

**UNIVERSIDAD COMPLUTENSE DE MADRID**

FACULTAD DE CIENCIAS MATEMÁTICAS



**Grado en Ingeniería Matemática**

**EL PROBLEMA DE REPOSICIONAMIENTO  
EN SISTEMAS DE BICICLETAS COMPARTIDAS**

**TRABAJO DE FIN DE GRADO**

Ángel Amando Gil Álamo

Tutora: Susana Muñoz López

Julio de 2022

# Índice de contenidos

Abstract .....	1
1. Introducción.....	2
2. Formulación matemática del problema: el modelo arco-flujo .....	6
3. Procedimientos de resolución heurísticos.....	12
3.1. Determinación de una solución inicial mediante un algoritmo voraz.....	12
3.2. Método PILOT.....	20
4. Experiencia computacional .....	23
4.1. Generación de los casos y ajuste de los modelos .....	23
4.2. Resultados computacionales .....	27
5. Conclusiones .....	33
Referencias .....	35
Apéndice: Código del algoritmo AVET .....	38

## Abstract

Urban areas are in need of efficient and sustainable mobility services. Public bicycle sharing systems (PBSS) stand out as a promising alternative and many cities have invested in their deployment. This fact has led to a continuous and fast implementation of PBSS systems around the world, while at the same time, researches devoted to understanding the system dynamics and deriving optimal designs are being developed. The most complex and yet most studied challenge regarding PBSS is the problem of repositioning, in which operators have to regularly redistribute bikes across their stations in order to avoid them getting overly full or empty. In this paper, we study the dynamic variant of the repositioning problem, when the process takes place during the system's activity hours. We address the problem of finding efficient vehicle tours by proposing a constructive greedy algorithm based on a space-time network. The algorithm is later extended and improved following the concept of the PILOT method. Both versions of the heuristic procedures are compared to a MIP model that is also implemented following the space-time formulation. In order to test the efficiency of the algorithms, several sets of large instances have been considered; some of them are based on the PBSS Bicimad, in Madrid, and the others have been randomly generated. The keys to the recent success of PBSS are briefly analyzed in the introduction. At the end of the work, possible ways to improve the performance of the proposed heuristics are also suggested.

**Keywords.** Public bike-sharing system, Vehicle routing, Heuristics, Greedy, PILOT.

## 1. Introducción

Un sistema de bicicletas compartidas (SBC), consiste en poner a disposición de los habitantes de una ciudad bicicletas para que puedan ser alquiladas. Para ello, se distribuyen estratégicamente diversas estaciones, cerca de lugares muy frecuentados, lo que permite a los usuarios recoger y devolver las bicicletas fácilmente. Pese a que los SBC comenzaron siendo en gran medida servicios privados, actualmente la mayoría son de servicio público y proporcionan un apoyo cada vez más importante para los sistemas de transporte multimodal puesto que presentan distintos beneficios para la sociedad, principalmente en salud, medioambiente, movilidad y economía. En las planificaciones de las mayores organizaciones sin ánimo de lucro y organismos públicos a nivel mundial (Organización Mundial de la Salud [ONU], 2018; Gouvernement français, 2019), se plantea la idea de los SBC como principal transporte urbano de futuro, con el objetivo de crear entornos sostenibles y una sociedad activa, proponiendo medidas de ayuda y mejora de infraestructuras. En cuanto a movilidad, existen estudios en grandes ciudades en los que se estima que uno de cada dos trayectos en automóvil es de menos de 3 km de distancia, situando la media en 2,8 km (Instituto Geográfico Nacional [IGN], 2022). En los desplazamientos urbanos, las bicicletas podrían ser más eficientes que los automóviles y autobuses (Gouvernement français, 2019). En cuanto a economía a nivel de usuario, aunque los sistemas de bicicletas urbanas tienen un coste, siguen siendo una solución muy asequible en comparación con otros medios de transporte urbano. El coste de un abono de bicicletas compartidas suele ser menor que el de un abono de transporte público y por supuesto muy inferior a la compra y mantenimiento de un coche. Dos ejemplos de ciudades en las que los SBC están teniendo éxito son Montreal, con su sistema BIXI (Bixi, s.f.), en el que se permite utilizar la tarjeta de transporte público para alquilar bicicletas en las estaciones y Madrid, con Bicimad (Bicimad, 2022), en el que el abono anual cuesta

entre 15 y 25 euros, con un coste adicional de un euro por hora de uso. A nivel social, al ser mayormente públicos, los SBC proporcionan una fuente de ingresos extra para los gobiernos de las ciudades. Por ejemplo, algunos estudios sugieren que la sociedad estadounidense podría ahorrarse hasta 24 billones de dólares en 35 años si se adoptaran de forma inmediata estrategias agresivas para la incorporación de SBC (Mason et al., 2015). En términos de CO<sub>2</sub>, es evidente que las bicicletas contaminan menos que el coche o el transporte público. Para cuantificar el impacto que podría llegar a suponer el desplazarse en bicicleta por la ciudad, se recomienda leer los estudios llevados a cabo por Christian Brand, profesor de transporte, energía y medio ambiente de la universidad de Oxford donde, por ejemplo, se llegó a la conclusión de que las emisiones de carbono ocasionadas por el transporte de las personas que montan en bicicleta diariamente son un 84% inferiores en comparación con las que no lo hacen (Brand et al., 2021a, 2021b).

Debido a estos factores, la presencia de los SBC ha crecido drásticamente en las grandes ciudades del globo: en 2010 se estimó que se habían implementado alrededor de 101 programas en 125 ciudades, incorporando alrededor de 139.000 bicicletas (Shaheen et al., 2010), para 2014 ya había 747 sistemas activos compuestos aproximadamente por 772.000 bicicletas (Ghosh et al., 2015) y en la actualidad hay más de 1.800 sistemas totalmente operativos con unas 8.937.000 bicicletas en más de 560 ciudades (Meddin et al., 2021). Estos datos, sumados a que existen numerosos proyectos a corto, medio y largo plazo (Bicimad, 2022; ONU, 2018) para hacer de las bicicletas un medio de transporte cada vez más presente, parecen indicar que los SBC están lejos de frenar su expansión. Si bien tienen un gran potencial, existen varios factores cruciales para que sean una alternativa real a largo plazo y para un gran volumen de usuarios. Entre ellos destacan el estudio previo de la densidad y distribución de las estaciones, la facilidad de uso del sistema y, por encima de todos, un servicio de reposicionamiento eficaz (Todd et al., 2021).

A continuación, se define el problema de reposicionamiento derivado de este último factor:

Un SBC suele tener varias estaciones repartidas por la ciudad. Al principio del día, cada estación está provista de un número predeterminado de bicicletas. Un usuario registrado puede alquilar una bicicleta en una estación y devolverla más tarde en otra. Debido a diversos factores como la altitud de las estaciones, las características demográficas o las paradas de transporte público cercanas, algunas estaciones tienden a vaciarse, mientras que otras, por el contrario, a llenarse. En el caso de una estación vacía, los clientes no podrán recoger bicicletas en dicha estación, mientras que, en el caso de una estación llena, los clientes no podrán devolver sus bicicletas. Por tanto, los operadores de SBC necesitan redistribuir las bicicletas de forma regular entre las estaciones para evitar, o al menos minimizar, la insatisfacción de los clientes. Normalmente, esta tarea la realiza una flota de vehículos que recoge las bicicletas de las estaciones con exceso y las deposita en estaciones con déficit. En este trabajo se aborda el problema de Reposicionamiento en Sistemas de Bicicletas Compartidas (PRSBC), cuyo objetivo es determinar una ruta para cada vehículo, de modo que el sistema alcance un estado de equilibrio y sea capaz de satisfacer las demandas de los usuarios en la medida de lo posible. Este problema es más complejo que los clásicos de rutas de vehículos y el del viajante, ya que se requiere además determinar el número de bicicletas a depositar en cada estación respetando las capacidades de los vehículos (Ho y Szeto, 2014). Las variantes principales del PRSBC son dos: estática y dinámica. La variante estática considera que las operaciones de reposicionamiento son nocturnas, pues la tasa de utilización durante este periodo es, en general, insignificante, así como las variaciones en la demanda de las distintas estaciones, mientras que la variante dinámica considera que las operaciones son diurnas, y debe tener en cuenta las variaciones en la demanda en tiempo real.

Una comparación directa de los trabajos existentes es difícil ya que la mayoría de ellos consideran diferentes casuísticas del problema. La variante más estudiada del PRSBC es la estática y se suelen aplicar técnicas de programación lineal entera mixta (MIP). Entre los trabajos más referenciados en la literatura se destacan los siguientes: Forma et al. (2010), proponen varias formulaciones matemáticas para el problema de equilibrio estático. Contardo et al. (2012), consideran la variante dinámica del problema y presentan un modelo MIP, y descomposiciones de Dantzig-Wolfe y Benders para abordar instancias más grandes. Raviv et al. (2013), presentan otras dos formulaciones MILP adicionales para el problema estático. Chemla et al. (2013), también abordan el problema estático y proponen un algoritmo exacto basado en la generación de columnas. Pfrommer et al. (2014), investigan una heurística para planificar recorridos con múltiples vehículos y también sugieren una estrategia de precios dinámica. Recalculan periódicamente los recorridos de los camiones y los precios dinámicos mientras el sistema está activo y lo prueban con una simulación basada en datos históricos. Kloimüllner et al. (2014), desarrollan varios enfoques metaheurísticos para el caso dinámico que escalan bien para instancias grandes. De forma similar, pero para la variante estática, Di Gaspero et al. (2013, 2016), proponen varios enfoques meta-heurísticos con resultados prometedores. Otros trabajos relacionados examinan los aspectos de planificación estratégica de los SBP, como la ubicación y el diseño de la red (Lin et al., 2013; Adham y Bentley, 2015) o las características del sistema y los patrones de uso (Todd et al., 2021). Sin embargo, estos aspectos no entran en el ámbito de actuación que nos ocupa.

En este trabajo se aborda el PRSBC dinámico. El problema se formula basándose en la construcción de una red espacio-temporal. Sobre esta formulación se presenta un algoritmo voraz y se mejora siguiendo la filosofía del procedimiento metaheurístico PILOT.

El objetivo principal del trabajo es proponer un nuevo algoritmo basado en la discretización del tiempo que determine, de manera eficaz, soluciones para grandes casos basados en Bicimad. También se pretende contribuir al desarrollo de la formulación espacio-temporal, analizando el impacto de la longitud de los intervalos de tiempo en casos de tamaño variable, y se espera que el procedimiento presentado pueda resultar útil para el desarrollo de futuros algoritmos que aborden el PRSBC dinámico.

El trabajo está estructurado de la siguiente manera: en la sección 2 se define formalmente el PRSBC dinámico y se adopta la formulación matemática espacio-temporal que Contardo et al. (2012) introdujeron para la variante dinámica. En la sección 3 se presentan el nuevo algoritmo y su extensión mediante el procedimiento metaheurístico PILOT. En la sección 4 se muestran y analizan los resultados computacionales, en una serie de experimentos basados tanto en casos generados aleatoriamente como en casos creados a partir de los datos del SBC Bicimad de Madrid. Por último, en la sección 5 se exponen las conclusiones y se proponen algunas formas de mejorar el algoritmo considerado.

## **2. Formulación matemática del problema: el modelo arco-flujo**

El conjunto de vehículos se denota por  $K$ . Cada vehículo  $k \in K$  tiene asociada una capacidad,  $Q_k$ , una carga inicial  $Q_k^0$  al comienzo del horizonte temporal, y una posición inicial dada por el punto  $u_k$ .

Se denota por  $V$  el conjunto de estaciones de la red. Cada estación  $v \in V$  tiene asociados una capacidad  $C_v$ , y un número inicial de bicicletas  $C_v^0$  al comienzo del horizonte temporal.

