GUROBI
OPTIMIZATION

Tips and tricks for optimal scheduling

Thank you for joining us. We will be starting shortly.

GUROBI
OPTIMIZATION

Welcome to the webinar

Tips and tricks for optimal scheduling

Speaker introduction

Dr. Nicole Taheri

- Consultant at End-to-End Analytics
- PhD in Computational and Mathematical Engineering from Stanford University
- Extensive experience applying operations research and mathematical modeling methods to find optimal solutions in a wide range of applications
- Work has centered around analyzing complex data to transform data insights into actionable recommendations that improve business processes

End-to-End Analytics

We partner with our customers to solve complex business challenges bringing the right balance of management consulting, analytics and technology.

Consulting

Management consulting focused on business strategy and process adoption

Analytics

Robust yet practical analytics to answer the hardest questions

Technology

Right-sized technology tailored to enable your business needs

- **Company:** Founded 2005, based in Palo Alto, CA
- **Team:** 60+ professional staff, ∼20 PhDs from major universities, resources in CA, MI, MA, Brazil, Peru, Hong Kong, and China
- **Work:** Over 75 clients for more than 700 projects to date, 30+ published articles, more than 15 patents

Agenda

- Overview of scheduling problems
  - Sample problem
- Solving with Gurobi
  - Modeling & Formulation
  - Implementation in Gurobi (Python)
  - Useful features for defining constraints
    - Special-ordered sets (SOS)
    - General constraints: MAX, AND, OR, and INDICATOR
  - Parameter tuning
- Wrap up & Questions

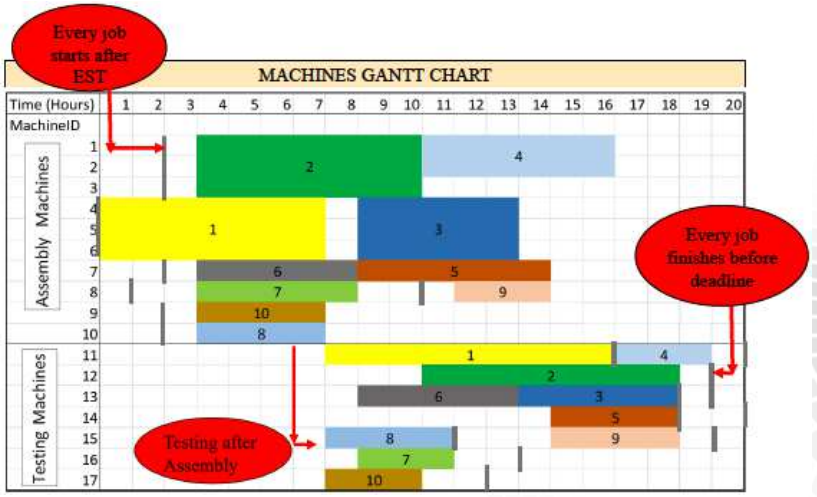
Scheduling problems

Definition

Determine the time periods in which to schedule a set of events to optimize a chosen objective

Examples include:

- Job shop scheduling (assigning jobs to machines in order to minimize makespan)
- Scheduling airline flights
- Assigning nurses to shifts in a hospital



Scheduling problems

Finding an optimal schedule is a combinatorial optimization problem that is NP-complete.

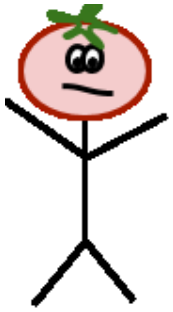
- An optimization problem consists of:
  - Objective
  - Constraints
  - Decision variables
  - Input data/parameters
- A Mixed-Integer Progam (MIP) is optimization problem with a mix of integer and continuous decision variables
- Scheduling decision variables are binary, either 0 or 1
- The characteristics of the constraints determine if the problem is a linear or non-linear program

Scheduling problems can be formulated as a mixed-integer program and often solved in reasonable time with an optimization solver like Gurobi.

