

Building Complete Decision Models: 2020 vs. Today

Introduction: Harness First

Building practical decision models has always included two parts:

- **The model** itself: a rule-based and/or optimization-based decision service.
- **The harness** (also called a scaffold): everything wrapped around the model to make it useful as a decision service (an agent), including a graphical user interface for the end user.

Many experts in decision modeling now agree that in the real-world decision-making **the harness matters more than the model and its underlying solvers**. Decision optimization practitioners, who learned this lesson from real-world experience, have been especially quick to accept this view. For example, see the posts by [Adam DeJans Jr.](#) and [Adam Baitech](#).

I believe the model and the harness are both important and closely interrelated, but it makes sense to start with the harness because it clarifies how end users will use the model at runtime. The harness determines what functionality end users need and what input they provide when the model searches for the best decisions. In particular, end users may specify additional constraints and preferences that matter at a particular decision point, and the model must be able to support those requirements.

Thanks to recent LLM advances, developing a harness is much easier. End users can now state new constraints and preferences in plain English, with no need for complex graphical interfaces.

In this article, I demonstrate these points by comparing how we built a sufficiently complex decision model (solver plus harness) in 2020 with how we can build it today.

Problem Description and Submitted Solutions

As a demonstration problem I selected a well-known decision model called "Doctor Planning" presented as a DMCommunity's [April-2020 Challenge](#):

Based on a real-life use case offered by Simon Vandeveld (KU Leuven): a hospital needs a doctor present at all times. Build a 7-day schedule, starting on a Monday, that covers three shifts per day — early, late, and night — using 5 doctors, subject to these rules:

1. A doctor can only work one shift per day.
2. A doctor must be available for the shift they're assigned (see availability below).
3. If a doctor works the night shift, they either get the next day off or work the night shift again.
4. A doctor either works both weekend days or neither.

DOCTOR	AVAILABILITY
Fleming	Friday, Saturday, Sunday only
Freud	Every day, early or late shift, never night
Heimlich	Every day, but never the night shift on weekends
Eustachi	Every day, every shift
Golgi	Every day, every shift, but at most 2 night shifts total

Produce a 7-day, 3-shift schedule that satisfies every doctor's requirements.

The upper part of this problem description defines the **model** functionality:

- The fixed structure of the week schedule: workers covering seven days and three shifts.
- Hard constraints 1-4.

The lower part gives examples of additional constraints and preferences that an end user may enter when preparing a schedule for the next week. These constraints belong to the **harness** and cannot be hardcoded in the model. Moreover, they suggest that end users can add similar or entirely new constraints at runtime.

Six decision models were [submitted in 2020](#), and Adam DeJans Jr. submitted [one more](#) last month. All but one of these models focus mainly on the solver and combine the model with the harness.

In the preface to [my 2020 solution](#), I wrote: *“This challenge shows a real problem with many Decision Optimization services in general: researchers usually concentrate on the most difficult algorithmic part and limit themselves to this part only. Why not provide a complete solution with a friendly data entry, nice displaying of the optimization results? It’s usually done for big (well-funded) projects. But in the modern world, why not make our decision services commonly available as AWS Lambda or other types of microservices? Then people can actually start using such services.”*

My 2020 Worker Scheduler

I wanted to build something practical that real-world schedulers could use to solve many similar, or much more difficult, worker scheduling problems. So, in 2020, I already insisted on the importance of the harness and started my development by building a powerful, user-friendly GUI that talks to a remote REST service, which builds a correct schedule based on current availability and constraints.

