Apéndice A

Programa de asignación de turnos `ShiftSolver`

Este apéndice incluye todo el programa que se ha escrito para resolver los dos ejemplos. La forma de usarlo es muy sencilla, sin embargo, antes de poder ejecutarlo hay que instalar una serie de librerías de Python sin las cuales no funcionará.

Configuración previa

Estos son los paquetes fundamentales que habrá que importar e instalar (ya sea en un entorno virtual o en la propia máquina) y sin los cuáles no funcionará el programa:

- **Numpy.** El paquete Numpy es necesario para todos los cálculos porque se usan arrays en lugar de listas para reducir el tiempo de ejecución.
- **csv.** El proceso de escritura de datos en archivos *csv* se hace usando la clase `DictWriter` de csv.
- **datetime.** Se usa en `Empolyee` para obtener las cotas de los intervalos de disponibilidad de los empleados.

Todos esos son necesarios para el funcionamiento básico del programa y para poder usar la función `solve()`. Los paquetes que se muestran a continuación son solo necesarios para ejecutar `pulp_solve()`, así que no son esenciales.

- **pulp.** PuLP es la librería en la que se basa `pulp_solve()` y es crucial para que funcione. En las referencias de este trabajo puede encontrarse el enlace a su página web donde se explica detalladamente los pasos a seguir para instalarlo. PuLP puede ser instalado mediante PyPi. PuLP incluye su propio solver, PULP_CBC_CMD, que no necesita ser instalado a parte.
- **glpk.** Esta es una librería *open source* que dispone de programas de resolución de problemas de optimización que implementa el algoritmo Simplex. Para usarlo, simplemente instalarlo a través de PyPi (o en caso de usar Conda, está disponible en Anaconda.org, conda-forge).
- **gurobi.** Este es verdaderamente difícil de conseguir, por lo que no recomiendo su instalación. No es gratuito y solo puede usarse bajo licencia. Si se consigue obtener

una licencia, se deben seguir los pasos de su página web a la vez que se instala gurobi a través de PyPi.

Adaptación del código

Una vez se ha completado la instalación previa, si solo se pretende usar `solve()` y no se añade ningún problema nuevo, solo hay que completar los pasos 2 y 3. Si se quisiera resolver un problema nuevo, completar también el paso 4 y solo en el caso de que se quiera usar Gurobi, completar el paso 1. Estos cambios hacen referencia a la función `main()`.

1. En caso de haber conseguido instalar Gurobi, cambiar la dirección de "GUROBI_CMD" por la de su instalación.
2. En caso de no estar ya creado, crear la carpeta "examples" en el mismo directorio donde se encuentren `Employee`, `Preference` y `ShiftSolver` y meter dentro los documentos de archivos que se quieran resolver.

   En este trabajo en particular se han resuelto dos, así que para mi caso particular serían "cafeteria.txt" y "clinica.csv".
3. Dentro de la carpeta "examples", si no está presente, crear la carpeta "results".
4. Si los ejemplos que se van a probar son los dos usados en este trabajo, entonces no es necesario hacer más cambios. Si se quisiera resolver un problema diferente, añadir a la estructura `if` de la función `main()` otra entrada de `elif`, siguiendo el mismo formato de las dos anteriores.

