CS/ECE/ISyE 524 — Introduction to Optimization — Spring 2021

Final Course Project: Due 5/2/21, 11:59pm

Automating Agent Scheduling for the DoIT Help Desk

Ray Smith (rsmith52@wisc.edu)

Table of Contents

1. Introduction
    - A. Overview
    - B. Problem Background and Specifics
    - C. Problem Data
    - D. Report Outline
2. Mathematical Model
    - A. Decision Variables
    - B. Constraints
    - C. Objective Function
    - D. Standard Form
3. Solution
    - A. Required Packages
    - B. Schedule Rules
    - C. Data Input Reading
    - D. Helper Functions
    - E. Scheduling Model
    - F. Display Schedule Function
4. Results and Discussion

   A. Example Weekday Schedule
   B. Example Weekend Day Schedule
   C. Model Limitations

5. Conclusion

1. Introduction

1.A. Overview

The DoIT Help Desk (https://kb.wisc.edu/helpdesk/) collects availability from its student agents each semester in order to create a weekly schedule. In the current system, 1-2 people spend many hours manually looking over this availability as well as the shifts that need to be filled and build the schedule for the semester. There are many different types of roles needed throughout the day each requiring different levels of training. Because of this, not every agent can fill every role, and the shifts required for each role depends on the day and time of day. In this project I developed an integer programming model that generates a schedule when given a list of agent availabilities and shifts that need to be filled each day.

1.B. Problem background and specifics

This project looks at a type of group multi-role assignment problem, a complex version of the Assignment Problem (https://en.wikipedia.org/wiki/Assignment_problem). In this problem, a group of agents must be assigned to different roles, and all of the desired roles must be filled. However, an agent can only be scheduled for 1 role at a time. While the basic version of the assignment problem can be formulated and solved as a linear program, this version requires integer programming.

A slightly different version of this problem was explored in 2011 called Group Multi-Role Assignment with Coupled Roles (https://ieeexplore.ieee.org/document/8743229). While this problem slightly differs, one of the important details discussed is relevent. Each role that an agent can be assigned to has different requirements, agents are limited in what roles they can be assigned to. For DoIT, the roles we look at are:

- HDQA - Quality assurance agents require the most training, there is always 1 scheduled, and they are the go to person for all other agents scheduled at the time.
- Phones - The majority of agents scheduled at any given time are answering phone calls. This position requires no additional training.
- Chat/Email - There is usually 1-2 agents responding to chats and emails and these agents are required to have completed the chat/email training.
- Walk-In - The walk-in portion of the help-desk is only open a portion of the time the rest of the help desk is, and requires 2 specially trained agents when it is open.

The schedule is broken down into 30 minute periods, and is open from 7AM to 11PM. Agents are additionally required to work a minimum of 15 hours per week, and cannnot work more than 25 hours per week. One final restriction of an acceptable solution that isn't immediately obvious in a generated schedule is that agents should not be scheduled sporadically. It would be unreasonable to have someone scheduled for only 30 minutes at a time. We can summarize the schedule constraints then as follows:

- Enough agents must be scheduled at any given time to meet the required demand in each role
- Agents should only be scheduled at times they are available
- Agents should only be scheduled in roles they have training for
- Agents should be scheduled such that: 15 hours <= number hours scheduled per week <= 25 hours
- Any given shift for an agent should be >= 2 hours long

These are the base constraints, though there are a few more less important preferences that DoIT usually follows when building schedules. Those constraints will be touched on later in this report when the model is discussed in detail.

1.C. Problem data

The data used as input to this model comes directly from the scheduling tool that the Help Desk currently uses. As a senior employee I have access to this data and took the following data from the Spring 2021 semester:

- The hourly available throughout the week for each agent
- The daily shift requirements for each role on weekdays
- The daily shift requirements for each role on weekends
- Which trainings each agent has completed

This data has been organized into 4 files that serve as input to the schedule building model along with a few settings in the code itself.

- agent_training.json - This file holds a record of every agent, listed by their unique 4-letter identification code along with a list of trainings they have completed. This file should list every agent that might appear in a given availability data file, but it is ok for extra agents to be listed here. As new agents are hired and complete trainings, this file should be updated to reflect those changes.
- availability.json - This file holds the specific availabilities of each agent to be scheduled. Each agent's availability is broken down by 30 minute period and day of the week. This is the primary input for each semester that a schedule needs to be generated.
    - R: Red - The agent is not available at this time
    - Y: Yellow - The agent is available but would prefer not to work at this time
    - G: Green - The agennt is available and would prefer to work at this time
- week_req.csv - This file holds the list of roles needed on a

weekday along with the amount of agents required in each role throughout each time period of the day.
- weekend_req.csv - This file holds the list of roles needed on a weekend day along with the amount of agents required in each role throughout each time period of the day.

