Mixed Integer Programming versus Evolutionary Computation for Optimizing a Hard Real-World Staff Assignment Problem

Jannik Peters and **Daniel Stephan** and **Isabel Amon** and **Hans Gawendowicz** and **Julius Lischeid** and **Lennart Salabarria** and **Jonas Umland** and **Felix Werner** and and **Martin S. Krejca** and **Ralf Rothenberger** and **Timo Kötzing** and **Tobias Friedrich**
Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

Abstract

Assigning staff to engagements according to hard constraints while optimizing several objectives is a task encountered by many companies on a regular basis. Simplified versions of such assignment problems are NP-hard. Despite this, a typical approach to solving them consists of formulating them as mixed integer programming (MIP) problems and using a state-of-the-art solver to get solutions that closely approximate the optimum.

In this paper, we consider a complex real-world staff assignment problem encountered by the professional service company KPMG, with the goal of finding an algorithm that solves it faster and with a better solution than a commercial MIP solver. We follow the evolutionary algorithm (EA) metaheuristic and design a search heuristic which iteratively improves a solution using domain-specific mutation operators. Furthermore, we use a flow algorithm to optimally solve a subproblem, which tremendously reduces the search space for the EA.

For our real-world instance of the assignment problem, given the same total time budget of 100 hours, a parallel EA approach finds a solution that is only $1.7\,\%$ away from an upper bound for the (unknown) optimum within under five hours, while the MIP solver Gurobi still has a gap of $10.5\,\%$.

Introduction

Every consulting company is faced with the problem of assigning employees to engagements, especially in the sector of professional services. Oftentimes, this is done manually, which is expensive and time-consuming. Hence, many companies strive for a semi-automatic solution.

We consider a staff assignment problem faced by the professional-service company KPMG AG Wirtschaftsprüfungsgesellschaft on a regular basis. In each instance, differently skilled employees have to be assigned on a day-to-day basis to engagements requiring specific needs (i.e., skills on specific levels) for a certain amount of hours, all while satisfying various hard constraints. Further, we have several conflicting objectives we want to optimize, such as covering as many engagements as possible with the best-fitting skills while using as few employees as possible (see section *Problem Definition* for more details). We combine all objectives into one objective by using a weighted sum, where the weights are given by management decisions. The resulting problem is strongly NP-hard, as can be shown by using a similar reduction from 3-Partition as done by Garey and Johnson (1975) and Salewski, Schirmer, and Drexl (1997).

We compare two different approaches for solving this optimization problem. The classical, first approach is a reformulation as a mixed-integer program (MIP). For this, we linearize all constraints and objectives by standard techniques. This approach was successfully used to solve another strongly NP-hard multi-skill IT staffing problem by Heimerl and Kolisch (2010). However, solving the resulting MIP instance of our problem by the state of the art solver Gurobi (Gurobi Optimization 2018) takes four days and offers only a mediocre assignment in terms of objective value.

Our second approach is a tailored evolutionary algorithm (EA). EAs are inspired by the biological

principle of evolution, especially of mutation and selection: an existing solution is copied, where the copy has a few random modifications; the former is called the parent, the latter the offspring. Out of parent and offspring, the individual with better objective value is selected to be the parent in the next iteration, resulting in the creation of better and better individuals. This strategy has been applied successfully to many optimization problems. For example, Globus et al. (2004) compare several evolutionary strategies for scheduling satellites, Jan, Yamamoto, and Ohuchi (2000) use one to schedule nurses, and Mesghouni, Hammadi, and Borne (2004) apply one to job-shop scheduling.

The EA puts no restrictions on how the objective function is defined. One benefit of this is that no linearization of objectives is required. Even more, the EA gives the flexibility to use a hybrid approach for finding a good assignment as follows. The algorithm only assigns different employees to engagements for specific dates, without assigning them to a *specific need* of an engagement. Given this (partial) assignment, we use a very efficient minimum-cost maximum-flow computation to find an optimal assignment of employees to needs. We describe this procedure in detail in the section *Minimum-Cost Maximum-Flow*. This approach reduces the dimension of the search space for the EA tremendously.