Sample problem

Sample scheduling problem: **The diner**

How should the diner staff burger-makers to minimize costs?



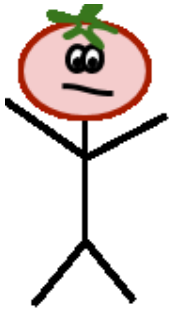
Tom Ato has an hourly rate of $10

Bri Ochebun has an hourly rate of $15

	8am	9am	10am	11am	12pm	1pm	2pm	3pm	4pm	5pm	6pm	7pm	8pm	9pm
# of burger makers needed	1	1	1	2	2	2	1	1	1	2	1	1	1	1

Sample scheduling problem: **The diner**

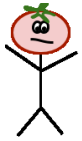
How should the diner staff burger-makers to minimize costs?



Tom Ato has an hourly rate of $10

Bri Ochebun has an hourly rate of $15

	8am	9am	10am	11am	12pm	1pm	2pm	3pm	4pm	5pm	6pm	7pm	8pm	9pm
# of burger makers needed	1	1	1	2	2	2	1	1	1	2	1	1	1	1



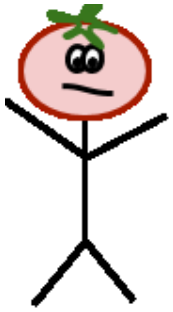
$140

$60

Total: $200

Sample scheduling problem: **The diner**

How should the diner staff burger-makers to minimize costs and make sure all hours worked are consecutive?



Tom Ato has an hourly rate of $10

Bri Ochebun has an hourly rate of $15

	8am	9am	10am	11am	12pm	1pm	2pm	3pm	4pm	5pm	6pm	7pm	8pm	9pm
# of burger makers needed	1	1	1	2	2	2	1	1	1	2	1	1	1	1

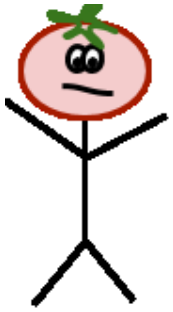
$140

$105

Total: $245

Sample scheduling problem: **The diner**

How should the diner staff burger-makers to minimize costs and make sure all hours worked are consecutive when an employee can work a max of 11 hours/day?



Tom Ato has an hourly rate of $10

Bri Ochebun has an hourly rate of $15

	8am	9am	10am	11am	12pm	1pm	2pm	3pm	4pm	5pm	6pm	7pm	8pm	9pm
# of burger makers needed	1	1	1	2	2	2	1	1	1	2	1	1	1	1

$110

$150

Total: $260

Characteristics of scheduling problems

Generally, constraints develop from restrictions on:

- consecutiveness of periods in a shift
- maximum and minimum limits on individual assignments
- minimum on overall required resources
- maximum on number of shifts
- simultaneous and nonsimultaneous scheduling
- ordering of jobs or shifts

Gurobi has an MIP solver with many useful features for modeling and solving scheduling problems

Formulation & Gurobi Implementation

Cooking up the formulation

minimize cost

subject to
- each burger-maker must work less than the maximum hours
- number of burger-makers at any time must be at least the minimum required
- burger-maker scheduled hours must be consecutive

Cooking up the formulation

Variable	Description	Type
$B \in \mathbb{Z}$	number of burger-makers	parameter
$T \in \mathbb{Z}$	number of time periods	parameter
$r \in \mathbb{R}^B$	burger-maker hourly rate	parameter
$\ell \in \mathbb{Z}^T$	minimum required hourly workers	parameter
$u \in \mathbb{R}$	maximum daily work hours	parameter
$s \in \{0,1\}^{B \times T}$	burger-maker shift schedule	decision variable
$w \in \{-1,0,1\}^{B \times T}$	shift changes	intermediary variables
$v \in \{0,1\}^{B \times T}$	shift starts	intermediary variables

minimize	$\sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t}$		**cost**
subject to	$\sum_{t=1}^{T} s_{b,t} \leq u$	$\forall b$	**max hours**
	$\sum_{b=1}^{B} s_{b,t} \geq \ell_t$	$\forall t$	**min workers**
	$w_{b,t} = s_{b,t} - s_{b,t-1}$	$\forall t > 0, b$	**shift changes**
	$w_{b,0} = s_{b,0}$	$\forall b$	
	$v_{b,t} = \max(0, w_{b,t})$	$\forall t, b$	**shift starts**
	$\sum_{t=1}^{T} v_{b,t} \leq 1$	$\forall b$	**consecutive shifts**