1.D. Report outline

The rest of this report will detail:

- The mathematical model of the schedule builder
- The implemented solution in julia
- The resulting schedule generated for the current data

2. Mathematical model

As mentioned earlier, this model is a complex version of the assignment problem. Our specific use case is scheduling agents in different roles for a help desk, but this general model could be expanded and applied to other types of businesses, or even to machines processing tasks.

2.A. Decision variables

This model has only a single decision variable, reffered to as sch but it contains a lot of information. This variable is a 4 dimensional binary matrix that contains an entry for every combination of:

- Agent with reported availability
- Day of the week
- Time period of the day
- Role or shift type

This decision variable is the schedule itself that the model assigns values to. An entry with a 1 shows that the corresponding agent is assignned that role for the given day and time period. This is a sparse matrix that will have mostly zeros.

2.B. Constraints

Most of the constraints used in the model depend on the input data in various ways. Rather than have numerous data matrixes referenced, I check against the data through a number of binary response helper functions. For example, to check if an agent is marked as available for a given day and time we check against: "agent_available(agent, day, time)".

The first constraint is simple but important. We ensure that each agent is only scheduled for 1 shift at any given time. An agent is not allowed to be on phones and walk-in at once for example.

```
for each day, time: sum(sch[agent, day, tim
e]) <= 1
```

The next basic constraint is that we only allow agents to be scheduled when their availability allows it. We are allowing scheduling whenever an agent is available, regardless of preference. That is, we are showing no difference between Green and Yellow availability.

```
for each agent, day, time, role:
    sch[agent, day, time, role] <= agent_av
ailable(agent, day, time)
```

The next 3 constraints work to ensure that there is the correct number of agents scheduled for each role at any given time. We check that the sum of entries in the schedule over all agents for a given day, time, and shift adds up to be at least the required amount of agents for that day, time, and shift. At the same time, we check that we only count agents with the required training for that shift. Additionally, we don't want to schedule too many agents, so we also check that the number of scheduled agents is not much higher than the required amount. We set a flexibility setting for how many extra agents are allowed by role. Finally, there is a special consideration where at any given time, we require 1 of the agents scheduled for phones to have "Advanced Phone Training".

```
for each day, time, role: sum(sch[agent] ∗
meets_requirements(agent, role)) >= demand
for each day, time, role: sum(sch[agent])
<= demand + flexibility
for each day, time: sum(sch[agent, "Phone
s"] ∗ adv_phones_trained(agent)) >= 1
```

The most complex constraint is the one that restricts the minimum duration of any given shift for each agent. A helper function is able to return the "neighboring shifts" to check that if an agent is scheduled at a given time, there should be a number of shifts around it that add up to at least the minimum shift length (2 hours = 4 half-hour time periods) where the agent is also scheduled.

```
for each agent, day, time, role:
    sum(sch in neighbor_shifts) >= sch[agen
t, day, time, role] ∗ 4
```

The last regular set of constraints ensures that each agent gets enough hours, but not too many hours. The minimum number of hours worked is typically set at 15 (30 time periods), but if an agent has less availability than that, it is allowed by special case. The maximum number of hours worked for each agent is set at 25 (50 time periods).

```
for each agent: sum(sch[agent]) >= 30
for each agent: sum(sch[agent]) <= 50
```

The remaining constraints are highly specific to the ideal schedule used by the DoIT Help Desk. It is requested that each agent trained for the HDQA role is ensured to be scheduled in that role for at least 4 hours (8 time periods) each week and to ensure a fair balance, no one agent gets more than 12 hours (24 time periods) in the HDQA role throughout the week.

```
for each agent: sum(sch[day, time, "hdqa"])
>= 8 ∗ hdqa_trained(agent)
for each agent: sum(sch[day, time, "hdqa"])
<= 24 ∗ hdqa_trained(agent)
```