El horizonte temporal se discretiza en un conjunto de períodos que se denota por  $T$ .

Esta formulación es similar a la formulación clásica de tres índices de otros problemas de rutas. Sin embargo, se permite tener en cuenta la posibilidad de visitar la misma estación en diferentes momentos (ver Forma et al., 2010).

Para construir la red espacio-temporal se definen un conjunto de estados y un conjunto de arcos, denominados  $S$  y  $A$  respectivamente (ver Contardo et al., 2012).  $S$  está compuesto por tres subconjuntos:  $S_1 \equiv \{(u_k, 0) : k \in K\}$  representa las posiciones de los vehículos en el instante 0,  $S_2 \equiv \{(v, t) : v \in V, t \in T\}$  representa los nodos para las estaciones en los diferentes periodos de tiempo y  $\varphi$  es un nodo ficticio que representa el final de las rutas de los vehículos en el horizonte temporal considerado. Se denota por  $v_s$  y  $t_s$  al nodo y al periodo correspondientes al estado  $s \in S \setminus \{\varphi\}$ . Para cada estado  $s \in S_2$  se define  $Pred_s = (v_s, t_s - 1)$  si  $t_s \geq 2$  y  $Suc_s = (v_s, t_s + 1)$  si  $t_s \leq |T| - 1$ . Además, cada estado  $s \in S_2$  tiene asociada una demanda  $Dem_s$  de bicicletas, de modo que  $Dem_s \geq 0$  si  $s$  es un punto de recogida, y  $Dem_s \leq 0$  si  $s$  es un punto de entrega.

Los tiempos de viaje se reescalan para tener las mismas unidades que en la discretización del tiempo realizada en  $T$  (por ejemplo, si un periodo de tiempo representa una ventana de 5 minutos, un viaje de 10 minutos que comienza en el periodo  $t$  finalizará en el periodo  $t+2$ ).

El conjunto de arcos  $A$  está compuesto por tres tipos de arcos:  $A_1 \equiv \{(s, s') : s, s' \in S, s \neq s', t_{s'} - t_s \geq Dist(v_s, v_{s'}) > t_{s'} - t_s - 1\}$  contiene todos los viajes directos posibles entre pares de estados, donde  $Dist(v_s, v_{s'})$  es el tiempo (reescalado) necesario para desplazarse desde  $v_s$  hasta  $v_{s'}$ ,  $A_2 \equiv \{(s, Suc_s) : s \in S_2, t_s \leq |T| - 1\}$  (cada arco de  $A_2$  representa la acción de esperar en una estación  $v_s$  durante un periodo de tiempo) y  $A_3 \equiv \{(s, \varphi) : s \in S \setminus \{\varphi\}\}$  contiene todos los arcos finales de las rutas de los vehículos. Es posible considerar que la función distancia depende del tiempo, lo que aportará mayor flexibilidad al enfoque de modelización. Sin embargo, se supone que los tiempos de viaje no dependen de las acciones de

recogida o entrega de bicicletas realizadas por un vehículo en una estación determinada; esta es una limitación del modelo que puede resolverse parcialmente incluyendo en los tiempos de viaje el tiempo medio de servicio en las estaciones.

En la *Figura 2.1* se muestra una red con dos vehículos y tres estaciones y en la *Figura 2.2*, la red espacio-temporal resultante tras una discretización del tiempo en cuatro períodos de una unidad. Como puede verse en este pequeño ejemplo, el número de arcos en la red espacio-temporal es mucho mayor que en el grafo original. Además, hay que tener en cuenta que el grafo espacio-temporal sólo permite arcos que vayan hacia delante en el tiempo. Esta propiedad se explotará más adelante en los procedimientos de resolución heurísticos. En el grafo espacio temporal, los arcos negros representan los viajes directos entre nodos, los azules simbolizan la acción de esperar en una estación y los arcos de línea discontinua son el final de ruta desde cualquier nodo. En el dibujo se puede ver que las distancias fraccionarias se redondean al entero superior.

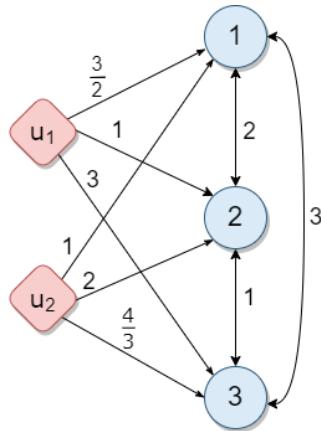


Figura 2.1. Red original

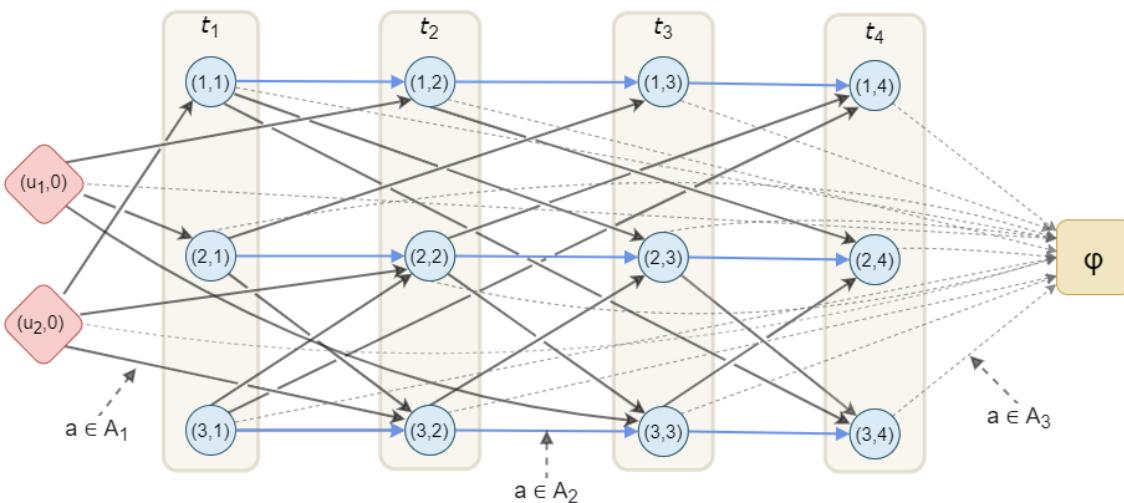


Figura 2.2. Red espacio-temporal

A continuación, se plantea el modelo propuesto por Contardo et al. (2012). El motivo de incluirlo en la presente memoria es poder realizar una comparación directa entre un método de resolución exacto y los algoritmos heurísticos propuestos, lo cual pondrá de manifiesto la complejidad del problema y permitirá añadir más profundidad al estudio del impacto de la longitud de los intervalos de tiempo en la convergencia de los algoritmos y la calidad del ajuste al caso real.

Para cada estado  $s \in S_2$  se definen las variables  $y_s^+, y_s^- \geq 0$  enteras que representan, respectivamente, la escasez y el exceso de bicicletas, es decir, la cantidad de bicicletas que no podrán ser recogidas o devueltas por los usuarios en la estación  $v_s$  durante el periodo  $t_s$ . Estos dos valores no pueden ser distintos de 0 simultáneamente y a partir de ellos se obtiene la demanda insatisfecha total. Además, se define la variable  $x_s \geq 0$  entera, que indica el número de bicicletas que quedan almacenadas en el estado  $s$ . Para cada par  $a \in A$  y  $k \in K$  se define la variable binaria  $\beta_{a,k}$  igual a 1 si el vehículo  $k$  pasa por el arco  $a$  en su ruta, y la variable  $q_{a,k} \geq 0$  entera, que indica la carga del vehículo  $k$  al pasar por el arco  $a$ . Aunque estas cargas son enteras, en Contardo et al. (2012) se presentan como continuas, puesto que se pueden relajar para ser recuperadas más tarde mediante la resolución de un problema de flujo de coste mínimo en la red espacio-temporal para valores enteros fijos de las variables  $\beta_{a,k}$ . Por último, para cada estado  $s \in S$  se denota por  $\delta_s^+$  al conjunto de arcos cuyo extremo final es  $s$  y por  $\delta_s^-$  al conjunto de arcos cuyo extremo inicial es  $s$ . Para tener una mejor visualización de los elementos descritos de la formulación elegida del problema, estos se recogen de manera resumida en la *Tabla 2.1*.

Conjunto	Dominio	Descripción	
$K$	$\{1, 2, \dots,  K \}$	Vehículos	
$V$	$\{1, 2, \dots,  V \}$	Estaciones	
$T$	$\{1, 2, \dots,  T \}$	Horizonte temporal	
$S$	$S_1 \cup S_2 \cup \{\varphi\}$	Estados $\equiv$ Nodos del grafo espacio temporal	
$S_1$	$\{(u_k, 0) : k \in K\}$	Posición inicial de los vehículos	
$S_2$	$\{(v, t) : v \in V, t \in T\}$	Estado de las estaciones en cada período	
$\{\varphi\}$		Fin de ruta	
$A$	$A_1 \cup A_2 \cup A_3$	Posibles rutas de los vehículos $\equiv$ Aristas	
$A_1$	$\{(s, s') : s, s' \in S, s \neq s'\}$	Viajes directos entre pares de estados	
$A_2$	$\{(s, Suc_s) : s \in S_2\}$	Acción de esperar un período sin moverse	
$A_3$	$\{(s, \varphi) : s \in S \setminus \{\varphi\}\}$	Fin de ruta desde cualquier estado	
Parámetro	Dominio	Descripción	
$Q_k$	$K$	Capacidad del vehículo $k$	
$Q_k^0$	$K$	Carga inicial del vehículo $k$	
$u_k$	$K$	Posición inicial del vehículo $k$	
$C_v$	$V$	Capacidad de la estación $v$	
$C_v^0$	$V$	Carga inicial de la estación $v$	
$Pred_s$	$S_2$	Predecesor del nodo $s$	
$Succ_s$	$S_2$	Sucesor del nodo $s$	
$Dem_s$	$S_2$	Demanda del nodo $s$	
$\delta_s^+$	$S$	Arcos que finalizan en $s$	
$\delta_s^-$	$S$	Arcos que parten de $s$	
Variable	Tipo	Dimensión	Descripción
$y_s^+$	Entera	$ S_2 $	Déficit de bicicletas en $s$
$y_s^-$	Entera	$ S_2 $	Exceso de bicicletas en $s$
$x_s$	Entera	$ S_2 $	Bicicletas almacenadas en $s$
$\beta_{a,k}$	Binaria	$ A  *  K $	1 si el vehículo $k$ pasa por $a$ en su ruta
$q_{a,k}$	Entera	$ A  *  K $	Carga del vehículo $k$ al pasar por $a$