A.1. Código de Employee

```
from datetime import datetime
import numpy as np

DEFAULT_DAYS = 5  # 5 porque en principio se considera |D| = 5 (ciclo de 5 días)
DEFAULT_R = 3


class Employee:
    """ Esta clase guarda los datos de cada empleado y crea un objeto para cada uno de ellos que contiene estos datos.
        :arg line línea del archivo datos.txt
        :returns Objeto que contiene los datos de cada empleado.
        :raises Error si no se han introducido todos los datos"""

    HOUR_MAP = {7: 1, 8: 2, '9': 3, '10': 4, '11': 5, '12': 6, '13': 7, '14': 8, '15': 9, '16': 10, '17': 11,
                '18': 12, '19': 13, '20': 14, '21': 15}

    def __init__(self, tc, line, n_days=DEFAULT_DAYS, n_rolls=DEFAULT_R):
        if type(line) is not str:
            raise TypeError("El argumento de entrada debe ser string")
        if tc == 't':
            line_list = line.rstrip().split("\t")
        elif tc == 'cex':
            line_list = line.rstrip().split(";")
        elif tc == 'c':
            line_list = line.rstrip().split(",")
        else:
            raise NotImplementedError("Archivo no soportado.")

        self.MAXHR = int(line_list[1])
        if int(line_list[2]) != int(line_list[3]) * self.MAXHR:
            raise ValueError("El máximo de horas a la semana debe ser igual al máximo de horas al día multiplicado "
                             f"por la cantidad de días que trabaja, {line_list[3]} en este caso.")
        self.HOURS = int(line_list[2])
        self.DAYS = int(line_list[3])
        self.S = np.zeros(n_rolls)
        for i in range(n_rolls):
            if int(line_list[4 + i]) == 0:
                self.S[i] = np.inf      # Para evitar problemas con 0 * inf
            else:
                self.S[i] = int(line_list[4 + i])
        self.AVL = np.zeros((n_days, 2))
        avl_list = line_list[4 + n_rolls].split()
        cont = 0
        for block in avl_list:
            if block != ".":
                block_limits = block.split("-")
                h_low, h_high = datetime.strptime(block_limits[0], "%H:%M"), datetime.strptime(block_limits[1], "%H:%M")
                if h_low.hour > h_high.hour:
```

```
                    raise Exception("La hora inicial debe ser menor que la final")
                if h_low.hour < 7 or h_high.hour > 22:
                    raise Exception("La hora de inicio y final no pueden sobrepasar los límites")
                self.AVL[cont, 0] = h_low.hour - 6
                self.AVL[cont, 1] = h_high.hour - 7
                cont += 1
            else:
                cont += 1
```

A.2. Código de Preference

```
import numpy as np

DEFAULT_DAYS = 5
DEFAULT_E = 4
DEFAULT_R = 3       # Se consideran de ejemplo: r0 = barman, r1 = camarero, r2 = cocinero


def len_b(block):
    return block[1] - block[0] + 1


class Preference:
    """
    Esta clase genera el tensor de cuatro dimensiones que contiene los valores de preferencia de un empleado para cada
    día d, bloque b y rol r. En la dimensión de b estarán los bloques de trabajo y estarán ordenados de menor a mayor
    según el largo del bloque.
    :arg employees Lista formada de objetos de la clase Employee. Contiene a todos los empleados.
    :arg blocks Matriz de ?x2 que contiene todos los bloques disponibles en un día.
    :arg n_employees # de empleados.
    :arg n_days # de días del ciclo.
    :arg n_rols # de roles.
    """

    def __init__(self, employees, blocks, n_employees=DEFAULT_E, n_days=DEFAULT_DAYS, n_rols=DEFAULT_R):
        self.P = np.zeros((n_days, blocks.shape[0], n_rols, n_employees))
        for i in range(len(employees)):
            emp = employees[i]
            for j in range(len(emp.AVL)):
                if np.array_equal(emp.AVL[j], [0, 0]):
                    self.P[j, :, :, i] = -np.inf
                else:
                    for k in range(len(blocks)):
                        if len_b(blocks[k]) > emp.MAXHR:   # Para chequear 3. y 4. (MINHR = MAXHR)
                            self.P[j, k, :, i] = -np.inf
                        else:
                            if blocks[k, 0] < emp.AVL[j, 0] or blocks[k, 1] > emp.AVL[j, 1]:
```

```
                                self.P[j, k, :, i] = -np.inf
                            else:
                                self.P[j, k, :, i] = 10 * (len_b(blocks[
    k]) / emp.MAXHR)
```