We compare Gurobi (the MIP solver we consider) and the EA experimentally on a real-world instance from KPMG and see that the EA outperforms Gurobi by far. Given an upper bound on an optimal solution, it takes Gurobi 4 days to create a solution that is $10.5\,\%$ away from that bound. In comparison, the EA beats Gurobi already after only 5 hours with a probability of $45\,\%$. Even further, it can be massively parallelized. Using parallelization with a total time budget of 100 hours, the EA typically finds in under 5 hours wall-clock time a solution that is only $1.7\,\%$ away from the upper bound, and it beats Gurobi already after 7 minutes.

Finally, we present our visualization software. The user interface we built allows loading, visualizing, editing, and saving assignments as well as calculating the effects of changes on each objective in real time. It also offers to run the EA with freely selectable objective weights and to adjust the resulting assignments manually.

Various approaches have been tried for assignment problems. One of the classical problems in the field of operations research and computer science is resource-constrained scheduling, for example defined by Brucker et al. (1999). However, in most cases, this problem only has a single objective.

An extension of resource-constrained scheduling takes the skills of employees into account, similar to our problem, as outlined in a survey by De Bruecker et al. (2015). However, in contrast to our setting, the skills are not used as a criterion for the value of their assignment but rather as a hard constraint.

Li et al. (2012) consider a multi-objective nurse scheduling problem, which is similar to our problem. While their amount of constraints and objectives is similar to ours, our problem has a far higher dimensionality and considers more aspects, such as employees having different skills of varying levels.

Ernst et al. (2004) and Van den Bergh et al. (2013) also consider staff scheduling problems, pertaining to health care, transportation, or financial services. While similar to our problem, none of them has an objective considering the continuity of work during the runtime of a project, as will be described in Objective 3. However, for a professional service company like KPMG, the continuity of work is a core measurement for the quality of an assignment and as such must be taken into account.

Problem Definition

Our assignment problem considers a set of engagements E to which members of a set of employees A should be assigned. We let D be a set of days, and we cluster days into weeks W. S is a set of skills. Each employee has a corresponding level of competence for each skill, which takes a value in $L \subseteq \mathbb{N}$, with a small value representing little experience. Furthermore, the employees have places of residence and days of absence.

Each engagement has a location, a set of employees that worked on this engagement in the last planning period, and a set of needs. The latter consists of skill–level combinations with corresponding needs in hours per week that should be worked on.

An assignment for this problem is a mapping

$$\Phi\colon A \times D \times E \times S \times L \to \mathbb{R}_{\geq 0}$$

such that, for each $a \in A$, $d \in D$, $e \in E$, $s \in S$, and $\ell \in L$, $\Phi(a, d, e, s, \ell)$ is the amount of hours employee a works on engagement e on day d, regarding skill s with level ℓ.

Hard Constraints

Due to the real-world nature of our problem, we have to consider several hard constraints, such as not doing more work than necessary. Other hard constraints are derived from legal regulations. The formal definition of these constraints is as follows.

Hard Constraint 1 Employees do not contribute to satisfying a need if that need is already satisfied. Hence, the assigned total hours per skill and level shall not be greater than the needed hours in this skill–level combination. Let $n_{ews\ell}$ be the amount of needed work on engagement e in week w on skill s and level ℓ. Hence, for all engagements e, weeks w, skills s, and levels ℓ, it has to hold that

$$\sum_{a \in A,\, d \in w} \Phi(a, d, e, s, \ell) \leq n_{ews\ell} \,. \tag{1}$$

