Universidad de Valladolid

Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en estadística

Estudio del problema de diseño de turnos y cómo resolverlo

Autor:

Diego Farto González

Tutor/es:

Ricardo Josa Fombellida

Jesús Sáez Aguado

Quería mostrar mi más sincero agradecimiento a Jesús Sáez, por su gran apoyo y ayuda en la culminación de este trabajo.

También quiero agradecer a mis compañeros y amigos, por sus ánimos en los momentos difíciles.

Por último, también quiero agradecer a mi pareja y familia, porque han sido una fuente de inspiración y esperanza.

RESUMEN

En este trabajo vamos a describir en qué consiste el problema de diseño de turnos y comentaremos tres técnicas para resolver este problema: la programación lineal entera multiobjetivo, el método de restricciones ε y los métodos heurísticos. Estos últimos, no los implementaremos ya que su estudio es muy extenso y requeriría de un análisis más profundo.

Todos estos métodos, se utilizan para tratar de obtener la solución óptima, o al menos, una aproximación a la misma. Para ello, hemos considerado un conjunto de restricciones de carácter general, que habitualmente se incluyen en este tipo de problemas y que son ampliamente tratadas en la literatura (Dantzig (3) y Kocabas (6)).

Finalmente, compararemos los resultados obtenidos con distintos *solvers*: Xpress Optimizer, Gurobi y Cplex utilizando métodos de programación lineal entera multiobjetivo.

ABSTRACT

In this work, we are going to describe what the shift design problem consists of and we will study three techniques for solving this problem: the multiobjective integer linear programming, the ε constraint method, and heuristic methods. We will not implement the latter, since their study is very extensive and would require a deeper analysis.

All of these methods are used to try to obtain the optimal solution, or at least, an approximation to it. To do this, we have considered a set of general constraints, which are usually included in this type of problem and they are widely discussed in the literature (Dantzig (3) and Kocabas (6)).

Finally, we will compare the results obtained with different *solvers*: Xpress Optimizer, Gurobi and Cplex using multiobjective integer linear programming methods.

Índice

Índice de figuras

Índice de tablas

1. Introducción

En este trabajo abordados el estudio del problema de turnos de trabajo. Se trata de un problema relativamente reciente que despierta gran interés y que evoluciona rápidamente. Es un problema de programación lineal multiobjetivo que busca encontrar la mejor distribución de los turnos de trabajo de una empresa, teniendo en cuenta que éstos se encuentran limitados por una serie de restricciones muy variadas, que dependen del problema concreto que estemos estudiando y del enfoque del mismo. Las restricciones más habituales que se suelen considerar son: los recursos disponibles y la demanda que se debe cubrir. Generalmente, la elaboración de estos turnos, constituye un problema demasiado costoso de realizar de forma manual debido al elevado número de variables que intervienen. Por ello, se busca una solución aproximada utilizando algoritmos adecuados mediante sistemas informáticos.

Dentro del problema de turnos podemos destacar los tres tipos siguientes:

- **El problema de diseño de turnos**: basado en elegir la mejor secuencia de turnos de trabajo y la cantidad de los mismos necesaria en un cierto periodo de tiempo, P, de forma que minimice *las faltas* (o necesidad de más trabajadores) y *excesos de demanda* (o sobras de trabajadores). En este proyecto nos centraremos en el estudio de este tipo de problemas.

- **El problema de diseño de descansos**: se encarga de la selección de los mejores lapsos o intervalos de tiempo para que los trabajadores descansen en sus jornadas laborales, gestionando de forma adecuada estos momentos con el fin de proporcionar a los empleados una mayor calidad en su vida laboral e incrementando, también así, su rendimiento en la empresa o entidad. Las restricciones que habitualmente se consideran para este tipo de problemas son: los tiempos máximos y mínimos de los descansos, y también, los tiempos máximo y mínimo de trabajo consecutivos que puede realizar un empleado. Además, se pueden añadir otras restricciones particulares como pueden ser: el mayor número de periodos de descanso de fines de semana consecutivos, el descanso para comer en turnos partidos, etc.

- **Problema de diseño de tareas o labores**: surge al tratar de ubicar los recursos de una entidad en distintos intervalos de tiempo con la finalidad de maximizar su rendimiento. Los recursos utilizados en este tipo de problemas pueden ser muy variados: empleados, maquinaria, instalaciones, etc. Las principales restricciones proceden de los recursos necesarios en cada etapa de tiempo y del propio tiempo necesario para culminar una tarea.

El problema de turnos se puede llevar a cabo en una gran variedad de entidades que necesiten mejorar sus turnos laborales o gestionar sus recursos. Ejemplos de este problema se pueden encontrar en hospitales, aerolíneas, limpieza, etc. Para intentar dar solución a estos problemas se suelen utilizar técnicas de programación entera multiobjetivo, gracias a las cuales, distintas empresas u organizaciones pueden incrementar la eficacia y eficiencia del uso de sus plantillas, así como reducir costes, generar turnos de trabajo y descansos o gestionar sus recursos.

1.1. Objetivos

En este trabajo vamos a tratar de explicar y resolver el problema de diseño de turnos.

La resolución de este problema, pasa por intentar encontrar un conjunto de soluciones eficientes que buscaremos mediante dos métodos: la programación lineal entera multiobjetivo y el método de restricciones ε. Compararemos los resultados obtenidos, tanto con en el solver Xpress Optimizer, como con los programas de optimización Cplex y Gurobi.

Además, explicaremos cómo se podría resolver este problema mediante métodos heurísticos, aunque no implementaremos estos últimos.

Por otro lado, a nivel personal los objetivos que pretendo alcanzar son en gran medida formativos: ampliar la experiencia tanto en la búsqueda de resultados eficientes como en el uso de solvers de optimización.

1.2. Estructura

Para el correcto desarrollo del trabajo, vamos a comentar las etapas o pasos que hemos dado para completar el mismo.

1. **Planteamiento del problema**: En esta etapa expondremos de forma exhaustiva los distintos problemas de turnos de trabajo.

2. **Formulación del problema**: Una vez explicados los distintos tipos de problemas, procederemos a formular el problema concreto de diseño de turnos. En esta etapa, comentaremos y describiremos tanto las variables, como las restricciones que intervienen. Propondremos tres tipos de métodos distintos que intentan dar solución a este problema. Estos métodos son: la programación lineal entera multiobjetivo, el método de restricciones ε y algunos algoritmos para los métodos heurísticos.

3. **Datos del problema**: En este apartado indicaremos la procedencia de las fuentes de datos que nosotros hemos utilizado para los métodos de resolución anteriores excepto los métodos heurísticos. Explicaremos todos los aspectos referentes a estas fuentes de datos, es decir, su contenido y características particulares.

4. **Discusión y resultados**: En esta etapa se analizarán todos los resultados obtenidos al implementar en el solver Xpress Optimizer nuestros métodos de resolución: programación lineal entera multiobjetivo y el método restricciones ε. Además, realizaremos una comparativa con los resultados obtenidos con otros dos solvers: Cplex y Gurobi.

5. **Conclusiones y futuros estudios**: En este apartado comentaremos las conclusiones extraídas del estudio realizado en este trabajo. También expondremos algunas líneas de trabajo futuras que se pueden llevar a cabo.

2. Descripción y planteamiento del problema de turnos

En esta sección vamos a describir los tres tipos de problemas de turnos que podemos encontrarnos.

2.1. Problema de diseño de turnos

Se trata de un problema donde, de forma general, se considera un cierto periodo de tiempo, P, en el cual debemos elegir la mejor secuencia de turnos de trabajo y la cantidad necesaria de los mismos, con el fin de minimizar las faltas y excesos de trabajadores.

Con esta finalidad, el problema se descompone en n intervalos consecutivos, $I_1, \ldots, I_n$, de la misma longitud, de forma, que cada uno de ellos tiene que cubrir su propia demanda, es decir, se busca generar k turnos de trabajo con los que se pueda cubrir la demanda en cada intervalo de tiempo $I_j, \quad j = 1, \ldots, n$, sin que se produzcan ni excedentes ni faltas en la oferta.

Habitualmente, el problema planteado es cíclico, debido a que los turnos de trabajo y la demanda se repiten periódicamente. Estas repeticiones suelen ser semanales o a lo sumo mensuales, de manera, que el primer intervalo del primer periodo y el último intervalo del siguiente periodo suelen tomarse iguales.

Uno de los principales plantemientos del problema de diseño de turnos que se realiza desde el campo de la *Investigación Operativa* es el siguiente:

- Se denota por $\boldsymbol{v}$ al vector de todos los posibles turnos básicos del problema.
- Las variables que se utilizan para describir estos turnos de trabajo son las siguientes:
  - La hora más temprana de inicio de un turno j, también llamada, tiempo de inicio o $\boldsymbol{v_j.minStart}$.
  - La hora de inicio más tardía de un turno j, denominada $\boldsymbol{v_j.maxStart}$.
  - La longitud o duración mínima de un turno j, llamada $\boldsymbol{v_j.minLength}$.
  - La duración máxima de un turno j, o $\boldsymbol{v_j.maxLength}$.

A modo de ejemplo la Tabla 1 recopila los valores de estas variables correspondientes a cuatro tipos de turnos.

Turno	**minStart**	**maxStart**	**minLength**	**minLength**
v_1	05:00	8:00	7:00	9:00
v_2	09:00	11:00	7:00	9:00
v_3	14:00	16:00	7:00	9:00
v_4	21:00	23:00	7:00	9:00

Tabla 1: Ejemplo de las variables de distintos turnos básicos.

En esta tabla, si observamos la primera fila, vemos que el turno v_1 puede iniciarse entre las 5:00 horas y las 8:00 horas y que su duración es variable pudiendo abarcar entre 7 y 9 horas. Cada una de estas posibles variaciones es considerada como un turno de trabajo diferente, siendo una variación del turno básico v_1. En este mismo turno básico, si el intervalo de tiempo tuviera una longitud de 1 hora, tendríamos cuatro posibles inicios distintos y tres posibles duraciones distintas, lo que daría un total de 12 turnos diferentes entre sí.

- También se suelen utilizar otras dos variables, llamadas **AS** y **AH**. La primera, impone una limitación en el número de turnos de trabajo semanales de los empleados. La segunda, es una media de horas semanales de trabajo por cada empleado (u otra medida de tiempo que se decida utilizar). En nuestro estudio, no las consideraremos debido a que necesitaríamos conocer más datos acerca de los empleados de la empresa y, a priori, no disponemos de esta información.

- En este tipo de problemas se suelen considerar estas cuatro funciones objetivo a minimizar:

  - $\boldsymbol{F_0}$, que es la suma de los excesos de trabajadores producidos a lo largo de los intervalos de tiempo del problema.

  - $\boldsymbol{F_1}$, que es la suma de la falta de trabajadores producida a lo largo de los intervalos de tiempo del problema.

  - $\boldsymbol{F_2}$, que es el número de turnos utilizados en el problema. Es necesario tener presente que se suele buscar la menor variedad posible de turnos de trabajo.

  - $\boldsymbol{F_3}$ que es la media de las distancias al número medio de turnos por empleado, es decir,

    $$F_3 = \frac{1}{s}\sum_{i=1}^{s} |t_i - \mu_t|$$

    donde s representa el número de empleados, t_i el número de turnos realizados por el empleado i, y μ_t es la media de turnos realizados por todos los empleados.

  En nuestro caso, y debido a que queremos centrarnos en un caso más general, no tendremos en cuenta en nuestro estudio la última función objetivo, F_3, que tiene como finalidad evitar que los empleados tengan un número de turnos muy fraccionados.

2.2. Problema de diseño de descansos

A continuación, abordamos la descripción formal del problema de diseño de descansos (break scheduling problem, Musliu *et al.* (8), Gärtner *et al.* (4)).