Finally, agents who have received the highest level of training and are in student leadership positions (SLP) should not be scheduled in the basic phones or chat/email shifts. We ensure that each student with this training can't be assigned to phones or chat/email shifts.

```
for each agent, day, hour: sum(sch["phone
s"]) <= not_slp_trained(agent)
for each agent, day, hour: sum(sch["chat/em
ail"]) <= not_slp_trained(agent)
```

2.C. Objective function

Since most of the work is done in the constraints that force the schedule to take an acceptable form, the objective function is actually very simple. As long as all the required rules are followed and demand for each type of agent is met, a schedule is valid. We then want to choose a schedule that requires the least amount of total agent hours, to save DoIT the most money. Thus, the objective function is just minimizing the sum of the entire schedule decision variable: sch.

```
min sum(sch[agent, day, time, role])
```

2.D. Standard form

$$
\begin{array}{rl}
\underset{sch \in \mathbb{R}^4}{\text{minimize}} & \text{sum}(sch) \\
\text{subject to:} & \text{sum}(sch[a,i,j]) <= 1 \\
 & sch[a,i,j,k] \leq avail[a] \\
 & \text{sum}(sch[a] * req[a,k]) \geq dem[i,j,k] \\
 & \text{sum}(sch[a]) \leq dem[i,j,k] + flex[k] \\
 & \text{sum}(sch[a, k=2] * adv[a]) \geq 1 \\
 & \text{sum}(sch[a,i,j \in J_{near},k]) \geq 4 * sch[a,i,j,k] \\
 & \text{sum}(sch[a]) \geq 30 \\
 & \text{sum}(sch[a]) \leq 50 \\
 & \text{sum}(sch[a, k=1]) \geq 8 * req[a, k=1] \\
 & \text{sum}(sch[a, k=1]) \leq 24 * req[a, k=1] \\
 & \text{sum}(sch[a,i,j,k=2]) \leq \text{not } slp[a] \\
 & \text{sum}(sch[a,i,j,k=3]) \leq \text{not } slp[a] \\
 & \\
 & sch \\
 & \\
\text{where:} & A = 1 \ldots n \\
 & I = 1 \ldots 7 \\
 & J = 1 \ldots 32 \\
 & K = 1 \ldots 4 \\
 & J_{near} = j_{j-4} \ldots j_{j+4}
\end{array}
$$

Where data matrixes are defined such that:

- $avail$ contains each agent's availability
- req contains whether an agent can do a role or not (hdqa, phones, chat/email, walk-in)
- dem contains the required number of agents for each day, time period, and role

3. Solution

3.A. Required packages

First we must import the required packages. Depending on the machine this is being run on, the gurobi home and license file paths might need to be updated.

```
In [1]:
```

```julia
# Imports

ENV["GUROBI_HOME"] = "/Library/gurobi911/mac64"
ENV["GRB_LICENSE_FILE"] = "/Library/gurobi911/guro
bi.lic"
import Pkg
Pkg.add("Gurobi")
Pkg.build("Gurobi")

using Gurobi
using JuMP
using CSV
using JSON
using DataFrames
using OrderedCollections
```

```
   Updating registry at `~/.julia/regi
stries/General`



   Updating git-repo `https://github.c
om/JuliaRegistries/General.git`


  Resolving package versions...
No Changes to `~/.julia/environments/v
1.5/Project.toml`
No Changes to `~/.julia/environments/v
1.5/Manifest.toml`
   Building Gurobi → `~/.julia/package
s/Gurobi/JcjAE/deps/build.log`
┌ Info: Precompiling Gurobi [2e9cd046-
0924-5485-92f1-d5272153d98b]
└ @ Base loading.jl:1278
```

3.B. Schedule rules

This section defines a collection of variables for rules the model will follow. This includes the structure of the schedule, defining trainings needed for an agent to work each type of shift, minimum and maximum hours agents can work each week, etc.

