetic algorithm has been inspired by the recent promising results provided by [42] for the OSSP. We adapted their approach to fit our problem characteristics. A genetic algorithm provides a stochastic optimization method that attempts to simulate Darwin's theory on natural selection. The method iteratively applies certain operators to a population of solutions so that on average each new population (i.e., generation) tends to be better than the previous one, according to a predefined fitness criterium. The fitness function measures the fitness of an individual to survive in a population. The algorithm seeks to maximize the fitness function. The termination criterion is usually specified as a fixed number of generations, or an execution time limit. At the end, the best solution found is returned. In our variant of the genetic algorithm (GA), the representation of an individual is divided into two parts: On the one hand, the resulting *schedule* is considered, and on the other hand, the allocation of *rooms* for the companies is taken into account.

For the schedule, we use a specific numbering in the representation in order to be able to quickly understand the respective student-company-matching (cf. [42]). The first assigned company of student 1 gets the number 1, the kth company of student 1 gets the number k, and the first company of student 2 gets the number $m + 1$, and so on. Thus, if the number x is given, it refers to the student $i = \lceil \frac{x}{m} \rceil$ and the $(x \mod m)$th assigned company. All numbers are stored in a one-dimensional array with $s \cdot m$ elements. Every element is assigned a number from the set $\{1, \ldots, s \cdot m\}$, which corresponds to the student-company-matching. A permutation of the numbers

Students	Companies to talk with (talk no.)
S_1	C_1 (1), C_2 (2)
S_2	C_1 (3), C_3 (4)
S_3	C_2 (5), C_3 (6)

(a) Student-company-matching



(b) Representation of a non-optimal individual

	$t=1$	$t=2$	$t=3$
S_1	C_1	C_2	
S_2		C_3	C_1
S_3	C_3		C_2

(c) Resulting schedule

Fig. 2 Example for representation of schedule chromosome and resulting schedule

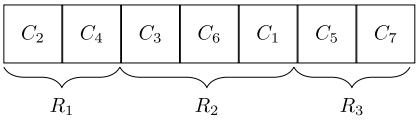
now represents a different solution. The order of the numbers indicates the order in which the student-company-matchings are scheduled in the current timetable (when decoding the representation). Each combination is scheduled at the earliest possible time, taking into account the students' preferences regarding companies, each student can meet only one company at a given time t and each company can only be assigned once at time t. Figure 2 shows an example with $s = 3$ students and $c = 3$ companies, with $m = 2$ meetings per student. Based on the special numbering from 1 to $s \cdot m = 6$ of student-company-matchings in (a), a schedule representation (chromosome) of a non-optimal solution is given in (b). When decoding the representation, we obtain the timetable in (c), where the assignments are made at the earliest possible times; the length of the resulting time horizon is $n = 3$.

The room allocation is also represented as a one-dimensional array. The array contains all c real companies and as many placeholders as additional "seats" are available in the rooms ($\sum \kappa^r - c$). The decoding of the representation is done very easily. The first κ^1 entries (companies or placeholders) are assigned to room 1, the next κ^2 entries to room 2, and so on. Figure 3 shows the procedure for 3 rooms with specific capacities (see (a)) and 7 companies contained in array (b) in random order. The resulting allocation is given in (c).

The fitness function value of one whole individual (consisting of schedule and room-allocation representation) depends on the phase the GA is currently in. In the first phase, the algorithm only searches for a feasible solution. For this purpose, the fitness function value is geared exclusively toward minimizing the makespan (i.e., the fitness that has to be maximized is equal to the negative makespan). However, in order to speed up this procedure even further, the fitness value includes not only the current makespan but also the average of the times of the conversations taking place. The idea behind this is based on the fact that schedules with the same makespan may have a different distribution of conversations. Schedules with overall earlier conversations then have gaps in later periods. For these solutions, it may be

Room	Capacity
R_1	2
R_2	3
R_3	2

(a) Room capacities



(b) Room chromosome

Room	Companies
R_1	$C_2,\ C_4$
R_2	$C_1,\ C_3,\ C_6$
R_3	$C_5,\ C_7$

(c) Resulting allocation

Fig. 3 Example for representation of room chromosome and resulting room allocation

sufficient to reschedule a single late conversation to shorten the makespan. These schedules should be preferred, since they have a greater potential for minimizing the makespan.

