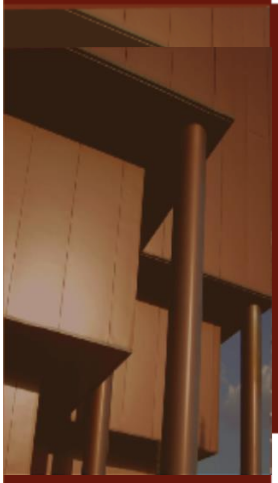
Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Optimización de los procesos de asignación de llamadas en grupos de ascensores de arquitectura Double Deck mediante heurísticas y metaheurísticas: análisis de tráfico *interfloor*

Autor: María de los Angeles Reyes López

Tutor: Pablo Cortés Achedad



Dep. Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Optimización de los procesos de asignación de llamadas en grupos de ascensores de arquitectura Double Deck mediante heurísticas y metaheurísticas: análisis de tráfico *interfloor*

Autor:

María de los Angeles Reyes López

Tutor:

Pablo Cortés Achedad

Dpto. de Organización Industrial y Gestión de Empresas II

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: Optimización de los procesos de asignación de llamadas en grupos de ascensores de arquitectura Double Deck mediante heurísticas y metaheurísticas: análisis de tráfico *interfloor*

Autor: María de los Angeles Reyes López

Tutor: Pablo Cortés Achedad

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia y amigos
A mis profesores

RESUMEN

El transporte vertical de un edificio es una de las instalaciones fundamentales a la hora de diseñarlo. Un diseño incorrecto del núcleo de ascensores puede originar graves problemas a los usuarios y un sobredimensionamiento puede elevar los costes de mantenimiento innecesariamente.

El estudio del transporte vertical de un edificio consta partes fundamentales. La primera, definir el núcleo de ascensores mediante un estudio del tráfico del edificio. La otra parte fundamental es el estudio mecánico y definición de todos los componentes de los ascensores. El presente proyecto se centra en la primera de las partes de este estudio.

Además del dimensionamiento del número de ascensores, este proyecto se centra en estudiar, según ciertos métodos y en ciertas situaciones, con cuál de ellos se consigue mejorar la calidad del servicio, y de esa manera, optimizar el funcionamiento de dichos ascensores.

Palabras clave: Transporte vertical, Ascensor, *Average Waiting Time*, *Average Transit Time*, Búsqueda Tabú, Algoritmos Genéticos, *Estimated Transit Time*.

ABSTRACT

The vertical transport of a building is one of the fundamental installations when designing it. Improper design of the elevator core can cause serious problems for users and oversizing can raise maintenance costs unnecessarily.

The study of the vertical transport of a building consists of fundamental parts. First, define the core of elevators by a study of the traffic of the building. The other fundamental part is the mechanical study and definition of all the components of the elevators. The present project focuses on the first part of this study.

In addition to the dimensioning of the number of lifts, this project focuses on studying, according to certain methods and in certain situations, which one is to improve the quality of the service, and in that way, to optimize the operation of these lifts.

Keywords: Vertical Transport, Lift, Average Waiting Time, Average Transit Time, Tabu Search, Genetics Algorithms, Estimated Transit Time.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

OBJETO DEL PROYECTO

"El hombre nunca sabe de lo que es capaz hasta que lo intenta".

-Charles Dickens-

Es de vital importancia diseñar adecuadamente las instalaciones dedicadas al transporte vertical de un edificio. Y a su vez, que éstas consigan satisfacer las necesidades de los usuarios que van a hacer uso de ellas. Es por ello por lo que se plantea este proyecto.

El principal objetivo de este trabajo fin de grado consiste en determinar el algoritmo más idóneo para optimizar el funcionamiento y la calidad de servicio en ascensores de arquitectura Double Deck.

El citado objetivo global enmarca y genera objetivos específicos, definidos a continuación:

1. Determinación del número óptimo de ascensores según las especificaciones del edificio.
2. Minimizar los tiempos de espera y tránsito de los pasajeros.
3. Minimizar tiempos totales de viaje.

La memoria de este trabajo se divide en 8 secciones. Al final de la misma se encuentran los anexos con gráficas obtenidas a lo largo del estudio y los códigos de edificios y ascensores que se usan a la hora de implementar el algoritmo en C++.