In [2]:

```
# Schedule Rules

# General Schedule Layout
hours = 1:32
hours_real = 7:0.5:22.5
days = 1:7

# Shift Types and Training Requirements
shifts = ["HDQA", "Phones", "Chat/Email", "Walk-I
n"]
training_levels = ["Pick 1", "Walk-In", "Pick 3",
"Chat/Email", "HDQA", "SLP"]
requirements = OrderedDict()
requirements[shifts[1]] = [training_levels[5]]
requirements[shifts[2]] = [training_levels[1]]
requirements[shifts[3]] = [training_levels[4]]
requirements[shifts[4]] = [training_levels[2]]

# Allowed Flexibility in (Extra) Agents Scheduled
flexibility = OrderedDict()
flexibility[shifts[1]] = 0 # Only ever 1 HDQA
flexibility[shifts[2]] = 2 # Extra phones agents u
sually ok
flexibility[shifts[3]] = 0
flexibility[shifts[4]] = 1 # Walk-In has limited s
pace

# Other Rules
adv_phones_req = 1 # min number of pick 3 trained
agents scheduled over a phones shift
min_shift_length = 4 # minimum shift duration in 1
/2 hour blocks
agent_min = 30 # minimum scheduled shifts/agent in
1/2 hour blocks
agent_max = 50 # maximum schedule shifts/agent in
1/2 hour blocks
sch_yellow = true # allow scheduling of agents whe
n availability is marked yellow
```

```
min_hdqa_shifts = 8 # minimum hdqa shifts each eli
gible agent gets
max_hdqa_shifts = 24 # maximum hdqa shifts each el
igible agent gets
```

Out[2]:

```
24
```

3.C. Data input reading

The schedule generated depends entirely on the agents available to work and the demand for each role at any given time. This section reads this information from the .csv and .json files to fill dictionaries that the model can reference.

This code depends on the 4 data files listed previously being available in the same directory.

In [3]:

```
# Data Input Reading

# Read the agent demands for each role, which vari
es between the week and weekend
raw_week = CSV.read(joinpath(@__DIR__, "week_req.c
sv"), DataFrame)
week_shift_types = names(raw_week)
week = Matrix(raw_week)
raw_weekend = CSV.read(joinpath(@__DIR__, "weekend
_req.csv"), DataFrame)
weekend_shift_types = names(raw_weekend)
weekend = Matrix(raw_weekend)
# Store and format the data in easy to access dict
ionaries
# - day[hour][shift] = # agents demanded of that t
ime at that time
weekday = [OrderedDict() for i in hours]
weekendday = [OrderedDict() for i in hours]
for i in hours
    j = 1
    for shift in week_shift_types
        weekday[i][shift] = week[i, j]
        j += 1
    end
    k = 1
    for shift in weekend_shift_types
        weekendday[i][shift] = weekend[i, k]
        k += 1
    end
end

# Read the agent availability data for each time p
eriod
string_data = join(readlines("availability.json"))
raw_avail = JSON.parse(string_data)
agents = keys(raw_avail) # list of all agents who
submitted availability
# Store and format the data in an easy to access d
```

```
ictionary
# - avail[agent] = dictionary by time period/day o
f R/Y/G availability
avail = OrderedDict()
for agent in agents
    raw_agent_avail = raw_avail[agent]
    agent_avail = OrderedDict()
    for i in hours
        agent_avail[i] = raw_agent_avail[i]
    end
    avail[agent] = agent_avail
end

# Read the agent training data for what role each
agent can do
string_data = join(readlines("agent_training.jso
n"))
raw_trainings = JSON.parse(string_data)
all_agents = keys(raw_trainings) # list of all age
nts with listed trainings
# Store and format the data inn an easy to access
dictionary
# - trainings[agent] = list of all trainings the a
gent has
trainings = OrderedDict()
for agent in all_agents
    raw_agent_trainings = raw_trainings[agent]
    base_job = raw_agent_trainings[:"Base"]
    agent_trainings = raw_agent_trainings[:"Traini
ngs"]
    if startswith(base_job, "SLP")
        append!(agent_trainings, ["SLP"])
    end
    trainings[agent] = agent_trainings
end
```

3.D. Helper functions

This section contains a number of helper functions that mostly return binary results about the data to the model, allowing the constraints to be simpler to understand. Each function is described with a comment above it.