A.3. Código de ShiftSolver

```
import timeit
import numpy as np
import csv
from pulp import gurobipy
import Preference
from Employee_v1_2 import Employee
from Preference import Preference as pr
import pulp as pl

SHIFT_LENGTHS = np.array([6, 7, 8])
SHIFT_HOURS = np.array(list(range(1, 16)))
MINNUM = np.array([1, 2, 1])  # Para el rol r0 (barman), r1 (camarero) y
    r2 (cocinero)
MAXNUM = np.array([2, 3, 1])
DEFAULT_DAYS = 5
DEFAULT_E = 4
DEFAULT_R = 3
DEFAULT_ROLLS = {0: "Barra", 1: "Camarero", 2: "Cocina"}


# Para chequear 2.
def e_free_for_day_d(e0, d0, shift_dict):
    """
    This function checks if a given employee is free for a given day, as
    each employee can only work in one shift per
    day.
    :param e0: index of the employee to check for.
    :type e0: numpy.ndarray
    :param d0: index of the day to check for.
    :type d0: numpy.ndarray
    :param shift_dict: Dictionary containing all the assigned shifts for
    each employee.
    :type shift_dict: dict
    :return: True if e0 is has no assigned shifts day d0, False
   otherwise.
    :rtype: bool
    """
    e_shifts = shift_dict[f"enp{e0}"]
    for d, b, r in e_shifts:
        if d == d0:
            return False
    else:
        return True


# Para chequear 6.
def e_days_filled(e0, employee, shift_dict):
    """
    This function checks if a given employee e0 has all of its days
   already occupied.
    :param e0: index of the employee to check for.
```

```
    :type e0: numpy.ndarray
    :param employee: Employee type object of index e0 in the list of employees.
    :type employee: Employee
    :param shift_dict: Dictionary containing all the assigned shifts for each employee.
    :type shift_dict: dict
    :return: True if e0 has a shift assigned to each day.
    :rtype: bool
    """
    e_shifts = shift_dict[f"enp{e0}"]
    if len(e_shifts) >= employee.DAYS:
        return True
    return False


# Para chequear 5.
def e_hours_filled(e0, employee, shift_dict):
    """
    This function checks if a given employee e0 has all of its hours already occupied.
    :param e0: index of the employee to check for.
    :type e0: numpy.ndarray
    :param employee: Employee type object of index e0 in the list of employees.
    :type employee: Employee
    :param shift_dict: Dictionary containing all the assigned shifts for each employee.
    :type shift_dict: dict
    :return: True if all of e0 shifts sum equal to its HOURS property.
    :rtype: bool
    """
    e_shifts = shift_dict[f"enp{e0}"]
    h_total = 0
    for d, b, r in e_shifts:
        h_total += Preference.len_b(b)
    if h_total >= employee.HOURS:
        return True
    return False


# Para chequear 9.
def occupation_less_than_min_for_day_d(d0, r0, shift_dict, minnum):
    """
    This function checks if the occupation for a certain rol r0 in a given day d0 has reached it's minimum or not.
    :param d0: index of day to check for.
    :type d0: numpy.ndarray
    :param r0: index of rol to check for.
    :type r0: numpy.ndarray
    :param shift_dict: Dictionary containing all the assigned shifts for each employee.
    :type shift_dict: dict
    :param minnum: Array containing the minimum number of employees needed in each rol.
    :return: True if minimum is filled, False otherwise.
    :rtype: bool
    """
```

```
    min_for_this_rol = minnum[r0]
    counter = 0
    all_shifts = []
    for val in shift_dict.values():
        all_shifts.extend(val)
    for d, b, r in all_shifts:
        if d == d0:
            if r == r0:
                counter += 1
                if counter >= min_for_this_rol:
                    return False
    return True


# Para chequear 8.
def occupation_full_for_day_d(d0, r0, shift_dict, maxnum):
    """
    This function checks if the occupation for a certain rol r0 in a
    given day d0 is full or not.
    :param d0: index of day to check for.
    :type d0: numpy.ndarray
    :param r0: index of rol to check for.
    :type r0: numpy.ndarray
    :param shift_dict: Dictionary containing all the assigned shifts for
    each employee.
    :type shift_dict: dict
    :param maxnum: Array containing the maximum number of employees
    needed in each rol.
    :return: True if maximum is filled, False otherwise.
    :rtype: bool
    """
    max_for_this_rol = maxnum[r0]
    counter = 0
    all_shifts = []
    for val in shift_dict.values():
        all_shifts.extend(val)
    for d, b, r in all_shifts:
        if d == d0:
            if r == r0:
                counter += 1
                if counter >= max_for_this_rol:
                    return True
    return False


def translate_block(block):
    """
    This function takes an array with the start and end times of a shift
    as integers and rewrites the block as an hour
    interval.
    :param block: Array with start and end of shift as integers.
    :type block: numpy.ndarray
    :return: Interval of start and end hours as string.
    :rtype: str
    """
    return f"{block[0] + 6}:00-{block[1] + 7}:00"