If a solution with a maximum of n time slots could be found, the fulfilment of the corresponding maximum makespan becomes a necessary condition for the feasibility of the individuals. Then, the second phase of the genetic algorithm begins, in which the fitness function value now aims at minimizing the number of necessary room changes.

Genetic operators such as selection, crossover, and mutation are used in order to create the next generation. The selection of the individuals is done by a *tournament selection*, whereby β individuals are randomly selected from the population and the two best (according to the fitness) are chosen as parents. Furthermore, ordered one-point and two-point crossovers are applied as crossover operators. In one-point crossover, a cutting point is determined and the part before the cutting point of the first parent is transferred to the child. The missing genes (elements with positions) are transferred in the order in which they appear in the second parent. In two-point crossover, two cutting points are determined and the middle part of the first parent is passed on to the child. Missing genes are transferred to the child according to their occurrence in the second parent (cf. Fig. 4). The crossover operators are applied to both the schedule and the room-allocation representation, the choice of the operator is random.

The mutation occurs according to two of the four mutation operators presented in [42]. In the displacement mutation, two random positions are selected and the corresponding genes are swapped. In shift mutation, two random positions are selected and all genes in between are shifted one position to the right, with the last gene inserted at the beginning of the genes to be shifted (Fig. 5). The randomly selected mutation operator is applied only to the schedule representation, only to the room representation or to both.

Once λ children are generated, the next generation consisting of μ individuals is built. When selecting individuals for the next generation, the individuals are sorted by descending fitness values and the best 20% are chosen. The remaining individuals are taken randomly from the current population and identical individuals are excluded. The above steps are repeated until a stopping criterion is satisfied.

For the IT-speed dating 2020, we have tested different sizes for n. Already with the instance size of $m = 22$ conversations with $s = 42$ students and companies each (cf. the case study), the described method was not able to generate a schedule with less than $n = 25$ slots in an acceptable time (i.e., less than 10 minutes).

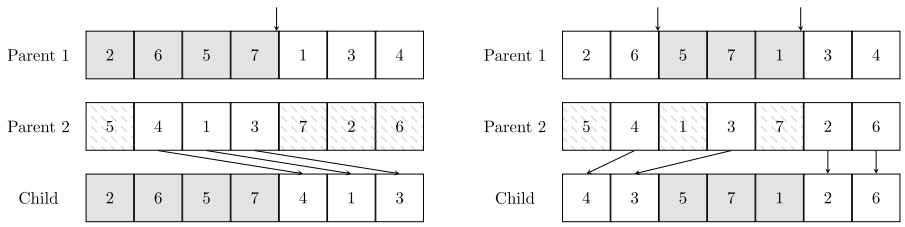


Fig. 4 Example for one- and two-point crossover

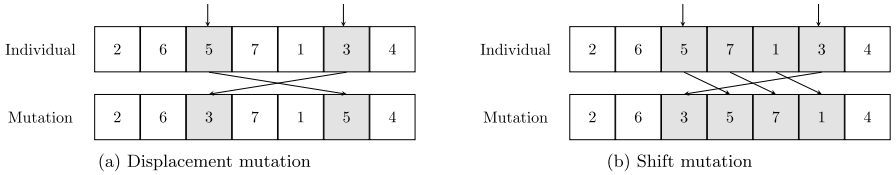


Fig. 5 Example for displacement and shift mutation

5.4 Algorithm Discussion

As explained previously, we used the minimization of the total number of room changes as the primary objective in the modeling and both algorithms. However, additional objectives could be taken into account. While the current version just minimizes the total number of room changes, their distribution among the students is not considered. Similar to the matching problem, fairness metrics might be integrated to balance the number of necessary room changes for each student. Additionally, the distribution of breaks during the event could be considered as part of the objective function. Breaks can occur for two different reasons: First, when a talk with a fictitious company is scheduled and second, when the GA is used and the number of available time slots is bigger than the number of talks per participant. While the total number of free slots and thus the total duration of the breaks per participant is specified by the matching algorithm and the given parameters for the GA, the distribution of these breaks is determined by the scheduling algorithm. Multiple short breaks evenly distributed over the time horizon might be desirable to reflect past talks. A few very long and/or unevenly distributed breaks might be evaluated negatively by the participants. Therefore, different metrics like the variance describing the distribution and length of the breaks could be integrated in the evaluation of the individual schedules. Here, fairness among participants could be considered as well.