Tabla 2.1. Elementos del modelo propuesto

La formulación del problema es la siguiente:

$$\text{minimizar} \sum_{s \in S_2} (y_s^+ + y_s^-)$$

sujeto a:

$$\sum_{k \in K} \sum_{a \in \delta_s^+} q_{a,k} - \sum_{k \in K} \sum_{a \in \delta_s^-} q_{a,k} - x_s + y_s^+ - y_s^- = Dem_s - C_{v_s}^0 \quad \forall s \in S_2 \text{ con } t_s = 1 \quad (1)$$

$$\sum_{k \in K} \sum_{a \in \delta_s^+} q_{a,k} - \sum_{k \in K} \sum_{a \in \delta_s^-} q_{a,k} + x_{Pred_s} - x_s + y_s^+ - y_s^- = Dem_s \quad \forall s \in S_2 \text{ con } t_s \geq 2 \quad (2)$$

$$\sum_{k \in K} \sum_{a \in \delta_s^+} \beta_{a,k} \leq 1 \quad \forall s \in S_2 \quad (3)$$

$$q_{a,k} \leq Q_k \beta_{a,k} \quad \forall k \in K, \forall a \in A \quad (4)$$

$$\sum_{a \in \delta_s^-} \beta_{a,k} - \sum_{a \in \delta_s^+} \beta_{a,k} = 0 \quad \forall k \in K, \forall s \in S_2 \quad (5)$$

$$\sum_{a \in \delta_{(u_k,0)}^-} \beta_{a,k} = 1 \quad \forall k \in K \quad (6)$$

$$\sum_{a \in \delta_{(u_k,0)}^-} q_{a,k} = Q_{k,0} \quad \forall k \in K \quad (7)$$

$$y_s^+, y_s^- \geq 0 \text{ enteras}, \quad x_s \geq 0 \text{ entera}, \quad q_{a,k} \geq 0 \text{ entera},$$

$$0 \leq x_s \leq C_{v_s} \text{ entera}, \quad \beta_{a,k} \in \{0,1\}$$

La función objetivo representa la demanda total no satisfecha, es decir, el número de usuarios que intentaron recoger bicicletas en estaciones vacías o entregar bicicletas en estaciones completas. Los conjuntos de restricciones (1) y (2) son los de conservación de flujo en cada estación para cada periodo de tiempo. El papel de las variables  $\{y_s^+, s \in S_2\}, \{y_s^-, s \in S_2\}$  es compensar el desequilibrio de la red. En una red perfectamente equilibrada, estas variables tomarán el valor cero para cada  $s \in S_2$ . Además, en las soluciones óptimas no pueden tomar valores positivos simultáneamente. El conjunto de restricciones (3) garantiza que cada nodo sea visitado como máximo una vez en cada periodo de tiempo. El conjunto (4) vincula

el uso de cada arco a la carga máxima permitida del vehículo que lo atraviesa. El conjunto de restricciones (5) es el de conservación del flujo de vehículos; obliga a los vehículos a abandonar las estaciones visitadas previamente. El conjunto de restricciones (6) obliga a que cada vehículo sea utilizado exactamente una vez. Nótese que esto incluye la opción de no utilizar un vehículo k introduciendo el arco  $((u_k, 0), \varphi)$ . El conjunto de restricciones (7) garantiza que los vehículos salgan de su posición inicial con sus cargas correspondientes.

En su trabajo, Contardo et al. (2012) exponen casos particulares del modelo a los que denominan patológicos, pues, aunque las penalizaciones tienden a minimizar el uso de las variables, estas variables pueden tomar valores poco realistas, creando o destruyendo bicicletas, si esto compensase en el futuro. Sin embargo, en su trabajo también demuestran que, dada una solución del modelo que contenga algún caso patológico, es posible construir una solución que no contenga ninguno de estos con, a lo sumo, el mismo coste. De esta manera, siempre se podrá obtener una solución óptima que no contenga ningún caso patológico, por lo que no es necesario preocuparse por estos a la hora de implementar el modelo.

### 3. Procedimientos de resolución heurísticos

En esta sección se plantea una heurística de construcción voraz, llamada AVET, para crear soluciones iniciales y posteriormente se presenta una versión ampliada que utiliza el método PILOT (ver Voss et al., 2005).

#### 3.1. Determinación de una solución inicial mediante un algoritmo voraz

Para generar eficazmente una solución factible inicial de calidad, se emplea un algoritmo voraz. La solución se construye creando iterativamente un recorrido para cada vehículo siguiendo una estrategia del mejor sucesor local. A partir del último estado  $s$  de un recorrido parcial, se evalúan todos los estados candidatos a ser la

siguiente parada del vehículo. Estos estados son precisamente los pertenecientes al conjunto  $\delta_s^-$ . Para cada  $s' \in \delta_s^-$  se calcula el número máximo de bicicletas que es posible recoger o entregar por unidad de tiempo, en base a la carga del vehículo tras operar en  $s$  (a la que se denotará por  $Q_k^{t_s}$ ), a la distancia del vehículo a la estación  $v_{s'}$ , (esto es,  $Dist(v_s, v_{s'})$ ) y al nivel de llenado esperado de las estaciones en el instante de tiempo de llegada del vehículo, que se obtiene restando, al nivel de llenado de  $v_{s'}$ , en tiempo  $t_s$ , la demanda acumulada entre  $t_s + 1$  y  $t_{s'}$ , (nótese que  $t_{s'} = t_s + Dist(v_s, v_{s'})$ ). Se denota por  $C_s$  al nivel de llenado de la estación  $v_s$  en el instante  $t_s$ , por  $s^*$  al estado  $(v_{s'}, t_s)$  y por  $Dem_{s^*}^{t_{s'} - t_s}$  a la demanda acumulada ( $Dem_{s^*}^{t_{s'} - t_s} = \sum_{i=t_s+1}^{t_{s'}} Dem_{v_{s'}, i}$ ). Además, se define  $Val_{s'}$  de la siguiente manera:

$$Val_{s'} = \begin{cases} \min\{C_{s^*} - Dem_{s^*}^{t_{s'} - t_s}, Q_k^0 - Q_k^s\} / Dist(v_s, v_{s'}) & \text{si } v_{s'} \text{ es de recogida} \\ \min\{C_{v_s} - C_{s^*} + Dem_{s^*}^{t_{s'} - t_s}, Q_k^s\} / Dist(v_s, v_{s'}) & \text{si } v_{s'} \text{ es de entrega} \end{cases}$$

Con estos valores se pretende minimizar de manera indirecta la función objetivo del modelo arco flujo presentado en la sección 2, ya que se intenta llenar tanto como sea posible las estaciones que tienden a vaciarse, y a vaciar aquellas que se llenen. En la práctica, esta forma de proceder da lugar a que los vehículos intenten alternar entre cargar y descargar el número máximo posible de bicicletas por unidad de tiempo, encontrando la estación adecuada para hacerlo. Sin embargo, si solo se tiene en cuenta este criterio, el algoritmo no reacciona totalmente en consonancia con el objetivo, pues puede tender a balancear estaciones que no necesiten ser equilibradas a corto plazo, dejando desatendidas estaciones que sí lo necesiten.

Con el fin de minimizar esta problemática, se añade un criterio adicional introducido por Kloimullner et al. (2014), el cual hace referencia a la urgencia en equilibrar cada estación y, aunque en su obra estudian una formulación y objetivos distintos, definieron un criterio aditivo y otro multiplicativo que resultan interesantes. Se ha decidido incorporar una versión propia del criterio de urgencia aditiva siguiendo una filosofía similar. La propuesta es la siguiente:

$$Urg_{s'} = w_2 \frac{N_{v_{s^*}}}{Dist(v_s, v_{s'})} - w_1 M_{v_{s^*}}, \text{ donde}$$

$M_{v_{s^*}}$  representa el margen de tiempo que hay entre el instante en el que  $v_{s^*}$  entrará en estado de desequilibrio si no se hace nada y la distancia de la estación  $v_s$  a  $v_{s'}$ ,  $N_{v_{s^*}}$  es el número de bicicletas involucradas desde el momento en el que comenzó el desequilibrio y  $w_1, w_2 > 0$  representan el peso de cada parte del criterio.

Para obtener  $M_{v_{s^*}}$ , se comienza calculando  $m$ , el primer instante en el que habrá bicicletas que no se podrán entregar o recoger de la estación. El valor de  $m$  se obtiene sumando a  $C_{s^*}$  la demanda de la estación  $v_{s^*}$  en los instantes  $t_{s^*} + 1, t_{s^*} + 2, \dots, t_{s^*} + m$  hasta que se cumpla que  $C_{v_{s^*}} - C_{s^*} + Dem_{s^*}^m > C_{v_{s^*}}$  para estaciones de carga, o  $C_{s^*} + Dem_{s^*}^m < 0$  para estaciones de descarga, donde  $Dem_{s^*}^m \equiv \sum_{i=t_s+1}^{t_s+m} Dem_{s^*}$ . Así pues, se tiene que  $M_{v_{s^*}} = \max\{0, m - Dist(v_s, v_{s'})\}$ . Por una parte, si  $M_{v_{s^*}} = 0$ , la estación ya se encontraba en desequilibrio o lo hará antes de que el vehículo pueda llegar; por otra parte, si se llega a  $t_{s^*} + m = |T|$  y no se cumplen las condiciones de balanceo, entonces la estación ya no necesitará ser balanceada y se tomará  $Urg_{s'} = -Val_{s'}$ , puesto que el criterio del algoritmo voraz para  $s'$  es  $h_{s'} = Val_{s'} + Urg_{s'}$ .  $N_{v_{s^*}}$  entra en juego solo cuando  $M_{v_{s^*}} = 0$ . Su cálculo es más delicado, pues conlleva un registro de los déficits o excesos en las estaciones desde instantes anteriores a  $t_s$ . Antes de proceder a explicar el cálculo de  $N_{v_{s^*}}$  se presenta el *pseudocódigo* del Algoritmo Voraz Espacio-Temporal (AVET), y se procede a explicar su funcionamiento:

Los elementos adicionales del algoritmo son los siguientes:  $t$  se inicializa a 0.  $Rutas_k$  contiene las rutas de cada vehículo y se inicializa con su posición inicial. Lo más importante en cada iteración es el último estado  $s$  añadido a la ruta, ya que se utiliza para determinar si el vehículo llega en el instante  $t$  (esto sucede cuando  $t_s = t$ ).  $Ocupados$  es un conjunto en el que se incluyen las estaciones ocupadas en

todo momento. Si un vehículo se encuentra de camino a una estación, esta no podrá volver a ser valorada hasta quedar desocupada.  $Fobj$  es el valor de la función objetivo del modelo arco flujo presentado en la sección 2, que se inicializa a 0.  $Ytemp_v$  se inicializa a 0 para cada estación y registra el número de bicicletas que no ha sido posible entregar/recoger por parte de los usuarios desde la última vez que se visitó dicha estación.

---

### **Algoritmo Voraz Espacio-Temporal (AVET)**

---

```

Iniciar Rutask, Ocupados, Fobj, Ytempv, t
while t ≤ |T|:
    if t > 0:
        (I)      for k in K:
                  if LlegaVehículo:
                      Actualizar cargas, desocupar estación, reiniciar Ytempv
        (II)     for v in V:
                  Actualizar demandas, Fobj e Ytempv
                  if t < |T| and LlegaVehículo:
                      (III)   Inicializar Valor, elegido y n_bicicletask.
                              for s in δs'- (s' ≡ Último estado en Rutask):
                                  calcular hs
                                  if hs > Valor:
                                      Actualizar Valor, elegido y n_bicicletask
                                      Ocupar estación elegida
                                      Añadir elegido a la ruta del vehículo
        t = t + 1

```

---

El funcionamiento del algoritmo es más fácil de entender si se comienza desde abajo, describiendo uno a uno los tres bloques que componen el pseudocódigo. El bloque III es el encargado de elegir el siguiente nodo a incorporar en la ruta del vehículo  $k$  en base a los criterios definidos. Para cada vehículo que llegue en el instante  $t$ ,  $Valor$  y  $n\_bicicletas_k$  se inicializan a 0 mientras que  $elegido$  se inicializa a  $\varphi$ , el estado auxiliar que representa el fin de ruta. Sea  $s$  el último estado incorporado en  $Rutas_k$ . Se escoge el estado más prometedor  $\hat{s}$  entre los  $s' \in \delta_s^-$  y se añade a  $Rutas_k$ . Durante el proceso de evaluación se guarda también el número

de bicicletas que se cargarán/descargarán en la estación. Este número se ha calculado previamente multiplicando  $Val_s$  por  $Dist(v_s, v_s)$  para las estaciones de descarga y por  $-Dist(v_s, v_s)$  para las de carga. Finalmente, se añade  $v_s$  a la lista de estaciones ocupadas.