Hard Constraint 2 Employees cannot work if they are absent. Let i_{ade} be 1 if employee a works on day d on engagement e and 0 otherwise, and let v_{ad} be 1 if employee a is on vacation on day d and 0 otherwise. Thus, for all employees a, days d, and engagements e, it has to hold that

$$i_{ade} \leq 1 - v_{ad} \,. \tag{2}$$

Hard Constraint 3 An employee may only be assigned to one engagement per day. Therefore, reusing the notation from Hard Constraint 2, for all employees a and days d, we require

$$\sum_{e \in E} i_{ade} \leq 1 \,. \tag{3}$$

Hard Constraint 4 In our setting, it is illegal to work more than 10 hours per day. Let h_{ade} be the amount of hours employee a works on engagement e on day d. Thus, for all employees a, days d, and engagements e, it has to hold that

$$h_{ade} \leq 10 \,. \tag{4}$$

Objectives

Based on a valid assignment, we can define an objective value. In our case, we want to maximize a weighted sum of seven sub-objectives, which will be introduced in the following subsection. In this section, we introduce the seven sub-objectives to our assignment problem.

Objective 1 Employees who have worked on an engagement in the previous planning period have an advantage in experience on this engagement over other employees who have not worked on it before. Hence, they will be able to work faster and deliver work with a potentially higher quality. Thus, we want to maximize the sum of hours spent on an engagement by employees who worked on it in the previous planning period. Let h_{ae} be the amount of hours employee a works on engagement e, and let ℓ_{ae} be 1 if a has worked on e in the previous planning period and 0 otherwise. We want to maximize

$$\sum_{a \in A,\, e \in E} h_{ae} \cdot \ell_{ae} \,. \tag{5}$$

Objective 2 As professional services require a lot of cooperation between the employees, we want to keep the team on an engagement constant and small. Thus, we want to minimize the number of employees working on an engagement. Let i_{ae} be 1 if employee a works on engagement e and 0 otherwise. Then we want to minimize

$$\sum_{a \in A,\, e \in E} i_{ae} \,. \tag{6}$$

Objective 3 Employees are more efficient when they work on an engagement continuously. They do not need to familiarize themselves with the engagement again, and their work flow is not interrupted. Thus, the amount of days an employee does *not* work on an engagement when they work on it at most 5 work days later should be minimized. Let j_{ade} be 1 if employee a does not work on engagement e on day d but works on e at most 5 days later and 0 otherwise. Then we want to minimize

$$\sum_{a \in A,\, d \in D,\, e \in E} j_{ade} \,. \tag{7}$$

Note that this can lead to some anomalies in the first 5 days, which can easily be solved by adding 5 days without needs before those first days.

Objective 4 In order to ensure both the well-being of the employees and to waste as little time as possible on traveling, the traveling distance should be minimized. Let t_{ae} be the traveling distance of an employee a to an engagement e. Reusing notation from Hard Constraint 3, we want to minimize

$$\sum_{a \in A,\, d \in D,\, e \in E} t_{ae} \cdot i_{ade} \,. \tag{8}$$

Objective 5 In our problem instance, the amount of needs is too high to be covered fully. To satisfy most of the needs, all employees should work as much as possible. Thus, we want to maximize

$$\sum_{a\in A,\, d\in D,\, e\in E,\, s\in S,\, \ell\in L} \Phi(a,d,e,s,\ell)\ . \quad (9)$$

Objective 6 Every engagement has needs in certain skill–level combinations per week in hours. As some tasks need a certain expertise, experienced employees are expected to perform better than inexperienced ones. However, it might be beneficial to prefer a slightly under-qualified employee to an overqualified one in order to ensure the growth of all employees. Hence, it is also reasonable to assign a penalty to overqualified employees. Overall, the mismatch of needed level and the actual level of an employee working on a task should be as small as possible. Let ℓ_{as} be the actual level of a in s, and let $p(\Delta)$ be the penalty for a skill difference of Δ levels. Subsequently, we want to minimize

$$\sum_{\substack{a\in A,\, d\in D,\, e\in E,\\ s\in S,\, \ell\in L}} \Phi(a,d,e,s,\ell)\cdot p(\ell_{as}-\ell)\ . \quad (10)$$

Objective 7 Regular work days last 8 hours, additional work (up to the national legal limit of 10 hours; see Hard Constraint 4) is considered overtime. In order to reduce both stress and bonus payment, the amount of overtime work should be minimized. Hence, we want to minimize

$$\sum_{a\in A,\, d\in D,\, e\in E} \max\{h_{ade}-8,0\}\ . \quad (11)$$

Let the objectives be $o_1,\ldots,o_7$, and let their corresponding (instance-defined) weights be $w_1,\ldots,w_7$. Then, for an assignment Φ, we want to maximize the objective function $f(\Phi)=\sum_{i=1}^{7} w_i\cdot o_i$.