In [4]:

```
# Model Helper Functions

# This function returns true if the agent has the
# specified training and false otherwise
function has_training(agent, training_level)
    return training_level in trainings[agent]
end

# This function returns true if the agennt has all
# required trainings for the specified type of shi
ft
# and false otherwise
function meets_requirements(agent, shift)
    shift_req = requirements[shift]
    for training in shift_req
        if !has_training(agent, training)
            return false
        end
    end
    return true
end

# This function returns true if the agent is avail
able
# to be scheduled at a given day/time period
# Green + Yellow vs. just Green counting as availa
ble
# is defined in the rules section
function shift_available(agent, day, time)
    agent_avail = avail[agent]
    time_slice = agent_avail[time]
    color = time_slice[day]

    if color == "G" || (color == "Y" && sch_yello
w)
        return true
    else
        return false
    end
end

# This function returns a list of roles/shift type
s
# with any demand in a given day of the week
function get_shift_types(day)
    if (day == 1 || day == 7)
        return weekend_shift_types
    else
        return week_shift_types
    end
end

# This function returns the number of agents deman
ded
# for each role/shift type on a given day of the w
eek
function get_day_req(day)
    if (day == 1 || day == 7)
        return weekendday
    else
        return weekday
    end
end

# This function counts up and returns the total
# number of available hours that an agent has
function get_total_hours(agent)
    total = 0
    for day in days
        for hour in hours
            if (shift_available(agent, day, hour))
                total += 1
            end
        end
    end
    return total
end

# This function returns the minimum value between
# the rule set minimum hours for agents in a week
# and the total hours an agent is scheduled for
function get_min_hours(agent)
    return min(get_total_hours(agent), agent_min)
end

# This function returns a list of time periods tha
t
# neighbor a given time period in both directions,
# length defined by the minimum length shift set
# in the rules
function get_neighbor_shifts(time)
    range = []
    for hour in hours
        if abs(hour - time) < min_shift_length
            append!(range, hour)
        end
    end
    return range
end

# This function calculates the maximum number of a
gents
# allowed to be scheduled for a given shift at a c
ertain
# day/time period
function get_max_allowed(shift, day, time)
    shift_types = get_shift_types(day)
    day_req = get_day_req(day)

    base_needed = day_req[time][shift]
    if base_needed == 0
        return 0
    else
        return base_needed + flexibility[shift]
    end
end
```

Out[4]:

```
get_max_allowed (generic function with
1 method)
```

3.E. Scheduling model

This section holds the actual JuMP model using Gurobi optimizer. The decision, variables, constraints, and objective function are all as defined in the above sections, and listed in the same order as they were referenced in earlier.

In [5]:

```
# Role Assigning Scheduling Model

# Set optimizer to be Gurobi for MIP
m = Model(Gurobi.Optimizer)
set_optimizer_attribute(m, "OutputFlag", 0)

#--------------------------------#
# Decision Variables
#--------------------------------#
# If agent scheduled and for what role
@variable(m, sch[agents, days, hours, shifts], Bi
n)

#--------------------------------#
# Constraints
#--------------------------------#
# Only allow each agent to be scheduled for 1 shif
t at a time
for agent in agents
    for day in days
        for hour in hours
            @constraint(m, sum(sch[agent, day, hou
r, shift] for shift in shifts) <= 1)
        end
    end
end

# Only allow agents to be scheduled when green/yel
low
for agent in agents
    for day in days
        for hour in hours
            for shift in shifts
                @constraint(m, sch[agent, day, hou
r, shift] <= shift_available(agent, day, hour))
            end
        end
    end
```

```
end

# Shift requirement constraints
for day in days
    shift_types = get_shift_types(day)
    day_req = get_day_req(day)

    for hour in hours
        for shift in shift_types
            # Ensure enough trained agents on each 
shift
            @constraint(m, sum(sch[agent, day, hou
r, shift] * meets_requirements(agent, shift) for a
gent in agents) >= day_req[hour][shift])

            # Ensure no excessive scheduling of ex
tra workers
            @constraint(m, sum(sch[agent, day, hou
r, shift] for agent in agents) <= get_max_allowed
(shift, day, hour))
        end

        # Advanced phone agent requirement
        @constraint(m, sum(sch[agent, day, hour, "
Phones"] * has_training(agent, "Pick 3") for agent
in agents) >= adv_phones_req)
    end
end

# Don't allow shifts of < minimum duration
for agent in agents
    for day in days
        for hour in hours
            for shift in shifts
                @constraint(m, sum(sch[agent, day,
h, shift] for h in get_neighbor_shifts(hour)) >= s
ch[agent, day, hour, shift] * min_shift_length)
            end
        end
    end
end
```

```
# Ensure each agent works within the min/max allow
ed hour range
for agent in agents
    @constraint(m, sum(sch[agent, day, hour, shif
t] for day in days, hour in hours, shift in shift
s) >= get_min_hours(agent))
    @constraint(m, sum(sch[agent, day, hour, shif
t] for day in days, hour in hours, shift in shift
s) <= agent_max)
end

# Minimum HDQA Shift Requirement
for agent in agents
    @constraint(m, sum(sch[agent, day, hour, "HDQ
A"] for day in days, hour in hours) >= min_hdqa_sh
ifts * has_training(agent, "HDQA"))
    @constraint(m, sum(sch[agent, day, hour, "HDQ
A"] for day in days, hour in hours) <= max_hdqa_sh
ifts * has_training(agent, "HDQA"))
end

# No SLPs on Phones or Chat/Email Shifts
for agent in agents
    for day in days
        for hour in hours
            @constraint(m, sch[agent, day, hour, "
Phones"] <= !has_training(agent, "SLP"))
            @constraint(m, sch[agent, day, hour, "
Chat/Email"] <= !has_training(agent, "SLP"))
        end
    end
end

#--------------------------------#
# Objective Function
#--------------------------------#
# Schedule the minimum amount of agents possible
@objective(m, Min, sum(sch))

# Solve the scheduling problem
```

```
optimize!(m)
println("Total Hours Scheduled: ", sum(value.(sc
h)))

#--------------------------------#
# For Reference
#--------------------------------#
# 7542 is when scheduling all agents using all gre
en availability
# 12092 is when scheduling all agennts using all g
reen/yellow availability
```

```
--------------------------------------
------
Warning: your license will expire in 5
days
--------------------------------------
------

Academic license - for non-commercial
use only - expires 2021-05-06
Total Hours Scheduled: 2218.0
```