The integration of the proposed aspects into the objective function for the algorithms is rather straightforward. The main challenge now is to decide on the most suitable way of dealing with multiple objectives. The simplest way is a weighting of the different aspects, where the choice of the weights plays a decisive role. A lexicographic ordering can be implemented in the same way. Also, a *Pareto front* may be created by storing dominant solutions, but there would be a final selection criterion needed to choose among the *Pareto optimal* solutions. An integration into the mathematical formulation of the problem, however, is more complicated, since most of the desired behaviors would usually be modeled with non-linear functions.

A possible area of future research may be a more problem-specific modification of the MCTS and the GA. In the MCTS, candidates for new slots are created randomly, both at the current stage and during the simulation. The implemented Edmonds Karp algorithm could be enhanced by a problem-specific heuristic to increase the likelihood of good slots. This would decrease the exploration,

but increase the exploitation. In the GA, we already implemented a two-phase approach with changing objective functions in order to accelerate the creation of feasible solutions. Additional improvements are possible by guiding the GA toward promising solutions through the use of special crossover and mutation operators. By integrating problem-specific knowledge, the random selection of crossover and mutation points could be replaced by a heuristic selection of promising points in order to keep beneficial parts and destroy bad parts of a solution (see, e.g., [43]).

6 Case Study and Algorithm Comparison

The *speed dating* 2020 of the University of Hildesheim took place in January with $c = 34$ companies. The interested students received six weeks in advance a list with the short descriptions (max. 1000 characters) of the internships offered by the companies (e.g., in the field of ERP systems, software development or marketing). It was specified that each student has to select 6 $\mathcal{A}$-, 16 $\mathcal{B}$-, and 12 $\mathcal{C}$-companies. As a compromise between fulfilling as many wishes as possible, maintaining an appropriate length of conversations, and adhering to the maximum time horizon of 4 hours, 22 rounds of talks, each lasting 5 minutes, were initially planned. From a group of 60 interested students, $s = 42$ participants were selected according to the progress of their studies. The most popular company received an $\mathcal{A}$-wish from 22 participants, so that with $m = 22$ talks, all $\mathcal{A}$-wishes can usually be satisfied.

In what follows, we first describe the evaluation of the algorithms. Afterward, we present details regarding the realization for the IT-speed dating event 2020 as well as the feedback received from the participants. Please note that the presented algorithms were not supposed to directly improve the quality of the event compared to the last years but were necessary to implement the concept change. Therefore, we are just able to evaluate the quality of the new concept compared to the round robin procedure and the quality of the algorithm results compared to the specified constraints.

6.1 Algorithm Evaluation and Comparison

A performance analysis was conducted to evaluate the algorithms. All experiments were performed on a *Dell OptiPlex 5050* with *Intel Core i7-6700* 3.4 GHz processor and 64 GB RAM. In order not to rely on the only real data set, the experiments were (exclusively) performed with numerous synthetic data sets. These were generated by assuming that there are different classes of companies whose offerings are similar to each other and differ greatly across classes. In addition, different classes of students were constructed, who had special preferences regarding the classes of companies. In combination with an individual random preference for each company, a preference of each student in each class could be generated for each company in each class. Subsequently, m of the companies were randomly chosen, with the selection probability being proportional to the individual preference of the student. The

selected companies were sorted by preference in descending order. The first companies were assigned to class $\mathcal{A}$, the other selected ones to class $\mathcal{B}$, and the unselected ones to class $\mathcal{C}$. A comparison with the real data set showed that this approach reflected the voting behavior very well.

First, the matching algorithm with 20 iterations of the multi-start procedure was executed. The minimization of the variance has been configured such that it is followed until a stable state is reached. Thus, each iteration lasted about 5 seconds. As intended, the simulated students showed a very heterogeneous voting behavior. In total, each company was preferred by 11 to 35 students ($\mathcal{A}$- or $\mathcal{B}$-wish). With this condition, all $\mathcal{A}$-wishes could be fulfilled. In addition to the $\mathcal{A}$-wishes, an average of 55% of the $\mathcal{B}$-wishes (between 6 and 12 out of 16) received attention.

After the matching, the scheduling and room allocation were carried out. For the event, $|R| = 4$ rooms with capacities $\kappa^i \in \{7, \dots, 10\}, i = 1, \dots, 4$, tables (one per company) were available. In addition to the Monte Carlo tree search (approach for a time-constrained schedule) and the genetic algorithm (approach for a time-relaxed schedule), a solver solution was used for the cases of $n = 22$ and $n = 27$ time slots. Thereby, the mixed-integer linear model (5)–(17) was solved with CPLEX 12.10 (using GAMS) applying an optimality gap of 1%, and a time limit of 24 hours. The schedule approaches have been implemented in Java 8.