Gurobi implementation: Parameters

Variable	Description	Type
$B \in \mathbb{Z}$	number of burger-makers	parameter
$T \in \mathbb{Z}$	number of time periods	parameter
$r \in \mathbb{R}^B$	burger-maker hourly rate	parameter
$\ell \in \mathbb{Z}^T$	minimum required hourly workers	parameter
$u \in \mathbb{R}$	maximum daily work hours	parameter

```
# define parameters
B                = 2                                  # num of burger makers
T                = 14                                 # num of time periods
timeperiods      = range(T)                           # time periods
burgermakers     = ["Tom","Bri"]                      # burger makers
hourly_rates     = {"Tom": 10,"Bri": 15}              # rates
minimum_workers = [1,1,1,2,2,2,1,1,1,2,1,1,1,1] # min hourly workers
maximum_hours    = 11                                 # max worker hours
```

Gurobi implementation: Model

Variable	Description	Type
$s \in \{0,1\}^{B \times T}$	burger-maker shift schedule	decision variable
$w \in \{-1,0,1\}^{B \times T}$	shift changes	intermediary variables
$v \in \{0,1\}^{T \times B}$	shift starts	intermediary variables

```
# Create a new model
m = Model("diner_schedule")

# Create variables
s = m.addVars(burgermakers, timeperiods, vtype=GRB.BINARY, name="s")
w = m.addVars(burgermakers, timeperiods, lb=-1, ub=1, name="w")
v = m.addVars(burgermakers, timeperiods, name="v")
```

Gurobi implementation: Objective

$$
\begin{array}{lll}
\text{minimize} & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} & \\
\text{subject to} & \sum_{t=1}^{T} s_{b,t} \leq u & \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t & \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} & \forall t > 0, b \\
& w_{b,0} = s_{b,0} & \forall b \\
& v_{b,t} = \max(0, w_{b,t}) & \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 & \forall b
\end{array}
$$

```
s[(b,t)]
```

Gurobi relies heavily on the *tuple*. Lists and tuples are both ordered collections of Python objects, but tuples are immutable– they cannot be modified once created.

List: `x = [3,7,9]` Tuple: `x = (3,7,9)`

Gurobi implementation: Objective

$$
\begin{array}{lll}
\text{minimize} & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} & \\
\text{subject to} & \sum_{t=1}^{T} s_{b,t} \leq u & \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t & \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} & \forall t > 0, b \\
& w_{b,0} = s_{b,0} & \forall b \\
& v_{b,t} = \max(0, w_{b,t}) & \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 & \forall b
\end{array}
$$

```
[(hourly_rates[b]*s[(b,t)]) for b in burgermakers for t in timeperiods]
```

Gurobi implementation: Objective

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} & \\
\text{subject to} \quad & \sum_{t=1}^{T} s_{b,t} \leq u & \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t & \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} & \forall t > 0, b \\
& w_{b,0} = s_{b,0} & \forall b \\
& v_{b,t} = \max(0, w_{b,t}) & \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 & \forall b
\end{aligned}
$$

```
sum([(hourly_rates[b]*s[(b,t)]) for b in burgermakers for t in timeperiods])
```

