Un sistema basado en programación de restricciones para la asignación de turnos en una empresa de suministro de combustible

A constraint programming-based system for shift scheduling in a fuel supply company

Gabriel Echeverría Canque[1*] https://orcid.org/0009-0002-6313-5909
Ricardo Valdivia Pinto[1] https://orcid.org/0009-0008-7131-3844

Recibido 5 de abril de 2024, aceptado 03 de mayo de 2024
Received: April 05, 2024 Accepted: May 03, 2024

RESUMEN

Este artículo presenta la implementación de una solución para la asignación de turnos de trabajadores en una empresa de suministro de combustibles. Esta solución se basa en la especificación del caso como un problema de satisfacción de restricciones (model) y su resolución mediante el uso del solucionador (solver) proporcionado por el módulo CP-SAT de la herramienta OR-Tools. Se implementó un sistema denominado ARGOS que provee de una adecuada interfaz a nivel usuario para gestionar la información de los empleados y proveer de la información necesaria al solucionador para que este genere la planificación de turnos.

Palabras clave: Timetabling, programación de restricciones, CSP, OR-Tools, CP-SAT.

ABSTRACT

This article presents the implementation of a solution for worker shift assignment in a fuel supply company. The solution is based on specifying the case as a constraint satisfaction problem (model) and resolving it using the solver provided by the CP-SAT module of the OR-Tools tool. A system called ARGOS was implemented. It provides a user-friendly interface for managing employee information and supplies the solver with the necessary information for generating shift schedules.

Keywords: Timetabling, constraint programming, CSP, OR-Tools, CP-SAT.

INTRODUCCIÓN

La Sociedad Comercial Ariaca es una empresa orientada al rubro suministro de combustible para los aviones. Ariaca opera las veinticuatro horas con un equipo de empleados que trabajan en tres turnos diarios para brindar los servicios necesarios a sus clientes. Entre los principales clientes de Ariaca se encuentran las aerolíneas SKY, LATAM y JETSMART.

En la Sociedad Comercial Ariaca, hay un empleado que se encarga de realizar la planificación de turnos de los empleados de la empresa. Para llevar a cabo esta tarea, es necesario que las líneas aéreas entreguen sus itinerarios de vuelo correspondiente al mes siguiente. Una vez obtenidos los itinerarios de vuelo de las líneas aéreas, el empleado genera una plantilla que calendariza los vuelos programados para el mes (Figura 1).

[1] Universidad de Tarapacá. Facultad de Ingeniería. Departamento de Computación e Informática. Arica. Chile.
E-mail: gabriel.echeverria.canque@alumnos.uta.cl; rvaldivi@academicos.uta.cl
* Autor de correspondencia: gabriel.echeverria.canque@alumnos.uta.cl

De la Figura 1, lo más importante son las horas que están marcadas en amarillo, ya que en base a esas horas se debe asignar una cierta cantidad de empleados para el turno. A estas horas marcadas se les conoce como "itinerario").

Hay dos condiciones del ámbito de operación de la empresa:

– Existen 3 turnos en el día, estos son: 07:00 a 15:00, 15:00 a 23:00 y 23:00 a 07:00.
– Existen 5 empleados para la distribución de los turnos.

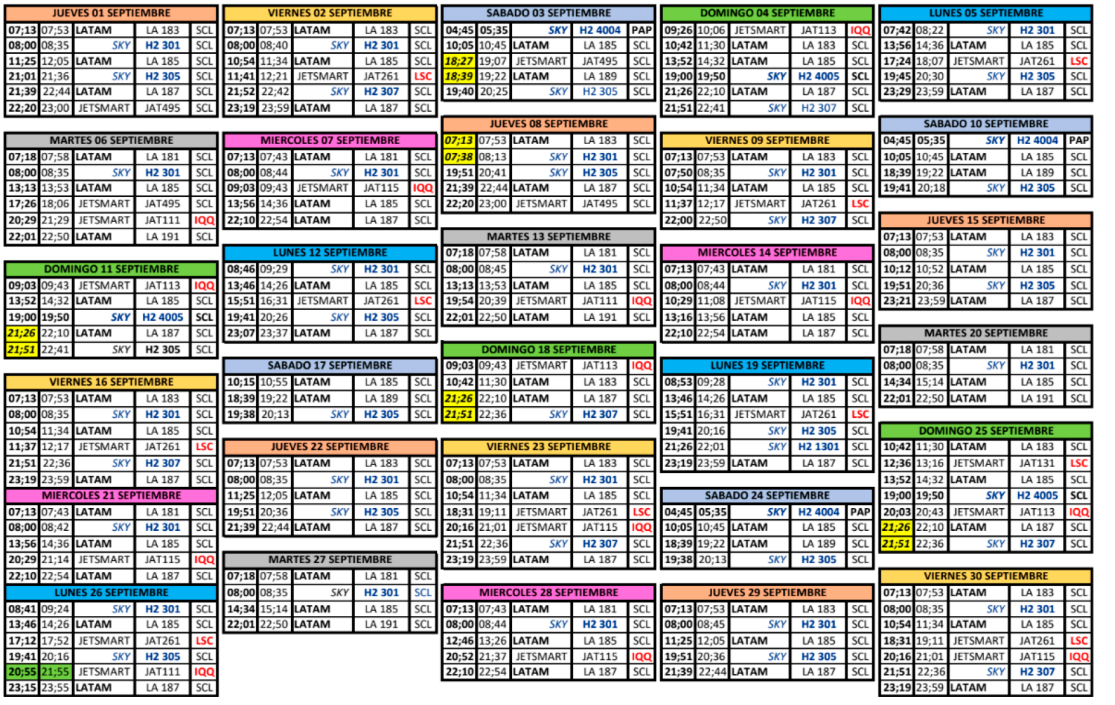
Adicionalmente, existe un conjunto de restricciones que deben cumplirse en la planificación de turnos:

1. Cada empleado trabaja como máximo un turno al día.
2. Un empleado que sea asignado en el turno de las 23:00 a 07:00 no puede ser asignado al día siguiente en el turno 07:00 a 15:00, ya que trabajaría dieciséis horas seguidas.
3. Cada turno debe tener como mínimo un empleado al día.
4. Cada empleado tiene un día libre a la semana (la semana no incluye los domingos).
5. Cada empleado tiene dos domingos libres al mes.
6. Se asignará una cantidad determinada de empleados a un turno específico en un día, según lo que indique el itinerario (Figura 1).
7. Los empleados restantes que no hayan sido asignados a un turno se distribuirán equitativamente desde el primer turno como prioridad, ya que hay más tráfico de aviones, seguido del segundo turno hasta el tercer turno.
8. Los meses que tengan cuatro domingos deberán asignar a un empleado especial adicional (distinto de los cinco empleados) para trabajar en dos domingos en cualquier turno, a este último se le denominará como "comodín".
9. La carga de trabajo de los empleados debe estar distribuida equitativamente para cada turno.

Con esta información se realiza la planificación que se puede observar en la Figura 2.

De lo anterior, la problemática reside en automatizar esta planificación, ya que la situación actual presenta

Fuente: Ariaca.



JUEVES 01 SEPTIEMBRE					
07;13	07;53	LATAM	LA 183	SCL
08;00	08;35	SKY	H2 301	SCL
11;25	12;05	LATAM	LA 185	SCL
21;01	21;36	SKY	H2 305	SCL
21;39	22;44	LATAM	LA 187	SCL
22;20	23;00	JETSMART	JAT495	SCL

VIERNES 02 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
08;00	08;40	SKY	H2 301	SCL
10;54	11;34	LATAM	LA 185	SCL
11;41	12;21	JETSMART	JAT261	LSC
21;52	22;42	SKY	H2 307	SCL
23;19	23;59	LATAM	LA 187	SCL

SABADO 03 SEPTIEMBRE				
04;45	05;35	SKY	H2 4004	PAP
10;05	10;45	LATAM	LA 185	SCL
18;27	19;07	JETSMART	JAT495	SCL
18;39	19;22	LATAM	LA 189	SCL
19;40	20;25	SKY	H2 305	SCL

DOMINGO 04 SEPTIEMBRE				
09;26	10;06	JETSMART	JAT113	IQQ
10;42	11;30	LATAM	LA 183	SCL
13;52	14;32	LATAM	LA 185	SCL
19;00	19;50	SKY	H2 4005	SCL
21;26	22;10	LATAM	LA 187	SCL
21;51	22;41	SKY	H2 307	SCL

LUNES 05 SEPTIEMBRE				
07;42	08;22	SKY	H2 301	SCL
13;56	14;36	LATAM	LA 185	SCL
17;24	18;07	JETSMART	JAT261	LSC
19;45	20;30	SKY	H2 305	SCL
23;29	23;59	LATAM	LA 187	SCL

MARTES 06 SEPTIEMBRE				
07;18	07;58	LATAM	LA 181	SCL
08;00	08;35	SKY	H2 301	SCL
13;13	13;53	LATAM	LA 185	SCL
17;26	18;06	JETSMART	JAT495	SCL
20;29	21;29	JETSMART	JAT111	IQQ
22;01	22;50	LATAM	LA 191	SCL

MIERCOLES 07 SEPTIEMBRE				
07;13	07;43	LATAM	LA 181	SCL
08;00	08;44	SKY	H2 301	SCL
09;03	09;43	JETSMART	JAT115	IQQ
13;56	14;36	LATAM	LA 185	SCL
22;10	22;54	LATAM	LA 187	SCL