The parameters for the GA were chosen based on previous studies. For example, in the Tournament Selection, we randomly select $\beta = 5$ individuals from the population. The two best ones then form the parent individuals. Once $\lambda = 25$ children are generated, the next generation of $\mu = 25$ individuals is created. Please note that the best parameter combination may depend on the current instance size. An essential factor in the parameterization of the MCTS approach is the branching factor b, which influences how many different perfect matchings have to be considered per time slot. Due to the importance of the factor b, we have carried out some experiments, too. Figure 6 shows the solution quality of the MCTS (measured by the mean room changes) for different numbers of b. In our tests, we performed the MCTS 20 times for 30,000 iterations (run time ca. 10 minutes) for each of the considered suitable b-values. The different results for the b-values can be seen in the box plot, $b = 400$ produces the best results, and is therefore used in the further process. Note that an increase of the branching factor does not necessarily lead to better results, despite the assumed construction of a larger tree. Before a deeper node can be examined, the parent node must be fully expanded, i.e., $b - 1$ "siblings" of the current node have to be considered. As we left the number of iterations and thus the number of nodes to generate constant, the branching factor b represents a trade-off between breadth-first and depth-first search. Because of the countless possible combinations over m levels of the tree, a very good solution can be constructed from $b = 400$ child nodes.

Figure 7 visualizes the results of the heuristic approaches (as well as the solver solution with GAMS) considering different run time limits in the range of 200–900 seconds. The genetic algorithm was executed with different numbers $n \in \{25, \dots, 32\}$ of time slots. As described above, the consideration of less than 25 time slots in the given computation time resulted in no adequate solution. MCTS creates a time-constrained schedule and always considers the number of $n = m = 22$ time slots. The objective function values shown are mean values based on 20 runs each.

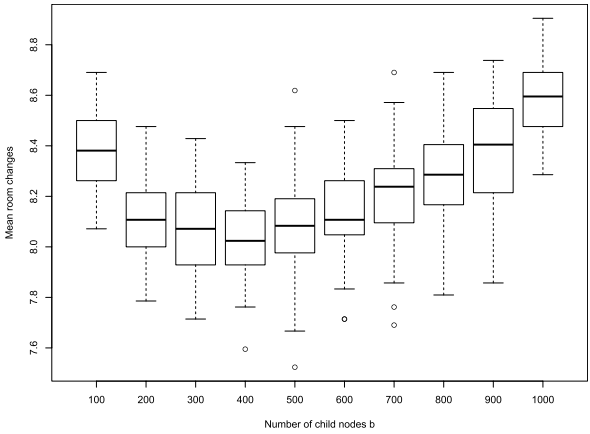


Fig. 6 Solution quality depending on the branching factor b

As expected, the number of average room changes decreases with the run time taken by the heuristic approaches. In addition, when looking at the GA, it becomes clear that an increasing number n of possible time slots causes lower numbers of average room changes. At $n = 25$, solutions with a mean of 10 room changes are generated, whereas at $n = 32$, solutions with a mean of only 5 room changes are possible. The MCTS is able to generate solutions with an average of 8 room changes for $n = m = 22$ time slots. The quality of the results of the MCTS approach is thus roughly comparable to the GA for 27 slots.