Gurobi implementation: Objective

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} \\
\text{subject to} \quad & \sum_{t=1}^{T} s_{b,t} \leq u && \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t && \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} && \forall t > 0, b \\
& w_{b,0} = s_{b,0} && \forall b \\
& v_{b,t} = \max(0, w_{b,t}) && \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 && \forall b
\end{aligned}
$$

```
quicksum([(hourly_rates[b]*s[(b,t)]) for b in burgermakers for t in timeperiods])
```

Gurobi implementation: Objective

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} & \\
\text{subject to} \quad & \sum_{t=1}^{T} s_{b,t} \leq u & \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t & \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} & \forall t > 0, b \\
& w_{b,0} = s_{b,0} & \forall b \\
& v_{b,t} = \max(0, w_{b,t}) & \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 & \forall b
\end{aligned}
$$

```
# Set objective
m.setObjective(
  quicksum([(hourly_rates[b]*s[(b,t)]) for b in burgermakers for t in timeperiods])
      , sense=GRB.MINIMIZE)
```

Gurobi implementation: Bound constraints

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} \\
\text{subject to} \quad & \sum_{t=1}^{T} s_{b,t} \leq u && \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t && \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} && \forall t > 0, b \\
& w_{b,0} = s_{b,0} && \forall b \\
& v_{b,t} = \max(0, w_{b,t}) && \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 && \forall b
\end{aligned}
$$

```
# Add constraints on maximum daily hours
m.addConstrs(quicksum([s[(b,t)] for t in timeperiods]) <= maximum_hours \
  for b in burgermakers, name="max_daily_hours")


# Add constraints on minimum hourly workers
m.addConstrs(quicksum([s[(b,t)] for b in burgermakers]) >= minimum_workers[t] \
  for t in timeperiods, name="min_workers")
```

Gurobi implementation: Shift constraints

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} & \\
\text{subject to} \quad & \sum_{t=1}^{T} s_{b,t} \leq u & \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t & \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} & \forall t > 0, b \\
& w_{b,0} = s_{b,0} & \forall b \\
& v_{b,t} = \max(0, w_{b,t}) & \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 & \forall b
\end{aligned}
$$

```
# Add constraints to count shift starts
m.addConstrs((w[(b,t)] == (s[(b,t)] - s[(b,t-1)]) \
  for b in burgermakers for t in range(1,T)), name="shift_changes")
m.addConstrs((w[(b,0)] == s[(b,0)] for b in burgermakers), \
  name="shift_starts_init")
m.addConstrs((v[(b,t)] == max_(w[(b,t)], 0.0) \
  for b in burgermakers for t in range(1,T)), name="shift_starts")


# Add constraints to place maximums on shift starts
m.addConstrs((quicksum([v[(b,t)] for t in timeperiods]) <= 1 \
  for b in burgermakers ), name="shift_start_max")
```

Fire it up

m.optimize()

```
> python burgers.py
Optimize a model with 46 rows, 84 columns and 166 nonzeros
Model has 26 general constraints
Variable types: 56 continuous, 28 integer (28 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+01, 2e+01]
  Bounds range     [1e+00, 1e+01]
  RHS range        [1e+00, 1e+01]
Presolve removed 26 rows and 61 columns
Presolve time: 0.00s
Presolved: 20 rows, 23 columns, 70 nonzeros
Variable types: 10 continuous, 13 integer (13 binary)
Found heuristic solution: objective 260.0000000
Root relaxation: cutoff, 6 iterations, 0.00 seconds
    Nodes    |    Current Node    |     Objective Bounds      |     Work
Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0     cutoff    0       260.00000  260.00000  0.00%     -    0s
Explored 0 nodes (6 simplex iterations) in 0.01 seconds
Thread count was 4 (of 4 available processors)
Solution count 1: 260
Optimal solution found (tolerance 1.00e-04)
Best objective 2.600000000000e+02, best bound 2.600000000000e+02, gap 0.0000%
>
```