JUEVES 08 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
07;38	08;13	SKY	H2 301	SCL
19;51	20;41	SKY	H2 305	SCL
21;39	22;44	LATAM	LA 187	SCL
22;20	23;00	JETSMART	JAT495	SCL

VIERNES 09 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
07;50	08;35	SKY	H2 301	SCL
10;54	11;34	LATAM	LA 185	SCL
11;37	12;17	JETSMART	JAT261	LSC
22;00	22;50	SKY	H2 307	SCL

SABADO 10 SEPTIEMBRE				
04;45	05;35	SKY	H2 4004	PAP
10;05	10;45	LATAM	LA 185	SCL
18;39	19;22	LATAM	LA 189	SCL
19;41	20;18	SKY	H2 305	SCL

DOMINGO 11 SEPTIEMBRE				
09;03	09;43	JETSMART	JAT113	IQQ
13;52	14;32	LATAM	LA 185	SCL
19;00	19;50	SKY	H2 4005	SCL
21;26	22;10	LATAM	LA 187	SCL
21;51	22;41	SKY	H2 305	SCL

LUNES 12 SEPTIEMBRE				
08;46	09;29	SKY	H2 301	SCL
13;46	14;26	LATAM	LA 185	SCL
15;51	16;31	JETSMART	JAT261	LSC
19;41	20;26	SKY	H2 305	SCL
23;07	23;37	LATAM	LA 187	SCL

MARTES 13 SEPTIEMBRE				
07;18	07;58	LATAM	LA 181	SCL
08;00	08;45	SKY	H2 301	SCL
13;13	13;53	LATAM	LA 185	SCL
19;54	20;39	JETSMART	JAT111	IQQ
22;01	22;50	LATAM	LA 191	SCL

MIERCOLES 14 SEPTIEMBRE				
07;13	07;43	LATAM	LA 181	SCL
08;00	08;44	SKY	H2 301	SCL
10;29	11;08	JETSMART	JAT115	IQQ
13;16	13;56	LATAM	LA 185	SCL
22;10	22;54	LATAM	LA 187	SCL

JUEVES 15 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
08;00	08;35	SKY	H2 301	SCL
10;12	10;52	LATAM	LA 185	SCL
19;51	20;36	SKY	H2 305	SCL
23;21	23;59	LATAM	LA 187	SCL

SABADO 17 SEPTIEMBRE				
10;15	10;55	LATAM	LA 185	SCL
18;39	19;22	LATAM	LA 189	SCL
19;38	20;13	SKY	H2 305	SCL

DOMINGO 18 SEPTIEMBRE				
09;03	09;43	JETSMART	JAT113	IQQ
10;42	11;30	LATAM	LA 183	SCL
21;26	22;10	LATAM	LA 187	SCL
21;51	22;36	SKY	H2 307	SCL

LUNES 19 SEPTIEMBRE				
08;53	09;28	SKY	H2 301	SCL
13;46	14;26	LATAM	LA 185	SCL
15;51	16;31	JETSMART	JAT261	LSC
19;41	20;16	SKY	H2 305	SCL
21;26	22;01	SKY	H2 1301	SCL
23;19	23;59	LATAM	LA 187	SCL

MARTES 20 SEPTIEMBRE				
07;18	07;58	LATAM	LA 181	SCL
08;00	08;35	SKY	H2 301	SCL
14;34	15;14	LATAM	LA 185	SCL
22;01	22;50	LATAM	LA 191	SCL

JUEVES 22 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
08;00	08;35	SKY	H2 301	SCL
11;25	12;05	LATAM	LA 185	SCL
19;51	20;36	SKY	H2 305	SCL
21;39	22;44	LATAM	LA 187	SCL

VIERNES 23 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
08;00	08;35	SKY	H2 301	SCL
10;54	11;34	LATAM	LA 185	SCL
18;31	19;11	JETSMART	JAT261	LSC
20;16	21;01	JETSMART	JAT115	IQQ
21;51	22;36	SKY	H2 307	SCL
23;19	23;59	LATAM	LA 187	SCL

SABADO 24 SEPTIEMBRE				
04;45	05;35	SKY	H2 4004	PAP
10;05	10;45	LATAM	LA 185	SCL
18;39	19;22	LATAM	LA 189	SCL
19;38	20;13	SKY	H2 305	SCL

DOMINGO 25 SEPTIEMBRE				
10;42	11;30	LATAM	LA 183	SCL
12;36	13;16	JETSMART	JAT131	LSC
13;52	14;32	LATAM	LA 185	SCL
19;00	19;50	SKY	H2 4005	SCL
20;03	20;43	JETSMART	JAT113	IQQ
21;26	22;10	LATAM	LA 187	SCL
21;51	22;36	SKY	H2 307	SCL

MARTES 27 SEPTIEMBRE				
07;18	07;58	LATAM	LA 181	SCL
08;00	08;35	SKY	H2 301	SCL
14;34	15;14	LATAM	LA 185	SCL
22;01	22;50	LATAM	LA 191	SCL

MIERCOLES 28 SEPTIEMBRE				
07;13	07;43	LATAM	LA 181	SCL
08;00	08;44	SKY	H2 301	SCL
12;46	13;26	LATAM	LA 185	SCL
20;52	21;37	JETSMART	JAT115	IQQ
22;10	22;54	LATAM	LA 187	SCL

JUEVES 29 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
08;00	08;45	SKY	H2 301	SCL
11;25	12;05	LATAM	LA 185	SCL
19;51	20;36	SKY	H2 305	SCL
21;39	22;44	LATAM	LA 187	SCL

VIERNES 30 SEPTIEMBRE				
07;13	07;53	LATAM	LA 183	SCL
08;00	08;35	SKY	H2 301	SCL
10;54	11;34	LATAM	LA 185	SCL
18;31	19;11	JETSMART	JAT261	LSC
20;16	21;01	JETSMART	JAT115	IQQ
21;51	22;36	SKY	H2 307	SCL
23;19	23;59	LATAM	LA 187	SCL

LUNES 26 SEPTIEMBRE				
08;41	09;24	SKY	H2 301	SCL
13;46	14;26	LATAM	LA 185	SCL
17;12	17;52	JETSMART	JAT261	LSC
19;41	20;16	SKY	H2 305	SCL
20;55	21;55	JETSMART	JAT111	IQQ
23;15	23;55	LATAM	LA 187	SCL

Figura 1. Itinerario de vuelo de septiembre de 2022.

Fuente: Ariaca.

DIA	FECHA	A. MONTANER	C. VEIRA	D. TRONCOSO	R. ZAVALA	R. ZUÑIGA
L	27	LIBRE	07;00 a 15;00	23;00 a 07;00	15;00 a 23;00	07;00 a 15;00
M	28	15;00 a 23;00	07;00 a 15;00	23;00 a 07;00	LIBRE	07;00 a 15;00
M	29	15;00 a 23;00	07;00 a 15;00	LIBRE	23;00 a 07;00	07;00 a 15;00
J	30	15;00 a 23;00	07;00 a 15;00	07;00 a 15;00	23;00 a 07;00	07;00 a 15;00
JULIO						
V	1	15;00 a 23;00	LIBRE	07;00 a 15;00	23;00 a 07;00	
S	2	15;00 a 23;00	07;00 a 15;00	07;00 a 15;00	23;00 a 07;00	
D	3	LIBRE	07;00 a 15;00	LIBRE	23;00 a 07;00	15;00 a 23;00

DIA	FECHA	A. MONTANER	C. VEIRA	D. TRONCOSO	R. ZAVALA	R. ZUÑIGA	
L	4	07;00 a 15;00	LIBRE	15;00 a 23;00	23;00 a 07;00	07;00 a 15;00	
M	5	07;00 a 15;00	23;00 a 07;00	15;00 a 23;00	LIBRE	07;00 a 15;00	
M	6	07;00 a 15;00	23;00 a 07;00	15;00 a 23;00	07;00 a 15;00	07;00 a 15;00	
J	7	07;00 a 15;00	23;00 a 07;00	15;00 a 23;00	07;00 a 15;00	07;00 a 15;00	
V	8	LIBRE	23;00 a 07;00	15;00 a 23;00	07;00 a 15;00		
S	9	15;00 a 23;00	23;00 a 07;00	LIBRE	07;00 a 15;00		9JUL JUNTA 09;40
D	10	15;00 a 23;00	LIBRE	23;00 a 07;00	LIBRE		Abaltolu hace turno dia

Figura 2. Planificación de turno de septiembre de 2022.

dos problemas en la empresa. Uno de ellos es el tiempo que toma en hacer esta planificación, ya que se tiene que considerar múltiples variables y restricciones a la hora de su elaboración. El segundo problema radica en que, al ser realizada por un empleado interno de la empresa, no existe imparcialidad desde el punto de vista de los demás empleados, lo que genera descontento.

Para abordar estas problemáticas, se presenta una solución basada en la implementación de un sistema que genera planificaciones de turnos, modelando el problema como un problema de satisfacción de restricciones (CSP) y su resolución mediante el solucionador CP-SAT proporcionado por OR-Tools. Esta solución no solo permite reducir significativamente el tiempo de planificación, sino que también elimina la intervención humana, garantizando una distribución imparcial y equitativa de los turnos entre los empleados.