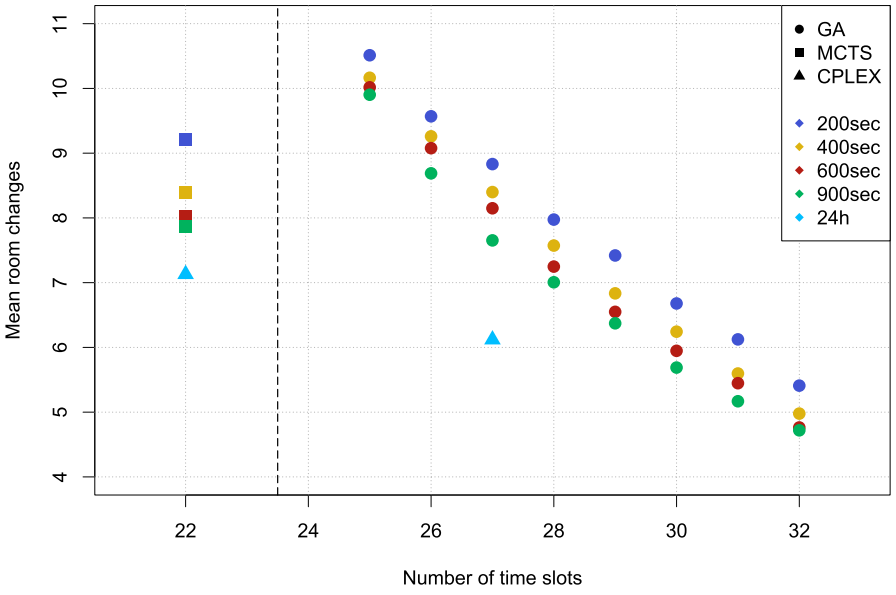


Fig. 7 Solution quality depending on the selected approach and maximum run time. The left part shows the quality of the MCTS for 22 slots (squares). The right part shows the quality of the GA for 25 to 32 slots (circles). For comparison, the quality of CPLEX for 22 and the finally selected 27 slots is added (triangles)

To investigate the influence of the instance size on the quality and runtime of both heuristics and the exact solver, we additionally performed experiments with larger data sets. For this purpose, we assumed that – for practical reasons – again only 22 conversations in 22 (MCTS) or 27 (GA) time slots should take place. The number of participating companies was set to 40, 56, and 80. The number of students was either the same or 12.5% larger to express a slight student surplus. The companies were divided equally among the available rooms, with each room providing space for exactly 8 companies.

All generated instances were solved with both heuristics (different run times up to 1 h) and CPLEX (24 h). Analogous to the previous experiments (cf. Fig. 7), the variants with 22 as well as with 27 time slots were also solved with GAMS and CPLEX. Figure 8 shows the results for all 6 instance sizes, whereby the average results of 10 runs each are given for the two heuristics. For the reason of comparability between the different instance sizes, the number of room changes is given as the proportion of those of a random solution. Therefore, a value of 0.6 means that in this solution only 60% of the room changes are necessary as in a random solution. With increasing runtime of the heuristic procedures, the number of room changes decreases. CPLEX could not generate a time-constrained schedule (22 time slots) for any instance size, therefore the CPLEX-results are only showed for 27 time slots. For the instances with 40 and 56 companies (but not 80 companies) CPLEX-solutions for 27 time slots were found, which are entered as constant lines in Fig. 8.

It can be observed that already with 40 companies and 40 students (the number of participants assumed in advance) the solutions of the genetic algorithm are better than those of the exact solver (with a time limit of 24 hours). Larger instances could not be solved with CPLEX and the applied solver configurations. For small instances, the number of necessary room changes was reduced by approximately

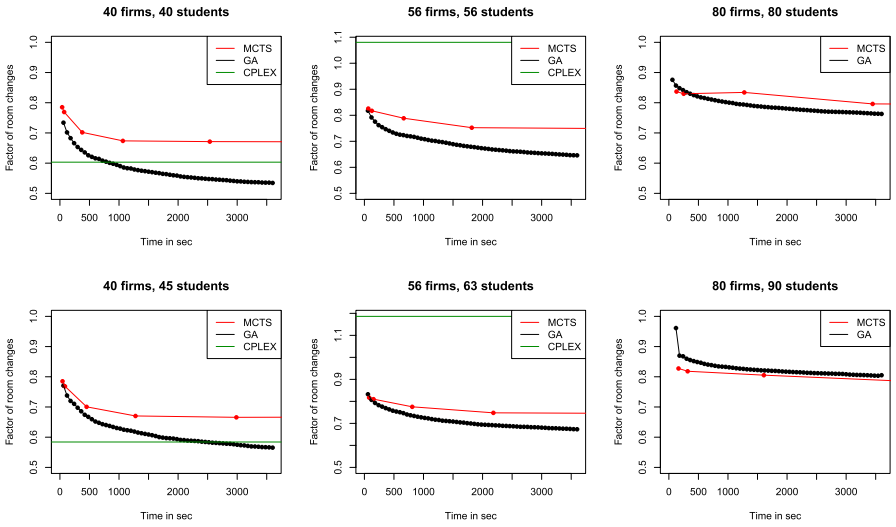


Fig. 8 Comparison of MCTS, GA, and CPLEX for different instance sizes

30% (time-constrained with MCTS) or 45% (time-relaxed with GA) within one hour solution time. For the largest instances with 80 companies and 80 or 90 students, reductions of about 20% could still be achieved within one hour. Please note that the performance of the MCTS can be rated as better here, as the results are quite similar and a time-constrained schedule typically requires more room changes. The results show that both scheduling algorithms can reduce the number of necessary room changes significantly and therefore provide high-quality schedules.