En el bloque II se actualiza el nivel de llenado de todas las estaciones y la función objetivo cuando proceda. Para las estaciones de carga se toma  $Fobj = Fobj + \max\{0, C_{v_s} - C_s + Dem_s\}$ ,  $C_s = \min\{C_s - Dem_s, C_{v_s}\}$ . El restante  $\max\{0, C_{v_s} - C_s + Dem_s\}$  se suma también a  $Ytemp_v$ , almacenando un registro del número de bicicletas que se han tratado de dejar sin éxito por parte de los usuarios desde la última vez que se visitó la estación. Se procede análogamente para las estaciones de descarga.

Ahora es sencillo inferir la definición de  $N_{v_{s^*}}$  para el criterio de urgencia:  $N_{v_{s^*}} = Ytemp_v - Dem_{s^*}^{t_{s'} - t_s}$  (nótese que si la estación es de recogida,  $Dem_s \leq 0$  en todo momento). Para las estaciones de entrega, se procedería multiplicando  $Dem_{s^*}^{t_{s'} - t_s}$  por -1 ( $Ytemp_v$  guarda el número de bicicletas que dejaría el nivel de llenado de la estación por debajo de 0 pero con signo positivo).

El bloque 1 representa la operación que se realiza al llegar a la estación: se cargan en el vehículo las bicicletas calculadas en el bloque III, se resta al nivel de llenado de las estaciones ese mismo valor y se desocupa la estación. El código de AVET se encuentra disponible en el *Apéndice*.

A priori, son tres los inconvenientes inherentes a este algoritmo:

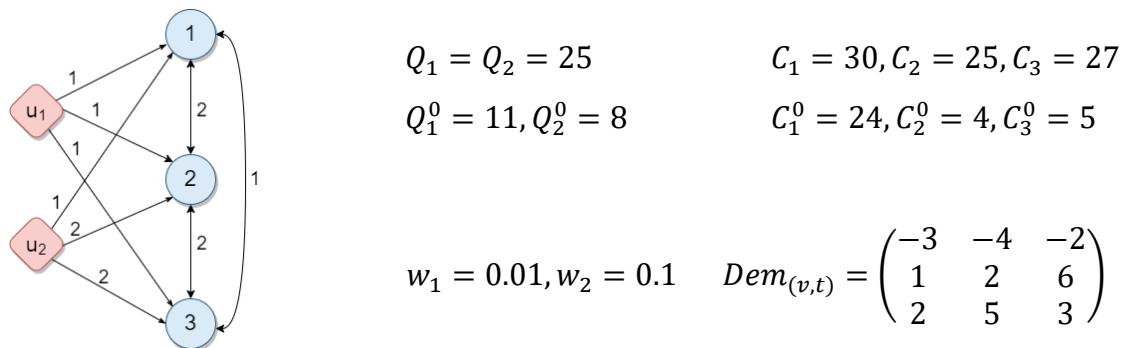
- 1) Si dos o más vehículos llegan en el mismo instante de tiempo, el orden en el que se evalúan puede alterar sus rutas, ya que el nodo elegido por el primer vehículo podría haber sido elegido por uno de los siguientes. Una permutación en el orden de evaluación de los vehículos podría dar mejores resultados.

2) El algoritmo voraz es ‘miope’, es decir, solamente se evalúa en base a la siguiente parada y no con visión general de las posibles soluciones.

3) En caso de empate en la valoración de dos o más candidatos, el siguiente nodo en la ruta del vehículo será el primero que se evaluó.

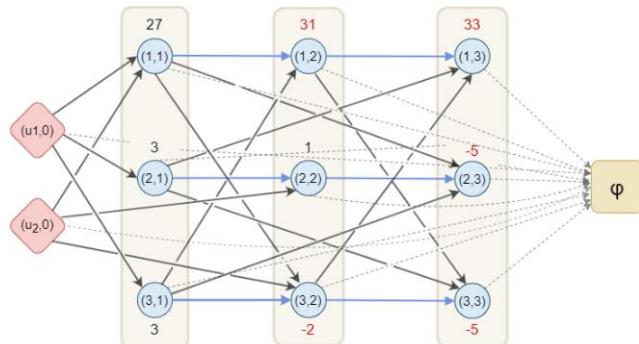
Los problemas 1) y 3) se parecen pues, si simplemente se cambiara el orden de evaluación en la llegada de los vehículos, las rutas podrían acabar por diferir considerablemente.

El siguiente ejemplo ilustra el funcionamiento de AVET y permite visualizar los inconvenientes comentados: Sea la red considerada en la *Figura 3.1*, con los datos iniciales especificados.



*Figura 3.1. Red original y datos iniciales de ejemplo*

La red espacio-temporal resultante es la mostrada en la *Figura 3.2*. En esta figura se incluye el nivel de llenado esperado en cada estado si no se llevara a cabo el reposicionamiento. Los números en rojo simbolizan un nivel de llenado imposible, ya sea por sobrepasar la capacidad de la estación o por dejarla con un número negativo de bicicletas. Para obtener una solución factible, esas bicicletas se acumularían, obteniendo  $Fobj = 20$ .



*Figura 3.2. Red espacio-temporal con niveles de*

El algoritmo procedería de la siguiente forma:

$$1) \quad t=0, Rutas_k = \{(u_k, 0)\}, k = 1,2. \quad Ocupados \equiv \emptyset, Fobj = 0, Ytemp_v \equiv \emptyset$$

Como el último estado  $s$  de  $Rutas_1$  cumple que  $t_s = t$ , se elige su siguiente parada:

- $h_{(1,1)} = \min\{24 + 3,25 - 11\} - 0 * w_1 + 0 * w_2 = 14$
- $h_{(2,1)} = \min\{25 - 4 + 1,11\} - 1 * w_1 + 0 * w_2 = 10,99$
- $h_{(3,1)} = \min\{27 - 5 + 2,11\} - 0 * w_1 + 0 * w_2 = 11$

$$Rutas_1 = \{(u_1, 0), (1,1)\}, Ocupados = \{1\}, n\_bicicletas_1 = 14$$

Como el último estado  $s$  de  $Rutas_2$  cumple que  $t_s = t$ , se elige su siguiente parada:

- $h_{(2,2)} = \frac{\min\{25-4+1,8\}}{2} - 0 * w_1 + 0 * w_2 = 4$
- $h_{(3,2)} = \frac{\min\{27-5+2,8\}}{2} - 0 * w_1 + 2 * w_2 = 4,2$

$$Rutas_2 = \{(u_2, 0), (3,2)\}, Ocupados = \{1,3\}, n\_bicicletas_2 = -8$$

2)  $t=1$  Se actualizan las cargas de los vehículos, el nivel de llenado y se desocupan las estaciones cuando proceda:

- $C_{(1,1)} = C_1^0 - n\_bicicletas_1 = 10, \quad Q_1^1 = Q_1^0 + n\_bicicletas_1 = 25.$
- $C_{(2,1)} = C_2^0 = 4, \quad C_{(3,1)} = C_3^0 = 5$

$$Ocupados = \{3\}$$

Se incluye la demanda y se actualizan  $Fobj$ ,  $Ytemp_v$  si procede:

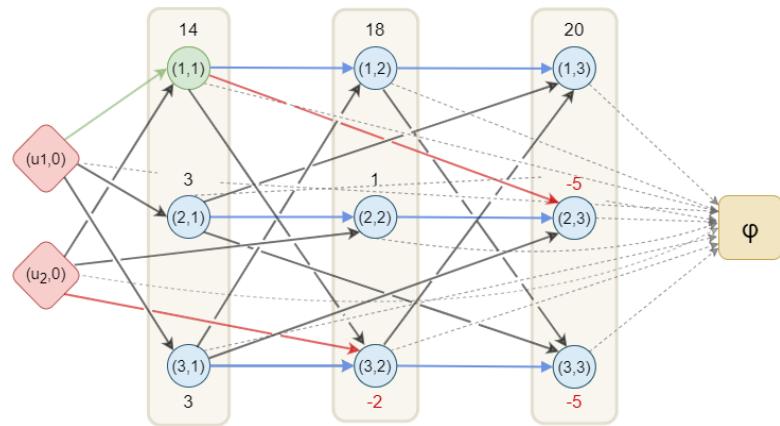
- $C_{(1,1)} = \min\{C_{(1,1)} - dem_{(1,1)}, C_1\} = \min\{10 + 3,30\} = 14$
- $C_{(2,1)} = \max\{C_{(2,1)} - dem_{(2,1)}, 0\} = \max\{4 - 1,0\} = 3$
- $C_{(3,1)} = \max\{C_{(2,1)} - dem_{(2,1)}, 0\} = \max\{5 - 2,0\} = 3$

Como el último estado  $s$  de  $Rutas_1$  cumple que  $t_s = t$ , se elige su siguiente parada:

- $h_{(1,2)} = 0$ , puesto que la estación 1 ya no necesitará ser balanceada.
- $h_{(2,2)} = \frac{\min\{25-3+(2+6),25\}}{2} - 0 * w_1 + 0 * w_2 = 12,5$
- La estación 3 no se considera porque está ocupada.

$$Rutas_1 = \{(u_1, 0), (1,1), (2,3)\}, Ocupados = \{3,2\}, n\_bicicletas_1 = 25$$

El estado actual de la red es el mostrado en la *Figura 3.3*. Nótese que se han eliminado los trayectos que ya no es posible realizar, a excepción de los que comparten estado de salida con los trayectos en curso. Se ha señalado en verde el viaje ya realizado y en rojo los que se encuentran en curso.



*Figura 3.3. Estado de la red en t=1*

3) t=2 Llega el vehículo 2 a (3,2):  $C_{(3,2)} = 11$ ,  $Q_2^2 = 0$ , *Ocupados* = {2}

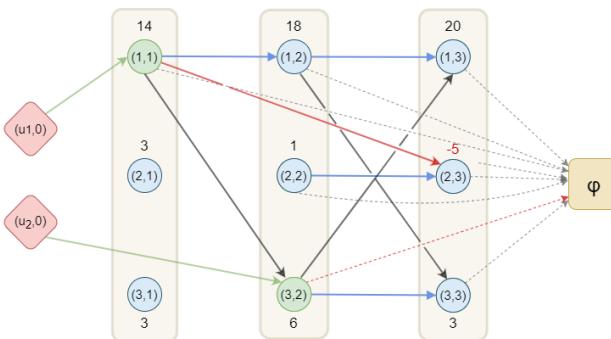
Se incluye la demanda:  $C_{(1,2)} = 18$ ,  $C_{(2,2)} = 1$ ,  $C_{(3,2)} = 6$ .

Como ni la estación 1 ni la estación 3 necesitan ser balanceadas, se acabó el trayecto del vehículo 2: *Rutas*<sub>2</sub> = {(u<sub>2</sub>, 0), (3,2), φ}

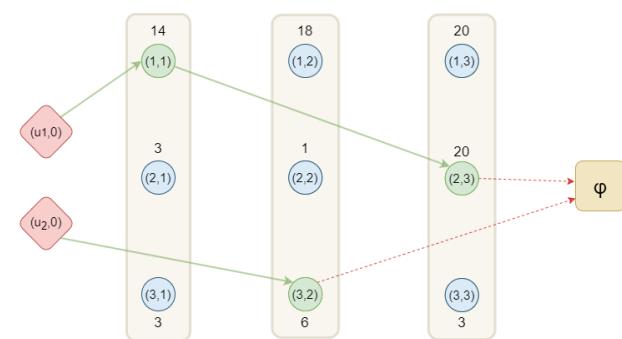
4) t=3 Llega el vehículo 1 a (2,3):  $C_{(2,3)} = 26$ ,  $Q_1^3 = 0$ , *Ocupados* ≡ φ

Se incluye la demanda:  $C_{(1,3)} = 20$ ,  $C_{(2,3)} = 20$ ,  $C_{(3,3)} = 3$

Como se ha acabado el tiempo, se acabó el trayecto del vehículo 1: *Rutas*<sub>1</sub> = {(u<sub>1</sub>, 0), (1,1), (2,3), φ}. Las *figuras 3.4* y *3.5* muestran el estado de la red para los instantes 2 y 3, procediendo de manera análoga a como se ha hecho en *3.3*.