```

```
151 def filter_match(f_blocks, blocks):
152     """
153     This function looks for the matches between the start or end block
    list (f_blocks) and the list of all the blocks
154     (blocks) and returns an array of indexes corresponding to the
    position of the start (or end) blocks in the list of
155     all blocks.
156     :param f_blocks: Filtered array containing shift blocks (arrays).
    The available filters will either leave all the
157     blocks containing the first shift or the last shift.
158     :type f_blocks: list
159     :param blocks: Set of all the available shifts.
160     :type blocks: list
161     :return:Array of the indexes of the matches.
162     :rtype: numpy.ndarray
163     """
164     matches = np.zeros((len(f_blocks)), dtype=int)
165     cont = 0
166     for i in range(len(blocks)):
167         for b in f_blocks:
168             if b == blocks[i]:
169                 matches[cont] = i
170                 cont += 1
171     return matches
172 
173 
174 def load_to_csv(filename, shift_dict, rolls=None, n_days=DEFAULT_DAYS,
    n_rolls=DEFAULT_R):
175     """
176     This function loads the obtained solution into a csv file named '
    filename'. If no rolls and days are specified it
177     defaults to some predetermined values.
178     :param filename: Name of the csv to save the data to.
179     :type filename: str
180     :param shift_dict: This is a dictionary where each item contains an
    employee as the key and a list of its assigned
181     shifts ordered from the first to the last day as value.
182     :type shift_dict: dict
183     :param rolls: This is a dictionary containing the different rolls
    that can represent different jobs, rooms, sections
184     , tasks, etc. available to a solution. Keys are indexes and values
    are strings.
185     :type rolls: dict
186     :param n_days: # of days.
187     :type n_days: int
188     :param n_rolls: # of rolls.
189     :type n_rolls: int
190     """
191     if rolls is None:
192         rolls = DEFAULT_ROLLS
193     if len(rolls) != n_rolls:
194         raise Exception("Cantidad de roles incompatible.")
195     with open(filename + ".csv", mode="w") as csvfile:
196         print("SEP=,", file=csvfile)  # Para que excel ponga el texto en
    columnas automáticamente
197         fieldnames = ['employee']
198         fieldnames.extend([f'day{i}' for i in range(n_days)])
199         writer = csv.DictWriter(csvfile, fieldnames=fieldnames, dialect=
```

```
    'excel')
200         writer.writeheader()
201         for emp in shift_dict:
202             writer_dict = dict.fromkeys(fieldnames, ' ')
203             writer_dict[fieldnames[0]] = emp
204             for d, b, r in shift_dict[emp]:
205                 block = translate_block(b)
206                 writer_dict[f'day{d}'] = f'{block} {rolls[r]}'
207             writer.writerow(writer_dict)
208
209
210 class ShiftSolver:
211     """
212     Segunda versión del código que resuelve el PLE. La clase Employee se
        usa para extraer los datos de cada trabajador,
213     que se guardan en una lista de objetos de esa misma clase. La clase
        Preference utiliza los datos de los trabajadores
214     y la lista de empleados para generar el tensor de preferencia. Con
        la función 'solve' se resuelve el PLE mediante un
215     algoritmo voraz. Con la función 'pulp_solve' se resuelve el PLE
        mediante un solver de la librería PuLP.
216     :arg file str de dirección a archivo que contiene los datos de los
        trabajadores.
217     """
218
219     def __init__(self, file, tc, emp_amount=DEFAULT_E, n_days=