El sistema Argos fue diseñado bajo el marco operativo que tienen las empresas que se dedican al rubro de suministro de combustible para aviones, lo cual justifica la necesidad de la mayoría de las restricciones. No obstante, cada empresa presenta su propio marco personalizado de trabajo, por lo tanto, dependiendo de la situación, es necesario que el sistema incorpore nuevas restricciones para que se adapte a la empresa. Para el caso de Ariaca, se presenta la restricción 8, la cual requiere de un empleado adicional que participe en la planificación de turno bajo ciertas normas. Esta restricción es muy particular, por lo no es posible encontrar un sistema que sea capaz de abordar este tipo de restricción.

REVISIÓN DE LA LITERATURA

Timetabling problem

Básicamente el problema de la asignación de los empleados a turnos de la empresa Ariaca es una variación del conocido timetabling (scheduling) problem, o problema de asignación de horarios [1], [2]. Un problema de asignación de horarios es un problema de optimización o de satisfacción de restricciones que considera la asignación de recursos.

Los problemas de asignación de horarios se consideran problemas difíciles [3], para los cuales han sido utilizadas variadas técnicas de resolución. Babaei, Karimpour, and Hadidi [4] clasifica estas técnicas en cuatro categorías:

- *Métodos de Investigación Operativa:* Esta categoría incluye técnicas basadas en coloreo de grafos, métodos de programación lineal y entera y programación de satisfacción de restricciones.
- *Métodos metaheurísticos:* El enfoque meta-heurístico incluye dos vertientes principales, basados en población (algoritmos genéticos y evolutivos, optimización basada en colonias de hormigas y otros) y de solución única (búsqueda tabú, enfriamiento simulado, búsqueda local y otros).
- *Métodos inteligentes novedosos:* Esta incluye métodos híbridos, métodos fuzzy, métodos basados en inteligencia artificial y otros.
- *Métodos basados en sistemas distribuidos multi agente:* Estos sistemas incluyen múltiples

agentes autónomos que resuelven parcialmente un problema, no consideran un sistema de control global, los datos son distribuidos y la computación es asíncrona.

Programación de restricciones

La programación de restricción es un paradigma utilizado en la resolución de problemas de búsqueda combinatoria que se basa en una amplia gama de técnicas de inteligencia artificial, informática, bases de datos, lenguajes de programación e investigación de operaciones.

Actualmente se aplica con éxito a muchos dominios, como programación, planificación, enrutamiento de vehículos, configuración, redes y bioinformática [5]-[12]. La idea básica en la programación de restricciones es que el usuario especifica las restricciones del problema y se usa un solucionador de restricciones de propósito general para resolverlas [13].

Un problema de satisfacción de restricciones (CSP) establece qué relaciones deben mantenerse entre las variables de decisión dadas. Los solucionadores de restricciones toman un problema del mundo real como este, representado en términos de variables de decisión y restricciones, y encuentran una asignación a todas las variables que satisface las restricciones. Los solucionadores de restricciones buscan en el espacio de la solución de forma sistemática, utilizando algoritmos de backtracking o branch and bound, o utilizan formas de búsqueda local que pueden encontrar soluciones no óptimas.

Un problema de satisfacción de restricciones requiere que valores seleccionados de un dominio finito dado sean asignados a cada variable en el problema, tal que se satisfagan todas las restricciones definidas en el problema [14]. Formalmente, un CSP puede ser representado por una terna (X, D, C), donde X es un conjunto de variables, D es una tupla de dominios finitos que contienen los posibles valores que pueden asignarse a cada variable y C es un conjunto de restricciones. Una solución a un CSP es una asignación (a1, a2, …, an) de valores a cada variable, de tal manera que se satisfagan las restricciones [13], [15], [16].

OR-tools

OR-Tools [17], [18] es un paquete de software de código abierto para la resolución de problemas de optimización, como problemas en enrutamiento de vehículos, flujos, programación lineal y entera, así como programación de restricciones [19]-[23]. OR-Tools, cuenta con metaheurísticas que buscan encontrar la mejor solución a un problema entre un conjunto de posibles soluciones.

Una de las ventajas potenciales de Google OR-Tools consiste en que, a pesar de estar desarrollado en C++, permite el modelamiento de problemas en múltiples lenguajes de programación (provee wrappers para Python, C# y Java) [24], [25], del mismo modo, cuenta con múltiples solucionadores específicos para resolverlos, algunos comerciales (Gurobi, CPLEX) y otros de código abierto (SCIP, GLPK, GLOP, CP-SAT) [17].

OR-Tools ha ganado entre tres y cuatro medallas de oro anuales (2018-2022) en el MiniZinc Challenge [26], [27], la competencia internacional de solucionadores de problemas de satisfacción de restricciones. OR-Tools como herramienta trabaja con una metáfora basada en un modelo (model) y un solucionador (solver). El modelo provee los métodos sobre el cual se definen las variables, dominios y las restricciones de integridad del problema, el solucionador provee los métodos para encontrar las soluciones al problema considerando las especificaciones que han sido incorporadas al modelo.

En el contexto de este proyecto se ha seleccionado Python como un lenguaje hospedero, el que proveerá de las estructuras, instrucciones, operadores y librerías de apoyo a la especificación del problema requerido por OR-Tools.

METODOLOGÍA

La estrategia fundamental para enfrentar el problema de asignación de turnos consistió en su especificación como un problema de satisfacción de restricciones, esto implica la definición de las variables, dominios y restricciones de integridad en el modelo y su posterior resolución mediante el uso del solucionador CP-SAT de OR-Tools.

Variables y dominio

En el problema de asignación de turnos, el proceso clave es la asignación de empleados en cada turno. En el caso usual, se tienen cinco empleados disponibles

que deben ser distribuidos en tres turnos por día de cada mes. Bajo este escenario, cada variable, definida sobre un dominio binario, representa si a un empleado se asignará (1) o no (0) a trabajar en un cierto turno de un día.

En la gestión de los meses se utilizó una estructura que facilita la manipulación de cada una de las variables relacionadas con los empleados y los días correspondientes. Para definir la estructura fue necesario saber la cantidad de días del mes y en qué día de la semana comenzaba. Para ello, se utilizó el método monthrange de la biblioteca Calendar de Python que permite obtener la cantidad de días en el mes y el día de la semana en que comienza.

Es importante destacar, que la idea de utilizar monthrange para el mes anterior y posterior, fue obtener una estructura que facilitase la aplicación de las restricciones requeridas. En especial si se considera que hay semanas en donde termina un mes y se inicia otro, por lo que es importante considerar estos días para aplicar ciertas restricciones como:

– Cada empleado tiene un día libre a la semana (la semana no incluye los domingos).
– Un empleado asignado al turno 3 no puede ser asignado al día siguiente en el turno 1.

Por ejemplo, si se quiere generar la planificación de turno del mes de junio 2023, se debe considerar cuando empieza el mes. En este caso empezaría el jueves 1, por lo tanto, si se quiere aplicar la restricción 1, es necesario identificar que empleados tuvieron días libres el lunes 29, martes 30 y miércoles 31 de mayo para asignar los días libres correspondientes a aquellos empleados que aún no los tengan en la primera semana de junio.

En el caso de la segunda restricción, es fundamental identificar que empleado trabajó en el turno 3 del día 31 de mayo, para asignar los empleados que deben trabajar en el primer turno del día 1 de junio.

La Figura 3 muestra la incorporación de las variables requeridas al modelo para determinar los turnos de empleados (modelo.NewBolVar). En el código proporcionado, se recorre una lista empleadoPlanificacion y se crean variables booleanas que representan la asignación de empleados a los turnos.

```
for empleado in empleadoPlanificacion:
  ...
  for _ in range(cant_turno):
    turnos.append(modelo.NewBoolVar(empleado))
```

Figura 3. Incorporación de las variables del problema al modelo.

Restricciones

En esta sección se describe gruesamente la implementación de cada restricción mostrando los fragmentos de código más significativos. El código completo de la solución desarrollada puede encontrarse en el sitio github del proyecto en [28], el seudocódigo asociado puede encontrarse en [29].

i. Cada empleado trabaja como máximo un turno por día.

Esta restricción limita la cantidad de turnos que puede tomar un empleado al día.

Para implementar esta restricción, se itera por cada empleado de cada semana y de cada día, asegurando que la suma de las variables de turno para ese día sea menor o igual a uno. Si la suma es igual a uno, significa que al menos una variable de turno tiene asignado este valor, lo que indica que el empleado ya tiene un turno asignado. Por otro lado, si la suma es igual a cero, significa que el empleado no trabaja ese día (Figura 4).

Esta restricción utiliza el método AddAtMostOne, que toma como parámetro un conjunto de variables y establece que como máximo una de las variables de dicho conjunto pueda tener el valor uno.

En la Figura 5, se presenta una comparación entre los resultados de asignación de turnos para los

```
for empleado in all_empleado:
  for semana in range(len(cantidad_semana)):
    for dia in range(cantidad_semana[semana]):
      modelo.AddAtMostOne(mes[semana][dia][3][empleado][turno] for turno in range(cant_turno))
```

Figura 4. Implementación de la restricción 1.