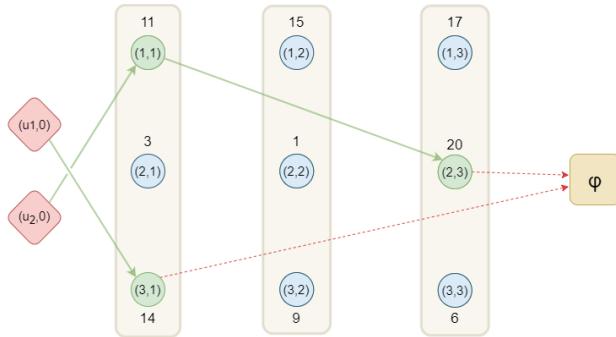


*Figura 3.4. Estado de la red en t=2*



*Figura 3.5. Estado de la red en t=3*

Este ejemplo también sirve para ilustrar alguna de las limitaciones de AVET. Si el primer vehículo evaluado hubiera sido el 2, se hubiera obtenido la solución mostrada en la *Figura 3.6*. Como se puede ver, la estación 1 queda ligeramente más vacía, y la 3, más llena, lo cual es preferible teniendo en cuenta que son estaciones que tienden a llenarse y vaciarse respectivamente. Además, uno de los vehículos se desocuparía un instante de tiempo antes, otorgando más margen de maniobra en un caso donde existieran más estaciones.



*Figura 3.6. Resultado de cambiar el orden de evaluación*

### 3.2. Método PILOT.

Un método apropiado para enfrentarse a los problemas que plantea el algoritmo voraz es PILOT (Preferred Iterative Look ahead Technique), cuyo nombre posee un acertado doble sentido. Por un lado, sus siglas vendrían a significar ‘Técnica Iterativa de Búsqueda Preferente hacia Adelante’ y, a su vez, las siglas forman la palabra ‘piloto’. Su metodología, descrita en detalle por Voss et al. (2005), es la siguiente: La idea principal de PILOT es realizar repeticiones utilizando un procedimiento heurístico como mecanismo de anticipación. En cada iteración de PILOT se calcula tentativamente para cada posible elección una solución llamada “piloto”, registrando los resultados óptimos. De esta manera, empezando de nuevo desde cada solución incompleta que pueda resultar de la inclusión de cualquier elemento aún no incluido en alguna posición de la solución incompleta actual, se inserta en la ruta real el sucesor que devuelva la solución completa temporal con menor coste de la función objetivo.

Así, el procedimiento de construcción PILOT extiende el algoritmo AVET (a partir de ahora AVET-PILOT) presentado en la sección 3 al evaluar cada

candidato de una manera más profunda: para cada posible sucesor se construye una ruta temporal completa y se elige el de menor función objetivo. En otras palabras, este procedimiento metaheurístico consiste en utilizar el valor de la función objetivo como criterio de evaluación para seleccionar la estación candidata con el mayor beneficio obtenido.

A continuación, se describe el *pseudocódigo* de AVET-PILOT. Los bloques I y II son similares a los de AVET, añadiendo una M al final del nombre de todos los parámetros del algoritmo. Estos parámetros representan la solución ‘maestra’, que se actualizan de manera similar a como se ha explicado en AVET tras cada iteración del algoritmo. La diferencia radica en el bloque III, donde se sustituye la valoración del criterio voraz por la función objetivo obtenida por AVET tras completar cada solución temporal.

---

### **Algoritmo AVET-PILOT**

---

```

Iniciar Rutask, OcupadosM, FobjM, Ytempv, tM
while tM ≤ |T|:
    if tM > 0:
        (I)      for k in K:
            if Llega Vehículo:
                Actualizar cargas, desocupar estación, reiniciar Ytempv
        (II)     for v in V:
                Actualizar demandas, FobjM e Ytempv
            if tM < |T| and Llega Vehículo:
                (III)   Iniciar ValorM, elegidoM, n_bicicletasMk
                Igualar Rutask, Ocupados, Fobj, Ytempv, t a sus contrapartes
                for s in δs' ( $s' \equiv$  Último estado en RutasMk):

                    Añadir s a Rutask, ocupar vs y calcular n_bicicletas
                    Ejecutar AVET al completo y obtener Fobj
                    if Fobj < ValorM:
                        Actualizar ValorM, elegidoM y n_bicicletasMk

                    Ocupar estación elegida
                    Añadir elegidoM a RutasMk
    tM = tM + 1

```

---

Para forzar a AVET a considerar cada sucesor  $s \in \delta_s^-$ , se igualan los parámetros de AVET con los de la solución maestra obtenida en el último paso de AVET-PILOT, añadiendo posteriormente  $s$ , (en otras palabras, cada solución temporal parte de la solución maestra añadiendo el candidato a evaluar). Esto se hace añadiendo  $s$  manualmente a  $Rutas_k$ , ocupando la estación  $v_s$  asociada y calculando el número de bicicletas que se cargarán/descargarán cuando el vehículo llegue a dicho estado. Partiendo de ahí se crea la solución completa mediante AVET y se guarda la mejor función objetivo devuelta, añadiendo el estado elegido a la ruta maestra y actualizando de manera definitiva el resto de parámetros asociados.

Aunque PILOT se presenta con el objetivo único de resolver los problemas de corto alcance que conllevan los algoritmos voraces, lo que de por sí ya lo convierte en un método interesante, resulta particularmente conveniente para AVET, ya que permite mitigar en gran medida sus otras dos problemáticas inherentes: Al construir una ruta completa para cada posible sucesor, se resuelve en gran medida el problema del orden de llegada de los vehículos; se prueban todos los candidatos para el primer vehículo y se deriva la solución completa resultante de incluir cada candidato, por lo que se da la oportunidad a los siguientes vehículos de incorporar cualquiera de sus sucesores si a la larga demostrarán mejoría. De igual manera se solventa el problema de los empates en la evaluación de los sucesores, puesto que pasan a probarse todos.

La complejidad estimada de aplicar PILOT sobre AVET es  $O(|K||V||T|)$ , lo que puede llegar a acarrear tiempos de cómputo algo elevados para los casos más grandes. Sin embargo, es posible limitar la profundidad  $\alpha$ , es decir, restringir el número de estaciones sucesivas que se evalúan recursivamente (esto es, se seguirían probando todos los sucesores, pero no hasta formar soluciones completas).

## **4. Experiencia computacional**

En esta sección se presentan los resultados de las pruebas realizadas con varias familias de casos y escenarios diferentes. Para ello, se han generado casos aleatoriamente de distintas dimensiones y se han utilizado casos del SBC Bicimad, en Madrid, de los cuales, tras tratar los datos de utilización de usuarios proporcionados en su página web, se han elegido aleatoriamente subconjuntos de estaciones con el fin de probar el rendimiento del modelo arco-flujo, así como de los procedimientos heurísticos basados en la red espacio-temporal. Estos últimos se pondrán a prueba posteriormente en casos de dimensiones cercanas a las de Bicimad, que incluyen la totalidad de las estaciones del sistema.

Para realizar la experiencia computacional se ha utilizado un procesador Intel Core i7-10750H de 2,60 GHz con 16 GB de RAM. Tanto el análisis de los datos como los algoritmos heurísticos se han implementado con el lenguaje de programación Python. Para resolver el modelo arco-flujo se ha utilizado Gurobipy, el módulo de Python de un potente optimizador comercial que incluye técnicas de preprocessado para agilizar los cálculos y proporcionar robustez al modelo.

### **4.1. Generación de los casos y ajuste de los modelos**

Bicimad es una de las compañías de SBC más prolíficas en España. Ha pasado de tener 173 estaciones a finales de 2019, a tener 264 en la actualidad, con una flota de 2.964 bicicletas y 6.315 anclajes. Además, se planea llegar a las 600 estaciones durante el primer semestre de 2023 (Ayuntamiento de Madrid, 2022). Los datos de utilización de las estaciones de Madrid, publicados en la página web de su ayuntamiento (Empresa Municipal de Transportes de Madrid [EMT], 2022), vienen agrupados por meses y están clasificados en dos tipos: estado de las estaciones y datos de uso. Sendos archivos, de tipo JSON, son mensuales y actualmente se dispone de registros desde abril de 2017 hasta junio de 2021. Para la realización de

la experiencia computacional se han elegido los de junio de 2021. El primer tipo de archivo contiene una actualización del estado de las estaciones cada hora y de él se ha obtenido el ID, nombre, capacidad, estado de llenado inicial y coordenadas de cada estación activa en el sistema. El segundo tipo de archivo contiene actualizaciones de los movimientos de bicicletas, también por horas y a partir de él se ha obtenido una estimación de las demandas de cada estación según la franja horaria y los tiempos de viaje entre estaciones.

Se han generado una serie de instancias con las siguientes características: se consideran diferentes números de estaciones, a saber, 25, 50, 100 y 264 distribuidas en un plano con coordenadas en el intervalo [0, 100] en cada eje. De forma similar, se considera un horizonte temporal invariable de 2 horas, pero discretizado con tres longitudes diferentes: 20 períodos de 6 minutos cada uno, 30 períodos de 4 minutos y 60 períodos de 2 minutos. La idea es probar la sensibilidad de los algoritmos a la longitud de los intervalos de tiempo, denominada *nmin*. Con el objetivo de mantener una demanda independiente de la longitud de los intervalos, en los casos generados aleatoriamente se han considerado valores de la uniforme discreta en el intervalo  $[1, 2nmin - 1]$ , es decir,  $U_d(1, 2nmin - 1)$ , ya que:

$$\begin{aligned} E[U_d(a, b)] &= \frac{1}{b - a + 1} \sum_{i=a}^b i \Rightarrow E[U_d(1, 2 \cdot nmin - 1)] = \frac{1}{2 \cdot nmin - 1} \sum_{i=1}^{2 \cdot nmin - 1} i = \\ &= \frac{1}{2 \cdot nmin - 1} * \frac{(2 \cdot nmin - 1)(2 \cdot nmin)}{2} = nmin \end{aligned}$$

Por ejemplo, para  $nmin = 2$  se tiene que  $E[U_d(1, 3)] = 2$  y como un intervalo de tiempo representa también 2 minutos, en una hora se tiene una demanda media esperada de 60 bicicletas. En otras palabras, como  $E[U_d(1, 2 \cdot nmin - 1)] = nmin$  la demanda esperada de cada estación es de una bicicleta por minuto en todo momento, independientemente de la longitud de los intervalos. El objetivo de diseño de estos casos no es discutir si el volumen o distribución de la demanda de las

estaciones es realista, sino presentar casos con demandas similares independientemente de la longitud de los intervalos de tiempo, de manera que se pueda evaluar de forma directa el impacto de su variación en base a los resultados.