Como ya comentamos previamente, se trata de seleccionar los intervalos de tiempo más adecuados para los descansos de los trabajadores a lo largo de sus jornadas laborales. La correcta gestión de estos momentos de descanso (o *slots de descanso*), proporcionará a los empleados una mayor calidad de trabajo, además, de una mejora en su rendimiento laboral.

De forma general, para analizar este problema se suele proceder como sigue:

- En el problema se divide la planificación de un periodo de tiempo P, en n intervalos consecutivos, $I_i = [a_i, a_{i+1}), \quad i = 1 \ldots, n$, con el mismo tamaño o longitud t (que habitualmente oscila entre 5 y 15 minutos), de forma que a lo largo de ellos se asignan los descansos de los trabajadores. Cabe destacar que estos intervalos tienen una estructura cíclica, con lo que se tiene que $I_1 = I_{n+1}$.

- Se establecen k turnos de trabajo, $s_1, \ldots, s_k, \quad k \in \mathbb{N}$, en los cuales debemos asignar los descansos. El tiempo de descanso de cada turno suele variar en función del tiempo de duración y horario del propio turno.

- Los intervalos de tiempo $I_j, \ j = 1 \ldots, n$, de la planificación verifican que:
  - El número de trabajadores necesarios en el intervalo I_j es w_j.
  - La necesidad de utilizar por los empleados un intervalo de tiempo extra para regresar a su puesto de trabajo tras finalizar su periodo de descanso.
  - Un trabajador se considera que está trabajando en un intervalo I_j si el empleado está en un turno de trabajo y no tiene asignado ningún descanso en ese intervalo.

- Cada turno de trabajo s_i, tiene asociado dos parámetros: inicio o $\boldsymbol{s.inicio_i}$ y final o $\boldsymbol{s.final_i}$.

- Cada periodo de descanso $\boldsymbol{b_i}$, está asociado a un turno de trabajo $\boldsymbol{s_i}$. Estos descansos pueden ser de dos tipos: el descanso regular y el descanso asociado al almuerzo o comida. A su vez, cada descanso tiene asociados dos parámetros: inicio o $\boldsymbol{b.inicio_i}$ y final o $\boldsymbol{b.final_i}$.

- A la hora de elaborar los descansos de cada turno, se deben tener en cuenta las siguientes restricciones:
  - Cada turno de descanso debe comenzar en un tiempo de inicio mayor o igual que el del propio turno de trabajo, es decir,
    $$b.inicio_i \geq s.inicio_i.$$
  - Cada turno de descanso debe terminar en un tiempo de final menor o igual que el del propio turno de trabajo, es decir,
    $$b.final_i \leq s.final_i.$$

  - El comienzo de un periodo de descanso debe ser menor o igual que el final del periodo de descanso, es decir,

$$b.inicio_i \leq b.final_i.$$

  - Dos periodos de descanso asociados al mismo turno de trabajo no deben solaparse, con lo que

$$b.inicio_i \leq b.final_i \leq b.inicio_j \leq b.final_j \quad i \neq j.$$

- La función objetivo a optimizar variará dependiendo del tipo de problema que estemos tratando y sus necesidades particulares.

Para una descripción más amplia y concreta de este problema véase Beer et al. (1).

2.3. Problema de diseño de tareas

Este problema trata de gestionar los recursos de una entidad asignando distintas tareas, o rutinas de tareas, principalmente a sus empleados, pero también puede ser utilizado en recursos como maquinaria, instalaciones, etc, con el fin de conseguir maximizar el rendimiento.

En el problema, al igual que en los casos anteriores, se divide la planificación de un periodo de tiempo P, en n intervalos consecutivos, $I_i = [a_i, a_{i+1}), \quad i = 1 \ldots, n$, con el mismo tamaño o longitud t. Además, los intervalos son cíclicos y la duración de los mismos varía entre 30 minutos y 2 horas, siendo este último el más habitual.

Las principales restricciones son:

- Cubrir las necesidades de cada uno de los intervalos de tiempo del problema.

- Minimizar los costes necesarios derivados del empleo de unos u otros recursos en cada intervalo de tiempo.

Este problema es más complejo que los expuestos anteriormente y suele resolverse mediante algoritmos basados en heurísticas (véase Jarrah *et al.* (5)), de hecho, en muchas ocasiones, abarca los dos problemas anteriores. Por todo esto, existe una gran variabilidad en el estudio de este problema (Bechtold y Showalter (2)). Para un estudio más profundo véase Mabert (7) y Thompson (10).

3. Métodos de resolución del problema de diseño de turnos

En este trabajo vamos a tratar un caso general del problema de diseño de turnos y por ello, y en adelante, nos centraremos exclusivamente en este primer tipo de entre los tres descritos en la sección anterior. A lo largo de esta sección propondremos tres métodos distintos para abordar el problema de diseño de turnos.

3.1. Resolución mediante programación lineal entera multiobjetivo

Un problema de programación lineal entera multiobjetivo busca encontrar un conjunto de soluciones *eficientes* de una función $F : \mathbb{R}^p \longrightarrow \mathbb{R}^m$ con

$$F(x_1, \ldots, x_p) = (F_0(x_1, \ldots, x_p), \ldots, F_m(x_1, \ldots, x_p)),$$

donde $x = (x_1, \ldots, x_p) \in S$ siendo S un conjunto formado por un número finito de restricciones y variables enteras. Las funciones $F_i(x), \quad i = 0, \ldots, m$, se denominan *funciones objetivo*, son lineales y de la forma

$$F_i(x) = \sum_{j=1}^{p} c_{ij} x_j, \quad c_{ij} \in \mathbb{Z}, \quad i = 0, \ldots, m.$$

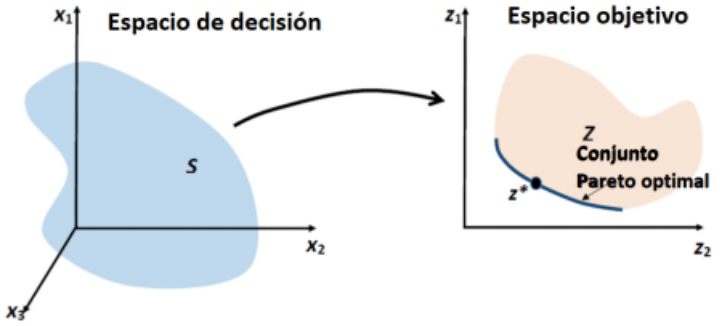

Figura 1: Conjunto de soluciones y solución *eficiente*.

Habitualmente, este tipo de problemas suele involucrar la resolución de otros problemas de optimización escalados mediante la selección de unos pesos adecuados ($W_i,\ i = 0, \ldots, m$). Estos últimos, suelen ser problemas de optimización con una sola función objetivo, que además de tener las propias restricciones del conjunto S, poseen otras restricciones añadidas a mayores.

Teniendo en cuenta lo anterior, nuestro problema de diseño de turnos trata de buscar la solución eficiente de un problema multiobjetivo, en el cual, las *restricciones*, suelen expresarse mediante un sistema de ecuaciones o inecuaciones también lineales.

Así, hemos formulado el problema de programación lineal entera multiobjetivo teniendo en cuenta las características que comentamos a continuación y que implementaremos utilizando el solver *Xpress Optimizer* para su resolución.

- **Las variables de entrada** son:

  - **La longitud del intervalo** que hemos denotado como **SlotLength**. Esta variable describe la amplitud o tamaño de los intervalos del problema (que suele variar entre 15 y 60 minutos, siendo su valor habitual 60 minutos).

  - **Días del periodo**. Variable que denotamos como **daysPerCycle** y que representa el número de días del periodo a planificar del problema. Típicamente son 7 días con una estructura cíclica.

  - **Número de intervalos** que hemos denotado como **n**. Esta variable es un número entero e indica el número de intervalos en que hemos dividido el problema. Se calcula como:

$$n = \frac{daysPerCycle \cdot 24 \cdot 60}{SlotLength}.$$

  - **Número de turnos posibles**. Hemos denotado esta variable como **m** y se trata de un valor entero que coincide con el número de todos los turnos posibles del problema a resolver. Se calcula de acuerdo con la fórmula:

$$m = \sum_{i=1}^{y} DistinctStart_i \cdot DistinctLength_i$$

donde y es el número de turnos base, $\boldsymbol{DistinctStart_i}$ se corresponde con el número de tiempos posibles de inicio que puede tener el turno $\boldsymbol{s_i}$ y $\boldsymbol{DistinctLength_i}$ es el número de distintas duraciones posibles del turno base $\boldsymbol{s_i}$.

– **Número de empleados requeridos**. Esta variable denotada como $\boldsymbol{x_t}$ es el número de empleados necesarios en el intervalo de tiempo t.

– **Día siguiente**. Variable denotada como $\boldsymbol{NextDay_s}$ que indica si el turno s se efectúa utilizando dos días distintos, por lo que hay que modificar este turno. Este tipo de turnos pertenecen al día en el que se iniciaron. El código en Xpress Mosel para crear esta variable es el siguiente:

forall s **in** turnos **do**
  if (ceil(daysPerCycle · (inicio_turno(s)+duracion_turno(s))/n) >
    ceil (daysPerCycle · inicio_turno(s)/n) **then**
      $NextDay(s) := 1$
  else
      $NextDay(s) := 0$
  end if
end do

– **Intervalos laborables de un turno** o $\boldsymbol{a_{ts}}$. Esta variable indica si un turno s realiza un trabajo o tarea en el intervalo de tiempo t. Su valor es:

$$a_{ts} = \begin{cases} 1 & \text{si el turno s realiza un trabajo en el intervalo t} \\ 0 & \text{en otro caso} \end{cases}$$

Su implementación en Xpress Optimizer es la siguiente:

forall (s **in** turnos) **do**
forall (k **in** 1..daysPerCycle) **do**
forall (i **in** 1..duracion_turno(s)) **do**
t1:=(inicio_turno(s)+(k-1) · round$(24 \cdot 60/\text{SlotLength}) + i$)
if (t1 $>$ n) **then**
$t1 :=$ (t1 mod n)
end if
a(t1, s) := 1
end do
end do
end do

– **Día del turno** o $\boldsymbol{diaturno_{ts}}$ es una variable que indica a que día pertenece el periodo de tiempo t del turno s. El código que hemos implementado para su cálculo:

forall (s **in** turnos, t **in** tiempo) **do**
if ((t-1) mod round$(24 \cdot 60/\text{SlotLenght}) <$ round$(24 \cdot 60/\text{SlotLenght})$
and NextDay(s) $= 1$) **then**
diaturno(t, s) := (ceil(daysPerCycle · t/n) + daysPerCycle $- 1$) mod daysPerCycle
else
diaturno(t, s) := ceil(daysPerCycle · t/n)
end if
end do

forall (s **in** $turnos, t$ **in** $tiempo \mid$ nextday$(s) = 1$ **and** $(t - 1) <$ round$(24 \cdot 60/\text{SlotLength})$
and $(t - 1)$ mod round$(24 \cdot 60/\text{SlotLength}) < 12 \cdot 60/\text{SlotLength})$ **do**
$diaturno(t, s) := daysPerCycle$
end do

– **Pesos de la función objetivo** que hemos denotado como $\boldsymbol{W_i}$ y se trata de una variable real verificando $W_i \geq 0, \quad \forall\, i$.

- **Las variables de decisión** son variables que inicialmente no tienen ningún valor, y éste, se calcula para encontrar la solución más eficiente. En nuestro caso estas variables son:

– **Indicador de turno activo** o $\boldsymbol{b_s}$. Es una variable binaria que indica si el turno s se utiliza o no a lo largo de los intervalos de tiempo del problema. De forma que:

$$b_s = \begin{cases} 1 & \text{si el turno se utiliza en el problema} \\ 0 & \text{en otro caso} \end{cases}$$

– **Número de empleados asignados** o $\boldsymbol{w_{sd}}$ es una variable entera que representa el número de empleados asignados al turno s el día d.