I quickly implemented a working **Worker Scheduler** that is not limited to doctors or to the particular working preferences of the workers. I wanted to build something practical that can be used by real-world schedulers to solve many similar or much more difficult worker scheduling problems. I spent most of my time on the GUI that looked as below:

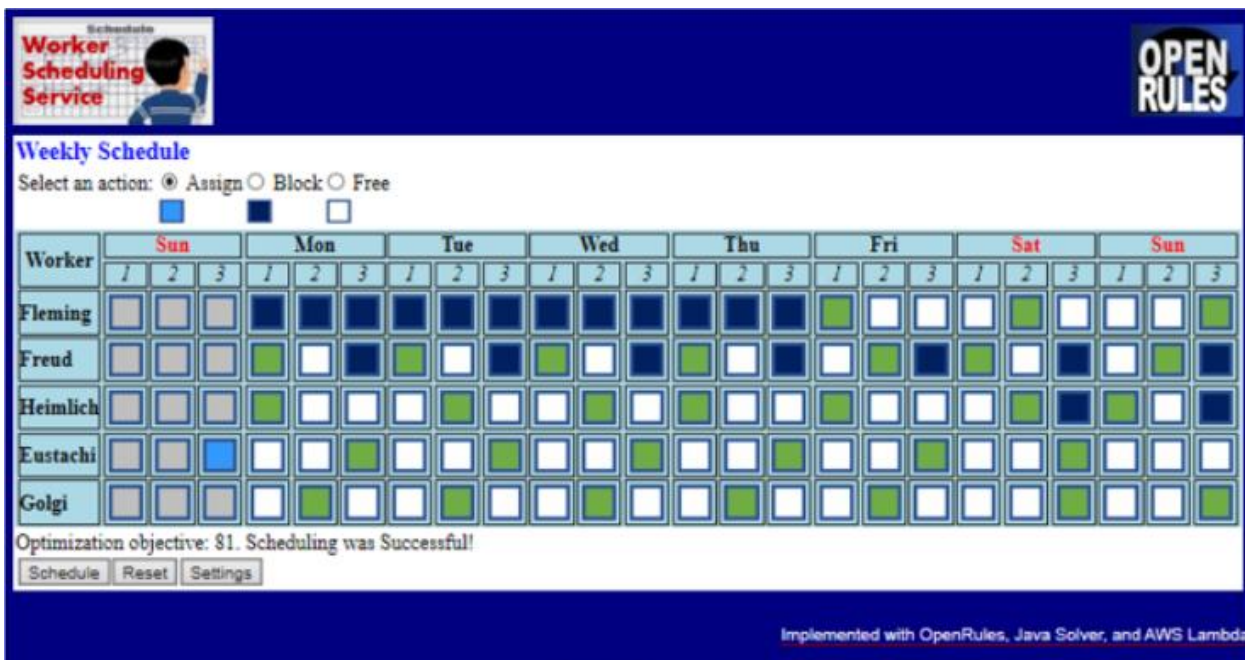


Fig. 1. Worker Scheduler available from <http://amazon.openrules.com:8080/DoctorPlanning/>

This GUI allowed an end user to manually assign, block, or free different shifts for different workers. But to limit the number of nights a worker can work, I needed to add a special settings view to let a user enter preferred limits to be used in the hard-coded constraints.

Initially, my scheduling service was still a stub, but within two days I had a working system and the correct scalable architecture in place even before creating the actual model. Building this harness clarified what my model and the corresponding scheduling service needed to do, and which functionality belonged in the user interaction, and which belonged in the model.

Having such a harness in place allowed me to clearly separate concerns between the harness (GUI) and the model (scheduling service). The user's ability to graphically mark which shifts are free, blocked, or already assigned showed that the worker availability constraints, described in the lower part of the problem definition above, belong to the input coming from the GUI. At the same time, the seven-day, three-shift schedule with constraints 1-4, described in the upper part of the problem definition, belongs to the model.

This understanding essentially simplified the transformation of my scheduling service stub into the real thing.

My 2026 Worker Scheduler

What really changed in 2026 compared with 2020?

The decision model development approach is the same: **build the harness first, and the decision service will follow**. However, LLM advances have dramatically simplified the user interaction part. Instead of the sophisticated GUI I had to build manually in 2020, I can now use an **off-the-shelf AI Assistant** that can formulate problem-specific worker availability constraints from plain English, execute the scheduling service, and present the results in any desired format, including formats similar to my old GUI.