De forma similar, para Bicimad se han tomado las demandas medias de cada estación en cada hora de funcionamiento y se han multiplicado por  $\frac{60}{nmin}$ , redondeando al número entero superior más cercano. Las estaciones del SBC Bicimad se han clasificado como de recogida o entrega en base al signo de su flujo total y en cada intervalo donde la demanda fuera de signo contrario al esperado se ha sustituido ese valor por un 0. Por ejemplo, la estación con  $id = 1$ , de nombre ‘Puerta del Sol A’, con capacidad para 30 bicicletas, con longitud  $\approx 29.50$  y latitud  $\approx 30.68$  presenta los flujos de demanda por hora (desde las 00:00 hasta 19:00) mostrados en la *Tabla 4.1*, que también incluyen la adaptación de las demandas para  $nmin = 6$ , que se realiza como sigue: el flujo total es de 221 bicicletas, por lo que esta estación se considera de recogida y todos los valores negativos pasan a ser 0. Una vez obtenidas las demandas discretizadas (multiplicándolas por  $\frac{60}{nmin}$ ), estas se simulan mediante procesos de conteo de Poisson con parámetro igual al valor de la demanda en cada instante (puesto que su esperanza coincide con este parámetro). Como para  $nmin = 2$  se necesitan 60 intervalos, se generan 3 valores por cada proceso de Poisson. Para  $nmin = 4$  y  $nmin = 6$  es necesario generar 2 y 1 valores respectivamente por cada distribución de Poisson. De esta manera, se simula el comportamiento real del SBC Bicimad (aunque de manera acelerada, al considerar un horizonte temporal de dos horas).

Hora	00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	08:00	09:00
Demanda	-1	-3	6	14	18	20	39	43	46	22
<b>nmin = 6</b>	0	0	1	3	3	4	7	8	8	4
Hora	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00
Demanda	23	-23	-26	-10	38	29	75	34	-40	-83
<b>nmin = 6</b>	4	0	0	0	7	5	13	6	0	0

*Tabla 4.1. Flujo de bicicletas en la estación ‘Puerta del Sol A’ y adaptación*

Las estaciones están distribuidas aleatoriamente o según los datos de Bicimad. En el primer caso, los puntos se generan aleatoriamente en el plano  $[0,100] \times [0,100]$  y las estaciones son alternativamente puntos de recogida o de entrega. En el segundo caso, se adaptan las latitudes y longitudes de las estaciones a coordenadas del plano de la siguiente manera:

$$EjeX_v = \frac{Lat_v - MinLat}{MaxLat - MinLat} \cdot 100 , \quad EjeY_v = \frac{Lon_v - MinLon}{MaxLon - MinLon} \cdot 100 ,$$

donde  $Lat_v$  y  $Lon_v$  representan, respectivamente, la latitud y longitud de la estación  $v$ ,  $MaxLat$  es la latitud máxima registrada entre todas las estaciones de Bicimad, y  $MinLat$ ,  $MaxLon$  y  $MinLon$  se definen de manera análoga. Las coordenadas sirven para poder representar las estaciones en el plano, calcular la distancia inicial de los vehículos a las estaciones y la distancia entre pares de estaciones. Los cálculos de la función distancia discretizada, ya sea para la distancia entre estaciones en el caso de las instancias aleatorizadas, se han calculado mediante la triangulación de las posiciones de los pares de estados:

$$dist_{s,s'} = ceil(hypot(\frac{|EjeX_{v(s)} - EjeX_{v(s')}|}{3 \cdot nmin}, \frac{|EjeY_{v(s)} - EjeY_{v(s')}|}{3 \cdot nmin})),$$

donde  $hypot$  calcula la longitud de la hipotenusa de los valores dados y  $ceil$  el entero superior más cercano. La *Tabla 4.2* muestra cálculos de la función distancia con las distintas longitudes de los intervalos de tiempo y nodos de ejemplo. Los nodos más distantes posibles,  $(0,0)$  y  $(100,100)$ , se consideran a una distancia de unos 30 minutos para las longitudes de 2 y 6 minutos, es decir, a 15 y 5 intervalos de tiempo respectivamente. Sin embargo, como  $nmin=4$  no es múltiplo de 30, con esta longitud de intervalo se considera que el viaje es de 32 minutos, es decir, de 8 intervalos. Estos pequeños desajustes provocados por la discretización del tiempo afectan a la aproximación del modelo al caso real. En el caso de Bicimad, se ha considerado que las distancias eran algo mayores, en base al análisis de los datos de utilización.

$(s, s')$	(0,0), (10,10)	(0,0), (0,100)	(0,0), (50,35)	(0,0), (100,100)
<b>nmin</b>				
<b>2</b>	2	10	7	15
<b>4</b>	1	5	4	8
<b>6</b>	1	4	3	5

Tabla 4.2. Ejemplos del cálculo de  $dist_{s,s'}$

En cuanto a los pesos del AVET, se ha optado por fijar  $w_1 = 0.01$  y  $w_2 = 0.1$ , para el criterio aditivo tras hacer una serie de pruebas iniciales. Con el margen de tiempo  $M_{v_{ss'}}$ , solo se pretende romper posibles empates entre estaciones con más margen que otras, y valores mayores de  $w_1$  provocaban que no se eligieran estaciones que a la larga deberían ser atendidas igualmente, lo que para algunos casos demostró ser contraproducente. En cuanto al número total de bicicletas en desequilibrio  $N_{v_{ss'}}$ , se encontró que este valor de  $w_2$  solía mejorar ligeramente las soluciones con respecto a otros valores probados, aunque se debería estudiar su contribución al criterio.

En relación con el procedimiento PILOT, no se limita la profundidad de AVET-PILOT para los primeros casos considerados, ya que no se van a aplicar procedimientos metaheurísticos adicionales sobre la solución obtenida y los tiempos no son excesivamente elevados. Para las instancias grandes sí se lleva a cabo la siguiente limitación: en lugar de construir soluciones temporales completas, se construyen soluciones con profundidad  $\alpha = \frac{12}{nmin}$ . Este valor de  $\alpha$  se deduce del horizonte temporal,  $T^* = 120$  minutos. Como  $|T| = \frac{T^*}{nmin}$ , se tiene que en realidad  $\alpha = \frac{|T|}{10}$ . De esta forma los candidatos son elegidos en base a una solución temporal proporcional a la discretización del tiempo elegida. Esta versión limitada es llamada AVET-PILOT $_{\alpha}$ .

## 4.2. Resultados computacionales

Para comprobar la eficacia de los procedimientos, se diseñan y realizan cuatro experimentos diferentes. Los resultados del primero están reflejados en la *Tabla 4.3*. En este experimento se compara el menor valor de la función objetivo ( $\bar{z}$ ) obtenido

con los siguientes procedimientos: número total de bicicletas que no podrían ser entregadas o recogidas si no se llevaran a cabo tareas de reposicionamiento (NR), resolver el modelo arco-flujo (AF), incluyendo las costas inferiores (CI) y superiores (CS), y aplicar los algoritmos AVET y AVET-PILOT, mostrando los tiempos de cómputo, expresados en segundos, requeridos por cada uno de ellos (CPU). Para el modelo exacto se utiliza el optimizador MIP de Gurobipy con la configuración por defecto durante un máximo de una hora y las variables se consideran continuas en lugar de enteras, ya que el optimizador no era capaz de obtener una solución factible en el límite de tiempo impuesto. Todos los resultados equivalen a resultados medios tras probar con conjuntos de 10 casos generados aleatoriamente. Los grupos de casos son de 25 y 50 estaciones, con un tamaño de flota fijo de 5 vehículos y las tres longitudes de los intervalos de tiempo propuestas, formando 6 grupos de casos.

<b> K </b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>
<b> V </b>	<b>25</b>	<b>25</b>	<b>25</b>	<b>50</b>	<b>50</b>	<b>50</b>
<b> T </b>	<b>20</b>	<b>40</b>	<b>60</b>	<b>20</b>	<b>40</b>	<b>60</b>
<b>NR</b>	<b><math>\bar{z}</math></b>	2678,7	2708,1	2728,8	5403,3	5436,7
<b>AF</b>	<b>CI</b>	704,7	374,3	126,7	2977,9	-
	<b>CS</b>	953,8	964,5	1182,3	3166,0	4372,1
<b>AVET</b>	<b><math>\bar{z}</math></b>	954,0	817,1	818,8	3199,4	2675,9
	<b>CPU</b>	< 0,01	~ 0,01	~ 0,02	~ 0,02	~ 0,03
<b>AVET-PILOT</b>	<b><math>\bar{z}</math></b>	868,3	660,2	688,5	3095,4	2256,7
	<b>CPU</b>	8,1	14,8	21,7	41,4	109,7
<b>186,7</b>						

Tabla 4.3. Resultados del primer experimento

El modelo AF solo fue capaz de devolver cotas aceptables para los casos con los mayores intervalos de tiempo. Para los casos de mayor dimensión, no fue capaz de resolver el problema relajado, por lo que solo se obtuvo la cota superior y en los intermedios las cotas son muy dispares. Se puede inferir que tanto el número de estaciones como la longitud de los períodos de tiempo tienen un gran impacto en el tiempo de cómputo del algoritmo exacto, siendo este último el factor más influyente. Sin embargo, para poder obtener conclusiones más precisas se necesitaría un tiempo

de cómputo aún mayor. La otra cara de la moneda viene dada por los algoritmos heurísticos: los tiempos de ejecución de AVET son ínfimos y se puede observar que las soluciones obtenidas minimizan la demanda insatisfecha en gran medida si se comparan con las obtenidas al dejar el SBC desatendido. AVET- PILOT realiza un buen trabajo mejorando las soluciones iniciales, con unos tiempos de ejecución bajos, aunque estos van aumentando con bastante rapidez.

En el segundo experimento se consideran longitudes de tiempo 2 y 6, tamaños de flota 5 y 15 en casos generados aleatoriamente (ver *Tabla 4.4*) y basados en Bicimad (ver *Tabla 4.5*) con 100 estaciones. Además, se incluye la desviación típica (SD) de los valores de la función objetivo medios obtenidos por cada algoritmo y se introducen los resultados de  $PILOT_\alpha$ . Con estas pruebas se tiene en mente varios objetivos: estudiar la diferencia en la función objetivo según el número de vehículos disponibles, comparar los casos confeccionados con los casos basados en demandas, distribución y tipo de estaciones reales, y comprobar el comportamiento de PILOT con profundidad reducida.

K	T	NR	AVET			AVET-PILOT			AVET- $PILOT_\alpha$		
			$\bar{z}$	SD	CPU	$\bar{z}$	SD	CPU	$\bar{z}$	SD	CPU
5	20	10953	8543,5	80,3	0,04	8467,2	75,6	167,6	9307,7	101,1	34,1
5	60		6319,7	74,4	0,12	5972,9	63,4	937,6	6911,1	130,6	197,0
15	20		4318,3	73,5	0,12	4213,1	86,1	1427,9	5378,2	138,2	284,4
15	60		1736,2	67,8	0,18	-	-	-	2364,3	153,7	1489,0

*Tabla 4.4. Resultados del segundo experimento – Casos generados aleatoriamente*

K	T	NR	AVET			AVET-PILOT			AVET- $PILOT_\alpha$		
			$\bar{z}$	SD	CPU	$\bar{z}$	SD	CPU	$\bar{z}$	SD	CPU
5	20	1676,0	1247,5	64,7	0,03	1164,9	63,3	65,5	1389,4	92,5	2,1
5	60	2175,5	1661,9	80,5	0,07	1528,4	75,8	152,3	1763,1	133,0	10,7
15	20	1652,3	709,2	72,0	0,07	628,8	73,1	288,5	1253,2	140,8	23,2
15	60	2164,4	876,0	99,9	0,17	734,5	84,6	862,8	1310,2	123,1	106,3

*Tabla 4.5. Resultados del segundo experimento – Bicimad*

Los valores de la función objetivo obtenidos para los casos basados en Bicimad son considerablemente menores que los casos generados aleatoriamente. El principal motivo es que la demanda real de las estaciones es mucho inferior que la supuesta en los casos de prueba. La heterogeneidad de las estaciones del caso real da lugar a casos más variados, lo que explica que las desviaciones sean superiores, en términos relativos, para los casos de Bicimad. En estos últimos, los tiempos de cómputo son considerablemente inferiores. De hecho, no se han tomado registros de AVET-PILOT con los casos generados aleatoriamente de mayor dimensión puesto que el tiempo de cómputo era demasiado elevado como para tenerlos en cuenta (en base a la complejidad  $O(|K||V||T|)$ ), se estima que se tardaría algo más de dos horas por prueba). El motivo de esta acusada diferencia en tiempos de cómputo es que las estaciones reales están, en general, a distancias mayores que las consideradas para los casos generados aleatoriamente, por lo que el número de arcos en casos del primer tipo es mucho menor. Con estas pruebas también se hace notar la disminución del valor de la función objetivo al aumentar el tamaño de flota, especialmente abultada en los casos aleatorios. De nuevo, esta diferencia se debe a las distancias: como las estaciones están más cerca, los vehículos pueden realizar mayor número de operaciones. Si bien  $PILOT_\alpha$  reduce los tiempos de cómputo de manera sustancial, se presenta una problemática con la que no se contaba: las soluciones temporales incompletas no tienen sentido, al menos, no si la profundidad es pequeña, ya que la distancia entre las estaciones suele ser mayor que  $\alpha$ . Esto provoca que las soluciones temporales no tengan margen para evaluar la visita de los vehículos a más estados que el propio nodo candidato (en algunas ocasiones ni siquiera este), lo que hace que frecuentemente se escojan candidatos arbitrariamente. Por esta razón, la variabilidad de las soluciones es muy elevada y en la inmensa mayoría de las ocasiones, lejos de mejorar la solución inicial obtenida por AVET, la empeoran. La única forma de aminorar el problema parece ser

aumentar la profundidad, lo que elevaría los tiempos de cómputo nuevamente. Así, se ha decidido descartar AVET- $PILOT_\alpha$  del siguiente experimento.

