Algoritmo GRASP para el problema de Programación de Tripulaciones de buses urbanos

David Pardo Peña

Universidad de los Andes
Cra 1 Nº 18A- 12, 111711, Bogotá, Colombia
d.pardop@uniandes.edu.co

Resumen

Este documento plantea un algoritmo tipo GRASP para la resolución del *Bus Driver Scheduling Problem* (BDSP) con el fin de encontrar buenas soluciones en tiempos razonables. Todo esto dentro de un contexto de la operación de un sistema de transporte masivo, donde se busca minimizar el costo de operación, y a su vez mejorar el ambiente laboral al tener en cuenta la satisfacción de los conductores con los turnos asignados. En el BDSP se tienen conductores encargados de cubrir la demanda de turnos, con una asignación que contiene varias restricciones como el mínimo y máximo de bloques de trabajo, mínimo de días de descanso y secuencias de turnos que no pueden ser asignados. Se adaptaron instancias clásicas de la literatura para el problema de asignación de turnos añadiendo la variable satisfacción y se resolvió el problema para una empresa operadora de vehículos articulados y alimentadores. Finalmente, la función de satisfacción dio un valor agregado a las soluciones mejorando sustancialmente el ambiente laboral y los resultados obtenidos fueron favorables en términos de tiempo y calidad de solución.

1 Introducción

El transporte público de pasajeros es determinante en la forma de vida de la cuidad. Un buen sistema de transporte proporciona confort en la movilidad de las personas y se convierte en una de las principales medidas de calidad de vida. En este sentido, el transporte público de pasajeros tiene un papel fundamental a la hora de tomar decisiones desde el punto de vista social, económico y ambiental. En línea con lo anterior, la planeación de la red de transporte es sumamente importante tanto para prestar un servicio de calidad como para el operador de transporte y los costos que asume. La planeación es un proceso complejo y de difícil solución por lo que se divide en tres fases que abarcan decisiones estratégicas, tácticas, y operacionales. En las decisiones estratégicas tenemos el diseño de la red de transporte, en las decisiones tácticas el conjunto de frecuencias y horario de la red de transporte y en las decisiones operacionales el problema de programación de vehículos, programación de turnos de trabajo y asignación de conductores a turnos. Sin embargo, cada uno de los problemas asociados a cada etapa de la planeación del transporte público de pasajeros ha sido ampliamente estudiado en la literatura debido a que son problemas complejos, matemática y computacionalmente se catalogan como problemas de tipo NP-duro, es decir, que no existe método o algoritmo que sea capaz de resolver el problema de manera general en tiempos de cómputo razonables. Teniendo en cuenta esto se debe resolver cada problema por separado y secuencialmente.

En este trabajo, se propone entonces resolver la etapa correspondiente a la planeación operacional, específicamente el problema de Asignación de Conductores a Turnos de Trabajo. Solucionar este problema se sustenta en que uno de los principales costos que asumen las compañías de transporte público de pasajeros son los costos operativos relacionados a la nómina de sus conductores. Con esto en mente, este artículo busca presentar una alternativa de solución del BDSP dentro del contexto de operadores de vehículos articulados y alimentadores de la empresa INTEGRA S.A. y comprobar la efectividad de la metodología propuesta mediante la adaptación de las instancias de Musliu (2006) que tratan un problema similar de asignación de turnos de enfermeras. El problema de INTEGRA S.A. consiste en un conjunto de turnos con cierta demanda que deben ser asignados a un conjunto de conductores. Los conductores tienen un mínimo y máximo de días de trabajo a la semana y un mínimo de días de descanso. Además, existen diferentes tipos de turnos con distintos horarios, por lo que hay restricciones donde se debe revisar los turnos que pueden ser asignados dependiendo del turno que se le fue asignado el día anterior. Actualmente se cuenta con un modelo de optimización propuesto por Cárdenas (2019) que trabaja el

mismo problema de asignación de turnos para la empresa INTEGRA S.A. y las instancias de Musliu (2006) por lo que fue de interés compararlo con la metodología propuesta. Se observo que el modelo de optimización al no tener en cuenta la satisfacción de los conductores muestra en general soluciones factibles para el problema, pero genera soluciones con baja satisfacción que a largo plazo puede implicar problemas con el rendimiento y permanencia de los trabajadores. La alternativa presentada obtuvo rendimiento similar en velocidad, y mayor rendimiento en cuanto a soluciones factibles encontradas. Por lo que se concluye que la metodología propuesta da un valor agregado a la solución del problema de asignación de conductores en términos de satisfacción del personal, generando soluciones de igual calidad al modelo de optimización en un tiempo similar.

2 Descripción del problema

El problema consiste en generar un horario cíclico, es decir, que se repita en un ciclo determinado de tiempo para un grupo de trabajadores, cumpliendo con una matriz de requisitos y otras restricciones. Entre las restricciones que se deben cumplir se encuentran la disponibilidad de los empleados para trabajar en los horarios programados, la cantidad de horas trabajadas, la cantidad de días libres, y la cantidad de turnos consecutivos. También se deben considerar restricciones específicas, como la prohibición de ciertas asignaciones de turnos de manera consecutiva. El objetivo es encontrar una solución óptima o cercana a la óptima que cumpla con todas las restricciones y que sea cíclica, lo que permitirá planificar los horarios de los trabajadores de manera eficiente y efectiva. Debido a la naturaleza combinatoria del BDSP un método exacto para instancias de gran porte (instancias de tamaño real) puede tener un alto costo computacional. Por tal motivo, la principal contribución de este artículo es encontrar estrategias de asignación de turnos que permitan obtener soluciones factibles en un tiempo razonable mientras se maximiza la satisfacción de los conductores mediante un algoritmo de optimización aproximado, de modo que el problema se pueda aplicar o adaptar a cualquier situación real

2.1 Trabajo relacionado

La mayoría de los modelos utilizados para la solución de *rostering* se basan en el modelo de *Set Covering* propuesto por Dantzig (1954) y sus variaciones. Adicionalmente se encuentra mucha relación con el modelo que proponen Martello and Toth (1992) de Asignación Generalizada GAP (*Generalized Assignement Problem*) donde se asignan n ítems a m unidades, de tal manera que el total de recursos disponibles no se excede y la sumatoria de las penalidades relacionadas a la asignación sea mínima. La formulación va muy de la mano, pero para una situación real tocaría adaptarlo a las restricciones de la empresa y se añadirían otras restricciones adicionales al modelo.