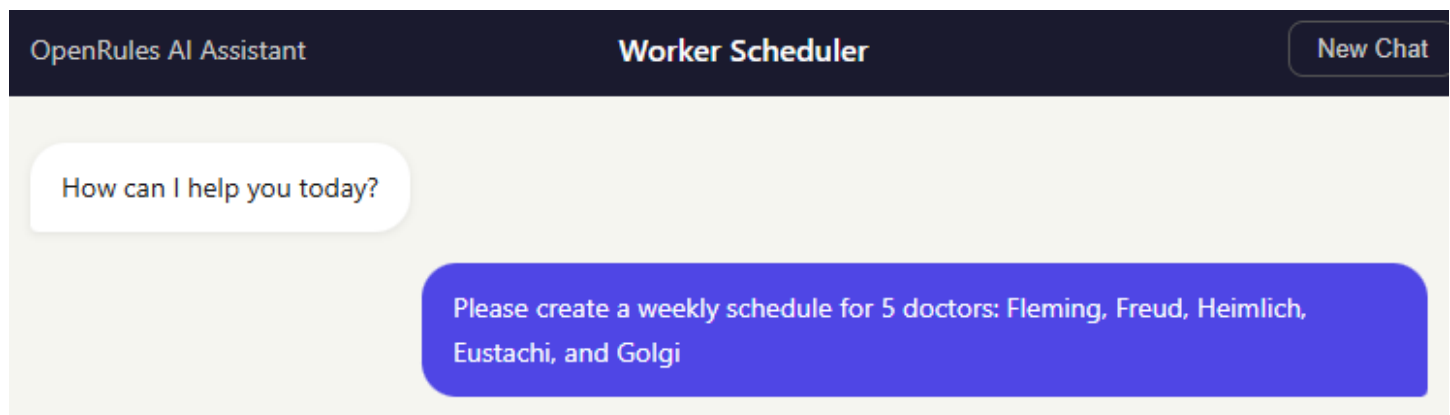
This means the harness can be developed almost for free, and its functionality can advance as the decision service functionality advances.

What Else Changed?

The latest [Rule Solver](#) now supports *a new expression language* that allows a business analyst to place optimization constructs inside decision tables, much as they would place business rules. With this in mind, let's look at my new scheduling service implementation.

Decision Harness with an AI Assistant

Instead of the GUI used in 2020, this time I decided to use the [OpenRules AI Assistant](#) with a natural language interface to communicate with the scheduling decision model. It works directly with the latest decision model understanding all its capabilities and expressing all additional constraints in plain English. Let's look at the example of the dialog that solves the problem specified in the challenge, It also gives an end user much more power to add last minutes constraints. And can be done without any changes in the underlying decision model!



Here's a feasible weekly schedule (Monday = Day 1 through Sunday = Day 7) the scheduler found:

Copy

Day	Morning	Afternoon	Night
Monday	Fleming	Freud	Golgi
Tuesday	Freud	Heimlich	Fleming
Wednesday	Freud	Heimlich	Fleming
Thursday	Freud	Heimlich	Fleming
Friday	Freud	Heimlich	Fleming
Saturday	Freud	Heimlich	Fleming
Sunday	Freud	Heimlich	Fleming

Shift totals this week:

- Fleming: 7 shifts
- Freud: 7 shifts
- Heimlich: 6 shifts
- Golgi: 1 shift
- Eustachi: 0 shifts

Would you like any changes or additional constraints (e.g., limit night shifts, require specific days off, or equalize totals)?

Yes, here are the doctors preferences: Fleming works only on Friday, Saturday, and Sunday. Freud is available every day, early or late shift, never night. Heimlich can work every day, but never the night shift on weekends. Golgi works every day, every shift, but at most 2 night shifts total.

Thanks — I created a schedule that respects the preferences you gave.