Special ordered sets

Alternate implementation

$$
\begin{array}{lll}
\text{minimize} & \sum_{t=1}^{T} \sum_{b=1}^{B} r_b \cdot s_{b,t} & \\
\text{subject to} & \sum_{t=1}^{T} s_{b,t} \leq u & \forall b \\
& \sum_{b=1}^{B} s_{b,t} \geq \ell_t & \forall t \\
& w_{b,t} = s_{b,t} - s_{b,t-1} & \forall t > 0, b \\
& w_{b,0} = s_{b,0} & \forall b \\
& v_{b,t} = \max(0, w_{b,t}) & \forall t, b \\
& \sum_{t=1}^{T} v_{b,t} \leq 1 & \forall b
\end{array}
$$

```
# Add constraints to count shift starts
m.addConstrs((w[(b,t)] == (s[(b,t)] - s[(b,t-1)]) \
  for b in burgermakers for t in range(1,T)), name="shift_changes")
m.addConstrs((v[(b,0)] == s[(b,0)] for b in burgermakers), \
  name="shift_starts_init")
m.addConstrs((v[(b,t)] == max_(w[(b,t)], 0.0) \
  for b in burgermakers for t in range(1,T)), name="shift_starts")
```

```
# Add constraints to place maximums on shift starts
m.addConstrs((quicksum([v[(b,t)] for t in timeperiods]) <= 1 \
  for b in burgermakers ), name="shift_start_max")
```

I'll order the special, please

$$\sum_{t=1}^{T} v_{b,t} \leq 1 \qquad \forall b$$

This can be modeled as a **Special Ordered Set (SOS)** constraint.

There are two types of SOS constraints:

1. **Type 1:** Given a set of variables, at most one of them can be non-zero
2. **Type 2:** Given a set of variables, at most two of them can be non-zero, and the non-zero values must be contiguous

I'll order the special, please

$$\sum_{t=1}^{T} v_{b,t} \leq 1 \qquad \forall b$$

This can be modeled as a **Special Ordered Set (SOS)** constraint.

There are two types of SOS constraints:

1. **Type 1:** Given a set of variables, at most one of them can be non-zero
2. **Type 2:** Given a set of variables, at most two of them can be non-zero, and the non-zero values must be contiguous

Linear constraint:

```
for b in burgermakers:
  m.addConstr((quicksum([v[(b,t)] for t in timeperiods]) <= 1 ))
```

SOS constraint:

```
for b in burgermakers:
  m.addSOS(GRB.SOS_TYPE1, [v[(b,t)] for t in timeperiods])
```

General Constraints

General constraints

Definition

General constraints allow for easy modeling, and Gurobi will transform it into the corresponding MIP formulation.

MAX: $r = \max\{x_1, \ldots, x_n, c\}$

- r will be equal to the maximum value over the set $\{x_1, \ldots, x_n, c\}$

MIN: $r = \min\{x_1, \ldots, x_n, c\}$

- r will be equal to the minimum value over the set $\{x_1, \ldots, x_n, c\}$

ABS: $r = |x|$

- r will be equal to the absolute value of x

AND: $r = \text{and}\{x_1, \ldots, x_n\}$

- For r, x_i binary, r will be 1 if and only if $x_i = 1$ for **all** i

OR: $r = \text{or}\{x_1, \ldots, x_n\}$

- For r, x_i binary, r will be 1 if and only if $x_i = 1$ for **some** i

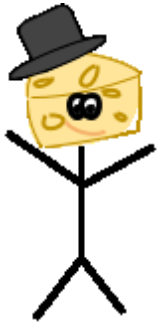
INDICATOR: $(y = f) \rightarrow (a^T x \leq b)$

- For y, f binary, if $y = f$, then $a^T x \leq b$ must be satisfied

Scrambled with cheese

Bri and Tom are *terrific* burgermakers, which has made the diner so popular that it needs two new employees!