3.F. Display schedule function

This section is used in the results section to display the daily schedules produced by the model.

In [6]:

```
# Display Schedule Function

# This  function returns a formatted dataframe tab
le for
# the provided day of the provided schedule
function display_schedule(sch, day_to_show)
    X = value.(sch)

    schedule = OrderedDict()
    for day in days
        shift_types = get_shift_types(day)
        schedule[day] = OrderedDict()
        for shift in shift_types
            schedule[day][shift] = OrderedDict()
            for hour in hours
                schedule[day][shift][hour] = []
                for agent in agents
                    if X[agent, day, hour, shift]
== 1
                        append!(schedule[day][shif
t][hour], [agent])
                    end
                end
            end
        end
    end

    shift_types = get_shift_types(day_to_show)
    data = schedule[day_to_show]
    cols = []
    for shift in shift_types
        col = []
        for hour in hours
            append!(col, [data[shift][hour]])
        end
        append!(cols, [col])
    end
    if length(shift_types) == 4
```

```
        df = DataFrame(time = hours_real, hdqa = c
ols[1], phones = cols[2], chat = cols[3], walkin =
cols[4])
    elseif length(shift_types) == 2
        df = DataFrame(time = hours_real, hdqa = c
ols[1], phones = cols[2])
    end

    return df
end

# Display Settings
ENV["COLUMNS"]=120
ENV["LINES"] = 32
```

Out[6]:

```
32
```

4. Results and discussion

This project's results are not easy to objectively quantify, but we can look at the individual days of the optimal schedule found and see how the agent assignments compare to the criteria we set at the beginning, and also see if subjectively it looks like a reasonable schedule.

- Enough agents must be scheduled at any given time to meet the required demand in each role
  - Checking against the provided data file, every role has enough agents at each time period each day of the week.
- Agents should only be scheduled at times they are available
  - While there are ~80 agents on this schedule, after manually checking about 10 of them, there are no scheduled agents at times where they marked unavailable. Running a code check confirms that no agents are scheduled except when they marked "Green" or "Yellow" availability.
- Agents should only be scheduled in roles they have training for
  - This is harder to check, but by iterating through everyone that appears in the HDQA column for each day to ensure they have HDQA training, and repeating for Chat/Email and Walk-In roles, we can confirm in fact that only properly trained agents are assigned to these roles.
- Agents should be scheduled such that: 15 hours <= number hours scheduled per week <= 25 hours
  - Again, I looked manually at about 10 agents to confirm none had too few or too little hours. As we are trying to schedule the minimum required number of agents, most only received 15 hours. No agents received more than 25 hours, and the only agents who had less than 15 had marked less than 15 total hours available in the first place.
- Any given shift for an agent should be >= 2 hours long

- This is the easiest to detect at a glance. No 4 letter code appears less than 4 times in a row at any given time throughout the generated schedule.