– **Número de trabajadores en el intervalo $\boldsymbol{t}$** que hemos denotado por $\boldsymbol{l_t}$. Es una variable entera que representa la suma de todos los trabajadores activos en cualquier turno en el intervalo de tiempo t. Su expresión es la siguiente:

$$l_t = \sum_{s=1}^{m} a_{ts} \cdot w_{s\,j}$$

con

$$j = \text{floor}\left(\frac{daysPerCycle \cdot t}{n}\right), \quad \forall\, t = 1, \ldots, n.$$

– **Las tres componentes de la función objetivo** son:

1. $\mathbf{F_0}$ o suma del exceso de trabajadores en cada uno de los intervalos de tiempo del problema. Viene dada por:

$$F_0 = \sum_{t=1}^{n} \text{máx}(l_t - x_t, 0).$$

2. $\mathbf{F_1}$ o suma de la demanda no cubierta por falta de trabajadores en todos los intervalos de tiempo. Su expresión es:

$$F_1 = \sum_{t=1}^{n} \text{máx}(x_t - l_t, 0).$$

3. $\mathbf{F_2}$ o número total de turnos utilizados en el problema. Viene dada por:

$$F_2 = \sum_{t=1}^{m} b_i.$$

– **Exceso de trabajadores**. Se trata de una variable entera que hemos denotado como $\boldsymbol{ex_t}$ y que coincide con el número de excesos de trabajadores en el intervalo de tiempo t. Esta variable nos permite expresar la componente F_0 de la función objetivo de la siguiente forma:

$$F_0 = \sum_{t=1}^{n} ex_t.$$

– **Falta de trabajadores**. Es una variable entera que denotaremos por $\boldsymbol{sh_t}$ y que representa la oferta sin cubrir por los trabajadores en el intervalo de tiempo t. Esta variable nos permite expresar la componente F_1 de la función objetivo de la siguiente forma:

$$F_1 = \sum_{t=1}^{n} sh_t.$$

- **Las restricciones del problema** son las ecuaciones o inecuaciones en las que intervienen las variables del problema y que limitan la función objetivo.

  En nuesto caso concreto estas restricciones son:

  – **Primera restricción**:

  $$ex_t \geq (l_t - x_t) \cdot SlotLength.$$

  Esta inecuación nos permite contabilizar los excesos de turnos introducidos en cada intervalo del problema. La importancia del tiempo de duración del excedente de oferta viene expresada mediante a través de la multiplicación por la variable **SlotLength** en la inecuación.

  – **Segunda restricción**:

  $$sh_t \geq (x_t - l_t) \cdot SlotLength.$$

La inecuación anterior nos permite contabilizar la demanda no cubierta en cada uno de los intervalos del problema. Se multiplica por el tamaño del intervalo por la misma razón que en el caso de la inecuación anterior.

– **Tercera restricción**:

$$ex_t \geq 0.$$

Esta ecuación indica que los excesos de turnos producidos es una variable no negativa.

– **Cuarta restricción**:

$$sh_t \geq 0.$$

Esta ecuación indica que la falta de demanda es una variable no negativa.

– **Quinta restricción**:

$$l_t \cdot SlotLength + sh_t - ex_t = x_t \cdot SlotLength.$$

La inecuación anterior nos permite contabilizar los excesos y faltas de oferta producidos a lo largo de los intervalos de tiempo del problema. Se multiplica por el tamaño del intervalo para dar importancia al tiempo de duración de dicho excedente, al igual que ocurría en algunas de las restricciones anteriores.

– **Sexta restricción**:

$$w_{sd} \leq M \cdot b_s \quad \forall\, s = 1, \ldots, m,$$

siendo

$$M = \max_{t=1,\ldots,n} \{x_t\}.$$

Mediante la restricción anterior se impone que el número de trabajadores utilizados en cada uno de los intervalos no supere al mayor valor de demanda de todos los intervalos del problema.

– **Séptima restricción**:

$$\sum_{s=1}^{m} b_s \leq K,$$

donde K es el número máximo de turnos permitidos.

Esta restricción sirve para limitar el número de turnos que se pueden utilizar en el problema. Aunque esta restricción puede ser muy útil, nosotros no la vamos a utilizar para poder, de esta forma, realizar las comparaciones de resultados.

Una vez concluida la exposición de las restricciones del problema, debemos hacer notar que la quinta restricción implica que se satisfacen las dos primeras, por tanto, se pueden utilizar todas a la vez o solo un grupo de ellas.

- La función multiobjetivo de este problema busca el mínimo de la suma de las tres componentes $F_0,\ F_1,$ y F_2 con los pesos $W_0,\ W_1,$ y W_2, es decir, $W_0 \cdot F_0 + W_1 \cdot F_1 + W_2 \cdot F_2$.

Finalmente, y teniendo en cuenta todo lo descrito previamente, el problema de diseño de turnos con programación lineal entera multiobjetivo, resulta:

minimizar $\quad W_0 \cdot F_0 + W_1 \cdot F_1 + W_2 \cdot F_2$

sujeto a:

$$ex_t \geq (l_t - x_t) \cdot SlotLength$$

$$sh_t \geq (x_t - l_t) \cdot SlotLength$$

$$ex_t \geq 0$$

$$sh_t \geq 0 \qquad .$$

$$M = max(x_t)$$

$$w_{sd} \leq M \cdot b_s \quad \forall\ s = 1, \ldots, m,$$

$$l_t \cdot SlotLength + sh_t - ex_t = x_t \cdot SlotLength$$

$$W_0 \geq 0$$

$$W_1 \geq 0$$

$$W_2 \geq 0$$

3.2. Resolución mediante el método biobjetivo de restricción ε

Este método, que en lo sucesivo denotaremos por ECM (ε constraint method), se utiliza para el cálculo exacto de la frontera de Pareto en problemas de programación entera multiobjetivo prestando especial atención a un conjunto de objetivos concretos (véase Sáez-Aguado y Trandafir (11)). Habitualmente, se fija la atención en los dos objetivos más importantes y los restantes son prefijados mediante restricciones generadas de forma artificial.

En el problema de diseño de turnos que estamos considerando en este proyecto, tenemos como objetivos destacados las componentes F_0 (suma total de los excesos de personal en los intervalos de tiempo del problema) y F_1 (suma total de la falta de personal en los intervalos de tiempo del problema) descritas en la sección anterior, ya que para una entidad, cubrir toda la demanda con la menor cantidad posible de recursos, suele ser el objetivo principal.

Para poder utilizar este método, debemos añadir una de estas dos restricciones que dependen del parámetro $\varepsilon \in \mathbf{Z},\ \varepsilon \geq 0$:

- $F_0 = \sum_{t=1}^{n} ex_t \leq \varepsilon.$
- $F_1 = \sum_{t=1}^{n} sh_t \leq \varepsilon.$

Una vez que hayamos seleccionado una de estas dos restricciones anteriores, la componente involucrada (F_0 o F_1) será la variable independiente y la representaremos en el eje de abscisas, y la otra componente, será la variable dependiente y la representaremos en el eje de ordenadas. O de otra forma, la componente de la restricción seleccionada se denominará variable restrictiva y la otra la variable a minimizar.

La componente F_2 de la función objetivo (número de turnos activos) la fijaremos utilizando la restricción

$$F_2 = \sum_{i=1}^{m} b_i \leq C,$$

siendo $C \in \mathbf{N}$ una constante arbitraria.

De esta forma, en nuestro problema de diseño de turnos, el algoritmo para implementar el método de restricciones ε es:

$\varepsilon = 10^{10}$
while $\varepsilon \geq 0$:
Función objetivo $= F_1$
sujeto a:

$$ex_t \geq (l_t - x_t)$$
$$sh_t \geq (x_t - l_t)$$
$$ex_t \geq 0$$
$$sh_t \geq 0$$
$$w_{sd} \leq M \cdot b_s \quad \forall\ s = 1, \ldots, m,$$
$$l_t + sh_t - ex_t = x_t$$
$$F_0 = \sum_{t=1}^{n} ex_t \leq \varepsilon \qquad .$$
$$F_2 = \sum_{i=1}^{m} b_i \leq C$$

minimize Función objetivo
If not exist Solución:
break:
else:
$\varepsilon = F_0.sol - SlotLength$
write$(F_0,\ F_1)$

3.3. Resolución mediante métodos heurísticos

Hay problemas en los que no existe, o no es viable, la utilización de un método algorítmico para obtener una solución, y ésta última debe ser generada y mejorada mediante un proceso de búsqueda.

Un método heurístico es un proceso que busca soluciones eficientes mediante un guiado adecuado del proceso de búsqueda capaz de determinar su proximidad a una solución y la calidad de la misma.

Se suele utilizar en problemas no abarcables en programación lineal entera para buscar de un modo rápido y sencillo una solución (aunque esta última no sea la óptima).

A continuación, vamos a describir dos tipos de métodos heurísticos aplicables al problema de diseño de turnos (Musliu *et al.* (9)).

- **Heurística de Greedy**.

  Es un método constructivo que no necesita de una *solución* de arranque inicial con la que llegar a una aproximación al óptimo. La solución que nos proporciona el método de Greedy se puede utilizar para inicializar otros tipos de métodos heurísticos que proporcionarán una corrección.

  Las etapas o pasos del algoritmo de Greedy para nuestro caso concreto del problema de diseño de turnos son:

  - **Paso 1:** Detectar todos los puntos temporales en los que se produce un incremento en las demandas no cubiertas del problema. Todos estos puntos se almacenan día a día.

  - **Paso 2:** Por cada punto temporal identificado en el paso previo, se determinan todos los turnos disponibles que pueden cubrir esa región de tiempo.

  - **Paso 3:** Se selecciona el primer turno de los identificados en la etapa anterior para cubrir los puntos temporales encontrados en el primer paso, asignando el turno elegido, al día en cuestión, con la cantidad necesaria de turnos en dichos puntos temporales.

  - **Paso 4:** Detectar los puntos de tiempo en los cuales se producen excesos de turnos asignados. Estos puntos se almacenan diariamente.

- **Paso 5:** Se eliminan todos aquellos turnos que no hayan sido seleccionados en el paso 3 y que producen excesos de oferta en los puntos de tiempo detectados en el paso 4.

- **Paso 6:** Todos los turnos que se han eliminado en la etapa anterior y que no están asignados a ningún día se deshechan y no se vuelven a tener en cuenta en el algoritmo.

- **Heurística de búsqueda local**.

  Conocida en la literatura como *local search*, es un método de mejora que requiere de una solución inicial (por ejemplo, la dada por el método de Greedy).

  En cada iteración del proceso sólo se busca en el entorno del estado actual del problema y es usualmente empleada para la generación o mejora de una solución. Tiene el inconveniente de poder quedar estancado en máximos o mínimos locales generando soluciones de baja calidad.

  En este algoritmo debemos tener en cuenta las siguientes acciones:

  - **Modificación del número de empleados** (denotada también como *ChangeStaff*) nos indica que el turno seleccionado puede aumentar o disminuir en una unidad la oferta de empleados para un día concreto.

  - **Modificación de la duración del turno** (denotada también como *ChangeLength*) indica que el turno seleccionado puede aumentar o disminuir su duración en un intervalo de tiempo.

  - **Modificación del inicio de turno** (denotada también como *ChangeStart*) indica que el turno seleccionado puede iniciarse en un intervalo de tiempo anterior o posterior.

  - **Crear nuevo turno** (o *CreateShift*) se produce cuando se aplica *ChangeStaff* a un turno no activo (sin empleados asignados).