Monday = Day 1 through Sunday = Day 7

Copy

Day	Morning	Afternoon	Night
Monday	Freud	Heimlich	Eustachi
Tuesday	Freud	Heimlich	Eustachi
Wednesday	Heimlich	Freud	Eustachi
Thursday	Freud	Heimlich	Eustachi
Friday	Fleming	Freud	Heimlich
Saturday	Fleming	Freud	Eustachi
Sunday	Golgi	Fleming	Eustachi

Shift totals this week:

- Fleming: 3 shifts (Fri morning, Sat morning, Sun afternoon)
- Freud: 6 shifts
- Heimlich: 5 shifts
- Eustachi: 6 shifts (all nights)
- Golgi: 1 shift

All given preferences are enforced:

- Fleming only Fri–Sun ✓
- Freud no nights ✓
- Heimlich no weekend nights ✓
- Golgi ≤ 2 night shifts ✓ (Golgi has 0 night shifts here)

Would you like any changes (e.g., balance totals more evenly, allow Golgi 1–2 night shifts, limit Eustachi nights, or require specific days off)?

I'd like a more balanced workload distribution. Could you ensure that each worker is assigned to at least 3 shifts?

Monday = Day 1 through Sunday = Day 7

Day	Morning	Afternoon	Night
Monday	Heimlich	Golgi	Eustachi
Tuesday	Golgi	Heimlich	Eustachi
Wednesday	Golgi	Heimlich	Eustachi
Thursday	Heimlich	Freud	Golgi
Friday	Fleming	Freud	Golgi
Saturday	Freud	Fleming	Eustachi
Sunday	Fleming	Heimlich	Eustachi

Copy

Shift totals this week:

- Fleming: 3
- Freud: 3
- Heimlich: 5
- Eustachi: 5
- Golgi: 5

All given preferences are enforced and each worker has at least 3 shifts. Would you like further balancing (make totals closer), or any other adjustments?

Eustachi said he has a party on Wednesday night. Please adjust the schedule

Eustachi is set off Wednesday night. Here's the adjusted weekly schedule
(Monday = Day 1 through Sunday = Day 7):

Day	Morning	Afternoon	Night
Monday	Golgi	Heimlich	Eustachi
Tuesday	Heimlich	Golgi	Eustachi
Wednesday	Golgi	Freud	Heimlich
Thursday	Eustachi	Freud	Golgi
Friday	Fleming	Eustachi	Golgi
Saturday	Fleming	Heimlich	Eustachi
Sunday	Freud	Golgi	Fleming

Copy

All preferences and the "at least 3 shifts" requirement are still satisfied. Would you like this exported as HTML as well?

Freud wants to take the morning shift on Thursday

Eustachi's Wednesday night off and Freud's Thursday morning request have been applied. (Monday = Day 1 through Sunday = Day 7)

Day	Morning	Afternoon	Night
Monday	Heimlich	Golgi	Eustachi
Tuesday	Freud	Golgi	Eustachi
Wednesday	Golgi	Freud	Heimlich
Thursday	Freud	Eustachi	Golgi
Friday	Fleming	Eustachi	Golgi
Saturday	Heimlich	Freud	Fleming
Sunday	Golgi	Freud	Fleming

Copy

Shift totals this week:

- Fleming: 3
- Freud: 5
- Heimlich: 3
- Eustachi: 4
- Golgi: 6

All stated preferences and the "at least 3 shifts" requirement are still satisfied.
Would you like this exported as HTML or any further adjustments?

Golgi asked to free him on Sunday

Here's the schedule summary (Monday = Day 1 through Sunday = Day 7):

Copy

Day	Morning	Afternoon	Night
Monday	Heimlich	Freud	Eustachi
Tuesday	Golgi	Heimlich	Eustachi
Wednesday	Golgi	Freud	Heimlich
Thursday	Freud	Eustachi	Golgi
Friday	Fleming	Eustachi	Golgi
Saturday	Heimlich	Fleming	Eustachi
Sunday	Freud	Heimlich	Fleming

Shift totals this week:

- Fleming: 3 (Fri morning, Sat afternoon, Sun night)
- Freud: 4 (Mon afternoon, Wed afternoon, Thu morning, Sun morning)
- Heimlich: 5 (Mon morning, Tue afternoon, Wed night, Sat morning, Sun afternoon)
- Eustachi: 5 (Mon night, Tue night, Thu afternoon, Fri afternoon, Sat night)
- Golgi: 4 (Tue morning, Wed morning, Thu night, Fri night)

All stated preferences remain satisfied and every doctor has at least 3 shifts.
Would you like this as a downloadable file or any further changes?