SIN RESTRICCIÓN	CON RESTRICCIÓN
empleado n°1 - turno: 3	empleado n°1 - turno: 3
empleado n°2 - turno: 1	empleado n°2 - turno: 1
empleado n°3 - turno: 1 2 3	empleado n°3 - turno: 2
empleado n°4 - turno: 2 3	empleado n°4 - turno: 2
empleado n°5 - turno: 1 2 3	empleado n°5 - turno: 2

Figura 5. Comparativa de asignación de turnos con la restricción 1.

empleados sin incluir la restricción y con la restricción aplicada. Al no incluir la restricción, se observa que algunos empleados tienen múltiples turnos asignados, mientras que, al agregar la restricción al modelo, los empleados tienen asignados un único turno, cumpliendo con la restricción de trabajar como máximo un turno por día.

ii. Un empleado que es asignado en el turno de las 23:00 a 07:00 no puede ser asignado al día siguiente en el turno 07:00 a 15:00, ya que trabajaría 16 horas seguidas.

Esta restricción evita que un empleado trabaje dos turnos seguidos en días consecutivos, ya que podría resultar que un empleado trabaje dieciséis horas seguidas.

Esta restricción sigue un enfoque similar al de la restricción anterior, donde se itera por cada empleado en cada semana y día. Si el día siguiente pertenece a la misma semana, se añaden a una lista el último turno del día actual y el primer turno del día siguiente. Por otro lado, si el día siguiente no pertenece a la misma semana, se verifica si existe un día siguiente válido. En este caso, se agregan a la lista el último turno del empleado y el primer turno del día siguiente de la siguiente semana (Figura 6).

Una vez que se han incorporado las variables a la lista, se aplica la restricción al modelo mediante el método Add que toma como parámetro una expresión lógica que debe cumplirse. En este caso, la suma de los elementos de la lista debe ser menor o igual a uno.

En la Figura 7, al analizar la asignación de turnos en el lunes 1, se puede observar que, en el escenario sin restricción, los empleados 4 y 5 trabajan en el

```
for empleado in all_empleado:
    for semana in range(len(cantidad_semana)):
        for dia in range(cantidad_semana [semana]):
            lista = []
            ...
                lista.append(mes[semana][dia][3][empleado][2]) # turno 3
                lista.append(mes[semana][dia+diaSiguiente][3][empleado][0]) # turno 1 día siguiente
        ...
        if lista: modelo.Add(sum(lista) <= 1)
```

Figura 6. Implementación de la restricción 2.

SIN RESTRICCIÓN		
Lunes 1	Martes 2	Miércoles 3
empleado n°1 - turno: 1	empleado n°1 - turno: 1	empleado n°1 - turno: 1
empleado n°2 - turno: 1	empleado n°2 - turno: 3	empleado n°2 - turno: 1
empleado n°3 - turno: 1	empleado n°3 - turno: 3	empleado n°3 - turno: 1
empleado n°4 - turno: 3	**empleado n°4 - turno: 1**	empleado n°4 - turno: 2
empleado n°5 - turno: 3	**empleado n°5 - turno: 1**	empleado n°5 - turno: 1
CON RESTRICCIÓN		
Lunes 1	Martes 2	Miércoles 3
empleado n°1 - turno: 1	empleado n°1 - turno: 1	empleado n°1 - turno: 1
empleado n°2 - turno: 1	empleado n°2 - turno: 2	empleado n°2 - turno: 1
empleado n°3 - turno: 1	empleado n°3 - turno: 2	empleado n°3 - turno: 1
empleado n°4 - turno: 3	empleado n°4 - turno: 3	empleado n°4 - turno: 2
empleado n°5 - turno: 3	empleado n°5 - turno: 2	empleado n°5 - turno: 1

Figura 7. Comparativa de asignación de turnos de la restricción 2.

último turno del día y al día siguiente, en el primer turno. Esta asignación genera que los empleados trabajen dos turnos seguidos, ya que el turno 3 refleja el turno de las 23:00 a 7:00 y el turno 1 el turno de las 7:00 a 15:00. No obstante, al incluir la restricción al modelo, este problema desaparece. En el mismo escenario del lunes 1, los empleados 4 y 5 son asignados a turnos distintos el día siguiente, eliminado el problema de trabajar dos turnos seguidos.

A pesar de haber solucionado el problema, aún queda otro problema por resolver. Si se observa de nuevo la Figura 7, se puede ver que los lunes 1 y miércoles 3 no hay empleados asignados para el turno 2 y el turno 3, respectivamente. Este se va abordar en la siguiente restricción.

iii. Cada turno debe tener como mínimo un empleado al día.

Con esta restricción se asegura que haya como mínimo un empleado para cada turno.

Esta restricción difiere de las demás al comenzar iterando por cada día de la semana, seguido por los turnos y finalmente los empleados. Esta estructura de iteración se utiliza para construir una lista que almacena los turnos correspondientes a cada empleado en un día determinado y turno en específico. La restricción impone que la suma de todos los turnos en la lista en un día debe ser mayor o igual a uno (Figura 8). Además, se suma uno más a la variable turnos_totales. Esta variable contabiliza la cantidad de empleados que se van asignando para cada turno.

En la Figura 9, se puede ver los resultados de incluir la restricción *iii* al modelo. Al incluir la restricción, cada turno del día tiene garantizado la asignación de al menos un empleado. Es importante destacar que esta restricción no se aplica los días domingos, ya que existe una restricción para esos días que se explica más adelante.

Además, en la Figura 9 se observa otro problema y es que no todos los empleados tienen un día libre a la semana. Es importante que cada empleado tenga un día libre a la semana, ya que va en contra de las regulaciones laborales. Este problema da introducción a la siguiente restricción que es donde se soluciona este problema.

iv. Cada empleado tiene un día libre a la semana (la semana no incluye los domingos).

Anteriormente, en la restricción "Cada empleado trabaja como máximo un turno por día", se explicó que, si la suma de los turnos de un empleado es igual

```
for semana in range(len(cantidad_semana)):
    for dia in range(cantidad_semana[semana]):
        if dia != Domingo:
            for turno in range(cant_turno):
                lista = []
                ...
                    if mes[semana][dia][0] == meses[month-1]:
                        turno_totales[turno]+=1
                    for empleado in all_empleado:
                        lista.append(mes[semana][dia][3][empleado][turno])
                ...
            modelo.Add(sum(lista) >= 1)
```

Figura 8. Implementación de la restricción 3.

Lunes 1	Martes 2	Miércoles 3
empleado n°1 - turno: 1	empleado n°1 - turno: 1	empleado n°1 - turno: 2
empleado n°2 - turno: 2	empleado n°2 - turno: 3	empleado n°2 - turno: 0
empleado n°3 - turno: 0	empleado n°3 - turno: 2	empleado n°3 - turno: 3
empleado n°4 - turno: 3	empleado n°4 - turno: 3	empleado n°4 - turno: 2
empleado n°5 - turno: 2	empleado n°5 - turno: 0	empleado n°5 - turno: 1

Figura 9. Resultados de la inclusión de la restricción 3 al modelo.

a cero, significa que el empleado no trabaja ese día. Esta restricción da posibilidad a que los empleados tengan el día libre. Por lo tanto, para garantizar que solo se otorgue un día libre a los empleados en la semana, se sumarán los turnos desde el lunes hasta el sábado. Al sumar los turnos de estos seis días, el resultado total será igual a cinco. Esto significa que habrá un día con un valor en cero, que corresponderá al día libre asignado a los empleados.

En la implementación de la restricción, se sigue el enfoque de iterar por cada empleado y cada día de la semana, incluyendo los turnos del día. En cada iteración, se verifica si el día es diferente a domingo para evitar incluirlo en la lista de turnos. Luego, se crea una lista llamada lista_semana que almacena los turnos de la semana correspondiente a un empleado en cada iteración (Figura 10).

El código anterior es efectivo si no existe una planificación del mes anterior o si este termina en domingo. En caso contrario, se requiere identificar los empleados que participaron en la planificación anterior y verificar si tuvieron su día libre (Figura 11).

Para identificar los empleados que participaron en la planificación anterior, se itera por empleado de la planificación anterior, seguido de la iteración del índice del empleado, con el fin de comprobar si el empleado de la planificación anterior participa en la planificación actual. Si participa, entonces se recorre cada turno del día, verificando en que turno trabajo y cual no. Los turnos del día que no trabaja el empleado se le asigna con un cero (0) al modelo, mientras que el turno que si trabajo con un uno (1). De esta forma cuando se aplique la restricción *iv* se tendrán en cuenta tanto los días que trabajaron y los que no.

En la Figura 12, se puede observar los resultados de incluir la restricción *iv* al modelo, considerando que había una planificación del mes anterior. Al incluir la restricción cada empleado tiene un día libre a la semana. Al inicio de la semana el empleado nº 3 tiene el día libre, seguido del empleado nº 5, empleado nº 2, empleado nº 1 y, finalmente el empleado nº 4.

v. Cada empleado tiene dos domingos libres al mes. Esta restricción, se aplica calculando la suma de todos los turnos de un empleado en los domingos del mes y se iguala al total de domingos menos dos, donde esta cifra representa los domingos libres que tiene un empleado en el mes (variable DomingosLibresAlMes).

