MEDICAL RESIDENTS SCHEDULING

Clemens Queckenberg, Dennis John, Felix Britzelmaier, Gaser Abdelaziz, Henning Erdweg, Luke Dreßen, Lynn Clemens, Simon Schürmann, Svenja Westphal, Theresa Täuber, Timothy Müller

Aachen, 12.07.2024

INTRODUCTION

- Postgraduate medical students need to complete additional training to become physicians
- Depending on the study program, students take different courses. Not all courses are given at all hospitals
- Therefore, they need to be assigned to hospitals to both work and further study in their program

PROBLEM

- The assignment has been done manually, so there is the need to automize and digitalize the scheduling process
- Plan needs to be readjusted
  - i.e. because of sickness, vacation,...

OBJECTIVES

Frontend for students and admins to input data

Database model to store and process data

Solve the underlying scheduling problem mathematically and generate study plan

Output the plans to students and admins

ALGORITHM
INTEGER LINEAR PROGRAM

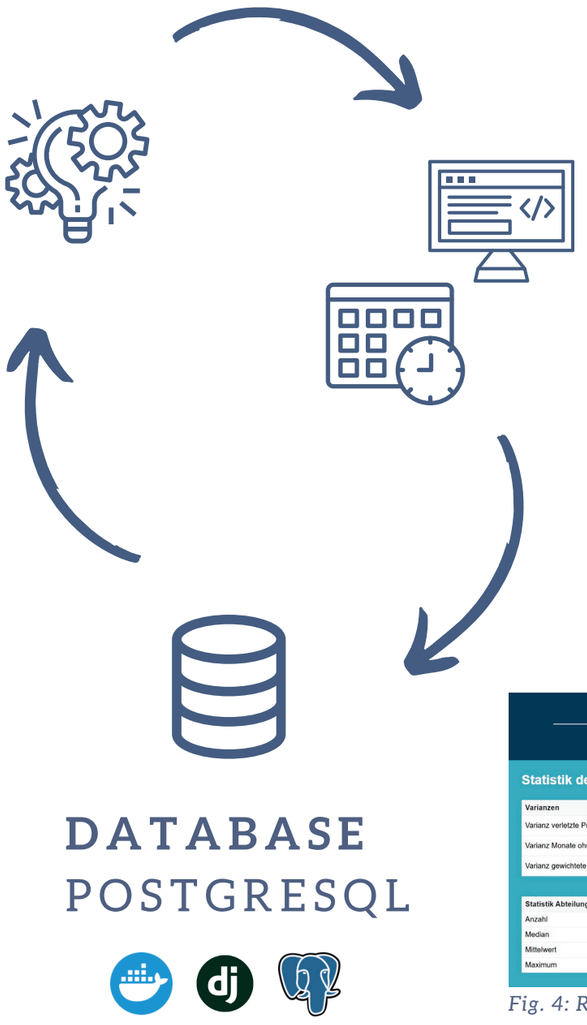
- Variables represent **all potential allocations** for each student and timeslot
- Restrictions /guidelines formulated as **linear constraints**
- Objective function: **weighted linear function** of individual objectives (e.g.: organizational and student preferences)
- (Potential) **optimal solution**
- **Warning**, if problem is infeasible
- Program offers **flexible adjustment**
- Can produce a solution
  - up to a certain **deviance** from the optimal solution, or
  - stop after a chosen amount of **time** (allows flexible scheduling)
- Solution represents feasible **allocation** of students to duty + training positions

```
# Every duty position has only the capacity for one student at the same time slot
#
for t_slot in range(amount_of_months):
    for h in hospitals:
        for d_pos in h_dpos_list[h]:
            if t_slot >= StartDate_to_Timeslot(d_pos.start_date) and t_slot <= EndDate_to_Timeslot(d_pos.end_date):
                model.addConstr(1 >= quicksum(x[t_slot, stud.id, h.id, str(h.id), d_pos.id] for stud in [stud for stud in persons for pp in stud_pp_list[stud]
                                                                                                         if  t_slot >= StartDate_to_Timeslot(pp.start_date)
                                                                                                         and t_slot <= EndDate_to_Timeslot(pp.end_date)
                                                                                                         and t_slot not in stud_uz_list[stud]
                                                                                                         and h.id in [aou.organisationsgruppe_id for aou in stud_pp_aou_list[stud, pp]]
                                                                                                         and pp.occupational_group.id in [og.occupational_group.id for og in h_dpos_og_list[h, d_pos]]]))
        for dep in h_dep_list[h]:
            for d_pos in h_dep_dpos_list[h, dep]:
                if t_slot >= StartDate_to_Timeslot(d_pos.start_date) and t_slot <= EndDate_to_Timeslot(d_pos.end_date):
                    model.addConstr(1 >= quicksum(x[t_slot, stud.id, h.id, dep.id, d_pos.id] for stud in [stud for stud in persons for pp in stud_pp_list[stud]
                                                                                                          if  t_slot >= StartDate_to_Timeslot(pp.start_date)
                                                                                                          and t_slot <= EndDate_to_Timeslot(pp.end_date)
                                                                                                          and t_slot not in stud_uz_list[stud]
                                                                                                          and h.id in [aou.organisationsgruppe_id for aou in stud_pp_aou_list[stud, pp]]
                                                                                                          and pp.occupational_group.id in [og.occupational_group.id for og in h_dep_dpos_og_list[h, dep, d_pos]]]))
```