  - **Eliminar turno** (o *RemoveShift*) tiene lugar cuando tras un *ChangeStaff* obtenemos como resultado un turno que no tiene oferta ningún día. Este turno se eliminaría de forma permanente.

Las posibles composiciones de acciones básicas son:

- **Movimiento de fronteras** o *MoveBorders* se produce cuando la frontera temporal entre dos turnos se desplaza a izquierda o derecha. El tiempo inicial y final de ambos turnos será, por tanto, modificado. Este movimiento es el resultado de la composición de *ChangeLength* y *ChangeStaff*.

- **Intercambio de empleados** o *ExchageStaff* se produce cuando en un día concreto un empleado pasa de formar parte de un turno a otro distinto. Es el resultado de la composición de dos acciones del tipo *ChangeStaff* y puede llegar a dar lugar a *RemoveShift* o *CreateShift*.

- **Unión de turnos** o *JoinShift* es una acción que tiene lugar cuando todos los empleados de un turno son añadidos a otro turno diferente de forma que el turno inicial pasa a ser eliminado. Esta acción es el resultado de la composición de varios *ChangeStaff* y da lugar a una acción del tipo *RemoveShift* e incluso puede llegar a producir la acción *CreateShift*.

- **División de turnos** o *SplitShift*. Se produce cuando un turno se divide en dos dando lugar a la creación de un nuevo turno además del original. Éste último tendrá una unidad de tiempo de inicio o de longitud diferente. Esta acción es el resultado de la composición de todas las acciones básicas salvo la de tipo *RemoveShift*.

- **Modificación de tiempo final o inicial de un turno**, también llamada *ChangeStartKeepEndFix*, tiene lugar cuando se modifica, o bien el inicio de un turno o bien su final, mediante un aumento o disminución en una unidad temporal. Es el resultado de la composición de las acciones *ChangeStart* y *ChangeLength*.

Las etapas de este algoritmo son:

- **Paso 1:** Encontrar el periodo de mayor longitud con excesos o faltas de oferta.

- **Paso 2:** Cuantificación (alta, media o baja) de la contribución a esa carencia o exceso de oferta de cada turno implicado.

- **Paso 3:** Selección del turno con la mayor contribución a esa falta o exceso de oferta.

- **Paso 4:** Determinar la posición (izquierda, media o derecha) en la que se produce esa carencia o exceso en el turno seleccionado en la etapa anterior.

- **Paso 5:** Clasificación de los excesos y faltas de oferta en cortos, medios o largos, teniendo en cuenta el turno seleccionado y sus turnos *vecinos.*

- **Paso 6:** Decisión de la acción a realizar en función de las dos etapas anteriores.

- **Paso 7:** Si se produce una mejora, se itera el algoritmo. Si no existe una mejora, pero hay otros tipos de turnos (paso 2) que contribuyen con faltas o excesos, se analiza el siguiente turno con mayor contribución y se continua con el paso 4. En caso de que no haya mejoras, ni turnos con faltas o excesos, se exploran todos los turnos vecinos involucrados en el paso 2.

4. Fuentes de datos del problema

A lo largo de esta sección vamos a trabajar con cuatro tipos distintos de fuentes de datos. Éstas, fueron inicialmente introducidas en Musliu (8) y Musliu *et al.* (9) y son las que utilizaremos en nuestras simulaciones.

Todas estas fuentes de datos tienen en común las siguientes variables:

- **MinuteInterval**: esta variable se corresponde con la longitud del intervalo de tiempo expresado en minutos. Es frecuente tomar 15, 30 o 60 minutos.

- ***DaysPerCycle***: esta variable representa el número de días del periodo P seleccionado y que posteriormente hemos procedido a dividir en subintervalos. La mayor parte de los datos considerados toman esta variable con un valor de 7 días.

- ***FlagCyclical***: esta variable expresa si el problema es cíclico o no. En nuestro caso, como ya hemos comentado previamente, esta variable siempre será *verdadera* indicando que el problema es cíclico.

- ***Requirements***: esta variable es un vector que contiene valores enteros expresando la demanda en cada intervalo de tiempo del problema.

- ***Number of shift templates***: esta variable nos indica el número de turnos base que tiene el problema.

- ***Shift templates***: esta variable nos muestra en forma de matriz las distintas características de cada uno de los turnos básicos.

- ***Duties per week***: esta variable representa el número de turnos que puede realizar un empleado en el periodo de tiempo P considerado. Nosotros no la hemos tenido en cuenta ya que su inclusión produce un resultado muy distante al valor óptimo. Además, es muy dependiente de la normativa de la entidad y de la situación particular de cada trabajador.

- ***Average number of hours per week***: representa el número medio de horas en nuestro periodo de tiempo P que realiza un empleado. Esta variable tampoco la tendremos en cuenta por las mismas razones que la variable anterior.

- ***Weights about the criteria***: es un conjunto de variables, W_1, W_2, W_3 y W_4, que

representan los pesos de la función objetivo. Son todos valores reales mayores o iguales que 0, es decir, $W_i \geq 0,\ i = 1,\ 2,\ 3,\ 4$.

Las cuatro fuentes de datos que vamos a utilizar las hemos obtenido de http://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html y sus características son las siguientes:

- **Fuente de datos 1**: sus instancias se generan aleatoriamente. En ningún intervalo se deberían producir ni excesos ni faltas de trabajadores. Por otro lado, el número de turnos básicos es generado también aleatoriamente oscilando entre 1 y 5. A su vez, cada turno seleccionado tiene el mismo número de tareas asignadas en días de diario (de lunes a viernes) con una probabilidad de 0,9. También, los turnos asignados el sábado y el domingo tienen el mismo número de tareas y su probabilidad es de 0,9. El peso que se ha dado tanto a las faltas como a los excesos es 1. El peso del número de turnos es uno de los tres valores siguientes: 15, 30 o 60, que además, debe coincidir con el tamaño del intervalo.

- **Fuente de datos 2**: sus instancias se generan también aleatoriamente. En ningún intervalo se deberían producir ni excesos ni faltas de trabajadores. En cuanto al número de turnos base, las 10 primeras instancias o intervalos se generan en base a 12 turnos; las instancias desde la 11 a la 20 se generan con 16 turnos básicos y las últimas 10 instancias (desde la 21 a la 30) tienen 20 turnos. Al igual que en el caso anterior, cada turno seleccionado tiene el mismo número de tareas asignadas en días de diario (de lunes a viernes) con una probabilidad de 0,9, así como también lo tienen los turnos asignados el sábado y el domingo (mismo número de tareas y probabilidad de 0,9). El peso, tanto de los defectos como de los excesos, es 1 y el peso del número de turnos de nuevo toma uno de los tres valores siguientes: 15, 30 o 60, que además, coincide con el tamaño del intervalo.

- **Fuente de datos 3**: en este caso los datos están generamos de forma aleatoria pero incluyendo algunos turnos no válidos. En cada intervalo de tiempo se generan tareas con el mismo número de turnos válidos y no válidos. No existe una solución que no tenga excesos o faltas de tareas en los intervalos. El número de turnos que se toma es el mismo que en el caso de la Fuente de datos 2, es decir, las 10 primeras instancias o intervalos se generan en base a 12 turnos; las instancias desde la 11 a la 20 se generan con 16 turnos básicos y las últimas 10 instancias (desde la 21 a la 30) tienen 20 turnos. El resto de valores se ha tomado igual que en las dos fuentes de datos anteriores.

- **Fuente de datos 4**: esta última fuente de datos se compone de 3 instancias. La primera es una instancia real tomada de un *Call Center*. La segunda y tercera instancia, se toman modificando la instancia número 5 de la Fuente de datos 3 y tienen un tamaño de intervalo de 30 y 60 minutos, respectivamente, es por ello que la instancia 3 tiene asignadas el doble de tareas que la instancia 2 en cada intervalo.

De cada fuente de datos tomaremos el resultado de 20 instancias, excepto de la última, de la que tomaremos las 3 que contiene.

5. Discusión y resultados

En esta sección vamos a exponer y comentar los resultados obtenidos al utilizar nuestro modelo del problema de turnos implementado en el solver Xpress Optimizer cuando utilizamos las fuentes de datos comentadas en el apartado anterior. El máximo tiempo de ejecución que hemos permitido es de 1 minuto.

El equipo utilizado para obtener los resultados es un ordenador MSI-7924, con un procesador i7-4790 @ 3.6GHz y 4 núcleos, con sistema operativo Windows 10 pro y 16 GB de memoria RAM.

5.1. Resultados obtenidos utilizando programación entera

Con programación entera hemos elaborado una tabla de resultados para cada uno de los cuatro tipos de fuentes de datos que describimos en la sección anterior. En estas tablas, cada fila, exceptuando la primera, corresponde a una instancia de la correspondiente fuente de datos. Para cada fila tenemos nueve columnas que indican lo siguiente:

- **1ª columna (Nº)**: indica el número de instancia de la fuente de datos.
- **2ª columna (Nº turnos posibles)**: indica el número de turnos que se pueden utilizar para resolver el problema. Son el resultado de todas las posibles variaciones de los turnos básicos.
- **3ª columna (Tamaño del intervalo)**: indica el tamaño de los intervalos de tiempo.
- **4ª columna (F_0)**: indica el valor de la componente F_0 (excesos de demanda) de la función objetivo.
- **5ª columna (F_1)**: indica el valor de la componente F_1 (faltas de demanda) de la función objetivo.
- **6ª columna (F_2)**: indica el valor de la componente F_2 (número de tunos utilizados) de la función objetivo.
- **7ª columna (Objetivo)**: indica el valor de la función objetivo.
- **8ª columna (Tiempo)**: indica el tiempo de ejecución del modelo.

- **9ª columna (Hueco)**: indica el valor del hueco relativo ($H.R$), de la instancia que mide la optimalidad relativa de la solución actual. Cuando el hueco tiene valor 0 significa que la solución obtenida es la óptima. Se calcula de la siguiente forma:

$$H.R = 100 \left(\frac{VO - CI}{VO} \right),$$

donde **VO** es el valor óptimo alcanzado y **CI** es la cota inferior.

Fuente de datos 1

Nº	**Nº turnos posibles**	**Tamaño del intervalo**	F_0	F_1	F_2	**Objetivo**	**Tiempo**	**Hueco**
1	39	60	0	0	8	480	0,039	0
2	110	30	0	0	10	300	2,054	0
3	39	60	0	0	10	600	0,033	0
4	110	30	0	0	15	450	4,361	0
5	39	60	0	0	8	480	0,069	0
6	39	60	0	0	7	420	0,025	0
7	110	30	0	0	9	270	0,214	0
8	360	15	0	0	10	150	0,671	0
9	360	15	0	0	10	150	0,21	0
10	110	30	0	0	11	330	0,506	0
11	360	15	0	0	2	30	0,101	0
12	360	15	0	0	6	90	0,166	0
13	360	15	0	0	7	105	0,157	0
14	360	15	0	0	13	195	7,277	0
15	39	60	0	0	3	180	0,022	0
16	360	15	0	0	15	225	4,438	0
17	110	30	0	0	18	540	24,43	0
18	39	60	0	0	12	720	0,205	0
19	360	15	0	0	12	180	3,129	0
20	39	60	0	0	9	540	0,032	0

Tabla 2: Fuente de datos 1. Costes de cada etapa.

Como se puede apreciar en la Tabla 2, correspondiente a la fuente de datos 1, no existen ni excesos ni faltas de oferta (variables F_0 y F_1 respectivamente). Además, como se puede observar en la columna 6 de dicha tabla, los tiempos de ejecución son muy bajos ya que la mayoría de la instancias no supera el segundo de ejecución. Por otro lado, el mayor coste temporal se produce en la instancia 17 con un tiempo de ejecución de 24,43 segundos que es debido a que esta instancia se genera utilizando el mayor número de tipos de turnos (18) de entre la batería de instancias de esta fuente.