En la primera sección se introduce al mundo del ascensor a través de una pequeña reseña histórica, junto con la tipología de ascensores que se encuentran en la actualidad. En esta sección se describen los ascensores *Double Deck*, que son objeto de estudio en este trabajo.

En la segunda sección se detallan los tipos de patrones de tráfico de pasajeros que pueden darse en los edificios y se explica el sistema de evaluación de la calidad del servicio que se va a emplear para comprobar la bondad de los algoritmos que se programan.

Es en la tercera sección donde se detalla el estudio de la planificación del transporte vertical. Se incluyen los elementos que lo constituyen además del "Estudio de casos" del cual se obtiene un número óptimo de ascensores para cada número de plantas de un edificio. Además, se definen los casos de estudio mediante los algoritmos.

La sección cuarta recoge las restricciones del problema que se estudia, así como una explicación de los tres algoritmos que se van a implementar. Se detallan diagramas de flujos y los pasos que siguen cada uno de ellos para la asignación de ascensores a llamadas.

La quinta sección presenta los dos softwares que se emplean en este trabajo: ELEVATE 8 y Microsoft Visual Studio.

Se explican las características de cada uno de ellos y se detalla el proceso de instalación de la interfaz de desarrollador que permite la conexión entre ambos para realizar las simulaciones.

La sección sexta es probablemente la más importante ya que en ella se refleja el objetivo principal de este trabajo.

Se realiza una experimentación consistente en simulaciones del funcionamiento de los algoritmos implementados en diferentes edificios, que varían su número de plantas y el número de ascensores del que disponen. Se obtienen así los tiempos medios de espera y los tiempos medios de tránsito para cada una de las situaciones. Una vez obtenidos todos los resultados, se comparan entre los tres algoritmos para comprobar cuál de ellos se ajusta mejor a cada una de las situaciones. Para el caso de minimizar los tiempos medios de espera, son los algoritmos genéticos y *Double Deck*-ETA los que consiguen mejores resultados. En cambio, para la optimización de los tiempos de tránsito es la búsqueda tabú el algoritmo que destaca entre los demás. Además, como análisis complementario, se compara el valor del tiempo total de viaje obtenido en cada una de las situaciones para cada algoritmo. Todos estos resultados se expondrán en esta sección.

En la sección séptima se elabora un análisis con las conclusiones que se extraen de la experimentación y estudio del problema.

En el primer Anexo se recogen las gráficas generadas por el software ELEVATE 8 para cada una de las simulaciones realizadas.

En el segundo Anexo se incluyen los códigos de los objetos empleados (edificios y ascensores) para la implementación de los algoritmos en C++.

1 INTRODUCCIÓN

«El científico no tiene por objeto un resultado inmediato. Él no espera que sus ideas avanzadas sean fácilmente aceptadas. Su deber es sentar las bases para aquellos que están por venir y señalar el camino».

- Nikola Tesla –

En esta sección se va a realizar un recorrido por la evolución histórica del ascensor, mostrando los sistemas de transporte vertical de los cuales se disponía en cada una de las épocas.

1.1 EVOLUCIÓN HISTÓRICA DEL ASCENSOR

El transporte vertical ha ido tomando protagonismo a medida que han pasado los años debido a la disminución de los lugares donde poder construir edificaciones, por lo que cada vez, éstas cuentan con más plantas. Es aquí donde el ascensor entra en juego, dada la necesidad por parte de los usuarios de poder acceder a todas las plantas del edificio sin suponer esfuerzo.

A continuación, se describe la evolución de los sistemas de transporte vertical a lo largo de los años.

Hay indicios de ascensores rudimentarios cuya fuerza motriz era la fuerza humana, de animales o de agua allá por el año 300 A.C. Aunque no es hasta el siglo XIX cuando se puede encontrar el ascensor cuyo concepto se tiene actualmente, que era propulsado mediante vapor dentro de los cilindros que movían la cabina.