Antes de aplicar la restricción, se realiza una búsqueda de todos los domingos que se encuentran en el mes

```
cantidadDiaSemana = 6 # lunes a sábado
diasLibreCadaEmpleadoPorSemana = 1

for empleado in all_empleado:
    for semana in range(len(cantidad_semana)):
        lista_semana = []
        for dia in range(cantidad_semana[semana]):
            if mes[semana][dia][2] != 'Domingo':
                for turno in range(cant_turno):
                    lista_semana.append(mes[semana][dia][3][empleado][turno])
        modelo.Add(sum(lista_semana) == cantidadDiaSemana - diasLibreCadaEmpleadoPorSemana)
```

Figura 10. Implementación de la restricción 4.

```
for empleadoAnterior in empleadoPlanificacionAnterior:
    for empleado in range(len(empleadoPlanificacionAnterior)):
        if empleadoAnterior == planificacionAnterior[dia][empleado][1]:
            for turno in range(cant_turno):
                if planificacionAnterior[dia][empleado][2] == turno+1:
                    modelo.Add(mes[semana][dia][3][empleado][turno]==1)
                else: modelo.Add(mes[semana][dia][3][empleado][turno]==0)
```

Figura 11. Identificación de los empleados de la planificación anterior. Restricción 4.

Lunes 31	Martes 1	Miércoles 2
empleado n°1 - turno: 1	empleado n°1 - turno: 1	empleado n°1 - turno: 2
empleado n°2 - turno: 2	empleado n°2 - turno: 3	empleado n°2 - turno: 0
empleado n°3 - turno: 0	empleado n°3 - turno: 2	empleado n°3 - turno: 3
empleado n°4 - turno: 3	empleado n°4 - turno: 3	empleado n°4 - turno: 2
empleado n°5 - turno: 2	empleado n°5 - turno: 0	empleado n°5 - turno: 1
Jueves 3	**Viernes 4**	**Sábado 5**
empleado n°1 - turno: 0	empleado n°1 - turno: 1	empleado n°1 - turno: 3
empleado n°2 - turno: 1	empleado n°2 - turno: 2	empleado n°2 - turno: 1
empleado n°3 - turno: 2	empleado n°3 - turno: 2	empleado n°3 - turno: 3
empleado n°4 - turno: 3	empleado n°4 - turno: 0	empleado n°4 - turno: 1
empleado n°5 - turno: 1	empleado n°5 - turno: 3	empleado n°5 - turno: 2

Figura 12. Resultados de la inclusión de la restricción 4 al modelo.

actual (lista domingos). Posteriormente, se crea una lista llamada lista_domingo donde se almacenan los turnos correspondientes a los domingos del mes para cada empleado en particular (Figura 13).

En la Figura 14, se observa los resultados de la inclusión de la restricción (*v)* al modelo, donde cada empleado tiene asignado dos domingos libres al mes. No obstante, la mayoría de los domingos (todos menos el domingo 9) no tienen asignados como mínimo un empleado en cada turno. Este problema se resuelve en la restricción *viii*.

vi. Se asignará una cantidad determinada de empleados a un turno específico en un día, según lo que indique el itinerario

Está restricción se va a aplicar siempre y cuando el usuario ingrese los itinerarios, es decir, los turnos en los que se requiere una cantidad específica de empleados.

La asignación de empleados basada en el itinerario comienza verificando si existe un itinerario para el día, además, este no puede ser de un domingo. Si no se cumple esta condición entonces, se consulta

```
DomingosLibresAlMes = 2

for empleado in all_empleado:
    lista_domingo = []
    for domingo, semana in domingos:
        for turno in range(cant_turno):
            lista_domingo.append(mes[semana][domingo][3][empleado][turno])
    modelo.Add(sum(lista_domingo) == len(domingos) - DomingosLibresAlMes)
```

Figura 13. Implementación de la restricción 5.

Domingo 2	Domingo 9	Domingo 16
empleado n°1 - turno: 1	empleado n°1 - turno: 1	empleado n°1 - turno: 2
empleado n°2 - turno: 0	empleado n°2 - turno: 2	empleado n°2 - turno: 3
empleado n°3 - turno: 3	empleado n°3 - turno: 3	empleado n°3 - turno: 2
empleado n°4 - turno: 0	empleado n°4 - turno: 2	empleado n°4 - turno: 0
empleado n°5 - turno: 1	empleado n°5 - turno: 0	empleado n°5 - turno: 3
Domingo 23	**Domingo 30**	
empleado n°1 - turno: 0	empleado n°1 - turno: 0	
empleado n°2 - turno: 0	empleado n°2 - turno: 3	
empleado n°3 - turno: 0	empleado n°3 - turno: 0	
empleado n°4 - turno: 1	empleado n°4 - turno: 2	
empleado n°5 - turno: 0	empleado n°5 - turno: 2	

Figura 14. Resultados de la inclusión de la restricción 5 al modelo.

por el siguiente día de la semana, pero si existe un itinerario entonces se inicializan dos variables: suma_itinerario y acumulador.
La variable suma_itinerario es una lista que se utilizará para almacenar la cantidad de empleados requeridos por el itinerario en el día, mientras que la variable acumulador tiene la cantidad de empleados disponibles para asignar como turno extra en ese mismo día.

En la implementación de esta restricción, se procede a iterar sobre los itinerarios, ya que puede darse el caso que haya dos itinerarios en un día, pero para turnos distintos. Para cada itinerario, se itera sobre los turnos, verificando si el turno del itinerario coincide con la iteración del turno del día.

Si hay una coincidencia se procede a guardar en la lista suma_itinerario la variable _itinerario[“aviones”], que representa la cantidad de empleados extras que se necesitan en el turno. A partir de este punto hay 3 escenarios que puede ocurrir con el itinerario (Figura 15).

- El primer escenario se produce cuando se puede realizar la asignación solicitada por el itinerario. Esto implica que la suma de la variable suma_itinerario y cant_turno (cantidad de turno) es menor o igual a la cantidad de empleado disponible.
- El segundo escenario ocurre cuando no es posible asignar la cantidad de empleados indicada por el itinerario, pero es posible asignar parte de él. Esto implica que la suma de (suma_itinerario) y cantidad de turno debe ser mayor a la cantidad de empleados. La parte que no se asignó se guarda en la lista lista_alarma, indicando el día turno y la cantidad de empleados que no se pudieron asignar para cumplir con el itinerario.
- El tercer escenario, es cuando no se puede cumplir con el itinerario, por lo tanto, se guarda

```
if itinerario_dia and dia != Domingo:
   suma_itinerario = []
   acumulador = cantidad_empleado - cant_turno
   for _itinerario in itinerario_dia:
    for turno in range(cant_turno):
     if _itinerario["turno"] == (turno+1):
      suma_itinerario.append(_itinerario["aviones"])

      # Alcanza y asigna esa cantidad a los empleados
      if   sum(suma_itinerario) + cant_turno <= cantidad_empleado and acumulador > 0
          and trabajo_extra >= _itinerario["aviones"]:
          ...
          if lista: modelo.Add( sum(lista) >= _itinerario["aviones"] + 1)
          turno_totales[turno]+= _itinerario[“aviones”]+1

      # No alcanza, pero es posible asignar algunos empleados al turno
      elif sum(suma_itinerario) + cant_turno > cantidad_empleado and acumulador > 0
          and trabajo_extra >= _itinerario["aviones"]:
          ...
          if lista: modelo.Add( sum(lista) >= cantidad + 1)
          lista_alarma.append([_itinerario["dia"],
          _itinerario["turno"],_itinerario["aviones"]-cantidad ])
          turno_totales[turno]+=cantidad

      # No alcanza y los manda como alarma
      else:
          lista_alarma.append([ _itinerario["dia"], _itinerario["turno"], _itinerario["aviones"]])
```

Figura 15. Implementación de la restricción 6.

en la lista lista_alarma el día, turno y la cantidad solicitada por el itinerario para posteriormente avisarle al usuario que no se puede cumplir con esta asignación.

Hay que destacar que para el primer y segundo escenario se siguen contabilizando la cantidad de empleados que se vayan asignado para cada turno.

En la Figura 16, se observa el ejemplo de un itinerario, el cual se utilizó para generar la planificación de la Figura 17. Al incluir la restricción al modelo, se toma como prioridad la asignación de los empleados para que cumplan con el itinerario.

vii. Los empleados restantes que no hayan sido asignados a un turno se distribuirán equitativamente desde el primer turno como prioridad, ya que hay más tráfico de aviones, seguido del segundo turno hasta el tercer turno.

Esta restricción se aplica siempre y cuando queden empleados por asignar. La asignación de los empleados sobrantes del itinerario se divide en tres escenarios: mes pasado, actual y siguiente.

Se mostrará el escenario correspondiente al mes actual (Figura 18). En primer lugar, si el día actual tiene itinerario, se verifica si hay otros itinerarios para ese mismo día. Esto se hace para validar que los empleados solicitados por los itinerarios no superen la cantidad de empleados disponible en el día. Entonces, se suma uno más a la variable turnos_totales en el índice que señale turnoAsignar y se resta uno a la variable cantidad_turno_extra. La variable cantidad_turno_extra representa la cantidad de turnos de empleados disponible, por lo tanto, al realizar una asignación se debe realizar la resta correspondiente. Se crea una lista vacía para guardar todos los turnos de los empleados de índice turnoAsignar.

Luego se suma todos los turnos en la lista para aplicar la restricción que afirma que la suma de todos los turnos de índice turnoAsignar debe ser mayor o igual a empleadoAdicionales + 1 (el incremento en uno corresponde a que cada turno tiene como mínimo un empleado).