La *Tabla 4.6* contiene los resultados del tercer experimento, en el que se prueban los casos basados en el sistema completo de Bicimad, incluyendo las 264 estaciones con las distintas longitudes de los intervalos de tiempo y tamaño de flota 5.

$ T $	NO	AVET			AVET-PILOT		
		$\bar{z}$	$\bar{z}$	SD	CPU	$\bar{z}$	SD
<b>20</b>	4364,1	3763,2	83,7	$\sim 0,05$	3704,3	54,0	402,5
<b>40</b>	4870,3	4071,0	36,7	$\sim 0,1$	4007,5	38,6	1110,9
<b>60</b>	5466,7	4714,7	42,5	$\sim 0,25$	4569,2	35,1	1805,9

*Tabla 4.6. Resultados del tercer experimento – Bicimad al completo*

AVET-PILOT obtiene consistentemente soluciones mejores que AVET, casi un 20% con respecto a la mejora del AVET sobre NR en los casos de menor longitud de los intervalos. Como cabría esperar, los tiempos de ejecución son elevados, alcanzando la media hora en los casos de mayor dimensión considerados. Una observación en cuanto a la longitud de los períodos de tiempo es que, en los casos de Bicimad, disminuirla tiene un efecto negativo en la solución obtenida, mientras que en los casos aleatorios una discretización más demandante suele ser equivalente a una mejora en los resultados. La causa es la estimación de la demanda. Se eligió redondear al número entero superior para evitar valores extremadamente bajos en las discretizaciones más finas provocando así demandas mayores para las mismas.

El cuarto y último experimento recoge los resultados obtenidos tras someter AVET a la aleatorización (AVET-AL) en la elección de los candidatos  $s$  en caso de empate en  $h_s$  y en el orden de evaluación de los vehículos. Para poder comparar resultados, se han realizado pruebas con los casos generados aleatoriamente (*Tabla 4.7*) y basados en Bicimad (*Tabla 4.8*) diseñados en los experimentos anteriores. AVET-AL se ha ejecutado en bucle durante un tiempo máximo de 10 minutos, conservando el mejor valor de la función objetivo obtenido.

<b> K </b>	<b> V </b>	<b> T </b>	5   25   20	5   25   60	5   50   20	5   50   60	5   100   20	5   100   60
<b>AVET</b>			954,0	818,8	3199,5	2310,2	8543,5	6319,7
<b>AVET-AL</b>			891,2	720,5	3101,8	2176,1	8502,7	6012,6
<b>AVET-PILOT</b>			868,3	688,2	3095,4	2093,3	8467,2	5972,9

Tabla 4.7. Resultados del cuarto experimento - Casos generados aleatoriamente

<b> K </b>	<b> V </b>	<b> T </b>	5   100   20	5   100   60	5   264   20	5   264   40	5   264   60
<b>AVET</b>			1287,5	1731,9	3763,2	4071,0	4714,7
<b>AVET-AL</b>			1254,6	1698,1	3749,2	4054,8	4694,3
<b>AVET-PILOT</b>			1164,9	1558,4	3704,3	4007,5	4569,2

Tabla 4.8. Resultados del cuarto experimento - Bicimad

La aleatorización de AVET demuestra mejorar notablemente la solución inicial obtenida en los casos de menor dimensión, pero esta mejoría disminuye a medida que aumenta el tamaño. La variedad de las soluciones devueltas por AVET-AL durante el bucle disminuye a medida que se aumenta el número de estaciones, llegando a obtener un número reducido de soluciones distintas en los casos de mayor tamaño. Tras una serie de pruebas adicionales, se ha concluido que esta limitación en la variedad de las soluciones viene dada por el criterio acumulativo del número de bicicletas  $N_{v_s}$ . Al haber una cantidad de estaciones tan elevada, este valor llega a imponerse sobre las demás partes del criterio voraz, independientemente del orden de evaluación de los vehículos. Se ha probado a eliminarlo y, aunque las soluciones empeoran, se obtiene un número de soluciones distintas considerablemente mayor. Esto demuestra que el impacto de  $N_{v_s}$  es positivo, pero debería ser estudiado en mayor profundidad. La combinación de la aleatorización en los empates del criterio voraz y en el orden de evaluación de los vehículos logra mejorar la solución inicial con aumentar el tiempo de cómputo un poco. Un paso más allá sería tratar de incorporar esta aleatorización en AVET-PILOT, pero para ello primero habría de conseguir limitarlo de manera eficiente.

## 5. Conclusiones

En este trabajo se afronta el problema dinámico de reposicionamiento en sistemas de bicicletas compartidas con un enfoque centrado en la discretización del tiempo. Se propone un algoritmo voraz que consigue soluciones iniciales factibles de manera extremadamente rápida, independientemente del tamaño del sistema. El algoritmo responde bien tanto en los casos generados aleatoriamente como en los casos basados en el SBC Bicimad. Sin embargo, cabe destacar que las distancias y demandas de un SBC real son muy dispares y es difícil encontrar una longitud de los intervalos de tiempo ideal que no provoque desajustes con respecto al caso de estudio. En un SBC de las características de Bicimad, todo parece indicar que no es necesario afinar los intervalos demasiado: con unos 6 minutos o incluso más se permitirá agilizar los cálculos y ajustar mejor la demanda, ya que los datos de utilización de las estaciones suelen venir agrupados en períodos de una hora. De esta forma, se podría considerar un horizonte temporal mayor sin comprometer la rapidez de los procedimientos. Aunque los resultados tras aplicar PILOT no han sido del todo favorecedores, se espera que el algoritmo pueda plantear alguna idea interesante de cara a la implementación de nuevos procedimientos o a la mejora del mismo.

La investigación futura podría probar procedimientos metaheurísticos adicionales, sin descartar PILOT de primera mano, puesto que queda pendiente el estudio de una limitación adecuada en la profundidad, como filtrar de antemano los sucesores a evaluar en cada paso de tiempo, pero seguir construyendo soluciones completas o, en cada paso, añadir estados solo si su inclusión no empeora los resultados de AVET para el mismo instante de tiempo, ya que el empate viene garantizado por el sucesor correspondiente de la ruta encontrada por el algoritmo voraz. En trabajos posteriores también se podría analizar el impacto de los coeficientes del algoritmo voraz en la calidad de las soluciones iniciales obtenidas, proponiendo alternativas o

mejorando los criterios planteados, buscando favorecer la versión aleatorizada de AVET, que logra obtener mejores soluciones factibles iniciales y permite ajustar el tiempo de cómputo tanto como se desee, pero no presenta una mejoría sustancial en los casos de mayor dimensión. Por último, una opción recomendable sería tratar de adaptar AVET a la naturaleza estocástica de la demanda de las estaciones, actualizando tras cada instante de tiempo la predicción del modelo en base a los datos históricos y las circunstancias particulares del caso de prueba, dejando cierto margen en el nivel de llenado de las estaciones puesto que, al suponer conocida la demanda, AVET no se preocupa por dejar estaciones completamente vacías o llenas, de hecho, es lo que busca.

## Referencias

Adham, M. T., & Bentley, P. J. (2015). An ecosystem algorithm for the dynamic redistribution of bicycles in London. In *International Conference on Information Processing in Cells and Tissues* (pp. 39-51). Springer, Cham.

[https://doi.org/10.1007/978-3-319-23108-2\\_4](https://doi.org/10.1007/978-3-319-23108-2_4)

Ayuntamiento de Madrid. (2022). *Almeida transformará BiciMAD por completo con aplicación, bicicletas y estaciones nuevas*.

<https://www.madrid.es/portales/munimadrid/es/Inicio/Actualidad/Noticias>

Bicimad.(2022). <https://www.bicimad.com/>

Bixi.(s.f.). <https://bixi.com/>

Brand, C., Dons, E., Anaya-Boig, E., Avila-Palencia, I., Clark, A., de Nazelle, A., & Panis, L. I. (2021). The climate change mitigation effects of daily active travel in cities. *Transportation Research Part D: Transport and Environment*, 93, 102764. <https://doi.org/10.1016/j.trd.2021.102764>

Brand, C., Götschi, T., Dons, E., Gerike, R., Anaya-Boig, E., Avila-Palencia, I., ... & Nieuwenhuijsen, M. J. (2021). The climate change mitigation impacts of active travel: Evidence from a longitudinal panel study in seven European cities. *Global environmental change*, 67, 102224. <https://doi.org/10.21203/rs.3.rs-149916/v1>

Contardo, C., Morency, C., & Rousseau, L. M. (2012). *Balancing a dynamic public bike-sharing system* (Vol. 4). Montreal: Cirrelt. Retrieved from <https://www.cirrelt.ca/documents/travail/cirrelt-2012-09.pdf>

Chemla, D., Meunier, F., & Wolfson-Calvo, R. (2013). Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2), 120-146. <https://doi.org/10.1016/j.disopt.2012.11.005>

Empresa Municipal de Transportes de Madrid (EMT). *OPENDATA: Datos estáticos BICIMAD*. Recuperado 18 de mayo de 2022, de [https://opendata.emtmadrid.es/Datos-estaticos/Datos-generales-\(1\)](https://opendata.emtmadrid.es/Datos-estaticos/Datos-generales-(1))

Forma, I. A., Raviv, T. & Tzur, M. (2010). The static repositioning problem in a bike-sharing system. In *Proceeding of the 7th triennial symposium on transportation analysis (TRISTAN)*, Tromsø, Norway (Vol. 6, pp. 280-283). <https://tristanconference.org/system/tristan-vii/documents//tristan-vii-details-current.pdf>

Gaspero, L. D., Rendl, A., & Urli, T. (2013). Constraint-based approaches for balancing bike sharing systems. In *International Conference on Principles and Practice of Constraint Programming* (pp. 758-773). Springer, Berlin, Heidelberg.  
[https://doi.org/10.1007/978-3-642-40627-0\\_56](https://doi.org/10.1007/978-3-642-40627-0_56)

Gaspero, L. D., Rendl, A., & Urli, T. (2016). Balancing bike sharing systems with constraint programming. *Constraints*, 21(2), 318-348.  
<https://doi.org/10.1007/s10601-015-9182-1>