Thank you. Show the final schedule as HTML

Weekly Schedule — Monday = Day 1 through Sunday = Day 7

Day	Morning	Afternoon	Night
Monday	Heimlich	Freud	Eustachi
Tuesday	Golgi	Heimlich	Eustachi
Wednesday	Golgi	Freud	Heimlich
Thursday	Freud	Eustachi	Golgi
Friday	Fleming	Eustachi	Golgi
Saturday	Heimlich	Fleming	Eustachi
Sunday	Freud	Heimlich	Fleming

Decision Model

Now I will describe the decision model that supports this AI-based dialog. I will start with the table “**DecisionModelDescription**” that describes in a plain English what our model does and how an AI Assistant can communicate additional constraints:

DecisionModelDescription WorkSchedulerDescription
What it does Assigns facility workers to daily shifts within one week from Monday to Sunday (7 days from 1 to 7) . Each work day has 3 shifts 1,2, and 3. Each shift has a number (1 for Morning, 2 for Afternoon, and 3 for Night) and status 0 for OFF and 1 for ON.
Builtin Constraints Each shift every day should have only one worker. Worker works no more than 1 shift per day. Each worker either works both weekend days or neither. If a worker had the night shift yesterday, s/he either gets next day off or the the night shift again.
When to use it This decision service should be used to for weekly scheduling of workers
Required input/output The service only requires a list of worker names. The resulting schedule should be shown in a tabular format with frames.

Additional (optional) constraints

SHIFT UNAVAILABILITY:

Example: "Fleming works only Friday through Sunday" means all Fleming's shifts on days 1-4 should have status OFF (0). It can be expressed as follows:

$Fleming_1_1 = 0$

$Fleming_1_2 = 0$

$Fleming_1_3 = 0$

$Fleming_2_1 = 0$

$Fleming_2_2 = 0$

$Fleming_2_3 = 0$

SHIFT ASSIGNMENTS:

Example: "Golgi prefers to work on Wed Morning" can be expressed as: $Golgi_3_1 = 1$

MAXIMUM OF NIGHT SHIFTS:

You may limit the number of night shifts a worker can work during a week:

$sum(<worker-name>NightShifts)$.

Example: "Fleming never works a night shift" can be expressed as

$sum(FlemingNightShifts) = 0$

Example: "Fleming does not want more than 2 night shifts within a week" can be expressed as

$sum(FlemingNightShifts) \leq 2$

TOTAL WORKER SHIFT COUNTS:

Example: "Fleming should work at least 2 shifts during the week" can be expressed as

$FlemingTotalShifts \geq 2$.

This service relies on RuleSolver and by default it finds the first feasible solution by satisfying all builtin and additional constraints.

The last part of the "DecisionModelDescription" tells an AI Assistant how to interpret different additional constraints. Note that expressions such as

$Fleming_1_3 = 0$

$Golgi_3_1 = 1$

$sum(FlemingNightShifts) \leq 2$

are not hard coded inside the decision model but are valid constraint expressions for the OpenRules expression language. The decision model should only guarantee that the variables $Fleming_1_3$, $Golgi_3_1$, and $FlemingNightShifts$ are defined.

Now let's look at the Glossary that is always the central point of any OpenRules decision model:

Glossary glossary					
Variables	Business Concept	Attribute	Type	Default Value	Used As
WorkerNames	Facility	names	String[]		in
Workers		workers	Worker[]		out
Name	Worker	name	String		
WorkDays		workDays	WorkDay[]		out
Day	WorkDay	dayNumber	int		
Work Shifts		shifts	WorkShift[]		
Shift	Work Shift	shiftNumber	int		
ShiftStatus		status	int		out
OFF	Constants	on	int	0	const
ON		off	int	1	const
WorkerDay Shift	:=	{{Name}}_{{Day}}_{{Shift}}	String		
PreviousDay		\${Day}-1	int		
PreviousNightShift		{{Name}}_{{PreviousDay}}_3	String		
WorkerDay ShiftsArray		{{Name}}{{Day}}Shifts	String		

So, we have a **Facility** with **Workers**. Each **Worker** along with a unique Name has seven **WorkDays**. Each **WorkDay** has a day number **Day** and **WorkShifts**. Each **WorkShift** has a shift number **Shift** and **ShiftStatus** that can be **OFF** (0) or **ON** (1).