En la Figura 19, se puede observar la aplicación de la restricción, donde cada día tiene asignado dos

Lunes 1	Martes 2	Miércoles 3
turno: 3	turno: 1	turno: 3
aviones: 3	aviones: 2	aviones: 2

Figura 16. Ejemplo de un itinerario

Lunes 1	Martes - 2	Miércoles - 3
empleado n°1 - turno: 3	empleado n°1 - turno: 0	empleado n°1 - turno: 1
empleado n°2 - turno: 3	empleado n°2 - turno: 3	**empleado n°2 - turno: 3**
empleado n°3 - turno: 3	empleado n°3 - turno: 2	empleado n°3 - turno: 1
empleado n°4 - turno: 1	**empleado n°4 - turno: 1**	empleado n°4 - turno: 2
empleado n°5 - turno: 2	**empleado n°5 - turno: 1**	**empleado n°5 - turno: 3**
Jueves - 4	Viernes - 5	Sábado - 6
empleado n°1 - turno: 3	empleado n°1 - turno: 1	empleado n°1 - turno: 1
empleado n°2 - turno: 0	empleado n°2 - turno: 2	empleado n°2 - turno: 2
empleado n°3 - turno: 3	empleado n°3 - turno: 0	empleado n°3 - turno: 0
empleado n°4 - turno: 1	empleado n°4 - turno: 3	empleado n°4 - turno: 3
empleado n°5 - turno: 2	empleado n°5 - turno: 0	empleado n°5 - turno: 0

Figura 17. Resultado de la inclusión de la restricción 6 al modelo.

```
turnos_totales[turnoAsignar]+=1
cantidad_turno_extra-=1
lista = []
for empleado in all_empleado:
    lista.append(mes[semana][dia][3][empleado][turnoAsignar])
if lista: modelo.Add(sum(lista) >= empleadoAdicionales + 1)
```

Figura 18. Implementación de la restricción 7.

Lunes 1	Martes - 2	Miércoles - 3
empleado n°1 - turno: 2	empleado n°1 - turno: 0	empleado n°1 - turno: 2
empleado n°2 - turno: 1	empleado n°2 - turno: 1	empleado n°2 - turno: 1
empleado n°3 - turno: 1	empleado n°3 - turno: 3	empleado n°3 - turno: 3
empleado n°4 - turno: 2	empleado n°4 - turno: 1	empleado n°4 - turno: 0
empleado n°5 - turno: 3	empleado n°5 - turno: 2	empleado n°5 - turno: 1
Jueves - 4	**Viernes - 5**	**Sábado - 6**
empleado n°1 - turno: 1	empleado n°1 - turno: 3	empleado n°1 - turno: 3
empleado n°2 - turno: 3	empleado n°2 - turno: 2	empleado n°2 - turno: 0
empleado n°3 - turno: 2	empleado n°3 - turno: 0	empleado n°3 - turno: 1
empleado n°4 - turno: 1	empleado n°4 - turno: 1	empleado n°4 - turno: 1
empleado n°5 - turno: 0	empleado n°5 - turno: 1	empleado n°5 - turno: 2

Figura 19. Resultado de la inclusión de la restricción 7 al modelo.

empleados para el turno 1, además se asignan dos empleados para turno 2 el lunes 1.

A diferencia de la figura anterior, la Figura 20 muestra el resultado de la inclusión de la restricción en conjunto con el itinerario de la Figura 16. En la figura se puede observar cómo se le dio prioridad a la asignación del itinerario para generar la planificación, ya que el día lunes tiene tres empleados asignados para el turno 3, dejando solo un empleado para el turno 1.

viii. Los meses que tengan cuatro domingos deberán asignar a un empleado especial adicional (distinto de los cinco empleados) para trabajar en dos domingos en cualquier turno, a este último se le denominará como "comodín".

En un mes con cuatro domingos, solo se pueden asignar dos empleados para cada domingo, ya que se debe respetar la restricción "Cada empleado tiene dos domingos libres al mes". Pero al respetar esta restricción, no se cumple la restricción "Cada turno tiene como mínimo un empleado", ya que quedarán dos turnos sin asignar.

Para resolver este dilema, se utiliza una lista llamada domingos_asignacion, la cual incluye sublistas que representan la cantidad de empleados a asignar en cada turno de cada domingo del mes. En esta estructura se verifica que los domingos no tienen empleados por asignar en un turno, a estos se les asigna un empleado denominado "Comodín". En particular, este empleado será asignado para trabajar en los dos turnos que faltan por asignar. Cabe destacar que estos turnos no pueden estar ubicados en un mismo domingo.

Para implementar esta restricción se itera por cada domingo, seguido de la iteración de cada turno, para inicializar una lista vacía que guardará todos los turnos de los empleados para el domingo. Se iteran por cada empleado, verificando que pertenezca a la planificación del mes actual para guardar los turnos. Una vez guardado los turnos de los empleados, se aplica la restricción a la lista domingos_asignación (Figura 21).

Previamente, a la asignación se realiza un recuento de la cantidad de empleados que se van asignar los días domingos y se sumarán a la lista turno_totales.

Lunes 1	Martes - 2	Miércoles - 3
empleado n°1 - turno: 2	**empleado n°1 - turno: 1**	empleado n°1 - turno: 2
empleado n°2 - turno: 3	empleado n°2 - turno: 3	**empleado n°2 - turno: 3**
empleado n°3 - turno: 3	empleado n°3 - turno: 0	**empleado n°3 - turno: 3**
empleado n°4 - turno: 1	**empleado n°4 - turno: 1**	empleado n°4 - turno: 1
empleado n°5 - turno: 3	empleado n°5 - turno: 2	empleado n°5 - turno: 1
Jueves - 4	**Viernes - 5**	**Sábado - 6**
empleado n°1 - turno: 1	empleado n°1 - turno: 3	empleado n°1 - turno: 0
empleado n°2 - turno: 3	empleado n°2 - turno: 0	empleado n°2 - turno: 3
empleado n°3 - turno: 2	empleado n°3 - turno: 2	empleado n°3 - turno: 1
empleado n°4 - turno: 1	empleado n°4 - turno: 1	empleado n°4 - turno: 0
empleado n°5 - turno: 0	empleado n°5 - turno: 1	empleado n°5 - turno: 2

Figura 20. Resultado de la inclusión de la restricción 7 al modelo con un itinerario.

```
for domingo, semana in domingos:
    for turno in range(cant_turno):
        lista = []
        for empleado in all_empleado:
            lista.append(mes[semana][domingo][3][empleado][turno])
        modelo.Add(sum(lista) == domingos_asignacion[semana][turno])
```

Figura 21. Implementación de la restricción 8.

En la Figura 22, se puede ver la inclusión de la restricción 8 sobre cómo se asignan los turnos de los empleados para un mes con cuatro domingos. También se puede ver la asignación del empleado "comodín" para los domingos 14 y 21, donde se asignan los turnos 3 y 2 respectivamente, ocupando así los turnos que no tienen asignado ningún empleado.

En la Figura 23, se ve la aplicación de la restricción pera para un mes que tiene cinco domingos. A diferencia de la figura anterior, no es necesario recurrir al empleado comodín, ya que cada turno tiene asignado al menos un empleado.

ix. La carga de trabajo de los empleados debe estar distribuida equitativamente para cada turno. En esta última restricción se utiliza la lista turnos_totales que tiene la cantidad de empleados que son asignados a cada turno. Adicionalmente, se utilizan las listas min_turno y max_turno que representan la cantidad de veces que un empleado debe trabajar en cada turno. Por ejemplo, si hay un turno con 57 instancias y cinco empleados, cada empleado debe trabajar entre 11 (min) y 12 (max) veces en el turno.

En la implementación de esta restricción (Figura 24) se itera por turnos, donde se aplica la restricción para cada turno que tenga el empleado (lista jornada).

Domingo 7
empleado n°1 - turno: 0
empleado n°2 - turno: 1
empleado n°3 - turno: 2
empleado n°4 - turno: 3
empleado n°5 - turno: 0

Domingo – 14
empleado n°1 - turno: 0
empleado n°2 - turno: 0
empleado n°3 - turno: 2
empleado n°4 - turno: 1
empleado n°5 - turno: 0
comodín - turno: 3

Domingo - 21
empleado n°1 - turno: 1
empleado n°2 - turno: 0
empleado n°3 - turno: 0
empleado n°4 - turno: 0
empleado n°5 - turno: 3
comodín - turno: 2

Domingo - 28
empleado n°1 - turno: 3
empleado n°2 - turno: 1
empleado n°3 - turno: 0
empleado n°4 - turno: 0
empleado n°5 - turno: 2

Figura 22. Resultado de la inclusión de la restricción 8 al modelo con un mes de cuatro domingos.