$$(GAP) = Min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \tag{1}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad \forall i = 1, ..., m \tag{2}$$

$$\sum_{i=1}^{n} a_{ij} x_{ij} \leq b_j \qquad \forall j = 1, ...n \tag{3}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i = 1, ..., m, \forall j = 1, ...n \tag{4}$$

En el modelo Martello y Toth (1992) la función objetivo es de minimización y relaciona la variable de decisión binaria x_{ij} (que toma el valor de 1 cuando el elemento i es asignado a una unidad j) con una matriz de costos c_{ij} para reducir los costos asociados a la asignación. La Ecuación (2) hace referencia a que solo pueda ser asignado un elemento i a una unidad j y que todos los elementos sean asignados. La Ecuación (3) asocia una matriz a_{ij} donde se representa la cantidad de recurso utilizado y la lógica es que los recursos gastados no sean mayores a los recursos disponibles de cada unidad.

$$\text{Min } Z = \sum_{j \in S} c_j * x_j \tag{5}$$

$$s.a.$$

$$\sum_{j \in S} a_{ij} * x_j \leq 1 \qquad \forall i = 1 \dots m \tag{6}$$

Podemos observar el modelo general de *set covering* de Dantzig (1954) que tiene como objetivo cubrir todas las filas de la matriz a_{ij} usando el mismo costo de subconjuntos de las columnas. La Ecuación (5) es la función objetivo donde se minimiza el costo asociado a las columnas seleccionadas y la Ecuación (6) nos permite acotar la cobertura de cada una de las filas al menos a una columna.

En este caso nos guiamos de la formulación propuesta por Cárdenas (2019) para el mismo problema de asignación de conductores en la empresa INTEGRA S.A. y solución a una selección instancias de Musliu (2006) con restricciones similares al problema específico de la empresa, pero en un contexto de asignación de turnos de enfermeras. En este caso se plantea una formulación general del modelo para las instancias de Musliu y se adaptó la información de la empresa para tener la misma estructura de estas instancias. A continuación, se presenta el modelo propuesto donde se formula matemáticamente el modelo de *rostering* como un modelo lineal entero con una variable de decisión de tres subíndices y una auxiliar de dos subíndices. Cárdenas (2019) propone esta formulación para el modelo general, pero puede ser modificado añadiendo conjuntos de restricciones que sean necesarios para representar diferentes problemas:

2.1.1 Definición de variables

- i: Índice de trabajadores disponibles, $i = 1,2, \dots, I$.
- j: Índice de días de horizonte de roster, $j = 1,2, \dots, J$.
- k: Índice de tipos de turnos, $k = 1,2, \dots, K$.
- x_{ijk}: Variable de decisión que toma el valor de 1 si el trabajador i es asignado al día j en el turno k y 0 de lo contrario.
- y_{ik}: Variable auxiliar que toma el valor del número de tipos de turnos k que realiza un trabajador i durante el róster.

2.1.2 Definición de parámetros

- P_S: Número de turnos que pueden ser asignados a un trabajador
- D_k: Duración del turno k
- JO: Tiempo máximo permitido que puede ser asignado a un trabajador en un róster
- L: Máximo número de días que pueden ser asignados a un trabajador en un horizonte de tiempo
- Req_{jk}: Número de trabajadores requeridos en un día j en un turno k
- Min_i: Mínimo de días que deben de ser asignados a un trabajador i en un horizonte de tiempo
- c_k: Costo asociado al tipo de turno k

2.1.3 Modelo matemático General propuesto

$$\text{Min } Z = \sum_{i=1}^{I}\sum_{k=1}^{K} c_k * y_{ik} \tag{7}$$

$s.a.$

$$\sum_{k=1}^{K} x_{ijk} \leq Ps \qquad \forall i,j \tag{8}$$

$$\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk} * D_k \leq JO \qquad \forall i \tag{9}$$

$$\sum_{j=1}^{J}\sum_{k=1}^{K} x_{ijk} \leq L \qquad \forall i \tag{10}$$

$$\sum_{i=1}^{I} x_{ijk} \leq Req_{jk} \qquad \forall j,k \tag{11}$$

$$\sum_{j=J}^{I}\sum_{k=1}^{K} x_{ijk} \leq Min_i \qquad \forall i \tag{12}$$

$$\sum_{i=1}^{I} x_{ijk} \leq y_{ik} \qquad \forall i,k \tag{13}$$

2.2 Función objetivo

Para tener en cuenta la variable satisfacción es necesario definirla ya que puede tener distintas interpretaciones dependiendo del contexto. A través de encuestas realizadas a los conductores se encontró que se puede representar esta satisfacción cuantificando el número de turnos favorables para el conductor, debido a que muchos de ellos cuentan con otras restricciones como estudio o tiempo que le quisieron dedicar a su familia o para realizar otras actividades. Con esto en mente se definió la función objetivo en términos de los turnos que les gusta, los turnos que no les gusta, y a los que son indiferentes. Se le asigno un peso de 1 en la función objetivo a todos turnos asignados a un conductor que se encuentre a gusto con el turno y un peso de -1 si no se encuentra a gusto. Además, se hizo el supuesto que los turnos a los que el conductor es indiferente no suman ni restan en la función objetivo, pero podría ser un tema interesante de profundizar en otra investigación, que tanto impacto tienen la indiferencia en la satisfacción ya que podemos encontrar casos donde haya mucha indiferencia con poca insatisfacción y casos con mucho satisfechos, pero también muchos insatisfechos. Fijémonos en que podríamos encontrar soluciones así con el mismo nivel de satisfacción, pero no saber cuál es mejor. En este caso se calcula entonces un puntaje a partir de los satisfechos y insatisfechos como se explicó anteriormente y se divide entre el máximo numero de satisfechos que hipotéticamente podríamos llegar a tener que es el caso donde todos los turnos sean asignados a conductores a los que les gusta el turno. A continuación, se puede observar la propuesta para la función objetivo:

$$Satisfacción(\%) = \frac{Cantidad\ de\ Satisfechos - Cantidad\ de\ insatisfechos}{\#\ de\ turnos\ por\ asignar}$$

Recordemos que el problema originalmente no tiene en cuenta la satisfacción por lo que fue necesario obtener información acerca de las preferencias en el caso de la empresa y en el caso de las otras instancias se modificaron generando aleatoriamente mediante una semilla una lista de preferencias para los trabajadores.