Below we see 2 of the scheduled days displayed, weekday and weekend day examples. As an employee of the DoIT Help Desk who has seen a lot of schedules my first hand experience confirms these results do look similar to the typical schedules I've seen, and my personal scheduled shifts ("RAYS" being my 4 letter code) for this semester actually line up almost exactly with my shifts I was manually scheduled for.

4.A. Example weekday schedule

In [7]:

```
# Friday Schedule
display_schedule(sch, 6)
```

Out[7]:

32 rows × 5 columns

	time	hdqa	phones	chat	walkin
	Float64	Any	Any	Any	Any
1	7.0	["TBOS"]	["IAND", "LBAN", "CKLE"]	[]	[
2	7.5	["TBOS"]	["IAND", "LBAN", "CKLE"]	[]	[
3	8.0	["TBOS"]	["IAND", "STEF", "LBAN", "CKLE"]	["PAKA", "AWGR"]	[
4	8.5	["TBOS"]	["IAND", "STEF", "LBAN", "CKLE", "SAWA"]	["PAKA", "AWGR"]	[
5	9.0	["TBOS"]	["STEF", "CKLE", "SAWA", "JLHE", "RWAS"]	["PAKA", "AWGR"]	["BCON", "JESO"
6	9.5	["SMDO"]	["STEF", "SAWA", "JLHE", "RWAS", "NINA"]	["PAKA", "AWGR"]	["BCON", "JESO"

	time	hdqa	phones	chat	walkin
	Float64	Any	Any	Any	Any
7	10.0	["SMDO"]	["STEF", "SAWA", "JLHE", "GENI", "RWAS", "NINA"]	["AWGR", "VRNA", "HAMC"]	["BCON", "JESO"]
8	10.5	["SMDO"]	["MJMO", "JLHE", "GENI", "RWAS", "GUZM", "NINA"]	["AWGR", "VRNA", "HAMC"]	["ALZA", "BCON", "JESO"]
9	11.0	["SMDO"]	["MJMO", "JLHE", "GENI", "RWAS", "GUZM", "NINA"]	["VRNA", "JLEE", "HAMC"]	["ALZA", "BCON", "JESO"]
10	11.5	["AARK"]	["CKLE", "MJMO", "JLHE", "GENI", "RWAS", "GUZM"]	["VRNA", "JLEE", "HAMC"]	["ALZA", "BCON", "JESO"]
11	12.0	["AARK"]	["CKLE", "ADWI", "MJMO", "JLHE", "GENI", "RWAS", "GUZM"]	["SPEK", "JLEE", "HAMC"]	["ALZA", "BCON"]
12	12.5	["AARK"]	["MRDA", "CKLE", "ADWI", "GENI", "RWAS", "GUZM"]	["SPEK", "JLEE", "HAMC"]	["BCON", "PETU"]

	time	hdqa	phones	chat	walkin
	Float64	Any	Any	Any	Any
13	13.0	["AARK"]	["ELLI", "MRDA", "CKLE", "ADWI", "RWAS"]	["SPEK", "JLEE"]	["BCON", "PETU"]
14	13.5	["LKRE"]	["KTEM", "IAND", "DANI", "ELLI", "MRDA", "ADWI"]	["SPEK", "JLEE"]	["BCON", "PETU", "RAYS"]
15	14.0	["LKRE"]	["KTEM", "IAND", "DANI", "ELLI", "MRDA", "ADWI"]	["LBAN", "SPEK"]	["BCON", "PETU", "RAYS"]
16	14.5	["LKRE"]	["KTEM", "MOME", "IAND", "DANI", "ELLI", "ADWI"]	["LBAN"]	["BCON", "RAYS"]
17	15.0	["LKRE"]	["KTEM", "MOME", "IAND", "DANI", "MJMO"]	["LBAN"]	["BCON", "KELL", "RAYS"]
18	15.5	["LKRE"]	["KTEM", "RIHA", "MOME", "IAND", "MJMO"]	["LBAN"]	["BCON", "KELL", "RAYS"]