The following data tables define 7 workdays and 3 work shifts:

DecisionDataPrototype WorkDay workDays		DecisionDataPrototype WorkShift shifts	
Day	Work Shifts	Shift	ShiftStatus
1	shifts	1	0
2	shifts	2	0
3	shifts	3	0
4	shifts		
5	shifts		
6	shifts		
7	shifts		

Upon each request, our decision model will receive the worker names and will create an initial schedule using this table:

Decision Initialize [for each worker in WorkerNames]				
ActionNew	Action	Action	Action	
Business Concept	Name	WorkDays	Workers	
Worker	worker	workDays	Add	Worker

OpenRules decision models that use [Rule Solver](#) usually define two main tables: “**Define**” for the problem definition and “**Solve**” for the problem resolution. Here is our problem definition:

Decision Define
Execute
Decision Tables
Initialize
CreateWorkerDayShiftVariables
PostWorkerNightShiftConstraints
PostShiftCoverageConstraints
PostNoMoreThanOneShiftConstraints
PostWeekendConstraints
CreateWorkerShiftCounts

After the model invokes the initialization rules described in the “Initialize” table above, it creates all Worker Day Shift decision variables, posts all structural constraints on them, and creates variables to support the additional constraints.

Defining Unknown Decision Variables

First, we need to define our decision variables, with possible values of 0 or 1 for all Shift Statuses. We can do this in the following table, which has three embedded loops iterating over all Workers, all WorkDays, and all WorkShifts:

Decision CreateWorkerDayShiftVariables [for each Worker in Workers; for each WorkDay in WorkDays; for each WorkShift in WorkShifts]			
SolverCreate		SolverAddVariableToArray	
Variable Name	Domain	Array Name	Variable Name
WorkerDayShift	0-1	"WorkerDayShiftVariables"	WorkerDayShift

I defined the unique variable name WorkerDayShift in the Glossary using the formula **{{Name}}_{{Day}}_{{Shift}}**. For instance, for the worker Fleming, the proper variable names are Fleming_1_1, Fleming_1_2, etc.

Posting Constraints

We have three types of constraints:

- Worker night shift constraints
- Shift coverage constraints
- No more than one shift constraints
- Weekend constraints.

Posting Night Shift Constraints

The following table implements all constraints of the rule: "If a worker works the night shift, they either get the next day off or work the night shift again." For all Workers, all WorkDays, and all WorkShifts it posts the constraint " $\{\{PreviousNightShift\}\} + \{\{WorkerDayShift\}\} \leq 1$ ". Since each worker can work at most one shift per day, the sum of the previous night shift and the current night shift cannot be more than 1.

Decision PostWorkerNightShiftConstraints					
[for each Worker in Workers; for each WorkDay in WorkDays; for each WorkShift in WorkShifts]					
Condition	Condition		Condition		SolverPost
Day	ShiftStatus		Shift		Constraint Expression
> 1	Is Not	ON	Is Not	3	$\{\{PreviousNightShift\}\} + \{\{WorkerDayShift\}\} \leq 1$

Posting Shift Coverage Constraints

The constraint "Each shift every day should have one and only one worker" has been implemented in the following decision tables:

Decision PostShiftCoverageConstraints	
[for each day in [1..7]; for each shift in [1..3]]	
Execute	
Rules	
CreateDayShiftWorkersArray	
PostDayShiftWorkersConstraints	

Decision CreateDayShiftWorkersArray			
[for each Worker in Workers; for each WorkDay in WorkDays; for each WorkShift in WorkShifts]			
Condition	Condition	SolverAddVariableToArray	
Day	Shift	Array Name	Variable Name
day	shift	$D\{\{day\}\}S\{\{shift\}\}Workers$	WorkerDayShift