        DEFAULT_DAYS, n_rolls=DEFAULT_R, shift_lengths=None,
220                  shift_start_end=None, minnum=MINNUM, maxnum=MAXNUM):
221         # Primero, cargamos los datos de cada empleado de la hoja "datos
        .txt o .csv"
222         if shift_lengths is None:
223             shift_lengths = SHIFT_LENGTHS
224         if shift_start_end is None:
225             self.shift_hours = SHIFT_HOURS
226         else:
227             try:
228                 self.shift_hours = np.array(list(range(shift_start_end
        [0], shift_start_end[1])))
229             except TypeError:
230                 raise TypeError("shift_start_end debe ser un iterable.")
231
232         with open(file, mode='r') as data_file:
233             file_lines = data_file.readlines()
234         self.minnum = minnum
235         self.maxnum = maxnum
236         self.employees = []
237         for line in file_lines[1:]:
238             self.employees.append(Employee(tc, line, n_days=n_days,
        n_rolls=n_rolls))
239         self.blocks = []
240         for i in range(len(shift_lengths)):
241             for j in range(1, len(self.shift_hours) + 1):
242                 if j + shift_lengths[i] - 1 >= len(self.shift_hours) +
        1:
243                     break
244                 self.blocks.append([j, j + shift_lengths[i] - 1])
245         np_blocks = np.array(self.blocks)
246         self.max_shifts = sum([e.DAYS for e in self.employees])
```

```
247         self.pr_tensor = pr(self.employees, np_blocks, n_employees=
    emp_amount, n_days=n_days, n_rols=n_rolls).P
248         self.sr_matrix = np.empty((n_rolls, emp_amount))
249         for r in range(n_rolls):
250             for e in range(emp_amount):
251                 self.sr_matrix[r, e] = self.employees[e].S[r]
252 
253     def solve(self):
254         """
255         Esta función resuelve el PLE. Para ello usa f(S,P) = S * P como
    función objetivo y lo resuelve mediante algorit-
256         mo voraz. Es decir, en cada iteración busca el valor máximo de
    la matriz resultante de S * P que cumpla con las
257         condiciones y lo añade a la matriz de la solución.
258         :return: assigned_shifts Diccionario que contiene los turnos
    asignados a cada trabajador. Clave: empleado,
259         valor: lista con formato [día, bloque, rol].
260         """
261 
262         T = self.pr_tensor * self.sr_matrix
263         all_shifts_assigned = False
264         assigned = 0
265         f_obj = 0
266         assigned_shifts = dict.fromkeys([f"enp{i}" for i in range(len(
    self.employees))], None)
267         for key in assigned_shifts:
268             assigned_shifts[key] = []
269         while np.amax(T) > -np.inf and not all_shifts_assigned:
270             # Obtener índices del valor máximo de la matriz, (d, b, r, e
    )
271             indexes = np.unravel_index(np.argmax(T, axis=None), T.shape)
272             d, b, r, e = indexes[0], indexes[1], indexes[2], indexes[3]
273             if np.amax(T) == np.inf:
274                 T[:, :, r, e] = -np.inf
275                 continue
276             if e_free_for_day_d(e, d, assigned_shifts):
277                 if not e_days_filled(e, self.employees[e],
    assigned_shifts):
278                     if not e_hours_filled(e, self.employees[e],
    assigned_shifts):
279                         if occupation_less_than_min_for_day_d(d, r,
    assigned_shifts, self.minnum):
280                             assigned_shifts[f"enp{e}"].append([d, self.
    blocks[b], r])
281                             assigned += 1
282                             f_obj += T[d, b, r, e]
283                             T[d, :, :, e] = -np.inf