Domingo - 2
empleado n°1 - turno: 0
empleado n°2 - turno: 2
empleado n°3 - turno: 3
empleado n°4 - turno: 0
empleado n°5 - turno: 1

Domingo - 9
empleado n°1 - turno: 0
empleado n°2 - turno: 0
empleado n°3 - turno: 1
empleado n°4 - turno: 3
empleado n°5 - turno: 2

Domingo - 16
empleado n°1 - turno: 2
empleado n°2 - turno: 0
empleado n°3 - turno: 0
empleado n°4 - turno: 3
empleado n°5 - turno: 1

Domingo - 23
empleado n°1 - turno: 3
empleado n°2 - turno: 2
empleado n°3 - turno: 1
empleado n°4 - turno: 0
empleado n°5 - turno: 0

Domingo - 30
empleado n°1 - turno: 3
empleado n°2 - turno: 2
empleado n°3 - turno: 0
empleado n°4 - turno: 1
empleado n°5 - turno: 0

Figura 23. Resultado de la inclusión de la restricción 8 al modelo con un mes de cinco domingos.

```
min_turno = []
max_turno = []

for turno in range(cant_turno):
    min_turno.append(turnos_totales[turno] // cantidad_empleado)
    max_turno.append(min_turno[turno] + (turnos_totales[turno] % cantidad_empleado != 0))
...
for turno in range(cant_turno):
    if jornada[turno]:
        modelo.Add(min_turno[turno] <= sum(jornada[turno]))
        modelo.Add(sum(jornada[turno]) <= max_turno[turno])
```

Figura 24. Implementación de la restricción 9.

En la Figura 25 se observa la cantidad de turnos que tienen asignados los empleados cada semana, de esta forma se puede ver cómo se distribuyen los turnos de los empleados en la planificación mensual. En la última fila se muestra la cantidad de veces totales que trabajó el empleado en un respectivo turno.

RESULTADOS

Argos

La Figura 26 presenta la arquitectura del sistema, en la cual se destaca el uso del framework Angular para el desarrollo del frontend. Para el backend, se utilizó el entorno de ejecución Node.js, mientras que para la administración de la base de datos se usó MySQL.

Un aspecto que destacar en esta arquitectura del sistema es el uso de un módulo llamado child_process de Node.js. Este módulo permite crear y controlar procesos secundarios de una aplicación, permitiendo la ejecución de programas o comandos externos

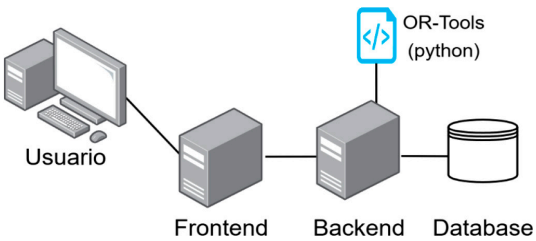


Figura 26. Arquitectura del sistema.

	empleado 1	empleado 2	empleado 3	empleado 4	empleado 5
semana 1	turno 1: 5 turno 2: 0 turno 3: 0	turno 1: 1 turno 2: 2 turno 3: 2	turno 1: 2 turno 2: 1 turno 3: 2	turno 1: 0 turno 2: 3 turno 3: 3	turno 1: 2 turno 2: 1 turno 3: 3
semana 2	turno 1: 2 turno 2: 2 turno 3: 2	turno 1: 1 turno 2: 3 turno 3: 2	turno 1: 4 turno 2: 0 turno 3: 2	turno 1: 2 turno 2: 1 turno 3: 2	turno 1: 3 turno 2: 2 turno 3: 0
semana 3	turno 1: 2 turno 2: 2 turno 3: 1	turno 1:2 turno 2:2 turno 3:2	turno 1:3 turno 2:1 turno 3:1	turno 1:4 turno 2:1 turno 3:0	turno 1:2 turno 2:2 turno 3:2
semana 4	turno 1:1 turno 2:1 turno 3:4	turno 1:5 turno 2:0 turno 3:0	turno 1:1 turno 2:4 turno 3:1	turno 1:4 turno 2:2 turno 3:0	turno 1:2 turno 2:1 turno 3:2
semana 5	turno 1:1 turno 2:2 turno 3:0	turno 1:2 turno 2:0 turno 3:1	turno 1:1 turno 2:1 turno 3:1	turno 1:1 turno 2:0 turno 3:2	turno 1:2 turno 2:1 turno 3:0
total:	turno 1:11 turno 2:7 turno 3:7	turno 1:11 turno 2:7 turno 3:7	turno 1:11 turno 2:7 turno 3:7	turno 1:11 turno 2:7 turno 3:7	turno 1:11 turno 2:7 turno 3:7

Figura 25. Resultado de la inclusión de la restricción 9.

al entorno de Node.js. Para este caso, se utilizó el método child_process.spawn, el cual permite crear e iniciar un nuevo proceso hijo. Este enfoque permitió ejecutar el script OR-Tools en un proceso secundario, evitando bloquear la ejecución principal de la aplicación, logrando una mejor gestión y control del flujo de la aplicación.

El sistema Argos está estructurado en dos grandes módulos: el módulo de gestión de la información de empleados y el módulo de la gestión de turnos. La Figura 27 muestra la información de los empleados registrados en el correspondiente módulo.

Para generar la planificación en el módulo de gestión de turnos, es necesario completar los siguientes campos: año, mes, comodín (ver restricción viii), empleados y horario de los turnos (Figura 28). En caso de que se necesite una cantidad específica de empleados para un turno y un día, se puede presionar el botón ITINERARIO y rellanar los campos correspondientes con el día, cantidad de empleados y turno. El botón AGREGAR permite ingresar nuevos turnos. Esta funcionalidad se agregó para el caso en el que dos aviones arriben simultáneamente y se necesitan empleados para este nuevo turno. Después de ingresar la información anterior, es posible generar la planificación de los turnos.

El frontend del sistema envía una solicitud HTTP al backend con los datos para generar la planificación de turno. El backend recibe la solicitud y verifica si la planificación de turno que el usuario quiere generar ya existe, es decir, se verifica si existe una planificación con el mismo año y mes. Si es el caso, se retorna un mensaje señalando el problema, en caso contrario, se procede a ejecutar la función que genera la planificación de turno.

Una vez creada la instancia planificación se realiza una petición a la base de datos para obtener la planificación del mes anterior. Estos datos son importantes a la hora de generar la planificación de turno, ya que hay restricciones que dependen de esta información para aplicarse correctamente. En particular:

– Cada empleado tiene un día libre a la semana (la semana no incluye los días domingos).
– Un empleado asignado al turno3 no puede ser asignado al día siguiente en el turno1.

Por último, la planificación generada por el solucionador de OR-Tools es mostrada por ARGOS (Figura 29) y almacenada en la base de datos.

En caso de que algún empleado se ausente a trabajar, el usuario puede hacer un registro de la ausencia, para que en la planificación se refleje la inasistencia (Figura 30). El empleado que se encarga de cubrir los turnos es el empleado comodín. El comodín es el mismo usuario por lo tanto, no hay necesidad de modificar la planificación como tal, pero sí dejar un registro del hecho.

Análisis de tiempo

En esta sección se analizan los tiempos de respuesta del solucionador CP-SAT para diferentes cantidades

Figura 27. Sistema Argos: Gestión de empleados.

Figura 28. Sistema Argos: Gestión de turnos.

Figura 29. Sistema Argos: Planificación de turno de febrero del 2023.

Figura 30. Sistema Argos: Registro de inasistencia de los empleados del 2023

de empleados (Tabla 1, Tabla 2 y Tabla 3), con el fin de evaluar su rendimiento. Para ello, se utilizó el método WallTime del solucionador, el cual devuelve el tiempo real transcurrido desde el inicio de la ejecución del solucionador hasta el momento de la llamada al método.

Al analizar los datos proporcionados sobre el tiempo de respuesta del solucionador en relación con la cantidad de empleados, se puede observar lo siguiente:

- Para una cantidad relativamente baja de empleados (de 5 a 30), el tiempo promedio de respuesta del solucionador se mantiene en valores tratables, aumentando de manera gradual a medida que aumenta la cantidad de empleados. Esto indica que el solucionador puede manejar eficientemente problemas con un número moderado de variables CSP.
- Para una cantidad de 40 empleados, los tiempos de respuesta muestra una mayor dispersión, abarcando desde 0.558632 hasta 8.158609 segundos. Esta variación nos indica que el solucionador empieza a encontrar dificultades llegados a este punto.
- Con 50 empleados, se observa un drástico incremento en el tiempo promedio de respuesta del solucionador, respecto al tiempo promedio de 40 empleados. En particular, para 60 empleados, donde la diferencia de tiempo promedio de respuesta es notoriamente alta. Esto sugiere que el aumento considerable de la cantidad

Tabla 1. Variación de los tiempos de respuesta en función del número de empleados del primer semestre.

Cantidad de Empleados	**Tiempo de respuesta (seg)**					
	Enero	**Febrero**	**Marzo**	**Abril**	**Mayo**	**Junio**
5	0,040045	0,041248	0,031142	0,035477	0,029621	0,031547
10	0,084042	0,066080	0,069787	0,073367	0,071111	0,070860
20	0,181559	0,185321	0,186579	0,223262	0,166307	0,166307
30	0,442444	0,348638	0,254314	0,367828	0,846598	0,369687
40	5,937749	0,558632	4,426362	4,497336	8,158609	1,275468
50	140,104017	1,471192	138,844006	90,832963	128,080234	4,991561
60	4727,210255	318,495243	302,909863	218,883723	272,872870	998,509414

Tabla 2. Variación de los tiempos de respuesta en función del número de empleados del segundo semestre.

Cantidad de Empleados	**Tiempo de respuesta (seg)**					
	Julio	**Agosto**	**Septiembre**	**Octubre**	**Noviembre**	**Diciembre**
5	0,043880	0,030712	0,030874	0,040483	0,029461	0,037116
10	0,084223	0,071751	0,076059	0,083769	0,091137	0,074574
20	0,433270	0,269891	0,194311	0,181919	0,173973	0,325963
30	0,835976	0,445489	0,381402	0,421586	0,294667	0,386746
40	4,504359	0,640986	3,626789	6,507982	3,661949	0,880090
50	143,439640	33,193986	138,917602	140,182025	4,108997	223,595776
60	204,466888	99,526295	160,679385	5091,62476	539,818110	345,462912

Tabla 3. Tiempo promedio de respuesta del primer y segundo semestre.