Fuente de datos 2

Nº	**Nº turnos posibles**	**Tamaño del intervalo**	F_0	F_1	F_2	**Objetivo**	**Tiempo**	**Hueco**
1	39	60	0	0	12	720	0,508	0
2	39	60	0	0	12	720	0,296	0
3	110	30	0	0	12	360	1,003	0
4	110	30	0	0	12	360	0,492	0
5	39	60	0	0	12	720	0,312	0
6	110	30	0	0	12	360	0,012	0
7	39	60	0	0	12	720	0,24	0
8	360	15	0	0	12	180	2,229	0
9	110	30	0	0	12	360	0,112	0
10	39	60	0	0	11	660	0,683	0
11	110	30	0	0	16	480	59,128	17,18
12	39	60	0	0	15	900	0,865	0
13	39	60	0	0	15	900	0,725	0
14	39	60	0	0	14	840	0,642	0
15	110	30	0	0	16	480	9,688	0
16	360	15	0	0	16	240	0,256	0
17	39	60	0	0	16	960	0,654	0
18	39	60	0	0	14	840	0,52	0
19	360	15	0	0	16	240	0,742	0
20	39	60	0	0	16	960	0,269	0

Tabla 3: Fuente de datos 2. Costes de cada etapa.

En la Tabla 3 se puede observar, nuevamente, que no se produjeron ni excesos ni faltas de oferta. Los tiempos de ejecución son, de nuevo, muy bajos, exceptuando las instancias 11 y 15, con tiempos de 59,128 y 9,688 segundos, respectivamente. La instancia 11 ha llegado prácticamente al tiempo máximo de ejecución, que como ya comentamos, es de 1 minuto (60 segundos), por lo que podría existir una mejor solución, aunque este hecho parece poco probable, pues el hueco de esta instancia es moderadamente bajo (17,18 segundos).

Fuente de datos 3

Nº	**Nº turnos posibles**	**Tamaño del intervalo**	F_0	F_1	F_2	**Objetivo**	**Tiempo**	**Hueco**
1	360	15	1365	825	13	2385	0,416	0
2	110	30	4560	2520	17	7590	8,773	0
3	110	30	3930	5160	15	9540	3,838	0
4	110	30	5070	1020	15	6540	15,873	0
5	39	60	5640	3420	11	9720	0,879	0
6	360	15	1350	510	14	2070	0,282	0
7	360	15	2865	2985	15	6075	0,38	0
8	110	30	4560	3630	13	8580	0,867	0
9	360	15	2895	2850	17	6000	4,441	0
10	110	30	900	1530	17	2940	1,101	0
11	110	30	2040	2730	14	5190	59,927	1,57
12	360	15	2910	825	25	4110	59,412	0,71
13	360	15	1905	2355	24	4620	59,382	2,16
14	39	60	6180	2580	14	9600	1,02	0
15	110	30	6360	4350	18	11250	5,913	0
16	39	60	3720	6300	10	10620	1,353	0
17	360	15	2505	1890	19	4680	6,703	0
18	110	30	4350	1650	18	6540	40,415	1,6
19	360	15	1590	2955	24	4905	59,314	2,5
20	110	30	1050	7320	18	8910	16,683	0

Tabla 4: Fuente de datos 3. Costes de cada etapa.

En la Tabla 4 observamos que los tiempos de ejecución son mucho más elevados que en las tablas de resultados anteriores. Además, se producen excesos y faltas de ofertas, con valores que pueden calcularse dividiendo el valor de las funciones objetivo F_0 y F_1 entre el tamaño del intervalo. Por último, indicar que hay varias instancias que alcanzan un tiempo cercano al tiempo límite de ejecución por lo que pueden contener un resultado distinto del óptimo, aunque es poco probable, pues sus huecos tienen valores relativamente bajos. Estas instancias son la 11, la 12, la 13 y la 19.

Fuente de datos 4

Nº	**Nº turnos posibles**	**Tamaño del intervalo**	F_0	F_1	F_2	**Objetivo**	**Tiempo**	**Hueco**
1	22	60	7380	10680	12	18420	0,077	0
2	110	30	5640	3420	11	9720	4,311	0
3	39	60	11040	7080	11	18780	0,748	0

Tabla 5: Fuente de datos 4. Costes de cada etapa.

En la Tabla 5, con datos reales en la instancia 1, observamos que los tiempos de ejecución son bajos. Sin embargo, en todas las instancias las variables F_0 y F_1, es decir, los excesos y faltas de oferta, tienen valores muy elevados. La instancia 1 del problema requiere el menor tiempo de ejecución (0,077 segundos), pero también tiene una gran cantidad de demanda no cubierta de forma exacta. Esto puede deberse al número, relativamente pequeño, de turnos posibles de esta instancia, tan solo 22, de forma que podrían mejorarse los resultados si se contara con más variedad de turnos distintos.

5.2. Resultados utilizando el método de restricciones ε

El método de restricciones ε se utiliza para ajustar algunas de las componentes de la función objetivo (sección 3.2). Prestaremos especial atención a las componentes F_0 y F_1 (excesos y faltas de oferta respectivamente), por lo que no tiene sentido estudiar las fuentes de datos 1 y 2 ya que no tienen ni excesos ni faltas. En concreto, la componente F_0 será la denominada *variable restrictiva* y observaremos las variaciones que se producen en los valores de la componente F_1 que será la *variable a minimizar*.

Los resultados se mostrarán a través de gráficas en las que el eje de abscisas representa los valores de F_0 y el eje de ordenadas los valores de F_1.

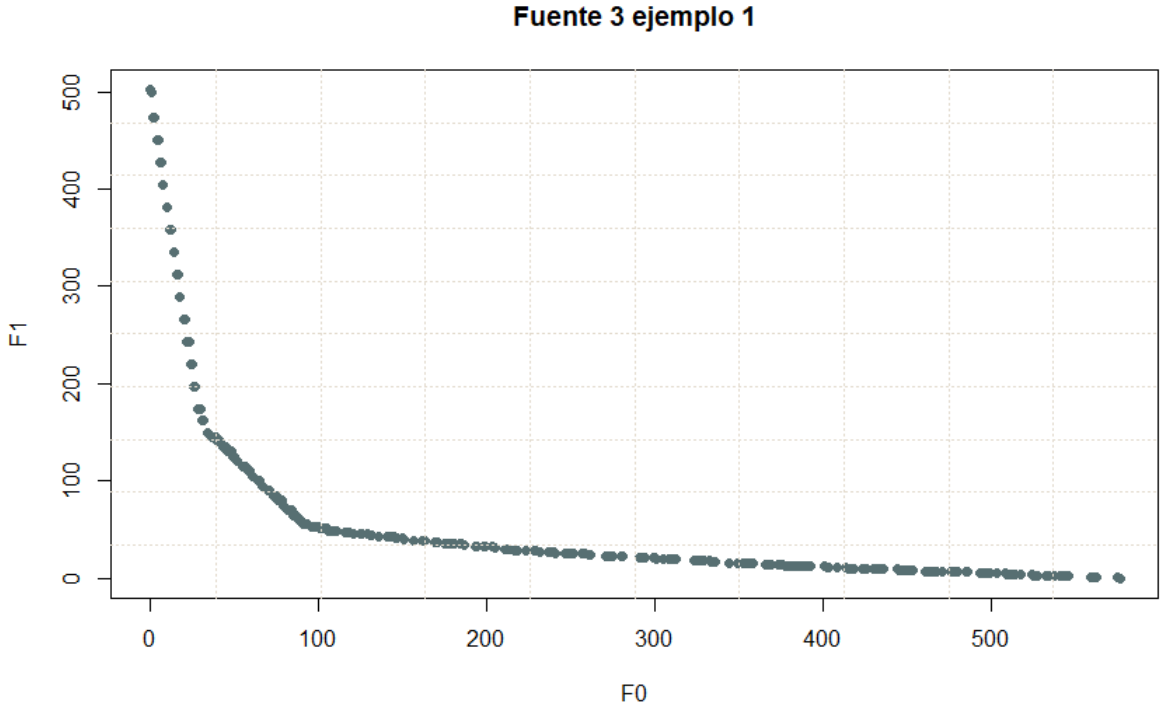

Figura 2: Resultados del método de restricciones ε en la instancia 1 de la fuente de datos 3.

En la Figura 2 se pueden observar los distintos valores óptimos de los excesos y faltas de oferta. A medida que se incrementa el valor de la variable F_0 se produce, como cabía esperar, una disminución en el déficit de demanda. Este descenso, es más pronunciado para valores de F_0 comprendidos entre 0 y 30 (como nos indica la pendiente de la recta que se observa en la figura). Por tanto, un valor equilibrado para la entidad se encontraría tomando la variable de excesos de demanda, F_0, con un valor oscilando entre 30 y 100.

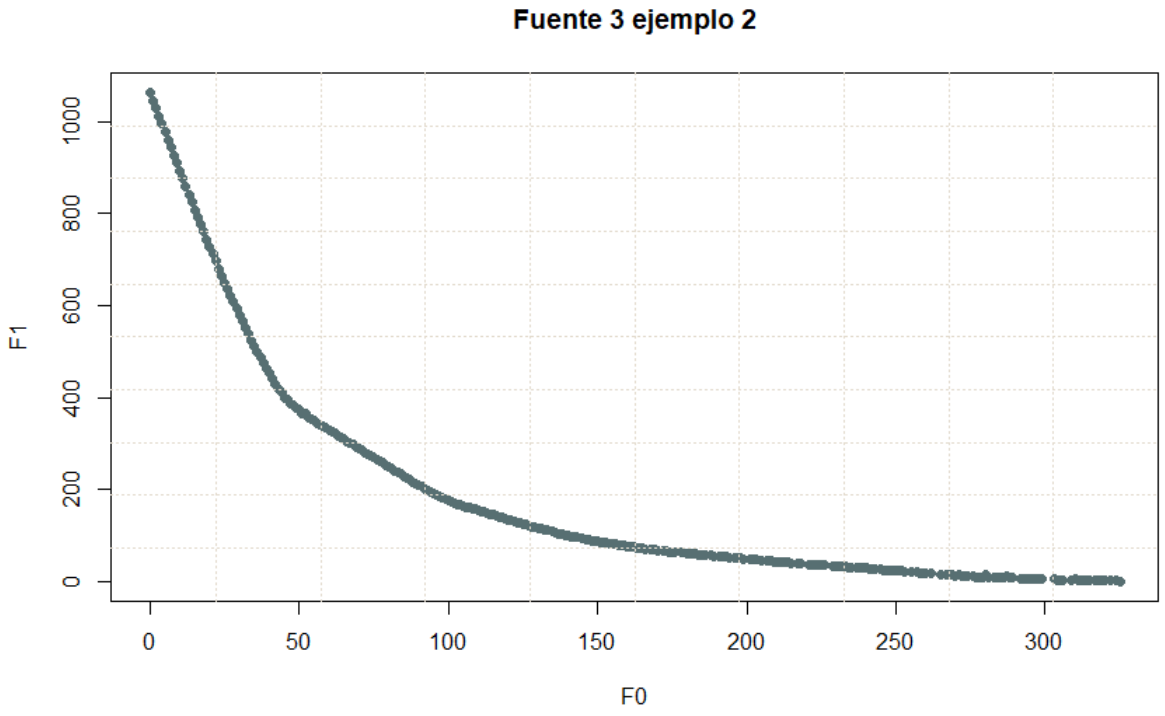

Figura 3: Resultados del método de restricciones ε en la instancia 2 de la fuente de datos 3.