284                         else:
285                             if not occupation_full_for_day_d(d, r,
    assigned_shifts, self.maxnum):
286                                 assigned_shifts[f"enp{e}"].append([d,
    self.blocks[b], r])
287                                 assigned += 1
288                                 f_obj += T[d, b, r, e]
289                                 T[d, :, :, e] = -np.inf
290                             else:
291                                 T[d, :, r, :] = -np.inf
292                     else:
```

```
293                         T[:, :, :, e] = -np.inf
294                 else:
295                     T[:, :, :, e] = -np.inf
296             else:
297                 T[d, :, :, e] = -np.inf
298             if assigned >= self.max_shifts:
299                 all_shifts_assigned = True
300         return assigned_shifts, f_obj
301 
302     def pulp_solve(self, solver=pl.PULP_CBC_CMD()):
303         """
304         Esta función resuelve el PLE usando uno de los solvers de la librería PuLP.
305         :param solver: Objeto de la librería PuLP que contiene los algoritmos necesarios para resolver el PLE. Su valor
306         por defecto es PULP_CBC_CMD.
307         :return:
308         """
309         T_coef = self.pr_tensor * self.sr_matrix
310         T_coef_flat = T_coef.flatten()
311         T_var = []
312         # Primero, definir todas las variables
313         for i in range(T_coef.shape[0]):  # d in DAYS
314             T_var.append([])
315             for j in range(T_coef.shape[1]):  # b in BLOCKS
316                 T_var[i].append([])
317                 for k in range(T_coef.shape[2]):  # r in ROLLS
318                     T_var[i][j].append([])
319                     for l in range(T_coef.shape[3]):  # e in EMPLOYEES
320                         T_var[i][j][k].append(pl.LpVariable(f'x{i}_{j}_{k}_{l}', 0, 1, pl.LpInteger))
321         T_var_arr = np.array(T_var)
322         T_var_flat = T_var_arr.flatten()
323 
324         # Segundo, definir problema y función objetivo
325         prob = pl.LpProblem("Shift_Assigning_Problem", pl.LpMaximize)
326         T_sum = []
327         for i in range(len(T_coef_flat)):
328             if T_coef_flat[i] == np.inf or T_coef_flat[i] == -np.inf:
329                 T_coef_flat[i] = -1E5
330                 T_sum.append(T_coef_flat[i] * T_var_flat[i])
331             else:
332                 T_sum.append(T_coef_flat[i] * T_var_flat[i])
333         prob += pl.lpSum(T_sum), "Objective_function"
334 
335         # Tercero, añadir restricciones
336         # 1 y 6
337         def filter_b_start(block):
338             if block[0] == 1:
339                 return True
340 
341         def filter_b_end(block):
342             if block[1] == self.shift_hours[-1]:
343                 return True
344 
345         filtered_start_blocks = list(filter(filter_b_start, self.blocks.copy()))
346         filtered_end_blocks = list(filter(filter_b_end, self.blocks.copy
```

```
    ()))
347         matching_start_indexes = filter_match(filtered_start_blocks,
    self.blocks)  # ndarray
348         matching_end_indexes = filter_match(filtered_end_blocks, self.
    blocks)  # ndarray
349         for i in range(T_coef.shape[0]):
350             for l in range(T_coef.shape[3]):
351                 # 1
352                 prob += pl.lpSum(
353                     [T_var[i][j][k][l] for j in range(T_coef.shape[1])
    for k in range(T_coef.shape[2])]) <= 1
354                 # 6
355                 prob += pl.lpSum(
356                     [T_var[i][js][k][l] + T_var[i][je][k][l]
357                      for js in matching_start_indexes for je in