	time	hdqa	phones	chat	walkin
	Float64	Any	Any	Any	Any
19	16.0	["PETU"]	["KTEM", "RIHA", "TATO", "MOME", "IAND", "MJMO"]	["LBAN"]	["BCON", "KELL", "RAYS"]
20	16.5	["PETU"]	["FOER", "RIHA", "TATO", "MJMO"]	["HAMC"]	["BCON", "KELL", "RAYS"]
21	17.0	["PETU"]	["FOER", "RIHA", "TATO", "STEF", "HAYA"]	["HAMC"]	[]
22	17.5	["PETU"]	["FOER", "TATO", "STEF", "HAYA"]	["HAMC"]	[]
23	18.0	["RWER"]	["FOER", "OCLA", "STEF", "HAYA"]	["HAMC"]	[]
24	18.5	["RWER"]	["RIHA", "OCLA", "STEF", "HAYA", "LBAN", "MNEL"]	["DUNN"]	[]
25	19.0	["RWER"]	["RIHA", "OCLA", "LBAN", "MNEL"]	["DUNN"]	[]

time	hdqa	phones	chat	walkin
Float64	**Any**	**Any**	**Any**	**Any**
		["RIHA", "OCLA"		

4.B. Example weekend day schedule

In [8]:

```
# Sunday Schedule
display_schedule(sch, 1)
```

Out[8]:

32 rows × 3 columns

	time	hdqa	phones
	Float64	Any	Any
1	7.0	["CACH"]	["DRDO", "BING"]
2	7.5	["CACH"]	["DRDO", "BING"]
3	8.0	["CACH"]	["DRDO", "BING"]
4	8.5	["CACH"]	["DRDO", "BING", "NINA"]
5	9.0	["EWJE"]	["NINA", "JKLO"]
6	9.5	["EWJE"]	["NINA", "JKLO"]
7	10.0	["EWJE"]	["NINA", "JKLO"]
8	10.5	["EWJE"]	["NINA", "JKLO"]
9	11.0	["HUNT"]	["AARK", "JKLO"]
10	11.5	["HUNT"]	["AARK", "JKLO"]
11	12.0	["HUNT"]	["AARK", "JLEE", "JKLO"]
12	12.5	["HUNT"]	["AARK", "JLEE", "JKLO"]
13	13.0	["STAD"]	["KTEM", "JLEE"]
14	13.5	["STAD"]	["KTEM", "JLEE"]
15	14.0	["STAD"]	["KTEM", "JJBE", "JLEE"]
16	14.5	["STAD"]	["KTEM", "JJBE"]
17	15.0	["EWJE"]	["KTEM", "JJBE"]
18	15.5	["EWJE"]	["KTEM", "JJBE"]

	time	hdqa	phones
	Float64	Any	Any
19	16.0	["EWJE"]	["MORI", "FOER"]
20	16.5	["EWJE"]	["MORI", "FOER"]
21	17.0	["EWJE"]	["MORI", "FOER"]
22	17.5	["HOST"]	["MORI", "FOER"]
23	18.0	["HOST"]	["MORI", "FOER", "SIEL"]
24	18.5	["HOST"]	["MORI", "SIEL"]
25	19.0	["HOST"]	["MORI", "SIEL"]
26	19.5	["HOST"]	["MORI", "BANT", "SIEL"]
27	20.0	["HOST"]	["MORI", "BANT"]

4.C. Model limitations

While this model does succesfully generate a technically valid schedule that would work for the DoIT Help Desk, it does have limitations in its current form. Typically, agents have a preference for a certain kind of shift/role that they work most of their hours in, even if they have training to work other types of roles. In the generated schedule, a lot of agennts are assigned to Walk-In who typically never work at Walk-In, but do have the training to do it for example. Additionally, there is no guarantee that agents' hours are spread evenly throughout their availability. An agent might have all 15 of their hours in the first day or two of the week, instead of having them spread out evenly which is likely preferred. Finally, when agents submit availabbility, they also submit a range of preferred hours. Some agents are looking for more hours, while others look for less. The data I was able to gather for availability did not include these preferences, so we assume all agents have no preference in the 15-25 hour range.

5. Conclusion

In summary, I think this schedule building model is a success that could be used in its current state by DoIT as at least a starting point for weekly semester schedules. I have already been talking with management and a student in a leadership position who works on the schedule application to possibly implement this project into the new system that is being worked on.

As mentioned in the limitations section, given more time I would like to take into account each agents' preferred amount of hours. This would just require collecting and formatting that data. Additionally I would like to distinguish between "Green" (preferred availability) and "Yellow" (non-preferred availability) marked time periods in the availability data. The objective function could be made more complex to weigh the preference of hours scheduled, so a more "optimal" schedule takes into account the most personal preferences being fulfilled.