6.2 Realization and Feedback

For the realization of the IT-speed dating in 2020, $m = 22$ talks were distributed over $n = 27$ time slots (solution of the GA). This procedure guaranteed an appropriate balance between individual break times and few room changes. In addition to generating a solution for the IT-speed dating, overview and statistics files were created to assess the quality of the solution identified. By means of simple input masks, parameters such as the concrete $\mathcal{A}, \mathcal{B}, \mathcal{C}$-weighting or the weighting of the objective function components can be adjusted if necessary. We tested different variants of the transfer from $\mathcal{A}, \mathcal{B}, \mathcal{C}$ to the preference values p and found that the matching approach was largely insensitive to the changes if $\mathcal{A}$ was rated over $\mathcal{B}$ and $\mathcal{B}$ over $\mathcal{C}$. In order to ensure that as many students as possible receive their $\mathcal{A}$-wishes, we rated $\mathcal{A}$ with 100 as particularly important. $\mathcal{B}$-wishes were rated with 2, $\mathcal{C}$-wishes with 1, thus ensuring that the fulfillment of one $\mathcal{A}$-wish is much more important than numerous $\mathcal{B}$-wishes and that $\mathcal{C}$-wishes are better than the fictitious company.

In the end, all 6 $\mathcal{A}$-wishes and between 9 and 14 $\mathcal{B}$-wishes (average 11.6) were fulfilled for all 42 participating students, which had between 20 and 22 real talks (average 21.9). The Jain index of the final solution with the described $\mathcal{A}, \mathcal{B}, \mathcal{C}$ -weighting was over 99.9%. Since all participants received their $\mathcal{A}$-wishes and the student scores showed only small deviations due to the high $\mathcal{A}$-weighting, the Jain index was also calculated with a weighting of $(p_{\mathcal{A}}, p_{\mathcal{B}}, p_{\mathcal{C}}) = (3, 2, 1)$. Even in this case, the Jain index with a value of more than 99.8% indicated that the solution is very fair. Based on these results, our matching algorithm can also be rated as very helpful in the presented scenario. The main goal of fulfilling all $\mathcal{A}$-wishes and as many $\mathcal{B}$-wishes as possible is met, as we fulfilled 100% of the $\mathcal{A}$-wishes and 72.5% of the $\mathcal{B}$-wishes. Additionally, the raw data, as well as the calculated fairness measures, indicate a given fairness well within our expectations.

At the end of the event, both students and companies were asked for feedback on the organizational scheme implemented. A total of 24 companies took already part in year 2019 and therefore could compare with the round robin procedure. The vast majority of companies (86%) was in favor of not returning to round robin. Additionally, of the 24 companies, 20 noticed a remarkable (11) or slight (9) reduction of "empty dates."

Both students and companies were asked to rate the talks following the $\mathcal{A}, \mathcal{B}, \mathcal{C}$ -scheme used before. The results are shown in Table 1. In about 25% of the talks evaluated by both sides, both parties found the other to be a good match. On average, each student met five companies, where both sides were confident about an

Table 1 $\mathcal{A}, \mathcal{B}, \mathcal{C}$-ratings by companies and students after the speed dating

		Companies			
		$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	Sum
Students	$\mathcal{A}$	202	108	42	352
	$\mathcal{B}$	108	110	65	283
	$\mathcal{C}$	30	50	101	181
	Sum	340	268	208	816

internship. It happened just with one of the 42 participating students that no suitable match was found. Only in every third conversation one of the parties rated the other as not suitable. Furthermore, 51% of the conversations were rated identically by both sides ($\mathcal{A}$-$\mathcal{A}$, $\mathcal{B}$-$\mathcal{B}$ or $\mathcal{C}$-$\mathcal{C}$), only 9% had the combination $\mathcal{A}$-$\mathcal{C}$ or $\mathcal{C}$-$\mathcal{A}$. This shows that both sides have evaluated the fit quite similarly.

Overall, the main goal of the new concept is met. While maintaining a maximum duration of 4 hours, we brought together a large number of students and companies to exchange offers and expectations for upcoming internships. Nearly every student was able to find at least one suitable match (on average even five) for the following application phase. The excluded talks – compared to the previous round robin procedure – were at least disproportionately often "empty dates," as the vast majority of the companies noticed a reduction of their ratio. Additionally, most of the companies preferred the new concept over the round robin procedure. The students' feedback also indicated satisfaction with the new concept. However, it must be mentioned here that no student experienced both concepts, as multiple participation is not allowed.