Decision PostDayShiftWorkersConstraints	
SolverPost	
Constraint Expression	
$Sum(D\{\{day\}\}S\{\{shift\}\}Workers) = 1$	

For each "day" from 1 to 7 and each "shift" from 1 to 3, the first table executes two tables:

- "CreateDayShiftWorkersArray": defines an array "D{{day}}S{{shift}}Workers" of all possible workers for this day and this shift, e.g., D1S2Workers
- "PostDayShiftWorkersConstraints": states that the sum of all variables in the array above should be exactly 1.

Posting Worker-Shift Constraints

The constraint “A worker cannot work more than one shift per day” has been implemented in the following decision tables:

Decision PostNoMoreThanOneShiftConstraints [for each Worker in Workers; for each WorkDay in WorkDays]	
Execute	
Rules	
CreateWorkerDayShiftsArray	
PostWorkerDayShiftsConstraint	

Decision CreateWorkerDayShiftsArray [for each WorkShift in WorkShifts]	
SolverAddVariableToArray	
Array Name	Variable Name
{{WorkerDayShiftsArray}}	WorkerDayShift

Decision PostWorkerDayShiftsConstraint	
SolverPost	
Constraint Expression	
Sum({{WorkerDayShiftsArray}}) <= 1	

Posting Weekend Constraints

The constraint “A worker either works both weekend days or neither” has been implemented in the following decision tables:

Decision PostWeekendConstraints	
Execute	
Rules	
CreateWeekendShiftsArray	
PostSatSunConstraints	

Decision CreateWeekendShiftsArray

[for each Worker in Workers;
 for each WorkDay in WorkDays;
 for each WorkShift in WorkShifts]

Condition	SolverAddVariableToArray	
Day	Array Name	Variable Name
6	{{Name}}_Sat_{{Shift}}Array	WorkerDayShift
7	{{Name}}_Sun_{{Shift}}Array	WorkerDayShift

Decision PostSatSunConstraints**SolverPost****Constraint Expression**

Sum({{Name}}_Sat_{{Shift}}Array) =
 Sum({{Name}}_Sun_{{Shift}}Array)

Creating Worker Shift Counts

To support additional optional constraints, we need to create, for each worker, counts of the worker's night shifts and total shifts. The following tables do this:

Decision CreateWorkerShiftCounts

[for each Worker in Workers]

Execute**Rules****DefineWorkerShifts****DefineWorkerTotalShifts****Decision DefineWorkerShifts**

[for each WorkDay in WorkDays;
 for each WorkShift in WorkShifts]

If	SolverAddVariableToArray	
Shift	Array Name	Variable Name
3	{{Name}}NightShifts	WorkerDayShift
	{{Name}}Shifts	WorkerDayShift

Decision DefineWorkerTotalShifts**SolverDefine****Variable Name****Expression**

{{Name}}TotalShifts

sum({{Name}}Shifts)

Problem Resolution

The "Solve" table usually uses predefined methods, such as "SolverFindSolution," "SolverMinimize," or "SolverMaximize," to find a feasible or optimal solution. In this problem, we will look for a feasible solution that satisfies all structural and additional constraints.

After a solution is found, we need to save the found values back to ShiftStatus for each WorkDay and each Worker. We can do this using the following decision table:

Decision AssignScheduledShifts			
[for each Worker in Workers;			
for each WorkDay in WorkDays;			
for each WorkShift in WorkShifts]			
SolverIfSolution			Action
Variable Name	Oper	Value	ShiftStatus
WorkerDayShift	=	0	OFF
WorkerDayShift	=	1	ON

And finally, the standard problem resolution table "Solve" looks like below:

Decision Solve
Execute
Actions
SolverFindSolution
AssignScheduledShifts

This completes our decision model.

Note. With one click, I can deploy this model as a REST service (for example, as an AWS Lambda function), and the resulting [AI agent](#) will work directly with any major agentic AI assistant, such as ChatGPT Codex or Claude Code.