3 Metodología propuesta

Se propone una metodología basada en GRASP con un procedimiento constructivo, un mecanismo de reparación de las soluciones y dos operadores de búsqueda local (operador INSERT y SWAP). La

idea es que el procedimiento constructivo haga una relajación de la restricción de máximos días de trabajo que se le pueden asignar al conductor para luego aplicar una estrategia de reparación y balanceo, permitiendo así la exploración de soluciones promisorias. Las soluciones construidas (reparadas y balanceadas) son mejoradas a través de una búsqueda local exhaustiva que utiliza dos diferentes vecindarios. El mecanismo de balanceo reasigna los turnos para nivel la carga de trabajo y recuperar la factibilidad de la solución en términos de la restricción relajada.

3.1 Algoritmo GRASP

El algoritmo GRASP propuesto consta de dos fases: la construcción y la búsqueda local, para esto utiliza el parámetro $itera$ que nos permite calibrar el numero de veces que se repetirán ambas fases (línea 2). Primero revisa si el parámetro $itera$ ya supero el umbral de soluciones sin reparación y si no se ha encontrado ninguna solución factible enciende el reparador del constructivo (línea 3 y 4). Luego se asignan los turnos mediante el algoritmo constructivo (línea 5). Si la asignación del constructivo es promisoria entra la a fase de mejoramiento (línea 7) donde el resultado de la asignación se válida para ver si logra mejorar la incumbente (líneas 8,9,10,11). Para ello debe revisar que la solución sea factible teniendo en cuenta el número de asignaciones hechas y el número de turnos (línea 6).

Algoritmo 1 GRASP

Parámetros: $itera$: Número total de iteraciones, $alpha$: tamaño de la lista restricta de candidatos, $umbral$: Proporción de iteraciones durante las cuales no se repara.

Input: Dict $ddia$: Diccionario con lista conductores disponibles por día, **Dict** $proh$: diccionario con lista de tipos de conductores prohibidos según el día y tipo de turno

Output: Dict *t_incumbent*: Diccionario de asignación de turnos, **List** *trabajados_incumbent*: Lista de días trabajos por conductor, **Integer** *obj_incumbent*:

1. $reparar = False$
2. **For** $i = 1$ **to** $itera$ **do**
3. **if** $i \geq itera * umbral$ *and* **then**
4. $reparar = True$
5. **Solution** $t, trabajados, obj \leftarrow constructivo(alpha, reparar)$
6. **if sum**$(trabajados) =$ **len**(t) **then**
7. **Solution** $t, trabajados, obj \leftarrow busquedaLocal(\ t, trabajados, obj)$
8. **if** $obj > obj\_incumbent$ **then**
9. $obj\_incumbent = obj$
10. $t\_incumbent = t$
11. $trabajados\_incumbent = trabajados$
12. **Return** $t\_incumbent$

3.2 Algoritmo Constructivo

El algoritmo constructivo está dividido en tres fases: una construcción pseudoaleatoria (línea 1), la reparación (línea 3) y el balanceo (línea 5). Como en la construcción se relaja la restricción de máximos días de trabajo la idea es que se haga una lista restricta de candidatos a partir del número de días que han trabajado los conductores como estrategia de acotación de la restricción y de cierta forma balancear la asignación de turnos. Con estas listas se seleccionan aleatoriamente conductores para asignarlos a los turnos revisando que se cumplan las demás restricciones. Luego está la fase de reparación que solo se realiza si se activa, faltan turnos por asignar y se han asignado al menos un porcentaje de los turnos (línea 2). En esta fase simplemente se revisan todos los conductores que pueden ser asignados a los turnos faltantes y se asigna el primero que encuentre, todo esto con el fin de que no se pierda el proceso de algunas soluciones que están cerca de ser factibles. Por último, se verifica que la solución sea factible

para pasar al balanceo (línea 4) donde se revisa la carga de trabajo y asigna turnos ya asignados a otros conductores para terminar de balancear los días de trabajo hasta llegar a una solución que además de factible cumpla con la restricción de máxima carga de trabajo. Hay que tener en cuenta que hay soluciones que pasan por el reparador y el balanceo y siguen sin ser factibles debido a que no llegan a ser aptas para balancear.

Algoritmo 2 Constructivo General

Parámetros: $alpha$: tamaño de la lista restricta de candidatos, $reparar$: Parámetro para activar el reparador

Input: **Dict** $ddia$: Diccionario con lista conductores disponibles por día, **Dict** $proh$: diccionario con lista de tipos de conductores prohibidos según el día y tipo de turno

Output: **Solution** $t, trabajados, obj$

1. $\boldsymbol{Solution}\ t, trabajados, ddia, proh \leftarrow constructivoPseudoaleatorio(alpha)$
2. $\boldsymbol{if}\ reparar = True\ \boldsymbol{and}\ alpha * t \leq \boldsymbol{sum}(trabajados) < \boldsymbol{len}(t)\ \ \boldsymbol{then}$
3. $\quad\boldsymbol{Solution}\ t, trabajados, ddia, proh \leftarrow reparar()$
4. $\boldsymbol{if}\ \boldsymbol{sum}(trabajados) = \boldsymbol{len}(t)\ \boldsymbol{then}$
5. $\quad\boldsymbol{Solution}\ t, trabajados, ddia, proh \leftarrow balancear()$

El algoritmo pseudoaleatorio revisa si se ha realizado alguna asignación cada iteración como condición de parada (línea 1). La idea es que recorra varias veces los turnos debido a que en ocasiones la elección aleatoria del conductor no es viable. Cada iteración empieza recorriendo los turnos (línea 3), se revisa si el turno ya fue asignado si tiene un valor de -1 (línea 4). Luego genera la lista restricta de candidatos a partir de la lista de conductores disponibles en el día del turno. Teniendo ya la lista restricta de candidatos inicial genera otras tres listas categorizando a cuáles conductores les gusta el turno, a cuáles les da igual y a cuáles no les gusta, luego se elige una de las tres listas como la lista restricta de candidatos (línea 5). El criterio de elección de la lista va cambiando cuando no encuentra conductores para asignar a los turnos hasta llegar a la lista de los que no les gusta (línea 11-13). Se verifica si lista contiene conductores para elegir uno aleatoriamente (línea 6 -7). Antes de asignar el conductor elegido al turno se debe revisar que no tenga prohibido el tipo de turno que se quiere asignar ese día (línea 8) para posteriormente asignarlo (línea 9) y actualizar las listas de prohibidos, disponibles por día y turnos asignados por conductor (línea 10). Cuando ya no encuentra conductores para asignar a los turnos el algoritmo para. En resumen, se determina cuál de las tres listas se debe utilizar y se seleccionan conductores aleatoriamente para luego asignarlos a los turnos.