Fig. 2: Code Snippet

$$\begin{aligned} \min \quad & x_1 + 2x_2 \\ \text{subject to} \quad & x_1 + x_2 \geq 3 \\ & 2x_2 \geq 4 \\ & x_1 \, , \, x_2 \in \mathbb{N}_0 \end{aligned}$$

Fig. 1: Example integer program



DATABASE
POSTGRESQL

FRONTEND
INPUT / OUTPUT

- Students:
  - Input of **priorities**
  - Input of **absence**
  - Evaluation of results
- Admins:
  - Input hospital
  - View / edit **hospitals**
  - Import / export button
  - Start evaluation with input of parameters
  - **Evaluation display** in detail
  - Evaluation display with colored blocks
- General:
  - Info page
  - Registration /Login

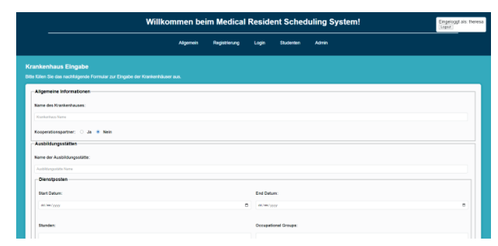


Fig. 3: Input mask hospital

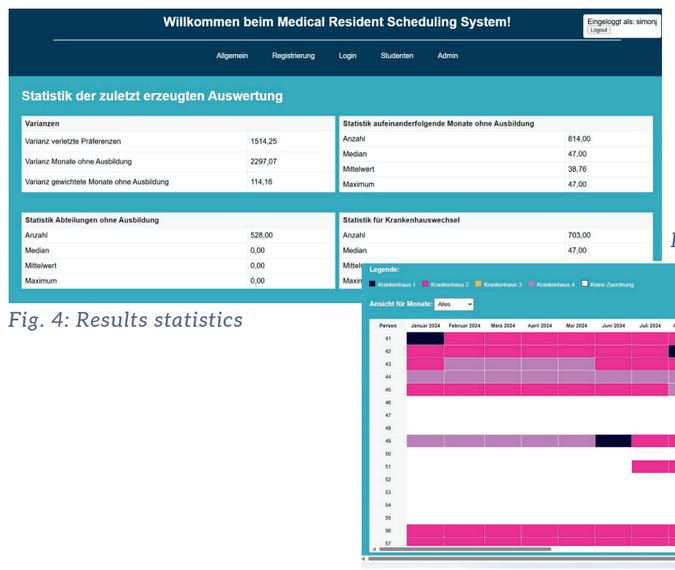


Fig. 4: Results statistics

Fig. 5: Admin results overview

FRAMEWORKS

Docker Django Gurobi Python PostgreSQL

RESULTS

- **Algorithm generates quickly a feasible solution**
- **Algorithm can be adjusted to strict or more flexible solutions**

RESOURCES

https://iconduck.com/icons/240263/docker
https://iconduck.com/icons/27845/python
https://iconduck.com/icons/27812/postgresql
https://www.cleanpng.com/png-django-web-development-web-framework-python-softwa-5166140/
https://github.com/Gurobi

FURTHER IDEAS

- Frontend improvements:
  - Check students (only real students can register)
  - Loading blocks with loading bar
  - Admin management of students
  - Calendar export function
  - Drag and drop of admin evaluation
  - More information on hospitals with priority assignment
  - Swap with other students
- More detailed explanation why this schedule is created
- Preprocessing the data to simplify objective function in algorithm