El periodo grecorromano (siglo X a.C a siglo V d.C) constituye una etapa de gran impulso en la evolución de la tecnología de la elevación. En la antigua Roma el ascensor era ya conocido. Se dispone de una descripción detallada del ascensor instalado en el Palacio de Nerón, el cual se muestra en la Figura 1 [15] y en la época del emperador Tito, en el año 80 d.C., en el Coliseo romano utilizaron también doce montacargas para elevar a los gladiadores.

Tras la caída del imperio romano, los ascensores desaparecieron durante un largo periodo.

Figura 1.- El ascensor del palacio de Nerón

En el periodo de la Edad Media (siglo V d.C. a siglo XVII d.C.) se conocen instalaciones de elevación que apenas se diferencian de las antiguas. El desarrollo del comercio, la navegación y la industria en los siglos XI y XII contribuyó a perfeccionar las máquinas de elevación y a ampliar los sectores de aplicación.

Cuando James Watt inventó la máquina de vapor, fue entonces cuando comenzó a considerarse la posibilidad de utilizar esta forma de energía en los dispositivos de elevación.

En 1845 Sir William Thompson diseñó el primer ascensor hidráulico para elevar cargas y en 1850 se utilizan en Estados Unidos por primera vez montacargas movidos por vapor.

Fue en 1853 cuando Elisha Graves Otis muestra un ascensor con "freno de emergencia", como se muestra en la Figura 2, que evitaba la caída libre en caso de rotura del cable que sostenía la cabina, lo cual lo hacía válido para el transporte de personas sin problemas de seguridad. Elisha demostró la eficacia de su sistema de emergencia instalando un ascensor en el edificio Crystal Palace de New York [22].

Figura 2.- Elisha Graves Otis en la presentación del ascensor en 1853.

Paralelamente en Inglaterra, Frost y Stutt desarrollaron un ascensor con contrapeso que se accionaba mediante tracción, al que denominaron “Teagle” (aparejo para elevación). Gracias a estos avances, se sentaron las bases para la aparición del ascensor seguro.

En el año 1857, se instaló el primer ascensor de pasajeros del mundo, fabricado por Otis Elevator Company, en un hotel de cinco pisos de Broadway. Siendo esto un revulsivo importante ya que, hasta dicha fecha, alojarse en niveles superiores de edificios era inaceptable ya que requería subir demasiadas escaleras con equipaje.

Es a partir de entonces cuando el número de ascensores comienza a crecer, no sin avanzar tecnológicamente. En 1867, Leon Edoux, presentó en la Exposición de París un ascensor con accionamiento hidráulico.

Siemens muestra un ascensor con accionamiento eléctrico en 1880 durante la exposición de Mannheim, aunque no es hasta 1889 cuando Norton Otis desarrolla el primer ascensor eléctrico accionado mediante corriente continua. En 1900 se introduce el motor de inducción para corriente alterna, siendo en 1903 cuando aparecen en Estados Unidos los ascensores con corriente de tracción sin engranajes.

Con el creciente tamaño y altura de los edificios construidos a comienzos del siglo XX, comenzaron a plantearse aspectos como la cantidad, el tamaño, la velocidad y la localización de los ascensores. Fue entonces cuando surgió la tecnología del tráfico vertical.

Desde el siglo XX hasta hoy día, la innovación en los sistemas de ascensores ha avanzado en la mejora del motor eléctrico, en la seguridad y en la rapidez con la que sirven las llamadas. Además, la arquitectura de edificios ha cambiado a lo largo de este tiempo haciendo que el requerimiento de ascensores más eficientes sea necesario, tal como se muestra en la Figura 3 [12].

Cabe destacar varias empresas que a lo largo de la historia han hecho evolucionar el mundo del ascensor, tales como Mitchubishi Electrics, Otis, Kone, ThyssenKrupp o Schindler. Aunque también encontramos otra serie de empresas de más reciente creación que favorecen la competitividad del sector además de la evolución del ascensor, como son MP u Orona.

Figura 3.- Evolución del ascensor

2 SISTEMAS DE TRANSPORTE VERTICAL

"Pregúntate si lo que estás haciendo hoy te acerca al lugar donde quieres estar mañana"

-Walt Disney-

En esta sección se van a presentar cada uno de los elementos que constituyen el sistema de transporte vertical. Entre ellos, se describirán los diferentes tipos de ascensores, los patrones de tráfico existentes y las situaciones en las que se producen cada uno de ellos, se definirá la calidad del servicio requerida por los usuarios y se dimensionará el número de ascensores necesarios para cada edificio, en función de ciertos parámetros que se definirán a lo largo del epígrafe.