Algoritmo 3 constructivo pseudoaleatorio

Parámetros: $alpha$: tamaño de la lista restricta de candidatos

Input: **Dict** $ddia$: Diccionario con lista conductores disponibles por día, **Dict** $proh$: diccionario con lista de tipos de conductores prohibidos según el día y tipo de turno

Output: **Solution** $t, trabajados, obj$

1. $\boldsymbol{while}\ asigno = True$
2. $\quad asigno = False$
3. $\quad\boldsymbol{For}\ tipo, dia, requerimiento\ \boldsymbol{in}\ t$
4. $\quad\quad\boldsymbol{if}\ t[tipo, dia, requerimiento] = -1\ \boldsymbol{then}$
5. $\quad\quad\quad\boldsymbol{Solution}\ lista \leftarrow RCL(tipo, ddia[dia], trabajados, alpha)[v]$
6. $\quad\quad\quad\boldsymbol{if}\ \boldsymbol{len}(lista) > 0\ \boldsymbol{Then}$

```
7.                          elegido ← random.choice(lista)
8.                          if elegido not in proh[dia, tipo] then
9.                              t[tipo, dia, requerimiento] = elegido
10.                             Update ddia, proh, trabajados, obj
11.         if asigno = False and v < 2:
12.             agrego = True
13.             v = v + 1
```

A continuación, se presenta el seudocódigo detallado del algoritmo que construye la lista restricta de candidatos para el constructivo pseudoaleatorio. Primero se encuentran los máximos y mínimos días trabajados para generar la lista restricta de candidatos general (línea 1-5). Luego se genera la lista restricta de candidatos general (línea 6-8). Por último, se hace el proceso de división en tres listas según las preferencias de los conductores (línea 9-15).

Algoritmo 4 Lista restricta de candidatos (RCL)

Parámetros: $alpha$: tamaño de la lista restricta de candidatos, $tipo$: Tipo del turno que se va a asignar, $ddia$: Diccionario con lista conductores disponibles por día, $trabajados$: Lista de turnos asignados por conductor

Input: Dict $proh$: diccionario con lista de tipos de conductores prohibidos según el día y tipo de turno **Dict** $preferencia$: diccionario con lista tupla de valores del tipo de turno que le gusta y no le gusta a cada conductor. La posición 0 contiene el tipo de turno que le gusta y la posición 1 el tipo de turno que no le gusta

Output: Solution $lista$: Lista de que contiene 3 listas con conductores aptos para asignar el turno, la primera contiene conductores a los que les gusta el turno, la segunda conductores a los que les es indiferente y la última conductores a los que no les gusta.

```
1.  For i in ddia
2.          if trabajados[i] < minimo then
3.                  minimo = trabajados[]
4.          if trabajados[i] > maximo then
5.                  maximo = trabajados[]
6.  For i in ddia
7.          if trabajados[i] ≤ math.ceil(minimo + alpha ∗ (maximo − minimo)) then
8.                  RCL.append(i)
9.  For p in RCL
10.         if  preferencias[p][0] = tipo then
11.                 lista[0].append(p)
12.         elif  preferencias[p][1] = tipo then
13.                 lista[2].append(p)
14.         else
15.                 lista[1].append(p)
```

Luego de realizar la construcción inicial se verifica que la solución sea apta y no hayan quedado turnos sin asignar. Para ello entra en la fase de reparación donde hay dos criterios indispensables para empezar con la reparación. Primero que el parámetro $reparar$ sea verdadero y segundo que la cantidad de turnos asignados sea menor al total de tunos y mayor a un porcentaje de los datos (línea 1). El porcentaje de asignaciones mínimas para reparar la solución se determinó como el mismo parámetro $alpha$ del tamaño de la RCL ya que si este disminuye se espera que más soluciones deban ser reparadas y por ello este porcentaje también. La lógica de la reparación es recorrer los conductores disponibles para el

día que se necesita el turno que no ha sido asignado y elegir el primero que reúna las condiciones en términos de que no se incumpla ninguna restricción (línea 5-7). En este caso no se tendrá en cuenta si el turno le gusta o no al conductor. Luego de asignado se actualizarán las listas de prohibidos, disponibles por día y turnos asignados por conductor (línea 8). En caso de que no encuentre ningún conductor apto se dejara de reparar la solución para evitar perder tiempo computacional en esa solución (línea 9-10).

Algoritmo 5 Reparación

Parámetros: $alpha$: tamaño de la lista restricta de candidatos, $reparar$: Parámetro para activar el reparador

Input: **Dict** $ddia$: Diccionario con lista conductores disponibles por día, **Dict** $proh$: diccionario con lista de tipos de conductores prohibidos según el día y tipo de turno

Output: **Solution** $t, trabajados, obj$

1. $\boldsymbol{if}\ alpha * \boldsymbol{len}(\boldsymbol{t}) \leq \boldsymbol{sum}(trabajados) < \boldsymbol{len}(t)\ \boldsymbol{and}\ reparar = True\ \boldsymbol{Then}$
2. $\quad \boldsymbol{For}\ tipo, dia, requerimiento\ \boldsymbol{in}\ t$
3. $\quad\quad \boldsymbol{if}\ t[tipo, dia, requerimiento] = -1\ \boldsymbol{then}$
4. $\quad\quad\quad asigno = False$
5. $\quad\quad\quad \boldsymbol{For}\ elegido\ \boldsymbol{in}\ ddia[dia]$
6. $\quad\quad\quad\quad \boldsymbol{if}\ elegido\ \boldsymbol{not\ in}\ proh[dia, tipo]\ \boldsymbol{then}$
7. $\quad\quad\quad\quad\quad t[tipo, dia, requerimiento] = elegido$
8. $\quad\quad\quad\quad\quad \boldsymbol{Update}\ ddia, proh, trabajados, obj$
9. $\quad\quad\quad \boldsymbol{if}\ asigno = False\ \boldsymbol{then}$
10. $\quad\quad\quad\quad \boldsymbol{break}$

El último paso de la construcción corresponde a la etapa del balanceo, donde se nivela la carga de trabajo para cumplir con la restricción de máximos días de trabajo que pueden ser asignados a un conductor. El algoritmo funciona de manera similar al constructivo pseudoaleatorio en el sentido de que recorre varias veces los turnos hasta que ya no se hayan hecho cambios y teniendo un criterio adicional de que todos los turnos deben estar asignados (línea 1). La lógica del algoritmo parte desde tres listas muy importantes, $falta$ contiene todos los conductores que no han llegado al máximo de carga de trabajo, $completo$ que contiene los conductores que ya tienen la máxima carga de trabajo y $excede$ que contiene los conductores que exceden el máximo de carga de trabajo. Se van recorriendo los turnos hasta que encuentre un turno que este en la lista $excede$ (línea 3-5), luego aleatoriamente se elige un conductor de la lista $falta$ (línea 6) y se verifica que sea apto para asignarlo al turno y cumpla todas las restricciones (línea 7). Luego de asignarlo (línea 8) se actualizan tanto las listas $falta, completo$ y $excede$ como las listas de prohibidos, disponibles por día y turnos asignados por conductor (línea 9-10).