En la Figura 3 se puede observar un decrecimiento similar al de una exponencial negativa, mayor entre $F_0 = 0$ y $F_0 = 50$ y algo más suavizado posteriormente. Además, podemos afirmar que si la entidad está buscando que no se produzca ningún exceso en la oferta de trabajadores, debe aceptar valores muy elevados de demanda no cubierta. Por otro lado, un equilibrio entre ambas componentes se encuentra para valores que oscilan entre 50 y 95 para F_0 y entre 200 y 390 para F_1.

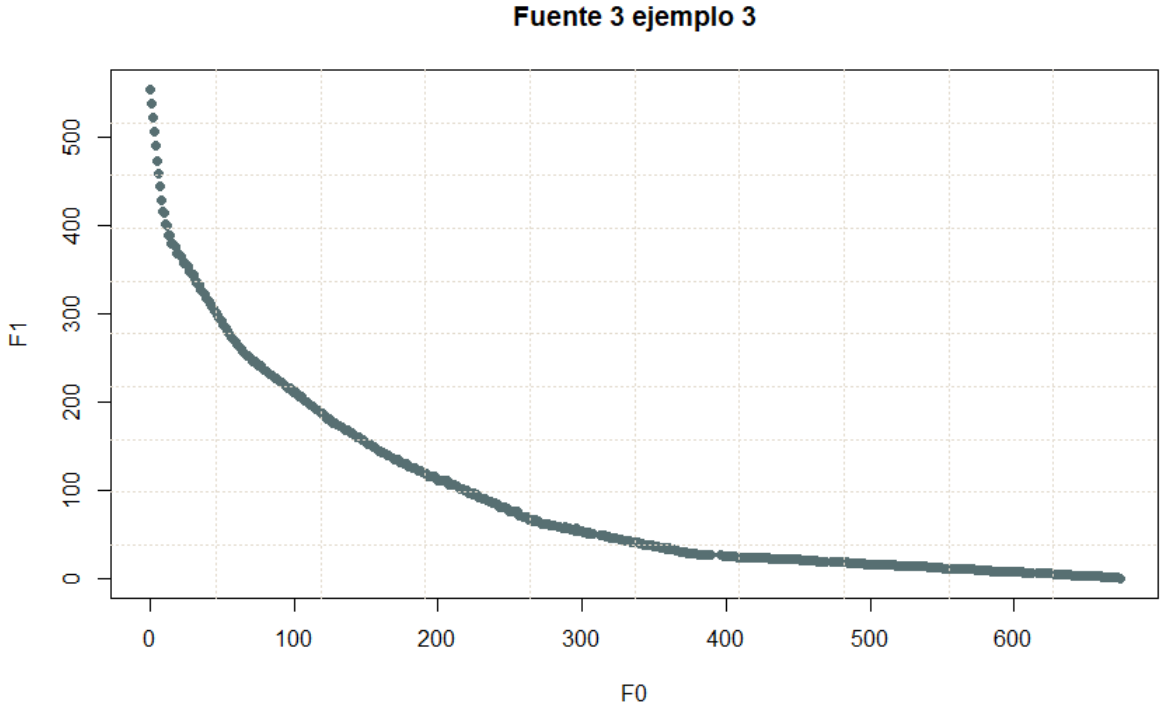

Figura 4: Resultados del método de restricciones ε en la instancia 3 de la fuente de datos 3.

La Figura 4 se corresponde con el último ejemplo para esta fuente de datos. Se puede apreciar un decrecimiento en la gráfica con una gran acumulación de datos y con valores para la componente F_0 oscilando entre 100 y 300, que son los que deberíamos tomar si quisiéramos un equilibrio entre ambas componentes. Si se desea dar más importancia a tener la menor cantidad de demanda sin cubrir, debemos tomar valores para la componente F_0 mayores que 300. Sin embargo, si se quiere una cantidad menor de exceso de oferta, tomaremos valores menores de 100 en la variable F_0.

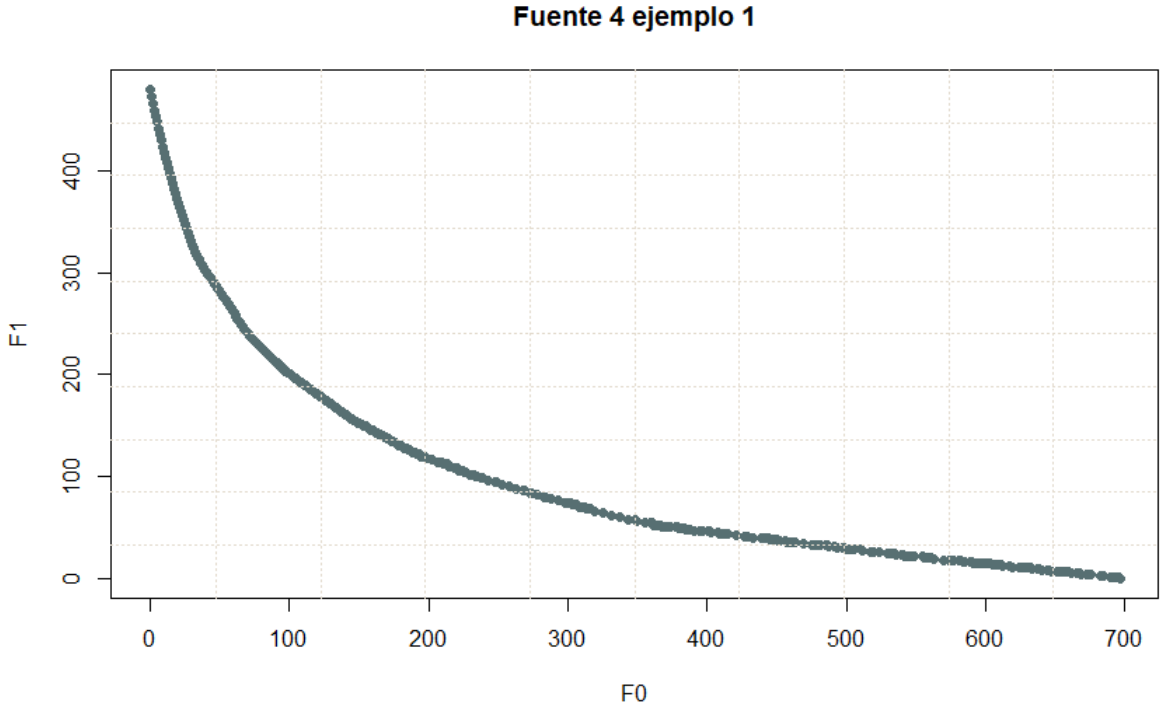

Figura 5: Resultados del método de restricciones ε en la instancia 1 de la fuente de datos 4.

La Figura 5 se corresponde con el primer y único ejemplo real de todas las fuentes de datos. Al igual que en las anteriores figuras, la gráfica obtenida es decreciente. En la gráfica también se produce un gran acúmulo de datos. Los valores de F_0 para los cuales obtedríamos un equilibrio entre las componentes F_0 y F_1, oscilan entre 180 y 290.

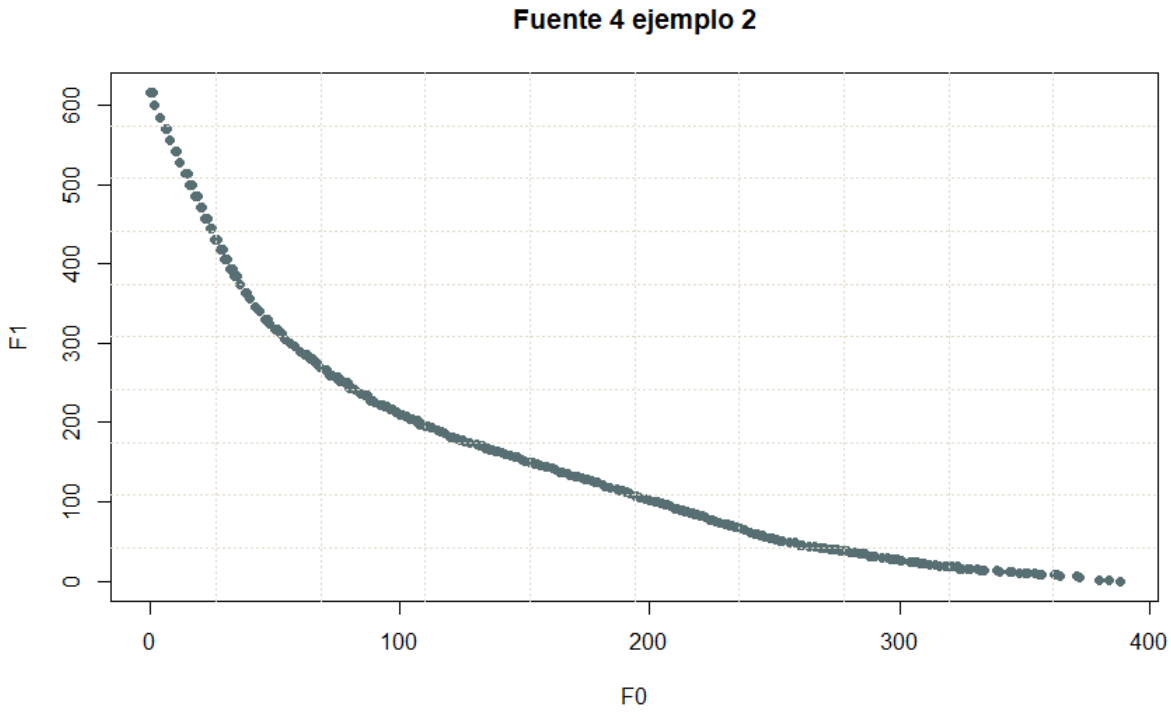

Figura 6: Resultados del método de restricciones ε en la instancia 4 de la fuente de datos 2.

En la Figura 6 se puede observar de nuevo un decrecimiento que parece exponencial. Si se desea tomar un valor que equilibre las dos componentes, F_0 y F_1, éste podría oscilar entre los valores 100 y 200 para F_0.

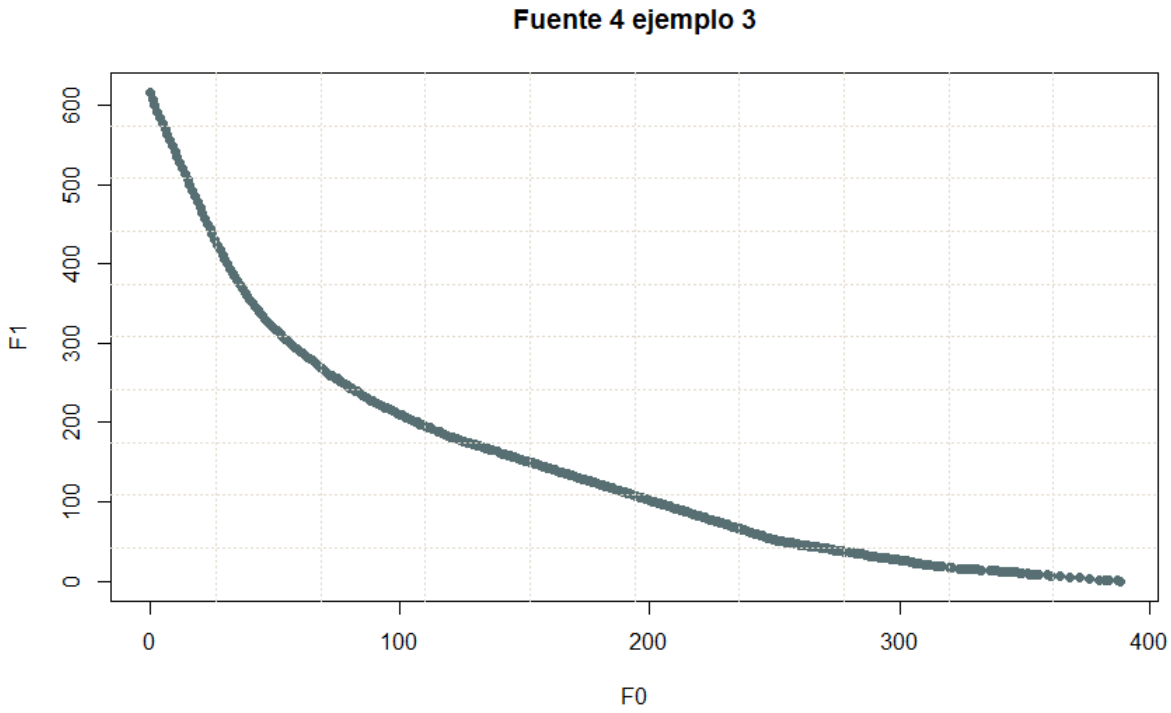

Figura 7: Resultados del método de restricciones ε en la instancia 3 de la fuente de datos 4.