7 Conclusion

In this paper, we presented a new methodology for IT-speed dating at the University of Hildesheim. Instead of the procedure that each student meets every company, the talks taking place are determined on the basis of previously given preferences by the students. Therefore, the companies provide an anonymous short description of their internships and the students divide them into classes $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ according to their preferences.

A matching algorithm assigns each participant (student as well as company) to exactly m conversation partners. The matching problem was formulated as a classical capacitated transportation problem and was solved by using a start and improvement procedure. By modifying the improvement procedure, the variance in the fulfillment of student wishes could be reduced in order not to put anyone in a better or worse position.

Two approaches were developed for the creation of a time schedule and a room allocation. The first method is based on the Monte Carlo tree search. The second method is a genetic algorithm that solves the problem as an open shop problem. While the first one is able to generate time-constrained schedules due to the use of some special problem characteristics, the genetic algorithm is more flexible and can also be used for slightly different conditions, such as different lengths of talks or different numbers of talks per participant.

With the help of the GA solution, a large number of talks between interested participants could be made possible despite the increasing number of participants. The share of talks in which both parties were not interested was only around 12%. According to the companies surveyed, this represents a significant reduction compared to previous years. The survey of all participants also revealed that the majority were satisfied with the new concept and the specified number of talks (about 2/3 of the possible).

The solution methods considered in the paper can be applied not only for the planning of the IT-speed dating event at the University of Hildesheim. Similar dating formats can also be planned in this way, e.g., a job speed dating or a hobby speed dating. In these cases, there are (as in our situation) two groups of people: job seekers/companies or hobby seekers/hobby people who want to meet in m conversations. The seekers usually have preferences for the partners, which, for example, may be categorized in the categories $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability Not available.

Code Availability Not available.

Declarations

Conflicts of Interest/Competing Interests The authors declare that they have no conflict of interest.

References

1. Pathak PA, Sethuraman J (2011) Lotteries in student assignment: An equivalence result. Theor Econ 6(1):1–17
2. Kesten O, Kurino M, Nesterov A (2017) Efficient lottery design. Soc Choice Welf 48:31–57
3. Abdulkadiroğlu A, Sönmez T (1998) Random serial dictatorship and the core from random endowments in house allocation problems. Econometrica 66(3):689–701
4. Bogomolnaia A, Moulin H (2001) A new solution to the random assignment problem. J Econ Theory 100(2):295–328
5. Duan R, Pettie S (2014) Linear-time approximation for maximum weight matching. J ACM 61(1):1–23
6. Budish E, Cantillon E (2012) The multi-unit assignment problem: Theory and evidence from course allocation at Harvard. Am Econ Rev 102(5):2237–2271
7. Sotomayor M (1999) Three remarks on the many-to-many stable matching problem. Math Soc Sci 38(1):55–70
8. Baïou M, Balinski M (2000) Many-to-many matching: Stable polyandrous polygamy (or polygamous polyandry). Discret Appl Math 101(1–3):1–12