Approaches

In this section, we describe the two different approaches for solving our assignment problem.

Mixed Integer Program

For our MIP, we can model all objectives almost exactly as described in the section *Problem Definition*. This leads to $\Theta(|A|\cdot|D|\cdot|E|\cdot|S|\cdot|L|)$ many continuous variables and $\Theta(|A|\cdot|D|\cdot|E|)$ many binary variables. Furthermore, we have $\Theta(|A|\cdot|D|\cdot|E|\cdot|S|\cdot|L|)$ many constraints.

Note that Objectives 2 and 3 are the objectives that are the hardest to optimize for MIP solvers, as those objectives create many additional binary variables with complex dependencies.

Evolutionary Algorithm

The EA (Algorithm 1) works by storing a single assignment, which it changes iteratively. We only keep the change if it improves the quality, otherwise we revert it.

A change consists of a sequence of mutation operators, chosen from a pool which we will describe later. The number of operators we apply is chosen randomly for each iteration via a Poisson distribution with expected value $\lambda=1$ with 1 added to the result. This way, we will mostly perform two mutations per iteration but still have a chance of performing exactly one change or more than two. For us, this is the reason why our algorithm is an EA and not a local search. Since we apply several operators an individual can be changed to arbitrarily far away individuals (with probability decreasing with the distance) before being subject to selection. In contrast, local search would apply one of our operators only once before using selection, which would give worse performance as this makes it harder to escape local optima.

After determining the number of mutations, we choose each operator uniformly at random from our pool (with repetition). However, since the operators mutation_cleanse_block and mutation_add_destructive are very efficient, they occur twice in our pool. That is, the probability to choose one of them is $1/4$ each, while the probability for choosing each other operator is $1/8$. Note that these operators are all hand-crafted. The ultimate choice of operators as well as the choice to add two of them with higher probability is made on grounds of extensive experiments the results of which we have to omit due to space constraints.

In describing our operators, it is important to note that they only work on engagements, employees, and days – not on skill–level combinations and hours. Assigning employees to respective skills and hours is always done via a minimum-cost maximum-flow after all mutations of the iteration have been applied. Note that the flow only has to be recomputed for weeks with changes.

Algorithm 1: Evolutionary Algorithm

input : problem instance[1] I
initial assignment Φ
number of desired iterations n
pool of our mutation operators OPs

for $i = 1$ **to** n **do**
  choose $k \sim \text{Pois}(1) + 1$;
  $f_{\text{old}} \leftarrow f(\Phi)$;
  foreach $i \in \{1, \ldots, k\}$ **do**
    $m \leftarrow$ choose Unif(OPs);
    $\Phi \leftarrow m(\Phi)$;
  if $f_{\text{old}} \geq f(\Phi)$ **then** revert all changes;

mutation_add We assign an employee to an engagement on a specific day. Hence, we choose a triple of employee, engagement, and day that is not in the assignment (uniformly at random) until the triple satisfies Hard Constraints 2 and 3 and add it.

mutation_delete Similar to mutation_add, we choose a triple of employee, engagement, and day from the set of already assigned employees uniformly at random and remove it.

The previous two operators only perform local changes, which is bad when in a local optimum. Thus, the following operators perform larger changes. We refer to them as block operators.

mutation_add_block We choose an employee, an engagement, and a day–each uniformly at random. This is followed by the selection of a consecutive range of days starting from the chosen day. The length of this range is chosen via a binomial distribution with 20 trials and a success probability of 0.5. Then the employee is added to the chosen engagement every day within the range if this addition satisfies the hard constraints. If it does not satisfy the hard constraints, we do not add the employee on this day and proceed to the next day.

mutation_delete_block This operator works similar to mutation_add_block, but deletes the employee within the range of days instead of adding.