Algoritmo 6 Balanceo

Input: **Dict** $ddia$: Diccionario con lista conductores disponibles por día, **Dict** $proh$: diccionario con lista de tipos de conductores prohibidos según el día y tipo de turno, **List** $falta$: Lista que contiene todos los conductores que no han llegado al máximo de carga de trabajo, **List** $completo$: Lista que contiene los conductores que ya tienen la máxima carga de trabajo, **List** $excede$: Lista que contiene los conductores que exceden el máximo de carga de trabajo

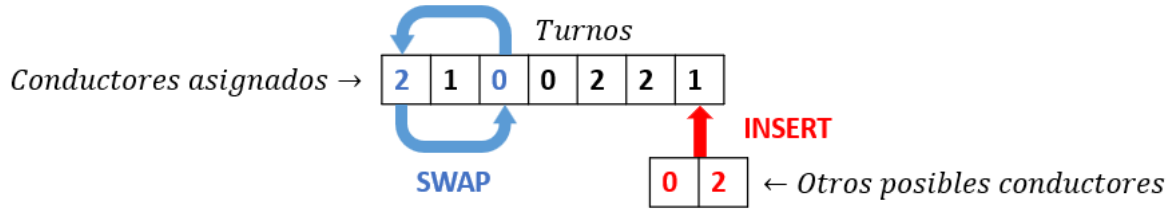
Output: **Solution** $t, trabajados, obj$

1. $\boldsymbol{While}\ cambio = True\ \boldsymbol{and}\ \boldsymbol{sum}(trabajados) = \boldsymbol{len}(t)$
2. $\quad cambio = False$
3. $\quad \boldsymbol{For}\ tipo, dia, requerimiento\ \boldsymbol{in}\ t$
4. $\quad\quad actual = t[tipo, dia, requerimiento]$
5. $\quad\quad \boldsymbol{if}\ actual\ \boldsymbol{in}\ excede\ \boldsymbol{then}$

6. $elegido = \boldsymbol{random.choice}(falta)$
7. $\boldsymbol{if}\ elegido\ \boldsymbol{in}\ ddia[dia]\boldsymbol{and}\ elegido\ not\ in\ proh[dia, tipo]\ \boldsymbol{then}$
8. $\quad t[tipo, dia, requerimiento] = elegido$
9. $\quad \boldsymbol{Update}\ ddia, proh, trabajados, obj$
10. $\quad \boldsymbol{Update}\ falta, completo, excede$

3.3 Algoritmo Búsqueda Local

Una vez ya se tiene una construcción inicial factible la idea es mejorarla a través de dos operadores (INSERT y SWAP). El primero consiste en recorrer los turnos e ir asignando diferentes conductores a los que están actualmente con el fin de mejorar la satisfacción de los conductores. El segundo operador consiste en intercambiar entre dos turnos los conductores que están asignados con el fin de ver si ambos conductores o alguno de ellos puede estar más satisfecho. A continuación, se muestra un diagrama de explicación de ambos:



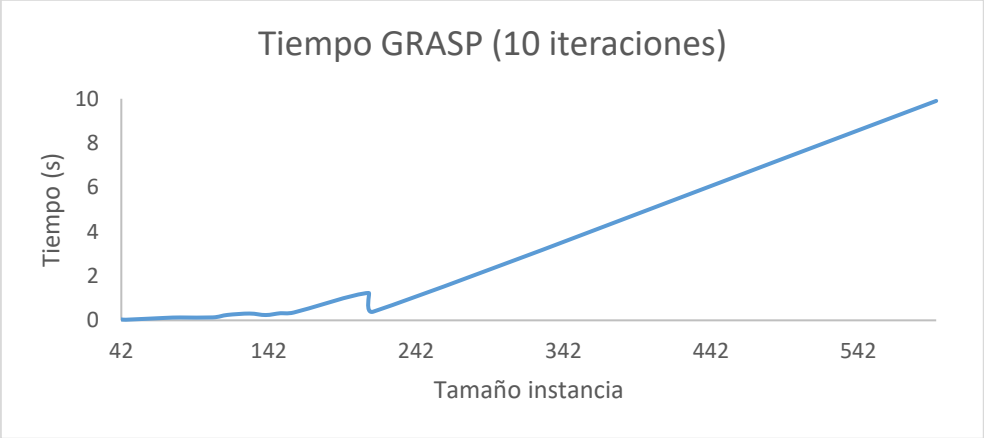
4 Resultados computacionales

Esta sección esta dividida en dos partes. En la primera se presenta la calibración de los parámetros junto con el rendimiento del algoritmo GRASP en cuanto al porcentaje de soluciones exitosas, no exitosas, reparadas y que no pudieron ser reparadas. La segunda parte consiste en comparar los resultados obtenidos con el modelo de Cárdenas (2019). El enfoque de solución fue codificado en Python 3.8.3 y fue ejecutado en un computador con un procesador Intel(R) Core (TM) i7-8750H CPU 2.20 GHz usando un sistema operativo Windows 11 y 16Gb de memoria RAM.

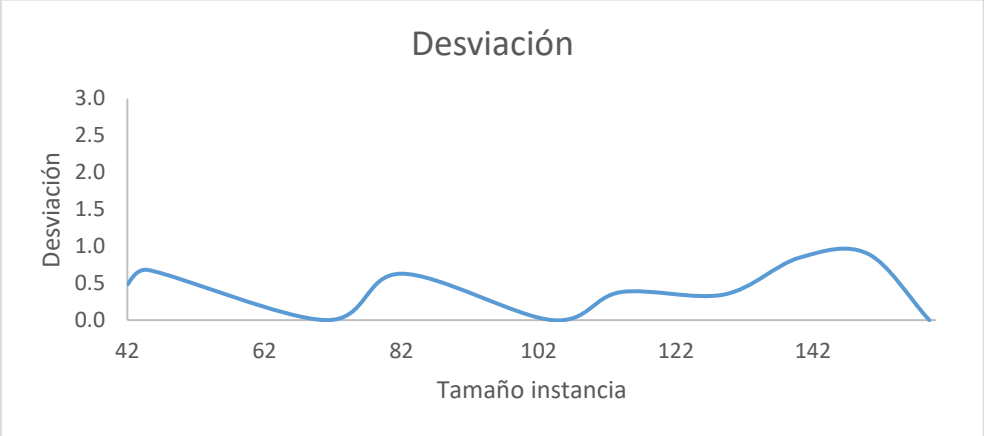
4.1 Calibración de parámetros y rendimiento de la metodología propuesta