Barb Ecuesauce has an hourly rate of $8



Jack Cheese has an hourly rate of $8

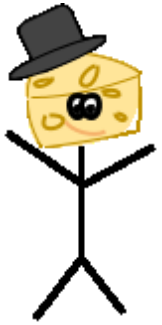
Bri hires her best friend Barb and her boyfriend Jack, which creates a couple complications:

- Bri and Jack cannot work at the same time
- Bri wants to only work when Barb is working

Scrambled with cheese

Bri and Tom are *terrific* burgermakers, which has made the diner so popular that it needs two new employees!

Barb Ecuesauce has an hourly rate of $8



Jack Cheese has an hourly rate of $8

Bri hires her best friend Barb and her boyfriend Jack, which creates a couple complications:

- Bri and Jack cannot work at the same time
- Bri wants to only work when Barb is working

The following constraint will ensure that Bri only works when Barb is working and Jack is *not* working:

$$s_{\text{"Bri"},t} = \text{AND}\{s_{\text{"Barb"},t}, 1 - s_{\text{"Jack"},t}\} \quad \forall t$$

Gurobi makes it over easy: AND

$$s_{\text{"Bri"},t} = \text{AND}\{s_{\text{"Barb"},t}, 1 - s_{\text{"Jack"},t}\} \quad \forall t$$

$x_1 = \text{and}\{x_2, x_3, x_4\}$ $\longleftrightarrow$ `m.addGenConstrAnd(x_1, [x_2, x_3, x_4])`

Gurobi implementation:

```
# Define opposite of the schedule
not_s = m.addVars(burgermakers, timeperiods, name="not_s")
m.addConstrs((not_s[(b,t)] == (1 - s[(b,t)]) \
  for b in burgermakers for t in timeperiods), name="not_s_c")

# Add constraints to enforce policies
for t in timeperiods:
  m.addGenConstrAnd(s[("Bri",t)], [s[("Barb",t)], not_s[("Jack",t)]])
```

Over hard: INDICATOR

$$s_{\text{"Jack"},t} = 1 \implies s_{\text{"Bri"},t} = 0 \quad \forall t$$

$$y = f \rightarrow a \cdot x = b$$

$$\updownarrow$$

```
m.addGenConstrIndicator(y, f, a*x, sense=GRB.EQUAL, rhs=b)
```

Gurobi implementation:

```
# Add constraints to enforce policies
for t in timeperiods:
  m.addGenConstrIndicator(s[("Jack",t)], 1, \
    s[("Bri",t)], sense=GRB.EQUAL, rhs=0.0, name="policy")
```

Would you like to place an order?

Opening is the worst part of working at a diner, and Bri wants to make sure she never has to do it.

The following constraints will ensure Bri's shift is not the first:

$$z_b = \max\{(t \cdot v_{b,t}) \quad \forall t\} \quad \forall b \quad \textbf{shift start time}$$

$$z_{\text{"Bri"}} \geq z_b \quad \forall b \quad \textbf{shift ordering}$$

Gurobi implementation:

```
# Define shift start times
z = m.addVars(burgermakers, name="z")
for b in burgermakers:
  m.addGenConstrMax(z[b], [t*v[(b,t)] for t in timeperiods])

# Add constraint to make sure Bri never has to open
m.addConstrs(z["Bri"] >= z[b] for b in burgermakers)
```

Parameters

Parameters

In the words of Gurobi

"While you should feel free to experiment with different parameter settings, we recommend that you leave parameters at their default settings unless you find a compelling reason not to."

Considering the above, these parameters could be useful for scheduling:

- **MIPFocus:** modifies solution strategy for MIPs
  - 0 : balance between finding feasible and optimal solutions
  - 1 : to find feasible solutions quickly
  - 2 : to focus on proving optimality
  - 3 : to focus on progressing the objective bound
- **IntFeasTol:** tolerance for how close an integer variable has to be to an integer (finite precision)
- **MIPGap:** Relative MIP gap
- **MIPGapAbs:** Absolute MIP Gap

Gurobi has an automated *Parameter Tuning Tool* that searches for the parameter settings to improve performance.

```
m.tune()
```

Thanks!