La Figura 7 es la última representación de las funciones objetivo del método de restricciones ε. Podemos observar que se vuelve a producir un importante decrecimiento, aunque en este caso, la gráfica está muy poblada de datos y no es posible destacar un punto de mayor densidad. Por último, si se quiere obtener un cierto equilibrio entre ambas componentes, éste podría obtenerse para valores de F_0 comprendidos entre 100 y 200.

5.3. Comparativa con otros solvers

Hemos realizado una comparativa cuando los resultados se obtienen usando también los dos solvers: Gurobi y Cplex (véase Kocabas (6)). Para ello, hemos construido distintas tablas (Tablas 6, 7, 8 y 9) correspondientes a los resultados obtenidos con cada solver y usando las diferentes fuentes de datos consideradas a lo largo del trabajo.

En estas tablas, las columnas contienen la siguiente información:

- **1ª columna (Nº)**: indica el número de instancia de la fuente de datos.
- **2ª columna (Tiempo Xpress)**: indica el tiempo de ejecución con el solver Xpress Optimizer.
- **3ª columna (Tiempo Cplex)**: indica el tiempo de ejecución con el solver Cplex.
- **4ª columna (Tiempo Gurobi)**: indica el tiempo de ejecución con el solver Gurobi.
- **5ª columna (Optimo Xpress)**: indica el valor de la función objetivo proporcionado por Xpress Optimizer.
- **6ª columna (Optimo Cplex)**: indica el valor de la función objetivo que proporciona Cplex.
- **7ª columna (Optimo Gurobi)**: indica el valor de la función objetivo con el solver Gurobi.

Nº	**Tiempo Xpress**	**Tiempo Cplex**	**Tiempo Gurobi**	**Optimo Xpress**	**Optimo Cplex**	**Optimo Gurobi**
1	0,141	0	0	480	480	480
2	2,687	14	75	300	300	300
3	0,141	0	1	600	600	600
4	8,57	14	169	450	450	450
5	0,141	0	1	480	480	480
6	0,125	0	0	420	420	420
7	0,172	0	8	270	270	270
10	0,624	1	17	330	330	330
15	0,032	0	0	180	180	180
17	12,461	29	1334	540	540	540
18	0,234	0	8	720	720	720
20	0,063	0	2	540	540	540
24	0,078	0	4	480	480	480
25	14,372	43	234	480	480	480
26	0,875	1	27	600	600	600
27	0,063	0	2	480	480	480
28	0,547	0	14	270	270	270
29	0,531	1	27	360	360	360

Tabla 6: Fuente de datos 1. Tiempos de cada solver.

Nº	**Tiempo Xpress**	**Tiempo Cplex**	**Tiempo Gurobi**	**Optimo Xpress**	**Optimo Cplex**	**Optimo Gurobi**
1	0,547	4	7	720	720	720
2	0,141	3	4	720	720	720
3	1,047	11	19	360	360	360
4	0,187	4	7	360	360	360
5	0,25	4	6	720	720	720
6	0,297	3	4	360	360	360
7	0,296	1	3	720	720	720
9	0,188	10	11	360	360	360
10	0,499	9	16	660	660	660
11	18,039	465	1947	480	480	480
12	1,5	41	9	900	900	900
13	2,218	29	5	900	900	900
14	0,766	9	3	840	840	840
15	10,842	318	528	480	480	480
17	0,859	24	2	960	960	960
18	1,156	28	14	840	840	840
20	0,343	11	1	960	960	960
21	26,008	101	79	600	600	600
22	1,047	20	20	1080	1080	1080
24	22,486	82	1300	600	600	600
25	6,145	39	118	600	600	600
26	1,156	10	29	1020	1020	1020
29	1,39	30	16	1140	1140	1140
30	5,827	28	53	1020	1020	1020

Tabla 7: Fuente de datos 2. Tiempos de cada solver.

Nº	**Tiempo Xpress**	**Tiempo Cplex**	**Tiempo Gurobi**	**Optimo Xpress**	**Optimo Cplex**	**Optimo Gurobi**
2	12,783	33	26	7590	7590	7590
3	2,75	19	23	9540	9540	9540
4	57,05	51	4698	6540	6540	6540
5	1,297	4	15	9720	9720	9720
8	0,906	1	24	8580	8580	8580
10	1,797	30	48	2940	2940	2940
11	59,802	199	3731	5190	5190	5190
14	2,218	61	61	9600	9600	9600
15	25,39	166	770	11250	11250	11250
16	2,656	49	18	10620	10620	10620
18	59,623	166	851	6540	6540	6540
20	25,285	21	> 7200	8910	8910	-
22	59,218	31	>7200	12630	12600	-
23	5,921	7	23	8280	8280	8280
24	0,656	5	5	10260	10260	10260
25	1,218	4	12	13020	13020	**600**
26	60,005	145	>7200	12780	12780	-
27	0,859	3	8	10020	10020	10020
28	6,39	7	34	10440	10440	10440
29	60,036	295	>7200	6510	6510	-
30	1,016	4	14	13320	13320	13320

Tabla 8: Fuente de datos 3. Tiempos de cada solver.

Nº	**Tiempo Xpress**	**Tiempo Cplex**	**Tiempo Gurobi**	**Optimo Xpress**	**Optimo Cplex**	**Optimo Gurobi**
1	0,844	1	1	18420	18420	18420
2	19,58	90	170	9720	9720	9720
3	1,812	18	28	18780	18780	18780

Tabla 9: Fuente de datos 4. Tiempos de cada solver.

Es importante hacer notar que mientras que los resultados procedentes del solver Xpress Optimizer se han realizado utilizando un ordenador LAPTOP-24V5G9GT, con un procesador i5-8250U @ 1.6GHz de 4 núcleos, con sistema operativo Windows 10 Home y 8 GB the memoria RAM, para los resultados procedentes de los solvers Cplex y Gurobi se ha utilizado un ordenador ligeramente inferior, con un procesador i5 @ 2.4GHz y 8 GB de RAM. A pesar de estas diferencias de hardware, creemos que se pueden realizar comparaciones válidas entre estos tres solvers.

Se puede apreciar que exceptuando los tiempos con valor 0 en algunas instancias de la primera fuente de datos, el solver Xpress es claramente superior a la hora de resolver estos problemas utilizando un tiempo de ejecución inferior al de los otros dos solvers (ver Tabla 6), con la única excepción en el tiempo de ejecución del solver Cplex con la instancia 22 de la fuente de datos 3 (ver Tabla **??**). En los ejemplos donde Xpress se acerca al límite que impusimos en el tiempo de ejecución (1 minuto), los otros dos solvers (con límite temporal de 2 horas), precisan de un tiempo cercano a los 10 minutos o incluso llegando hasta su límite de ejecución.

Pese a la diferente potencia de los equipos utilizados, siendo inferior el ordenador en el que hemos implementado Xpress, éste último ha obtenido mejores resultados poniendo de manifiesto su superioridad frente a las otras. Por último, podemos decir, que entre los dos solvers utilizados en el mismo ordenador (Cplex y Gurobi), Cplex es claramente superior en la gran mayoría de instancias de las fuentes de datos y sólo es superada por Gurobi en un total 10 instancias.

Por todo lo anterior, sería el método Xpress el utilizado en primer lugar, seguido Cplex y por último Gurobi.

6. Conclusiones y futuras líneas de trabajo

6.1. Conclusiones

Desde nuestro punto de vista, este trabajo se ha llevado a cabo de forma satisfactoria, ya que se han cumplido los objetivos que nos planteamos inicialmente.

Hemos distinguido y descrito, de forma general, los distintos problemas de turnos para posteriormente centrar nuestro trabajo en el estudio exhaustivo del denominado problema de diseño de turnos.

Posteriormente, hemos explicado en qué consiste el problema de diseño de turnos, así como los principales métodos utilizados para su resolución, centrándonos en la programación lineal entera y en el método de restricciones ε biobjetivo. Hemos proporcionado la formulación de estos dos métodos y hemos descrito los algoritmos que hemos utilizado para su posterior implementación en el solver Xpress Optimizer.

Por último, hemos analizado los resultados de los métodos de restricciones ε biobjetivo y de programación lineal entera. Además, observando las tablas comparativas entre los solvers, Xpress Optimizer, Cplex y Gurobi, podemos afirmar que hay una clara superioridad del solver Xpress sobre los otros dos.

En cuanto a los objetivos personales que me planteé al inicio del proyecto, puedo afirmar que éstos han superado mis espectativas, mostrándome la gran importancia de las técnicas de Investigación Operativa en problemas tan actuales como el analizado en este trabajo.

6.2. Futuras líneas de trabajo

Entre las futuras líneas de trabajo que contemplamos podemos citar las siguientes:

- Realizar un estudio más profundo de los métodos heurísticos y meta–heurísticos para el problema de diseño de turnos, llevando a cabo su implementación y realizando una comparativa con los métodos analizados en este trabajo.

- La utilización de otros solvers para el problema de diseño de turnos, además de los tres utilizados en este trabajo, y proceder a su comparación con las implementadas en este proyecto.

- Llevar a cabo un estudio paralelo al realizado en este trabajo, pero con el diseño de descansos y el diseño de labores.

- Utilizar los métodos descritos en este proyecto en el estudio de un caso real.

Bibliografía

[1] A. Beer, J. Gärtner, N. Musliu, W. Schafhauser. *An Ai–based break scheduling system for supervisory personnel*. Intelligent Systems, IEEE **25** (2), 60–73, 2010.

[2] S. E. Bechtold and M. J. Showalter. A methodology form labor scheduling in a service operating system. Decision Science **18**, 89–107, 1987.

[3] D. B. Dantzig. A comment on Eddie's traffic delays at toll booths. Operation Rechearch, **2** (3), 339–341, 1954.

[4] J. Gärtner, N. Musliu and W. Slany. Rota: A research project on algorithms for workforce scheduling and shift design optimization. A. I. Commmun **14** (2), 83–92, 2001.

[5] A. I. Z. Jarrah, J. F. Bard, A. H. deSilva. *Solving large–scale tour scheduling problems*. Management Science **4** (9), 1124–1144, 1994.

[6] D. Kocabas. Exact methods for shift design and break scheduling. PhD thesis, Vienna University of Technology, 2015.

[7] V. A. Mabert, C. A. Watts. *A simulation analysis of tour–shift constructions procedures*. Management Science **28** (5), 520–532, 1982.

[8] N. Musliu. Intelligent methods for workforce schedulling: New ideas and practical applications. PhD thesis, Vienna University of Thecnology, 2001.

[9] N. Musliu, A. Schaerf, W.Slany. Local search for shift design. *European Journal of Operational Rechearch*, **153(1)**, 51–64, (2004).

[10] G. M. Thompson. Assigning telephone operators to shifts at new brunswick telephone company. Interfaces **27**, 1–11, 1997.