Recordemos que los tres parámetros de calibración del algoritmo GRASP son $itera$ que es el numero de iteraciones que realizara nuestro algoritmo, $alpha$ que es el tamaño de nuestra lista restricta de candidatos y $umbral$ que es el porcentaje de iteraciones desde donde empezará a reparar soluciones en caso de no haber encontrado alguna factible. $Umbral$ se calibró bajo el supuesto de que, si bien no queremos que repare todo el tiempo por temas de tiempo computacional, en caso de requerirlo debería ser el 50% o un poco más. Para el número de iteraciones se tuvo más en cuenta el tiempo de ejecución, al incrementar el tamaño de la instancia se incrementa el esfuerzo computacional. La instancia más grande tiene un tamaño 595 turnos y se demora en promedio aproximadamente diez segundos, teniendo en cuenta que la instancia de la empresa tiene un tamaño 323 turnos, podemos observar en la gráfica que tardaría aproximadamente dos segundos en ejecución. Si bien se podría considerar aumentar el número de iteraciones para las instancias mas pequeñas no se genera un gran valor agregado ya que el rendimiento fue bastante bueno y la variación entre 10-1000 iteraciones (aumentando cada 100) fue mínima para las instancias pequeñas.

Gráfica 1 - Tiempo ejecución GRASP con 10 iteraciones vs tamaño de instancia

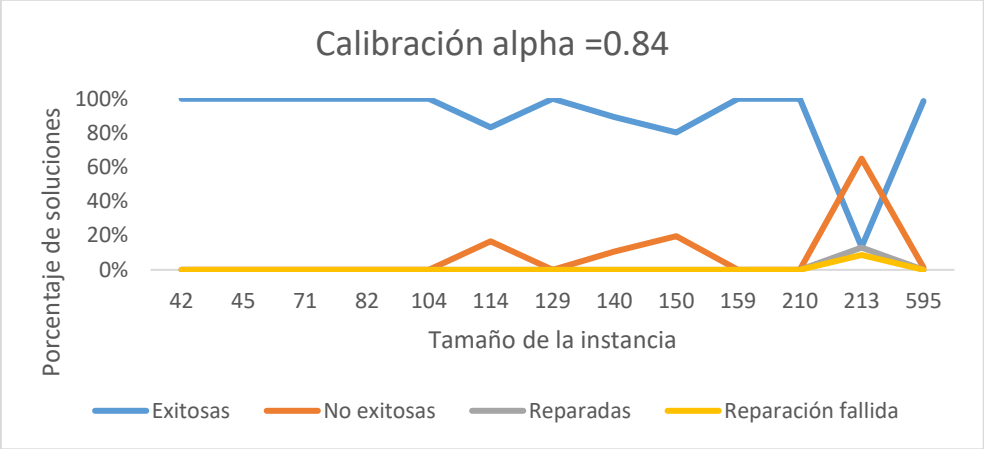


Gráfica 2 - Desviación cambiando iteraciones



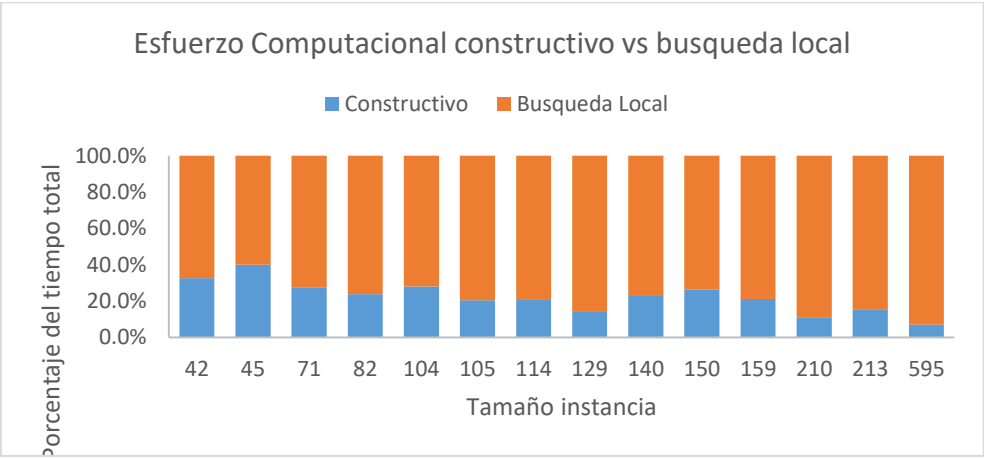
Por último, se calibró el $alpha$ que define el tamaño de la lista restricta de candidatos. En línea con la lógica del algoritmo que consiste en relajar la restricción de carga de trabajo no se debe disminuir mucho el $alpha$ ya que se estaría volviendo a acotar la restricción y la exploración de soluciones quedaría corta. Este parámetro debería estar por encima de 0.5 pero a la vez queremos dar cierto grado de restricción para que las soluciones no queden supremamente desbalanceadas. Por tal motivo la calibración se enfoco principalmente entre los valores de 0.6 y 0.9. Para instancias pequeñas se encuentran muy buenos resultados con un valor de 0.7 pero el resultado con instancias grandes no fue favorable debido a que no se les daba suficiente libertad para explorar soluciones y requerían ser reparadas todo el tiempo, esto es costoso a nivel computacional y además disminuye la calidad de las soluciones. En este caso se buscó encontrar un $alpha$ donde se le diera suficiente libertad de exploración a las instancias mas grandes. Se encontró que el $alpha$ optimo es 0.84 ya que todas las instancias tienen soluciones exitosas y el porcentaje de reparación es mínimo. Se puede observar además que el índice de soluciones exitosas de la metodología planteada es bastante alto con los parámetros utilizados. Solo se activó la estrategia de reparación para la instancia con 213 turnos. Lo cual nos indica que para todas se están encontrando soluciones factibles aproximadamente en un 80 % de las veces.