    matching_end_indexes for k in range(T_coef.shape[2])]
358                 ) <= 1
359         # 2 y 3 se cumplen en la creación P
360         # 4 y 5
361         for l in range(T_coef.shape[3]):
362             emp = self.employees[l]
363             # 4
364             prob += pl.lpSum(
365                 [Preference.len_b(self.blocks[j]) * T_var[i][j][k][l]
366                  for i in range(T_coef.shape[0]) for j in range(T_coef.
    shape[1]) for k in range(T_coef.shape[2])]
367             ) <= emp.HOURS
368             # 5
369             prob += pl.lpSum(
370                 [T_var[i][j][k][l]
371                  for i in range(T_coef.shape[0]) for j in range(T_coef.
    shape[1]) for k in range(T_coef.shape[2])]
372             ) <= emp.DAYS
373         # 7 y 8
374         for i in range(T_coef.shape[0]):
375             for k in range(T_coef.shape[2]):
376                 prob += pl.lpSum(
377                     [T_var[i][j][k][l] for j in range(T_coef.shape[1])
    for l in range(T_coef.shape[3])]
378                 ) >= self.minnum[k]
379                 prob += pl.lpSum(
380                     [T_var[i][j][k][l] for j in range(T_coef.shape[1])
    for l in range(T_coef.shape[3])]
381                 ) <= self.maxnum[k]
382         # Siguiente paso, resolver el problema
383         prob.solve(solver=solver)
384         # Reconstruimos T_var partiendo de su versión aplanada con los 0
    s y 1s colocados
385         T_var_flat = np.array(list(map(pl.value, T_var_flat)))
386         T_var = T_var_flat.reshape((T_coef.shape[0], T_coef.shape[1],
    T_coef.shape[2], T_coef.shape[3]))
387         return prob, T_var, pl.LpStatus[prob.status]
388
389     def matrix2shift_dict(self, var_arr):
390         assigned_shifts = dict.fromkeys([f'enp{i}' for i in range(len(
    self.employees))], None)
391         for key in assigned_shifts:
392             assigned_shifts[key] = []
```

```
393         assigned_positions = np.argwhere(var_arr)
394         for pos in assigned_positions:
395             assigned_shifts[f'enp{pos[3]}'].append([pos[0], self.blocks[
    pos[1]], pos[2]])
396         return assigned_shifts
397 
398 
399 def my_timeit(stmt='pass', number=100, repeat=10, setup='pass', scope=
    None):
400     lapse = timeit.repeat(stmt=stmt, number=number, repeat=repeat, setup
    =setup,
401                           globals=scope)
402     lapse = [x / number for x in lapse]
403     mean = sum(lapse) / repeat
404     std = (sum((x - mean) ** 2 for x in lapse) / (repeat - 1)) ** 0.5 if
     repeat > 1 else 0.0
405     if mean >= 1.0:
406         unit = 's'
407     elif mean >= 1E-3:
408         unit = 'ms'
409         mean *= 1E3
410         std *= 1E3
411     elif mean >= 1E-6:
412         unit = '\xb5s'  # 'mu'-ren unicodea
413         mean *= 1E6
414         std *= 1E6
415     else:
416         unit = 'ns'
417         mean *= 1E9
418         std *= 1E9
419     return f'{mean:.2f} {unit} \u00B1 {std:.2f} {unit}'  # +- zeinuaren
    unicode
420 
421 
422 def main():
423     paths_to_solvers = {
424         'CPLEX_PY': "/Users/aitor/.conda/envs/TFG_2023/lib/python3.10/
    site-packages/pulp/apis/cplex_api.py",
425         'GUROBI_CMD': "/Library/gurobi1001/macos_universal2/bin/
    gurobi_cl"
426     }
427 
428     solver1 = pl.COIN_CMD(path="/opt/miniconda3/pkgs/coin-or-cbc-2.10.8-
    h1ce7d08_0/bin/cbc")  # no va
429     solver2 = pl.CPLEX_CMD(path=paths_to_solvers['CPLEX_PY'])  # no va
430     solver3 = pl.GUROBI_CMD(path=paths_to_solvers['GUROBI_CMD'])
431     solver4 = pl.GUROBI()
432     solver5 = pl.GLPK_CMD()
433 
434     solvers = {'PULP_CBC_CMD': pl.PULP_CBC_CMD(),