	Cantidad de Empleados						
	5	**10**	**20**	**30**	**40**	**50**	**60**
Tiempo promedio de respuesta (seg)	0,036800	0,072709	0,206405	0,499406	3,514591	91,060991	1478,407480

de variables CSP puede llevar a un incremento exponencial en la complejidad del problema y, por tanto, al tiempo de resolución.

– El tiempo de respuesta extremadamente alto para 60 empleados indica que el solucionador enfrenta dificultades para encontrar una solución óptima o satisfactoria en tiempos razonables, lo cual es un comportamiento clásico en problemas de planificación de horarios [3].

CONCLUSIONES

El problema de asignación de turnos de la empresa Ariaca representa un caso real que puede ser modelado y resuelto como un CSP. La solución implementada usando el proceso de incorporar restricciones de integridad a un modelo y luego resolverlas invocando al solucionador CP-SAT de OR-Tools muestra como una herramienta basada en la programación de restricciones permite encontrar planificaciones satisfactorias a esta problemática.

La implementación del sistema ha logrado una notable mejora en la generación de planificaciones de turnos, reduciendo el tiempo requerido de aproximadamente dos horas a tan solo 0.036800 segundos. Adicionalmente, la imparcialidad en la distribución de turnos, es una característica relevante que la empresa valora sustancialmente.

Finalmente, el sistema no solo puede beneficiar a Ariaca, sino que también a otras empresas del sector de suministro de combustible que enfrenten este mismo problema, siempre y cuando no se superen los treinta empleados. A partir de ese punto, el tiempo de respuesta del solucionador aumenta considerablemente. Para solucionar esta ineficiencia, es necesario explorar nuevas estrategias para abordar las restricciones, como aquellas descritas en Variables y dominio de Metodología.

AGRADECIMIENTOS

Se desea agradecer a la empresa ARIACA por su apertura y buena voluntad en el desarrollo de este proyecto.

REFERENCIAS

[1] J.J. Kanet, S.L. Ahire, and M.F. Gorman, "Constraint programming for scheduling," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, J. Leung Ed., Boca Raton, FL, USA: CRC Press, 2004, pp. 47-1 – 47-21.

[2] P. Baptiste, C.L. Pape, and W. Nuijten, *Constraint-Based Scheduling*, Boston-Dordrecht- London, USA: Kluwer Academic Publishers, 2001.

[3] D. de Werra, A.S. Asratian, and S. Durand, "Complexity of some special types of timetabling problems," *Journal of Scheduling*, vol. 5, pp. 171-183, 2002, doi: 10.1002/jos.97.

[4] H. Babaei, J. Karimpour, and A. Hadidi, "A survey of approaches for university course timetabling problem," *Computers & Industrial Engineering*, vol. 86, pp. 43-59, 2015, doi: 10.1016/j.cie.2014.11.010.

[5] Z. Zhao and X. Li, "Scheduling elective surgeries with sequence-dependent setup times to multiple operating rooms using constraint programming," *Operations Research for Health Care*, vol. 3, no. 3, pp. 160-167, 2014, doi: 10.1016/j.orhc.2014.05.003.

[6] S. Topaloglu and I. Ozkarahan, "A constraint programming-based solution approach for medical resident scheduling problems," *Computers & Operations Research*, vol. 38, no. 1, pp. 246-255, 2011, doi: 10.1016/j.cor.2010.04.018.

[7] L. Trilling, A. Guinet, and D.L. Magny, "Nurse scheduling using integer linear programming and constraint programming," *IFAC Proceedings Volumes*, vol. 39, no. 3, pp. 671-676, 2006, doi: 10.3182/20060517-3-FR-2903.00340.

[8] A.M. Ham, "Integrated scheduling of m-truck, m-drone, and m-depot constrained by time-window, drop-pickup, and m-visit using constraint programming," *Transportation Research Part C: Emerging Technologies*, vol. 91, pp. 1-14, 2018, doi: 10.1016/j.trc.2018.03.025.

[9] J.M. Novas and G. P. Henning, "Integrated scheduling of resource-constrained flexible manufacturing systems using constraint programming," *Expert Systems with Applications*, vol. 41, no. 5, pp. 2286-2299, 2014, doi: 10.1016/j.eswa.2013.09.026.

[10] J.M. Novas, "Production scheduling and iot streaming at flexible job-shops environments

using constraint programming," *Computers & Industrial Engineering*, vol. 136, pp. 252-264, 2019, doi: 10.1016/j.cie.2019.07.011.

[11] S. Fatemi-Anaraki, R. Tavakkoli-Moghaddam, M. Foumani, and B. Vahedi-Nouri, "Scheduling of multi-robot job shop systems in dynamic environments: mixed-integer linear programming and constraint programming approaches," *Omega*, vol. 115, p. 102770, 2023, doi: 10.1016/j.omega.2022.102770.

[12] M. Küçük and S.T. Yildiz, "Constraint programming-based solution approaches for three-dimensional loading capacitated vehicle routing problems," *Computers & Industrial Engineering*, vol. 171, p. 108505, 2022, doi: 10.1016/j.cie.2022.108505.

[13] E.C. Freuder and A.K. Mackworth, "Constraint satisfaction: an emerging paradigm," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh Eds., 1st ed., Amsterdam, Netherlands, NL: Elsevier, 2006, pp. 11-25.

[14] S.C. Brailsford, C.N. Potts, and B.M. Smith, "Constraint satisfaction problems: algorithms and applications," *European Journal of Operational Research*, vol. 119, no. 3, pp. 557-581, 1999, doi: 10.1016/S0377-2217(98)00364-6.

[15] R. Barták, M.A. Salido, and F. Rossi, "Constraint satisfaction techniques in planning and scheduling," *Journal of Intelligent Manufacturing*, vol. 21, no. 1, pp. 5-15, 2010, doi: 10.1007/s10845-008-0203-4.

[16] F. Barber y M.A. Salido, "Problemas de satisfacción de restricciones (CSP)", en *Inteligencia Artificial: Técnicas, Métodos y Aplicaciones*, R.L. Marín y J.T. Palma Eds., 1era ed., España: McGraw-Hill Interamericana, 2008, pp. 385-432.

[17] L. Perron and V. Furnon, "OR-Tools," developers.google.com. [Online]. Available: https://developers.google.com/optimization (accessed Jun. 30, 2023).

[18] S. Kruk, *Practical Python AI Projects*, Rochester, Michigan, USA: Apress, 2018.

[19] G. da Col and E. Teppan, "Industrial-size job shop scheduling with constraint programming," *Operations Research Perspective*, vol. 9, pp. 100249, 2022, doi: 10.1016/j.orp.2022.100249.

[20] N. Rozario and D. Rozario, "Can machine learning optimize the efficiency of the operating room in the era of COVID-19?," *Canadian Journal of Surgery*, vol. 63, pp. E527-E529, 2020, doi: 10.1503/cjs.016520.

[21] A. Ibrahim, R. Abdulaziz, and J. Ishaya, "Capacitated vehicle routing problem," *International Journal of Research - Granthaalayah*, vol. 7, pp. 310-327, 2019, doi: 10.5281/zenodo.2636820.

[22] T. Li, E. Zekeriya, and R. Lagendijk, "Privacy-preserving bin-packing with differential privacy," *IEEE Open Journal of Signal Processing*, vol. 7, no. 3, pp. 94-106, 2022, doi: 10.1109/OJSP.2022.3153231.

[23] J. Koehler *et al.*, "Cable tree wiring - benchmarking solvers on a real-world scheduling problem with a variety of precedence constraints," *Constraints*, vol. 26, pp. 56-106, 2021, doi: 10.1007/s10601-021-09321-w.

[24] H. Kjellerstrand, "Constraint programming with implementations in Python, Java, and C#," hakank.org. [Online]. Available: http://www.hakank.org/google_or_tools/. (accessed Jun. 30, 2023).

[25] A. Falkner, A. Haselböck, G. Krames, G. Schenner, H. Schreiner, and R. Taupe, "Solver requirements for interactive configuration," *Journal of Universal Computer Science*, vol. 26, no. 3, pp. 343-373, 2020, doi: 10.3897/jucs.2020.019.

[26] MiniZinc, "The MiniZinc challenge," MiniZinc.org, [Online]. Available: https://www.minizinc.org/challenge.html (accessed Jun. 30, 2023).

[27] P.J. Stuckey, R. Becket, and J. Fischer, "Philosophy of the MiniZinc challenge," *Constraints*, vol. 15, no. 3, pp. 307-316, 2010, doi: 10.1007/s10601-010-9093-0.

[28] G. Echeverría, "Sistema de gestión de turnos", github.com. [En línea]. Disponible: https://github.com/thecarrot911 (accedido: 30 de junio de 2023)

[29] G. Echeverría, "Sistema de gestión de turnos "pseudocódigo", github.com. [En línea]. Disponible: https://github.com/thecarrot911/Sistema-gestion-turno-backend/blob/master/pseudocodigo.txt (accedido: 30 de junio de 2023)