Gráfica 3 - Porcentaje de efectividad soluciones con Alpha=0.84

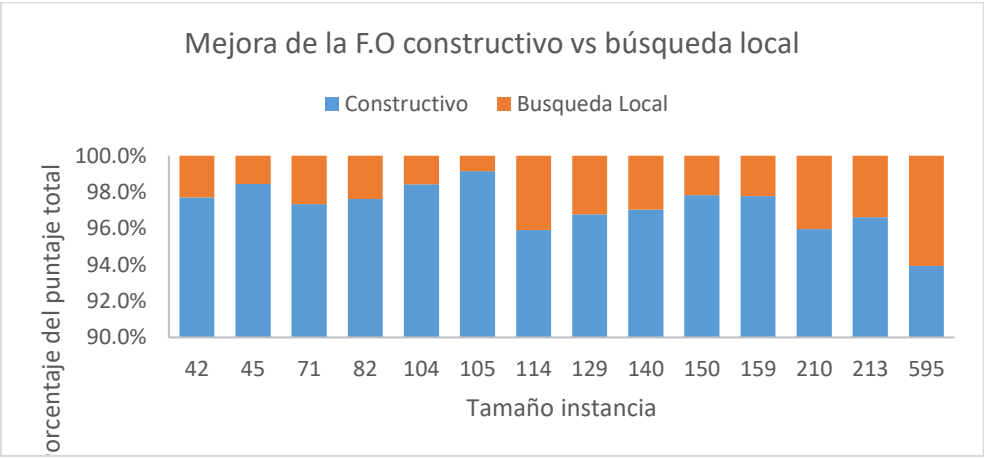


Por último, podemos observar el esfuerzo computacional y el porcentaje de mejora de la función objetivo del algoritmo constructivo y la búsqueda local. Donde observamos que la búsqueda local tiene un costo computacional casi de mas del triple del constructivo dentro de todo el algoritmo a pesar de la complejidad del constructivo. La mejora de la función objetivo se comporta de manera contraria, observamos que la mayoría del impacto viene del algoritmo constructivo. Sin embargo, a medida que crece la instancia vemos que empieza a ser más significativa la búsqueda local para la mejora de la solución.

Gráfica 4 - Comparación esfuerzo computacional constructivo y búsqueda local



Gráfica 5 - Comparación mejora F.O constructivo y búsqueda local



4.2 Comparación de la metodología propuesta con modelo lineal

Para validar el desempeño de la metodología GRASP se realizó una comparación con el modelo lineal de Cárdenas (2019) para las mismas instancias de asignación de turnos de enfermeras. Se corrió el algoritmo 30 veces para cada instancia con el fin de que la media de los indicadores se balanceara y de esta forma llevar a cabo un mejor análisis de su comportamiento. En términos de soluciones factibles y completas entregadas, el resultado de la metodología propuesta superó el modelo lineal con un 93% contra un 57%. Para ambas propuestas no se encontró una solución factible para la novena instancia. La diferencia radica en que el modelo lineal no encuentra una solución factible para otras dos instancias y entrega tres soluciones factibles incompletas. Otro punto de comparación es la satisfacción de los conductores. Es importante mencionar que el modelo lineal no tenia como objetivo maximizar la satisfacción de ninguna manera, pero nos demuestra que se le puede dar un valor agregado a la asignación de turnos de conductores sin necesidad de incumplir las restricciones y requerimientos de la empresa. En el modelo lineal se encuentra muy balanceado el número de satisfechos, indiferentes e insatisfechos pero la diferencia es clara al ver como en promedio es posible obtener un porcentaje promedio de satisfechos del 85% y en muchos casos un porcentaje de satisfechos superior al 90%.

Tabla 1 - Algoritmo GRASP

Instancia	Satisfacción Global	Satisfechos	Indiferentes	Insatisfechos	Factibilidad
1	89.85%	91.11%	7.63%	1.26%	Si (completo)
2	90.00%	90.32%	9.36%	0.32%	Si (completo)
3	90.89%	91.42%	8.05%	0.53%	Si (completo)
4	99.91%	99.91%	0.10%	0.00%	Si (completo)
5	94.37%	94.37%	5.63%	0.00%	Si (completo)
6	83.52%	85.39%	12.76%	1.86%	Si (completo)
7	97.57%	97.62%	2.33%	0.05%	Si (completo)
8	46.36%	64.36%	17.65%	18.00%	Si (completo)
9	-	-	-	-	No
10	60.38%	68.24%	23.89%	7.86%	Si (completo)
11	46.91%	64.33%	18.24%	17.43%	Si (completo)
12	91.19%	95.60%	0.00%	4.40%	Si (completo)
13	95.18%	95.19%	4.79%	0.02%	Si (completo)
14	55.06%	64.70%	25.67%	9.64%	Si (completo)
Promedio	**80.09%**	**84.81%**	**10.47%**	**4.72%**	**92.86%**

Tabla 2 - Modelo lineal

Instancia	Satisfacción Global	Satisfechos	Indiferentes	Insatisfechos	Factibilidad
1	6.67%	31.11%	44.44%	24.44%	Si (completo)
2	4.76%	35.71%	33.33%	30.95%	Si (completo)
3	-17.07%	21.95%	39.02%	39.02%	Si (completo)
4	-	-	-	-	No
5	8.45%	36.62%	35.21%	28.17%	Si (completo)
6	-9.91%	28.30%	33.49%	38.21%	Si (1 sin asignar)
7	-9.30%	26.36%	37.98%	35.66%	Si (completo)
8	4.00%	36.67%	30.67%	32.67%	Si (completo)
9	-	-	-	-	No
10	-4.39%	30.70%	34.21%	35.09%	Si (completo)
11	4.29%	35.71%	32.86%	31.43%	Si (completo)
12	11.54%	55.77%	0.00%	44.23%	Si (3 sin asignar)
13	-	-	-	-	No
14	-0.17%	34.18%	31.48%	34.34%	Si (1 sin asignar)
Promedio	**-0.10%**	**33.92%**	**32.06%**	**34.02%**	**57.14%**

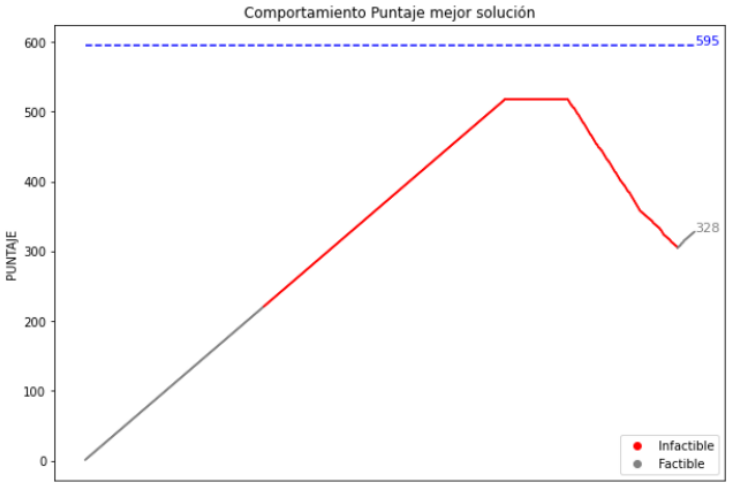
Además, el tiempo computacional de la metodología propuesta es comparable con el modelo lineal y en el caso de las instancias pequeñas no supera 1 segundo. Al igual que con las instancias de turnos de enfermeras se encontró una clara diferencia en la satisfacción entre ambos métodos para la solución de INTEGRA S.A. como se muestra a continuación:

Tabla 3 - Soluciones INTEGRA S.A.