435                'GLPK_CMD': solver5,
436                'GUROBI': solver4,
437                # 'GUROBI_CMD': solver3
438                }
439 
440     problem = input("Introduce nombre del datafile [cafeteria/clinica]:
    ")
441     f_type = input("Tipo de archivo [t/c/cex]: ")
```

```
    extension = ""
    if f_type == 't':
        extension = '.txt'
    elif f_type == 'c' or f_type == 'cex':
        extension = '.csv'

    if problem.lower() == 'cafeteria':

        # PROBLEMA A RESOLVER:
        examples_path = "../examples/"
        results_path = "../examples/results/"
        datafile_name = problem
        datafile = examples_path + datafile_name + extension
        my_solver = ShiftSolver(datafile, f_type, emp_amount=6)

        # 1) Usando mi solver
        shifts, f_obj = my_solver.solve()

        # 2) Usando PuLP
        pulp_shifts = {}
        f_objs = {}
        for key, val in solvers.items():
            try:
                lp_problem, lp_result, status = my_solver.pulp_solve(val
)
                pulp_shifts[key] = my_solver.matrix2shift_dict(lp_result
)
                f_objs[key] = pl.value(lp_problem.objective)
            except gurobipy.GurobiError:  # Para evitar problemas con la
 licencia de Guroby
                print("\nNo se ha podido completar el cálculo con Guroby
.\n")
                continue

        # 3) Cargamos los resultados
        load_to_csv(results_path + datafile_name + "_my", shifts)
        for key, shift in pulp_shifts.items():
            load_to_csv(results_path + datafile_name + key, shift)

        # 4) Resultados de las funciones objetivo
        print("Resultados de funciones objetivo:")
        print("Algoritmo voraz:\t\t", f_obj)
        for f_name, f in f_objs.items():
            print(f_name + ":\t\t", f)

    elif problem.lower() == 'clinica':

        # PROBLEMA A RESOLVER:
        examples_path = "../examples/"
        results_path = "../examples/results/"
        datafile_name = problem
        datafile = examples_path + datafile_name + extension
        my_solver = ShiftSolver(datafile, f_type, emp_amount=20, n_days
=7, n_rolls=5, shift_lengths=np.array([5, 6,
                                               7, 8]),
                                shift_start_end=[1, 16], minnum=np.array
([3, 3, 1, 1, 1]), maxnum=np.array([5, 6, 2, 2,
```

```
                                        2]))

        # 1) Usando mi solver
        shifts, f_obj = my_solver.solve()

        # 2) Usando PuLP
        pulp_shifts = {}
        f_objs = {}
        for key, val in solvers.items():
            try:
                lp_problem, lp_result, status = my_solver.pulp_solve(val)
                pulp_shifts[key] = my_solver.matrix2shift_dict(lp_result)
                f_objs[key] = pl.value(lp_problem.objective)
            except gurobipy.GurobiError:  # Para evitar problemas con la licencia de Guroby
                print("\nNo se ha podido completar el cálculo con Guroby.\n")
                continue

        # 3) Cargamos los resultados
        roles = {0: 'Enfermera', 1: 'Medico', 2: 'Auxiliar', 3: 'Secretario', 4: 'Celador'}
        load_to_csv(results_path + datafile_name + "_my", shifts, rolls=roles, n_days=7, n_rolls=5)
        for key, shift in pulp_shifts.items():
            load_to_csv(results_path + datafile_name + key, shift, rolls=roles, n_days=7, n_rolls=5)

        # 4) Resultados de las funciones objetivo
        print("Resultados de funciones objetivo:")
        print("ALgoritmo voraz:\t\t", f_obj)
        for f_name, f in f_objs.items():
            print(f_name + ":\t\t", f)


if __name__ == "__main__":
    main()
```