2.1 TIPOS DE ASCENSORES

Hay distintos aspectos a tener en cuenta a la hora de diseñar un edificio según el modelo de ascensor que se seleccione. Éstos se pueden clasificar según aspectos técnicos o según aspectos de diseño [20].

- Según su *aspecto técnico*, se pueden diferenciar:

    - **Ascensores autoportantes:** Esta clase de elevadores son muy requeridos en lugares como viviendas unifamiliares, "lofts", salones de fiestas, cines, y son aptos para cualquier instalación que, dada la arquitectura del edificio, deba prescindir de la sala de máquinas. A diferencia del ascensor hidráulico que precisa de un espacio determinado y reglamentario para colocar la central hidráulica, más las cañerías, instalación y tablero de comando, el ascensor autoportante aumenta las posibilidades de una correcta, económica y fácil instalación, ya que la máquina de tracción en su conjunto completo, va colocada en forma estructural dentro del mismo pasadizo, y en su parte superior (cielo de la caja), no siendo necesaria la construcción de la sala de máquinas arriba o abajo, ya que, dada la carencia de espacio del que los edificios citados al principio padecen, se dificulta su construcción. Se puede ver en la Figura 4 [23].

Figura 4.- Ascensor autoportante

- **Ascensores hidráulicos**: Este sistema es el ideal para edificios que no cuentan con posibilidades de modificar las estructuras interiores. Elimina la necesidad de una sala de máquinas superior y la instalación de la misma puede estar hasta 15 metros de distancia del hueco de la vertical del hueco. El esfuerzo del transporte no carga sobre la estructura de la construcción y el desgaste de la maquinaria es menor dado que todo el sistema funciona mediante aceite que es inyectado por una bomba a presión. Este tipo de ascensor es muy seguro en los casos de cortes de energía eléctrica ya que puede ser descendido manualmente quitando presión al equipo mediante una sencilla válvula. No se recomienda su implementación en alturas superiores a los 21 metros. A nivel general estos fueron los tipos de ascensores usados normalmente en las edificaciones modernas. Se puede ver en la Figura 5 [23].

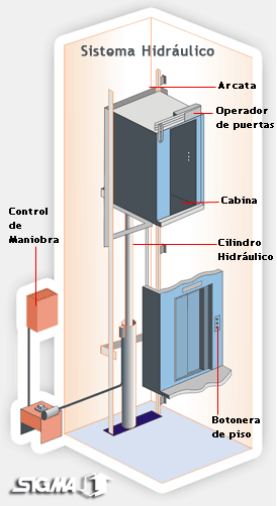


Figura 5. Ascensor hidráulico

- **Ascensores electromecánicos:** Son los más instalados en edificios de viviendas multifamiliares. A diferencia de los hidráulicos, necesitan máquina de tracción en sala de máquinas, ubicadas arriba o debajo de la instalación. Estos ascensores tienen la gran particularidad y funcionalidad de que una sola persona pueda asistir, en caso de persona encerrada, accionando la manivela del freno y el volante del motor (volante de inercia) al mismo tiempo, recordando que en éste u otro cualquier sistema, debe cortarse primero el suministro de energía del ascensor, antes de accionar los mecanismos. Se muestra en la Figura 6 [23].

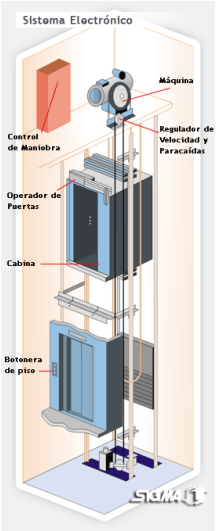


Figura 6.- Ascensor electromécanico

- Según su *aspecto de diseño*, se pueden encontrar:

  - **Ascensores panorámicos**: son un grupo de elevadores que se conciben como un espacio móvil abierto al exterior, un balcón desde el que contemplar el edificio y su entorno, permitiendo una relación visual del pasajero con el ambiente de forma directa, ya sea en el interior del edificio como en el exterior. Figura 7 [8].