9. Manea M (2007) Serial dictatorship and Pareto optimality. Games Econ Behav 61(2):316–330
10. Zhou L (1990) On a conjecture by gale about one-sided matching problems. J Econ Theory 52(1):123–135
11. Roth AE (1982) The economics of matching: Stability and incentives. Math Oper Res 7(4):617–628
12. Bouveret S, Lemaître M (2016) Characterizing conflicts in fair division of indivisible goods using a scale of criteria. Auton Agent Multi-Agent Syst 30(2):259–290
13. Heinen T, Nguyen NT, Rothe J (2015) Fairness and rank-weighted utilitarianism in resource allocation. In: Walsh (ed) Algorithmic Decision Theory. ADT 2015. Lecture Notes in Computer Science, vol 9346
14. Caragiannis I, Kurokawa D, Moulin H, Procaccia AD, Shah N, Wang J (2019) The unreasonable fairness of maximum Nash welfare. ACM Trans Econ Comput 7(3):1–32
15. Jain RK, Chiu DMW, Hawe WR (1984) A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Digital Equipment Corporation, Tech. rep
16. Mühlenthaler M, Wanka R (2016) Fairness in academic course timetabling. Ann Oper Res 239(1):171–188
17. Muklason A, Parkes AJ, Özcan E, McCollum B, McMullan P (2017) Fairness in examination timetabling: Student preferences and extended formulations. Appl Soft Comput 55:302–318
18. Nemhauser GL, Trick MA (1998) Scheduling a major college basketball conference. Oper Res 46(1):1–8
19. Cocchi G, Galligari A, Nicolino FP, Piccialli V, Schoen F, Sciandrone M (2018) Scheduling the Italian national volleyball tournament. INFORMS J Appl Anal 48(3):271–284
20. Durán G, Guajardo M, Wolf-Yadlin R (2012) Operations research techniques for scheduling Chile's second division soccer league. INFORMS J Appl Anal 42(3):273–285
21. Durán G, Guajardo M, Sauré D (2017) Scheduling the South American qualifiers to the 2018 FIFA World Cup by integer programming. Eur J Oper Res 262(3):1109–1115
22. Knust S (2010) Scheduling non-professional table-tennis leagues. Eur J Oper Res 200(2):358–367
23. Kostuk KJ, Willoughby KA (2012) A decision support system for scheduling the Canadian football league. INFORMS J Appl Anal 42(3):286–295
24. Kyngäs J, Nurmi K, Kyngäs N, Lilley G, Salter T, Goossens D (2017) Scheduling the Australian football league. J Oper Res Soc 68(8):973–982
25. Van Bulck D, Goossens D, Spieksma FCR (2019) Scheduling a non-professional indoor football league: a tabu search based approach. Ann Oper Res 275(2):715–730
26. Van Bulck D, Goossens D, Schönberger J, Guajardo M (2020) RobinX: A three-field classification and unified data format for round-robin sports timetabling. Eur J Oper Res 280(2):568–580
27. Kendall G, Knust S, Ribeiro CC, Urrutia S (2010) Scheduling in sports: An annotated bibliography. Comput Oper Res 37(1):1–19
28. Vangerven B, Ficker AMC, Goossens DR, Passchyn W, Spieksma FCR, Woeginger GJ (2018) Conference scheduling - A personalized approach. Omega 81:38–47
29. Gulati M, Sengupta A (2004) TRACS – Tractable conference scheduling. In: Proceedings of the decision sciences institute annual meetings (DSI 2004), pp 3161–3166
30. Zulkipli F, Ibrahim H, Benjamin AM (2013) Optimization capacity planning problem on conference scheduling. In: 2013 IEEE Business Engineering and Industrial Applications Colloquium (BEIAC), pp 911–915
31. Jaafar SN, Rahim SKNA, Omar N, Masrom S, Jaafar AH (2017) Domain transformation approach: Optimizing the preference-based conference schedules via room sharing matrix. In: 7th IEEE International Conference on System Engineering and Technology (ICSET 2017), pp 83–88
32. Wagner HM (1959) On a class of capacitated transportation problems. Manage Sci 5(3):304–318
33. Charnes A, Cooper WW (1954) The stepping stone method of explaining linear programming calculations in transportation problems. Manage Sci 1(1):49–69
34. Hall P (1935) On representatives of subsets. J Lond Math Soc 10(1):26–30
35. Galil Z (1986) Efficient algorithms for finding maximum matching in graphs. ACM Comput Surv 18(1):23–38
36. Ford LR, Fulkerson DR (1956) Maximal flow through a network. Can J Math 8:399–404
37. Chaslot G, Bakkes S, Szita I, Spronck P (2008) Monte-Carlo tree search: A new framework for game AI. In: Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, pp 216–217

38. Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of Monte Carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games 4(1):1–43
39. Dinic EA (1970) Algorithm for solution of a problem of maximum flow in a network with power estimation. Soviet Math Dokl 11(5):1277–1280
40. Edmonds J, Karp RM (1972) Theoretical improvements in algorithmic efficiency for network flow problems. J ACM 19(2):248–264
41. Gonzalez T, Sahni S (1976) Open shop scheduling to minimize finish time. J ACM 23(4):665–679
42. Rahmani Hosseinabadi AA, Vahidi J, Saemi B, Sangaiah AK, Elhoseny M (2019) Extended genetic algorithm for solving open-shop scheduling problem. Soft Comput 23(13):5099–5116
43. Chen SH, Chang PC, Cheng T, Zhang Q (2012) A self-guided genetic algorithm for permutation flowshop scheduling problems. Comput Oper Res 39(7):1450–1457