All operators so far are sufficient for optimizing our objectives in general. However, they still leave some potential in Objective 3, which describes the continuity of work during an engagement. Thus, we use two mutation operators tailored towards improving Objective 3 by allowing us to create a state where an employee can work on only one engagement during a range of days.

mutation_cleanse_block We choose an employee uniformly at random. Additionally, a range of days is chosen in the same manner as in mutation_add_block. We then identify the engagement the employee worked the most days on in the chosen range of days. If there are multiple possible engagements, we choose one of them uniformly at random. The employee is then deleted from other engagements in this range of days and added to the identified engagement on every day in the range where the addition satisfies the hard constraints.

mutation_add_destructive This operator works similarly to mutation_cleanse_block, but instead the engagement is chosen uniformly at random.

Note that the first two operators are straightforward and use basically no problem knowledge. The next two employ the knowledge that objectives benefit from scheduling employees for consecutive days. Finally, the last two operators are tailored to specific problems encountered by the algorithm during optimization to overcome local optima.

Minimum-Cost Maximum-Flow

The EA is only concerned with assigning employees to engagements for specific days, without considering which skill–level requirements should be satisfied by the scheduled employees. Given this assignment of employees to engagement and days, we can now decide which employee should be used to satisfy which need by constructing a flow network for every engagement e and week w as depicted in Figure 1. A flow unit represents an hour. For each employee working on e in week w, there is a node. Also, there is a node for every combination of skill and level the engagement has needs in (in w).

We have two edges from the source to every employee node. The first edge has a capacity equal to the amount of non-overtime hours that an employee a can work on e in week w. The cost is the negative weight of Objective 1 if a has worked on e in the previous planning period and 0 otherwise. The second edge represents Objective 7, that is, the overtime objective. The edge has a capacity equal to

[1] That is, all of the relevant information mentioned in the section *Problem Definition* that are necessary in order to compute all of the hard constraints and the objectives.

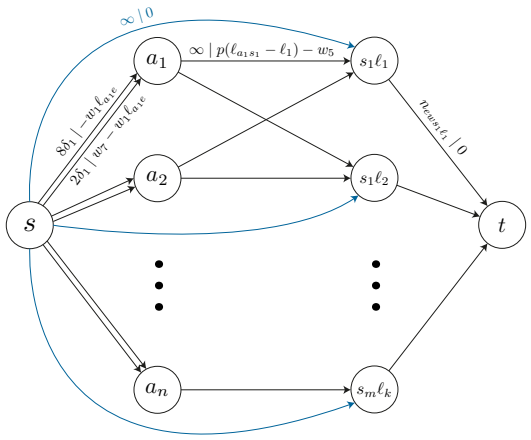


Figure 1: A flow network for assigning employees to exact hours on skills and levels for a given engagement and week. The flow units represent hours. Every edge has a capacity and cost, delimited by the symbol '|'. The blue edges represent the unsatisfied needs. The edges between the source s and the employee nodes symbolize the regular work an employee can do and the overtime work if they work δ_i days on the engagement in this week. The edges between the employee and skill–level (sl) nodes penalize an employee for working on a certain sl combination. Finally, the edges between the sl nodes and the sink t represent the needs the engagement has in this particular sl combination.

the amount of overtime hours a can work on e in w. This edge has the same cost as the first edge plus the weight of Objective 7. The total capacities of these edges represent the maximum hours a can work in w. The costs represent the penalty for overtime and the reward for having worked there in the last planning period. Furthermore, we add an edge between every pair of employee nodes and skill–level nodes, representing the amount of hours a works on this skill–level combination. The edge has a capacity of ∞ and a cost according to Objective 6 minus the weight of Objective 5 in order to represent the penalty for deviating from the required skill level and the reward for satisfying needs.

Now we connect all skill–level nodes to the sink t with a capacity of n_{ewsl} and a cost of 0. These edges ensure that only the required amount of hours are worked on each skill–level combination, according to Hard Constraint 1. Last, we add an edge from the source s to every skill–level node with a capacity of ∞ and a cost of 0, thus allowing employees to work fewer than the maximum possible amount of hours.