GUROBI
OPTIMIZATION

end-to-end
ANALYTICS

Questions

Please use the questions box on your GoToWebinar panel to submit questions

Next Steps

- If you haven't already done so, please register for an account at www.gurobi.com.
- For questions about Gurobi pricing contact: sales@gurobi.com or sales@gurobi.de.
- A recording of this webinar, including the slides and code, will be available in one week.

Appendix

Ugh, managers

Tom and Bri want to make sure that the quality of the burgers doesn't decline, so one of them wants to be working at all times.

The following constraints will ensure that either Tom or Bri is always working:

$$\hat{s}_t = \text{OR}\{s_{\text{"Bri"},t}, s_{\text{"Tom"},t}\} \quad \forall t$$
$$\hat{s}_t = 1 \quad \forall t$$

Ugh, managers

Tom and Bri want to make sure that the quality of the burgers doesn't decline, so one of them wants to be working at all times.

The following constraints will ensure that either Tom or Bri is always working:

$$\hat{s}_t = \text{OR}\{s_{\text{"Bri"},t}, s_{\text{"Tom"},t}\} \quad \forall t$$
$$\hat{s}_t = 1 \quad \forall t$$

Gurobi implementation:

```
# Add constraints so that either Tom or Bri is always working
s_hat = m.addVars(timeperiods, name="s_hat")
for t in timeperiods:
  m.addGenConstrOr(s_hat[t], [s[("Bri",t)], s[("Tom",t)]])
m.addConstrs(s_hat[t] == 1 for t in timeperiods)
```

Short shifting

Jack was getting short shifts and complained incessantly. After much ado, Bri and Tom agreed to a minimum shift length.

Previous shift constraints:

$$
\begin{aligned}
w_{b,t} &= s_{b,t} - s_{b,t-1} && \forall t > 0, b \\
w_{b,0} &= s_{b,0} && \forall b \\
v_{b,t} &= \max(0, w_{b,t}) && \forall t, b
\end{aligned}
$$

The following constraints ensure every shift is at least k hours:

$$
\begin{aligned}
\hat{v}_{b,t} &= \max(0, -w_{b,t}) && \forall t, b && \textbf{shift end} \\
\hat{v}_{b,T} &= s_{b,T} && \forall b \\
z_b &= \max\{(t \cdot v_{b,t}) \quad \forall t\} && \forall b && \textbf{start time} \\
\hat{z}_b &= \max\{(t \cdot \hat{v}_{b,t}) \quad \forall t\} && \forall b && \textbf{end time} \\
z_b - \hat{z}_b &\geq k && \forall b && \textbf{minimum shift length}
\end{aligned}
$$

Short shifting

The following constraints ensure every shift is at least k hours:

$$\hat{v}_{b,t} = \max(0, -w_{b,t}) \quad \forall t, b \quad \textbf{shift end}$$
$$\hat{v}_{b,T} = s_{b,T} \quad \forall b$$
$$z_b = \max\{(t \cdot v_{b,t}) \quad \forall t\} \quad \forall b \quad \textbf{start time}$$
$$\hat{z}_b = \max\{(t \cdot \hat{v}_{b,t}) \quad \forall t\} \quad \forall b \quad \textbf{end time}$$
$$z_b - \hat{z}_b \geq k \quad \forall b \quad \textbf{minimum shift length}$$

Gurobi implementation:

```
# Define shift end times
m.addConstrs(v_hat[(b,t)] == max_(-w[(b,t)], 0.0) \
  for b in burgermakers for t in range(0,T-1))

# Define start and end times, and the final shift end
for b in burgermakers:
  m.addConstr(v_hat[(b,T)] == s[(b,T)])
  m.addGenConstrMax(z[b], [t*v[(b,t)] for t in timeperiods])
  m.addGenConstrMax(z_hat[b], [t*v_hat[(b,t)] for t in timeperiods])

# Add constraint on minimum shift length
m.addConstrs(z[b] - z_hat[b] >= k for b in burgermakers)
```