Ghosh, S., Varakantham, P., Adulyasak, Y., & Jaillet, P. (2015, April). Dynamic redeployment to counter congestion or starvation in vehicle sharing systems. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 25, pp. 79-87). Retrieved from  
<https://ojs.aaai.org/index.php/ICAPS/article/view/13713>

Gouvernement français. (2019, Julio). 6 bonnes raisons de favoriser le vélo dans vos transports quotidiens. <https://www.gouvernement.fr/actualite/6-bonnes-raisons-de-favoriser-le-velo-dans-vos-transports-quotidiens>

Ho, S.C. & Szeto, W.Y. (2014). Solving a static repositioning problem in bike-sharing systems using iterated tabu search. *Transportation Research Part E: Logistics and Transportation Review*, 69(C), 180-198.  
<https://doi.org/10.1016/j.tre.2014.05.017>

Instituto Geográfico Nacional (IGN). (2022, Junio). *Transporte urbano. Atlas Nacional de España*. [http://atlasnacional.ign.es/wane/Transporte\\_urbano](http://atlasnacional.ign.es/wane/Transporte_urbano)

Kloimüllner, C., Papazek, P., Hu, B., & Raidl, G. R. (2014, April). Balancing bicycle sharing systems: an approach for the dynamic case. In *European conference on evolutionary computation in combinatorial optimization* (pp. 73-84). Springer, Berlin, Heidelberg. [https://doi.org/10.1007/978-3-662-44320-0\\_7](https://doi.org/10.1007/978-3-662-44320-0_7)

Lin, J. R., Yang, T. H. & Chang, Y. C. (2013). A hub location inventory model for bicycle sharing system design: Formulation and solution. *Computers & Industrial Engineering*, 65(1), 77-86. <https://doi.org/10.1016/j.cie.2011.12.006>

Mason, J., Fulton, L., & McDonald, Z. (2015). A global high shift cycling scenario. Retrieved from. <https://ecf.com/files/wp-content/uploads/A-Global-High-Shift-Cycling-Scenario -Nov-2015.pdf>

Meddin R., DeMaio P., O'Brien O., Rabello R., Yu Ch., Seamon J., Benicchio T. (2021). *The Meddin Bike-sharing World Map*. Accessed [DATE]. <http://bikesharingworldmap.com/>.

Nair, R., Miller-Hooks, E., Hampshire, R. C., & Bušić, A. (2013). Large-scale vehicle sharing systems: analysis of Vélib'. *International Journal of Sustainable Transportation*, 7(1), 85-106. <https://doi.org/10.1080/15568318.2012.660115>

Organización Mundial de la Salud (2018). *Plan de acción mundial sobre actividad física 2018-2030. Más personas activas para un mundo sano*. Washington, D.C.: Organización Panamericana de la Salud. <https://doi.org/10.37774/9789275320600>

OpenOrienteeringMap (OOMap). *Bike share map: Global Map of Bikeshare*. <https://bikesharemap.com>

Pfrommer, J., Warrington, J., Schildbach, G., & Morari, M. (2014). Dynamic vehicle redistribution and online price incentives in shared mobility systems. *IEEE Transactions on Intelligent Transportation Systems*, 15(4), 1567-1578. <http://doi.org/10.1109/TITS.2014.2303986>

Raviv, T., Tzur, M., & Forma, I. A. (2013). Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3), 187-229. <https://doi.org/10.1007/s13676-012-0017-6>

Shaheen, S. A., Guzman, S., & Zhang, H. (2010). Bikesharing in Europe, the Americas, and Asia: Past, Present, and Future. *Transportation Research Record*, 2143(1), 159–167. <https://doi.org/10.3141/2143-20>

Szeto, W. Y., Liu, Y., & Ho, S. C. (2016). Chemical reaction optimization for solving a static bike repositioning problem. *Transportation research part D: transport and environment*, 47, 104-135. <https://doi.org/10.1016/j.trd.2016.05.005>

Todd, J., O'Brien, O., & Cheshire, J. (2021). A global comparison of bicycle sharing systems. *Journal of Transport Geography*, 94, 103119. <https://doi.org/10.1016/j.jtrangeo.2021.103119>

Voss, S. Fink, A. & Duin, C. (2005, January). Looking ahead with the pilot method. *Annals of Operations Research*, 136, 285-302. <https://doi.org/10.1007/s10479-005-2060-2>

## Apéndice: Código del algoritmo AVET

A continuación, se plasma el código de AVET precedido por la generación de un caso aleatorio, listo para ser ejecutado en Python:

```
import time
import numpy as np

# Semilla del caso
np.random.seed(60)

# Registro del tiempo
start = time.time()

# Horizonte temporal
T = 60
nmin = 120/T
t = [i for i in range(1,T+1)]

# Vehículos
K = 5
k = [i for i in range(1,K+1)]
Q = 25 # Misma capacidad para todos los vehículos
Q0 = [np.random.randint(10,16) for i in k] # Carga inicial vehículo k

# Estaciones
N = 50
n = [i for i in range(1,N+1)]
C = [np.random.randint(20,31) for i in n]
C0 = [np.random.randint(10,16) for i in n]

# Demanda
dem = {}
tipo={}
for i in n:
    # Clasificamos las estaciones
    tip = np.random.randint(1,3)
    tipo[i] = tip
    if tip == 1:
        for j in t:
            dem[(i,j)] = np.random.randint(1,2*nmin)
    if tip == 2:
        for j in t:
            dem[(i,j)] = -np.random.randint(1,2*nmin)

# Estados
S1 = [(-i,0) for i in k]
S2 = [(i,j) for i in n for j in t]
SS = S1+S2

end = time.time()
print('Estados',end - start)
start = time.time()

# Conjuntos de predecesores y sucesores
Pred = {(i,j):(i,j-1) for i in n for j in t if j > 1}
Suc = {(i,j):(i,j+1) for i in n for j in t if j < T}
```

```

for i in k:
    Suc[(-i,0)]=[]

# Localización de los nodos:
U = np.random.rand(N+K)*20
V = np.random.rand(N+K)*20

# Distancia discretizada entre estaciones (== intervalos de tiempo)
dist ={(i,j): round(np.hypot(((abs(U[i-1]-U[j-1]))//g)+1,((abs(V[i-1]-V[j-1]))//nmin)+1))
       for i in n for j in n if i!=j}
for i in n:
    dist[i,i]=1
    # Se añade distancia vehículo - estación
for i in k:
    for j in n:
        dist[(-i,j)]= round(np.hypot(((abs(U[i]-U[j-1]))//3)+1,((abs(V[i]-V[j-1]))//3)+1))

end = time.time()
print('Distancias',end - start)
start = time.time()

# Cálculo de los arcos
A1 = [(i,j) for i in SS for j in S2 if j[0]!=i[0]
       and j[1]-i[1]>=dist[(i[0],j[0])] and dist[(i[0],j[0])]>j[1]-i[1]-1]
       # Acción de esperar un tiempo en la misma estación
A2 = [(i,Suc[i]) for i in S2 if i[1] < T]
AA=A1+A2

end = time.time()
print('arcos',end - start)
start = time.time()

# Cálculo de deltas (arcos incidentes).
IncidPred = {s:{} for s in SS} # {Arco:[ArcoPred1,ArcoPred2...]}
IncidSucc = {s:{} for s in SS}
for a in AA:
    if a[0] not in IncidPred[a[1]]:
        IncidPred[a[1]].append(a[0])
    if a[1] not in IncidSucc[a[0]]:
        IncidSucc[a[0]].append(a[1])

end = time.time()
print('Incidencias',end - start)
start = time.time()

# CONSTRUCCIÓN AVET
# Inicialización
rutas = {i:[(-i,0)] for i in k}
cantidad = {i:0 for i in k}
Qt = [np.random.randint(10,16) for i in k]
Ct = [np.random.randint(10,16) for i in n]
Fobj = 0
temp = 0
ocupados = []
w1=0.01
w2=0.1
ytemp ={i:0 for i in n}

```

```

# Programa principal
start = time.time()
while temp <= T:
    if temp != 0:

        for i in k: # Operaciones de carga y descarga. BLOQUE I
            if temp != 0 and rutas[i][-1][1] == temp:
                ocupados.remove(rutas[i][-1][0])
                ytemp[rutas[i][-1][0]] = 0
                Ct[rutas[i][-1][0]-1] += cantidad[i]
                Qt[i-1] -= cantidad[i]

        for s in n: # Actualización de las demandas. BLOQUE II
            if tipo[s]==1: # estación de descarga, dem > 0
                Fobj = Fobj + max(0,dem[(s,temp)]-Ct[s-1])
                Ct[s-1] = Ct[s-1] - min(dem[(s,temp)],Ct[s-1])
                ytemp[s] += max(0,dem[(s,temp)]-Ct[s-1])
            elif tipo[s]==2: # est de recogida, dem < 0
                Fobj = Fobj + max(0,Ct[s-1] - dem[(s,temp)] - C[s-1])
                Ct[s-1] = min(C[s-1],Ct[s-1] - dem[(s,temp)])
                ytemp[s] += max(0,Ct[s-1] - dem[(s,temp)] - C[s-1])

        for i in k: # Elección del siguiente estado a visitar. BLOQUE III
            if rutas[i][-1][1] == temp:
                pos = rutas[i][-1] # Se renombra por comodidad
                valor = 0
                elegido = ['dum']
                nbicis=0
                for s in IncidSucc[pos]:
                    if s[0] not in ocupados or s == Suc[pos]:
                        Nvalor = 0
                        demanda = 0
                        mrg = 0
                        invol = 0

                        # Demanda acumulada del candidato en el
                        # momento de llegada
                        for j in range(1,dist[(pos[0],s[0])]+1):
                            if temp+j <= T:
                                demanda+= dem[s[0],temp+j]

                        # Cálculo del margen
                        aux = demanda
                        tp = temp+dist[(pos[0],s[0])]+1
                        if tipo[s[0]]==2: # estación de carga, dem <0
                            while Ct[s[0]-1] - aux < C[s[0]-1] and tp
                            <= T:
                                mrg += 1
                                aux+= dem[s[0],tp]
                                tp+=1
                            # Si ya no hace falta balancear la
                            # estación:
                            if tp > T and Ct[s[0]-1] - aux <= C[s[0]-1]:
                                Criterio = 0
                            else:
                                invol = (ytemp[s[0]]+ Ct[s[0]-1] -
                                demanda - C[s[0]-1])/dist[(pos[0],s[0])]
                                Nvalor = min(Ct[s[0]-1]-demanda,Q-
                                Qt[i-1])/dist[(pos[0],s[0])]
                                Criterio = Nvalor - w1*mrg + w2*invol

```

```

# Procedimiento análogo para estaciones de
descarga
elif tipo[s[0]]==1:
    while Ct[s[0]-1] - aux > 0 and tp <= T:
        mrg += 1
        aux+= dem[s[0],tp]
        tp+=1
    if tp > T and Ct[s[0]-1] - aux >= 0:
        Criterio = 0
    else:
        invol = (ytemp[s[0]]+ demanda -
Ct[s[0]-1])/dist[(pos[0],s[0])]

        Nvalor = min(C[s[0]-1]-Ct[s[0]-
1]+demanda,Qt[i-1])/dist[(pos[0],s[0])]

        Criterio = Nvalor - w1*mrg + w2*invol

    # Se selecciona el mejor candidato
    if Criterio > valor:
        elegido = [s]
        valor = Criterio
        nbicis = int(Nvalor*dist[(pos[0],s[0])])

    # Se actualizan los valores asociados al estado elegido
    s = elegido[0]
    if s == 'dum':
        pass
    elif tipo[s[0]]==2: # estación de carga, dem < 0
        cantidad[i] = -nbicis
    elif tipo[s[0]]==1:
        cantidad[i] = nbicis
        rutas[i].append(s)
        ocupados.append(s[0])
        temp+=1
end = time.time()
print('AVET',Fobj,end - start)

```