In practice, the minimum-cost maximum-flow can be computed very fast, see Goldberg and Tarjan (1989) and Orlin (1993). Thus, the EA only needs to consider the problem of assigning employees to engagements on days, as the combined value of Objectives 1, 5, 6, and 7 is the negated summed cost of the minimum-cost maximum-flows for all weeks and engagements.

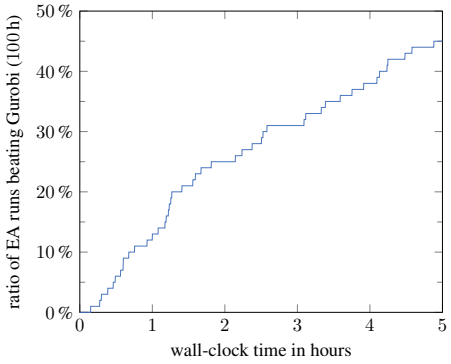
Experimental Comparison

We consider a real-world data set from KPMG consisting of 52 employees, 400 days, 90 engagements, and 18 skills with four levels each. The weights of the objective function were chosen based on the knowledge and experience of domain experts and are considered part of the input instance. The highest weight was given to Objectives 1 and 5; the lowest to Objectives 4 and 7. Unfortunately, we are not allowed to release the data set.

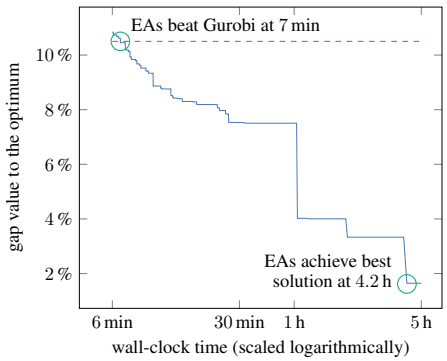
Gurobi (the MIP solver we consider) yields an upper bound for an optimal solution, which we call the *optimum*. For each algorithm we define the *gap value*; it always denotes the percentage a solution is away from the optimum, i.e., a gap of $2\,\%$ is synonymous to a 98%-approximation of the optimum. We use the gap values in order to compare the solution quality of both of our algorithmic approaches.

Gurobi achieves a gap value of only $10.5\,\%$ after 100 hours (roughly 4 days) and not before. Our optimum is also taken after the same amount of time to get a good upper bound. The EA yields far better results, which we describe in the following.

One problem is that while Gurobi is deterministic, the quality of the EA after t seconds is a distribution. Thus, we consider two experimental setups, whose results are depicted in Fig. 2. For the first setup, we let the EA run for 5 hours, for a total of 210 times. Fig. 2a shows the percentage of these 210 runs that were able to create a solution of a quality at least as good as Gurobi. This suggests that the EA gets a solution better than Gurobi after significantly less time with moderately high probability, e.g., nearly $45\,\%$ after 5 hours. Note that this probability roughly follows a linear trend. This allows for a massive parallelization approach: consider a fixed



(a) The ratio of EA runs beating Gurobi after the denoted amount of time. At each point depicted, 210 independent runs of the EA were performed. Note that Gurobi's solution we compare against was only achieved after 100 h.



(b) The best gap value achieved when running multiple EAs in parallel, each run only for the depicted wall-clock time, for a total time budget of 100 h. The dashed (red) line denotes the gap value of Gurobi after 100 h and is only added for comparison.

Figure 2: Our experimental results. Fig. 2a shows that the percentage of EA runs beating Gurobi (running for 100 h) grows roughly linearly in the time spent per run. Thus, multiple EAs can be run in parallel. This approach is depicted in Fig. 2b. See the section *Experimental Comparison* for details.

time budget of 5 h. Then it does not matter whether we let one EA run for 5 h or five EAs for 1 h each – the success ratio of beating Gurobi will roughly stay the same. However, the wall-clock time needed will only be 1 h in the latter case.