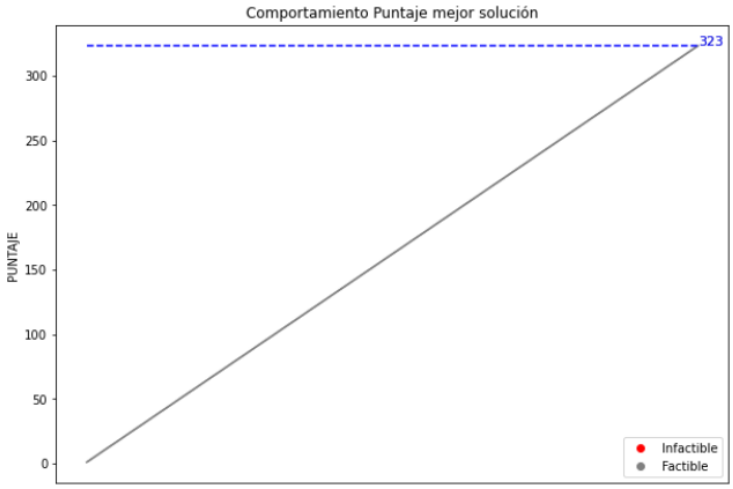
Instancia	Tiempo (s)	Satisfacción Global	Satisfechos	Indiferentes	Insatisfechos	Factibilidad
GRASP	1.91	100.00%	100.00%	0.00%	0.00%	Si (completo)
Modelo lineal	1.07	4.95%	37.46%	30.03%	32.51%	Si (completo)
Promedio	**1.49**	**52.48%**	**68.73%**	**15.02%**	**16.25%**	**100.00%**

Por último, se muestra el comportamiento de la función objetivo de la mejor solución para la empresa INTEGRA S.A. (ver Gráfica 5) y para la instancia más grande de Musliu (2006) que tiene 595 turnos por asignar (ver Gráfica 4). Se puede observar que la solucion obtenida para la empresa es bastante eficiente y no implica mucho trabajo para la metodologia a comparación de la otra. De hecho se observa en la Gráfica 4 que una parte de la solución es infactible y corresponde al algoritmo constructivo donde se busca explorar soluciones y luego recobrar la factibilidad gracias a la relajación de la restricción y la estrategia de balanceo. Sin embargo en la Gráfica 5 no observamos ese comportamiento debido a que con las restricciones actuales de la empresa los trabajadores tienen bastante tiempo libre y es posible una configuración de turnos donde todos estan satisfechos. De hecho sería posible cumplir con todos los requerimientos con menos empleados de los que tienen actualmente.

Gráfica 6 - Instancia 14 Musliu



Gráfica 7 - Instancia INTEGRA S.A.



5 Conclusiones

La metodología planteada resulto en una mejora en la generación de soluciones en comparación con el modelo lineal encontrando un mayor número de soluciones factibles y completas para el problema de asignación de turnos conductores. El algoritmo GRASP probó ser más eficiente en un 36% con una diferencia muy pequeña en esfuerzo computacional.

En cuanto al enfoque de maximización de la satisfacción, se puede concluir que implica una mejora en términos de ambiente laboral bastante grande. Por lo que sería interesante su implementación en la empresa para observar los beneficios que puede traer esto a largo plazo y para estudiar otros factores que pueden ser agregados y ser tenidos en cuenta en el cálculo de la satisfacción. Una propuesta de trabajo a futuro podría ser como mejorar esta función de satisfacción para que sea mas robusta y que tanto es el impacto que trae una solución de este tipo en comparación con una donde solo se busca cumplir con las restricciones.

Por último, es una metodología flexible que podría ser implementada en otros contextos de asignación de turnos como el personal de salud en hospitales. Seria interesante observar como difiere el calculo de la satisfacción dependiendo del problema de interés y que tan viable es trabajarlo de la manera propuesta en este artículo.

6 Referencias

Dantzig, G. B. (1954). A comment on Edie's “Traffic delays at toll booths”. *Journal of the Operations Research Society of America*, *2*(3), 339-341.

Martello, S., & Toth, P. (1992). Generalized assignment problems. In *Algorithms and Computation: Third International Symposium, ISAAC'92 Nagoya, Japan, December 16–18, 1992 Proceedings 3* (pp. 351-369). Springer Berlin Heidelberg.

Musliu, N. (2006). Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, *2*(4), 309-326.

Cárdenas Parra, K. (2019). Modelo matemático lineal flexible para la rotación de turnos de trabajos de los conductores del sistema de transporte masivo del área metropolitana centro occidente.

Beasley, J. E., & Cao, B. (1996). A tree search algorithm for the crew scheduling problem. *European Journal of Operational Research*, *94*(3), 517-526.

Paias, A., Mesquita, M., Moz, M., & Pato, M. (2021). A network flow-based algorithm for bus driver rerostering. OR Spectrum, 43, 543-576.

Lin, D. Y., Juan, C. J., & Chang, C. C. (2020). A branch-and-price-and-cut algorithm for the integrated scheduling and rostering problem of bus drivers. *Journal of Advanced Transportation*, *2020*, 1-19.

Barbosa, V. M. M. (2018). *Bus driver rostering by hybrid methods based on column generation* (Doctoral dissertation, Universidade de Lisboa (Portugal)).

Er-Rbib, S., Desaulniers, G., El Hallaoui, I., & Bani, A. (2021). Integrated and sequential solution methods for the cyclic bus driver rostering problem. *Journal of the Operational Research Society*, *72*(4), 764-779.

Xie, L., Merschformann, M., Kliewer, N., & Suhl, L. (2017). Metaheuristics approach for solving personalized crew rostering problem in public bus transit. *Journal of Heuristics*, *23*, 321-347.

Desaulniers, G., & Hickman, M. D. (2007). Public transit. *Handbooks in operations research and management science*, *14*, 69-127.