[11] J. Sáez–Aguado, P. C. Trandafir. *Variants of the ε–constraint method for biobjective integer programming problems: application to p–median–cover problems*. Mathematical Methods of Operations Research **87**, 251–283, 2018.

7. ANEXO 1: Código de lectura de datos

```
forward function calcular_periodo(ch:string):integer

declarations
  vacio :string
  SlotLength :integer  !longitud de los intervalos
  daysPerCycle :integer  !numero de días
  number :integer
  n :integer !numero de franjas horarias
  m :integer    !numero de posibles turnos de trabajo cada franja
  !horaria
  ciclico :boolean   !muestra si es cíclico o no

  archivo_datos = "Datos/RandomExample1.txt"
  Objective:linctr
end-declarations
fopen(archivo_datos, F_INPUT)

readln(vacio)  !las lineas leidas por vacio no se utilizan.
readln(SlotLength)

readln(vacio)
readln(vacio)
readln(daysPerCycle)


n := integer((24*daysPerCycle*60)/SlotLength)
declarations
tiempo = 1..n
x :array(tiempo)of integer  !demanda de empleados
end-declarations

readln(vacio)

readln(vacio)
readln(vacio)
if(vacio="yes")then ciclico := true
else
ciclico := false
```

```
end-if
!writeln("ciclico: ", ciclico)


readln(vacio)

readln(vacio)

final:=1
while(final <= n )do
read(x(final))
final:= final+1
end-do

while(TRUE)do
readln(vacio)
if(vacio = "#Number")then
readln(number)
break
end-if
end-do

declarations
tipoturno = 1..number
indice = 1..6
turnos = 1..m
letra, nombre, fechaInicio, ch :array(tipoturno) of string
masMinutos, menosMinutos, longitud, masLongitud,
menosLongitud:array(tipoturno) of integer
turnos_minimos, turnos_maximos: integer
horas_semanales : real
W1, W2, W3, W4 : integer
inicio_turno, duracion_turno, tipo_turno:array(range)of integer
a :array(tiempo, turnos) of integer   !0 si el tiempo n no pertenece
!al turno m y 1 si sí
end-declarations

while(TRUE)do
readln(vacio)
if(vacio = "#Abbr,")then
```

```
final2:=1
while(final2 <= number)do
read(letra(final2), nombre(final2), fechaInicio(final2),
masMinutos(final2), menosMinutos(final2), longitud(final2),
masLongitud(final2), menosLongitud(final2), ch(final2))

!writeln(letra(final2), " ",nombre(final2), " ",fechaInicio(final2), "
!masMinutos(final2), " ",menosMinutos(final2)," ",longitud(final2),"
!",masLongitud(final2)," ",menosLongitud(final2))
final2 := final2+1
end-do
break
end-if
end-do

while(TRUE)do
readln(vacio)
if(vacio = "#Duties")then
readln(turnos_minimos, turnos_maximos)
break
end-if
end-do


while(TRUE)do
readln(vacio)
if(vacio = "#")then
readln(horas_semanales)
break
end-if
end-do

while(TRUE)do
readln(vacio)
if(vacio = "#UnderCover:")then
readln(W1)
break
end-if
end-do
```

```
while(TRUE)do
readln(vacio)
if(vacio = "#Overcover:")then
readln(W2)
break
end-if
end-do

while(TRUE)do
readln(vacio)
if(vacio = "#DutiesPerWeek:")then
readln(W4)
break
end-if
end-do

while(TRUE)do
readln(vacio)
if(vacio = "#Shifts:")then
readln(W3)
break
end-if
end-do


fclose(F_INPUT)


writeln("\nW1 = ",W1)
writeln("\nW2 = ",W2)
writeln("\nW3 = ",W3)
writeln("\nW4 = ",W4)

m:=0
forall(k in tipoturno)do
hora:=fechaInicio(k) ! hora
perbase:=calcular_periodo(hora) ! per?odo base
durbase:=longitud(k) ! duracion base
forall(i in perbase-round(menosMinutos(k)/SlotLength)..perbase
+round(masMinutos(k)/SlotLength))do
```

```
forall(j in durbase-menosLongitud(k)..durbase+masLongitud(k)|j
mod SlotLength = 0)do
m:=m+1
inicio_turno(m):=i
duracion_turno(m):=round(j/SlotLength)
tipo_turno(m):=tipoturno(k)
end-do
end-do
end-do
writeln("\nNumero de turnos: ",m)

declarations
turnos = 1..m
a:array(tiempo, turnos)of integer
diaturno:array(tiempo, turnos)of integer
nextday:array(turnos)of integer

t1:integer
end-declarations


forall(s in turnos)do
if(ceil(daysPerCycle*(inicio_turno(s)+duracion_turno(s))/n) >
ceil(daysPerCycle*inicio_turno(s)/n))then
nextday(s):=1
else
nextday(s):=0
end-if
end-do


forall(s in turnos)do
forall(k in 1..daysPerCycle)do
  forall(i in 1..duracion_turno(s))do
      t1:=(inicio_turno(s)+(k-1)*round(24*60/SlotLength)+i)
      if(t1>n)then
      t1:=(t1 mod n)
      end-if
      a(t1, s):=1
```

```
  end-do
end-do
end-do


forall(s in turnos,t in tiempo)do
if((t-1) mod round(24*60/SlotLength)< round(12*60/SlotLength)
and nextday(s)=1)then
diaturno(t, s):=(ceil(daysPerCycle*(t)/n)+daysPerCycle-1 )
mod daysPerCycle
else
diaturno(t, s):=ceil(daysPerCycle*(t)/n)
end-if
end-do

forall(s in turnos,t in tiempo|nextday(s)=1 and (t-1) <
round(24*60/SlotLength) and (t-1) mod
round(24*60/SlotLength)<12*60/SlotLength )

diaturno(t, s):=daysPerCycle


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!FUNCIONES!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
function calcular_periodo(ch:string):integer
declarations
per:integer
end-declarations

per:= round((60*parseint(ch,1)+parseint(ch,4))/SlotLength)
returned:=per
end-function
```

8. ANEXO 2: Código de Programación entera

```
!!MODELO
declarations
dias = 1..daysPerCycle

b: array(turnos) of mpvar  !indica si el turno está activo
l: array(tiempo) of mpvar  !número de trabajadores en tiempo t

ex: array(tiempo) of mpvar !exceso de trabajadores
sh: array(tiempo) of mpvar !falta de trabajadores
w: array(turnos, dias) of mpvar  !empleados asignados al turno s
!el dia d

F1: mpvar
F2: mpvar
F3: mpvar
M : integer

objetivo:linctr
end-declarations

M:=max(j in tiempo)x(j)

forall(i in tiempo) rl(i):=
l(i) = sum(s in turnos)a(i, s)*w(s, diaturno(i, s))

forall(t in tiempo)rex(t):= ex(t)>=(l(t)-x(t))*SlotLength

forall(t in tiempo)rsh(t):= sh(t)>=(x(t)-l(t))*SlotLength

rF1:= F1=sum(t in tiempo)ex(t)

rF2:= F2= sum(t in tiempo)sh(t)

rF3:= F3=sum(s in turnos)b(s)

(!forall(i in tiempo) restricciones_temporales(i) :=
sum(s in turnos)a(i, s)*w(s, diaturno(i, s))-ex(i)+sh(i) = x(i)!)
```

```
forall(s in turnos,d in dias)w(s,d)<=M*b(s)

forall(s in turnos, d in dias)w(s,d)is_integer

forall(t in tiempo)sh(t)is_integer

F1 is_integer

F2 is_integer

F3 is_integer

!rF3Max := F3 <= 20

forall(t in tiempo)ex(t)is_integer

forall(s in turnos)b(s)is_binary

    !!!!!!!!!!Solucion
objetivo := W1*F1 + W2*F2 + W3*F3
setparam("XPRS_MAXTIME", 60)
tiempo_inicial := gettime

minimize(objetivo)
writeln("duracion del algoritmo: ", gettime-tiempo_inicial)
writeln("")
writeln("valor optimo = ",integer(getobjval))


writeln("valor objetivo F1 = ",F1.sol)
writeln("valor objetivo F2 = ",F2.sol)
writeln("valor objetivo F3 = ",F3.sol)

writeln(F1.sol, " ", F2.sol, " ",F3.sol, " ",getobjval)
    if (getobjval<>0)then
    writeln("Hueco final = ", strfmt(100*(getobjval
    -getparam("XPRS_bestbound"))/getobjval, 6, 2), "%")
    end-if
writeln("\n\nvalores x demandados:")
forall(i in tiempo)write((x(i)), "\t")
```

```
writeln("")
writeln("excesos")
forall(i in tiempo)write((ex(i).sol), "\t")
writeln("")
writeln("defectos")
forall(i in tiempo)write((sh(i).sol), "\t")


writeln("")

exportprob(EP_MIN,"shift5",objetivo)

writeln("valores obtenidos")
forall(i in tiempo, j in turnos)do
if( w(j, diaturno(i, j)).sol >= 1 )then
writeln("tiempo: ", i, " turno: ",  j, " valor: ",
w(j, diaturno(i, j)).sol)
end-if
end-do

writeln("")

writeln("\n\n\nTurno\tInicio\tDurac\tNextday\tTipo")
forall(s in turnos)do
forall (t in tiempo)do
if( w(s, diaturno(t, s)).sol >= 1 )then
writeln(s,"\t",inicio_turno(s),"\t",duracion_turno(s),"\t",nextday(s),
"\t",tipo_turno(s))
break
end-if
end-do
end-do

writeln("\nNumero de trabajadores cada dia:\n")
forall(s in turnos,d in dias|w(s,d).sol>=0.9)do
write("\nturno ",s,", dia ",d," -> ",w(s,d).sol," trabajadores")
end-do
```

9. ANEXO 3: Código de Programación método ε

```
!!MODELO
declarations
dias = 1..daysPerCycle

b: array(turnos) of mpvar  !indica si el turno está activo
l: array(tiempo) of mpvar  !número de trabajadores en tiempo t

ex: array(tiempo) of mpvar !exceso de trabajadores
sh: array(tiempo) of mpvar !falta de trabajadores
w: array(turnos, dias) of mpvar  !empleados asignados al turno s
!el dia d

F1: mpvar
F2: mpvar
F3: mpvar
M : integer

objetivo:linctr
end-declarations

valor_f1:=10000000
while(valor_f1 >= 0)do

M:=max(j in tiempo)x(j)

forall(i in tiempo) rl(i):=
l(i) = sum(s in turnos)a(i, s)*w(s, diaturno(i, s))

forall(t in tiempo)rex(t):= ex(t)>=(l(t)-x(t))*SlotLength

forall(t in tiempo)rsh(t):= sh(t)>=(x(t)-l(t))*SlotLength

rF1:= F1=sum(t in tiempo)ex(t)

F1 <= valor_f1

rF2:= F2= sum(t in tiempo)sh(t)
```

```
rF3:= F3=sum(s in turnos)b(s)

(!forall(i in tiempo) restricciones_temporales(i) :=
sum(s in turnos)a(i, s)*w(s, diaturno(i, s))-ex(i)+sh(i) = x(i)!)

forall(s in turnos,d in dias)w(s,d)<=M*b(s)

forall(s in turnos, d in dias)w(s,d)is_integer

forall(t in tiempo)sh(t)is_integer

F1 is_integer

F2 is_integer

F3 is_integer

rF3Max := F3 <= 20

forall(t in tiempo)ex(t)is_integer

forall(s in turnos)b(s)is_binary


objetivo := F2
setparam("XPRS_MAXTIME", 60)-
tiempo_inicial := gettime

minimize(objetivo)

writeln(F1.sol, " ", F2.sol, " ",F3.sol, " ",getobjval, " ",
gettime-tiempo_inicial)

valor_f1:= integer(F1.sol) - SlotLength
writeln("")
end-do
```