For the second experiment, we build upon the just mentioned approach of parallelization. We consider a total time budget of 100 h, i.e., the time it took Gurobi to achieve a reasonable solution. Given an amount of EAs that can be run in parallel, we determine the uniform amount of time each EA can be given for the time budget. We then take the best solution we could find up to that point in time as our overall solution. The medians of these results are depicted in Fig. 2b. We see that a time budget of 7 min per EA (given the respective number of cores) is already sufficient for creating a solution that outperforms the quality of the solution that Gurobi achieved after 100 h. If, instead, we give 4.2 h of time per EA, we achieve a gap value of 1.7 %. Hence, given enough computing power, the EA approach greatly outperforms Gurobi both in wall-clock time and solution quality.

Visualization

In order to make it easier for us and our industry partners to inspect assignments, we developed a graphical user interface. The interface has three different views: an assignment overview, an assignment creation view, and a detail view.

In the assignment overview, assignments can be saved, loaded, and compared. The objective values of an assignment are displayed using parallel coordinates, and different assignments are shown in the same diagram, allowing for an easy comparison.

In the assignment creation view, assignments can be calculated using the EA. The weights of the objectives can be chosen freely before starting a run, and it is possible to start several runs in parallel. All running jobs are viewed by displaying their objective values as parallel coordinates, which are updated periodically.

In the detail view, the actual assignment of employees to days and engagements can be viewed (see Figure 3) and edited. Editing an assignment is done via a new window, in which all of the relevant information for assigning an employee with a certain skill for a specific day or range of days to a certain engagement can be entered. Such changes can always be reverted or saved into a new assignment. Whenever an assignment is changed, its new quality is calculated on the fly and compared to that of the previous assignment. This immediate feedback enables the user to get an impression of why certain

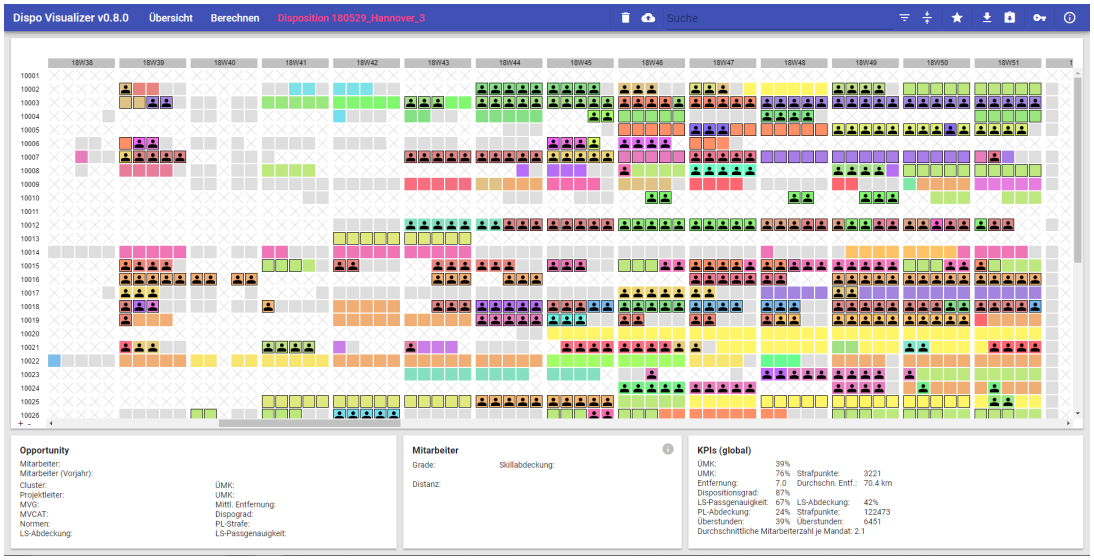


Figure 3: Example visualization of a staff assignment in the detail view with anonymized employees on the left and weeks of the year on top. Each box represents a day. The boxes are filled if the employees are assigned on that particular day, and the colors represent different engagements. If a box is selected, the fields at the bottom show information about the engagement and the fulfillment of its objective values (left), the employee and the skill and level they work on that day (center), and the global objective values of the assignment (right).

assignments of employees to days and skills are beneficial over other assignments that seem more natural and thus provides an indirect explanation of the assignment.

Discussion and Conclusion

The findings of our experiments show that EAs are a very good alternative to off-the-shelve solvers such as MIP solvers, even when the problem formulation is already close to a MIP problem. Our outcome showed that the EA did not require a lot of tuning and problem knowledge, but it benefited from simple yet specific mutation operators, especially concerning Objective 3. This is because the block operators help the algorithm to escape local optima and greatly boost the runtime, as beneficial operations can often be combined to a block operation. Furthermore, the EA allowed for formulating a subproblem which can be solved separately by a classical flow algorithm. Last, the EA can be massively parallelized (see Fig. 2b), which is a great benefit when given enough computing power by, e.g., a cloud-computing service.

Note that, in general, MIP solvers can find a good approximate solution for assignment problems similar to ours. However, Objectives 2 and 3 both require computationally expensive constraints. If we do not consider them in our approaches, Gurobi can generate a solution of a similar quality (i.e., a gap value of 3 %) as the EA in about 1 hour. However, the EA is still efficient as long as the objective can be computed efficiently and incrementally. In a nutshell, the advantages of the EA are that it

- is more flexible (no linearization of objectives),
- is easier to implement (see Alg. 1),
- is free, whereas Gurobi requires a license,
- finds comparable solutions significantly faster,
- finds better solutions (1.7 % gap vs. 10.5 %),
- can be massively parallelized (running 857 EAs for 7 min each is similar to 100 h Gurobi).

While the result on the given data set is sufficient to showcase the feasibility of the approach, further research should study the problem on more real-world data sets. Furthermore, other metaheuristics, such as genetic algorithms, could be considered.

References

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* 112(1):3–41.

De Bruecker, P.; Van den Bergh, J.; Beliën, J.; and Demeulemeester, E. 2015. Workforce planning incorporating skills: State of the art. *European Journal of Operational Research* 243(1):1–16.

Ernst, A. T.; Jiang, H.; Krishnamoorthy, M.; and Sier, D. 2004. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research* 153(1):3–27.

Garey, M. R., and Johnson, D. S. 1975. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing* 4(4):397–411.

Globus, A.; Crawford, J.; Lohn, J.; and Pryor, A. 2004. A comparison of techniques for scheduling earth observing satellites. In *Proc. of AAAI*, 836–843.

Goldberg, A. V., and Tarjan, R. E. 1989. Finding minimum-cost circulations by canceling negative cycles. *Journal of the ACM* 36(4):873–886.

Gurobi Optimization, L. 2018. Gurobi optimizer reference manual.

Heimerl, C., and Kolisch, R. 2010. Scheduling and staffing multiple projects with a multi-skilled workforce. *OR spectrum* 32(2):343–368.

Jan, A.; Yamamoto, M.; and Ohuchi, A. 2000. Evolutionary algorithms for nurse scheduling problem. In *Proc. of CEC*, volume 1, 196–203. IEEE.

Li, J.; Burke, E. K.; Curtois, T.; Petrovic, S.; and Qu, R. 2012. The falling tide algorithm: a new multi-objective approach for complex workforce scheduling. *Omega* 40(3):283–293.

Mesghouni, K.; Hammadi, S.; and Borne, P. 2004. Evolutionary algorithms for job-shop scheduling. *International Journal of Applied Mathematics and Computer Science* 14(1):91–104.

Orlin, J. B. 1993. A faster strongly polynomial minimum cost flow algorithm. *Operations Research* 41(2):338–350.

Salewski, F.; Schirmer, A.; and Drexl, A. 1997. Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. *European Journal of Operational Research* 102(1):88–110.

Van den Bergh, J.; Beliën, J.; De Bruecker, P.; Demeulemeester, E.; and De Boeck, L. 2013. Personnel scheduling: A literature review. *European Journal of Operational Research* 226(3):367–385.