Figura 7.- Ascensor panorámico

  - **Ascensores multicabina**: los sistemas de ascensores multicabina son una apuesta del futuro cuyo objetivo principal es que el usuario que llama a un ascensor sea atendido en un tiempo muy breve, gracias a que cada hueco de ascensor habrá más de una sola cabina. El grupo ThyssenKrupp es pionero en este tipo de tecnología y ahora trabaja en un grupo de ascensores que denomina MULTI.
    El sistema MULTI no emplea ruedas convencionales sobre raíles, sino que hace uso de la tecnología de propulsión lineal magnética. El objetivo de este sistema es crear un bucle continuo en el que las cabinas suben por un conducto y bajan, por el contrario. Figura 8 [9].

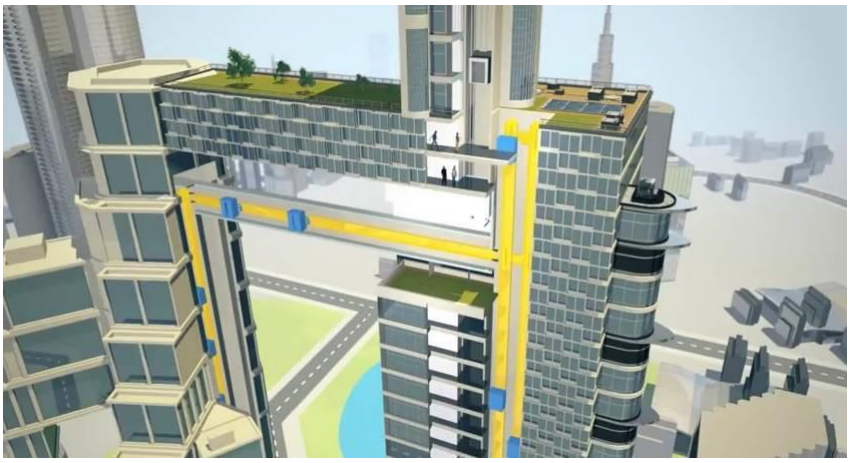


Figura 8.- Ascensor multicabina

- **Ascensores *Double Deck***: Se trata de ascensores con dos cabinas solidarias, una sobre otra, de manera que ambas recorren el hueco del ascensor conjuntamente y permite que los pasajeros de dos pisos consecutivos puedan utilizar el ascensor simultáneamente. Gracias a este sistema, es posible atender y desplazar el doble de pasajeros por cada viaje que se realiza. Además, el tiempo que los usuarios deben esperar para ser atendidos por un ascensor, se va reducido, y el consumo energético es inferior que con ascensores convencionales.

  No todos los ascensores de dos pisos se utilizan para transportar a los pasajeros simultáneamente en ambas cabinas. A veces uno o más ascensores en un edificio tienen estructura *Double Deck*, donde la segunda cabina se utiliza para el transporte de mercancías, por lo general fuera de los períodos de mayor tráfico. Esta técnica tiene las ventajas de evitar daños a los accesorios interiores debido al impacto de los carros, y no requiere un eje dedicado exclusivamente dedicado a una cabina de ascensor de mercancías. Durante las horas punta, el coche se cambia a modo de pasajeros, donde puede acelerar el movimiento de pasajeros dentro o fuera del edificio.

  Se muestran ascensores con esta arquitectura en la Figura 9 [11].

Figura 9.- Ascensor Double Deck

Se puede encontrar gran variedad de edificios que presentan este tipo de ascensores. A continuación, en la Tabla 1, se detallan algunos de ellos:

Tabla 1.- Edificios con ascensores de arquitectura Double Deck

Shanghai Tower: Rascacielos ubicado en el distrito de Pudong en Shanghai. Tiene una altura de 632 metros, con 128 pisos. Actualmente es el edificio más alto de China y el segundo del mundo. Está organizada en base a 9 edificios cilíndricos apilados uno encima de otro. Además de tratarse de uno de los principales centros financieros de Asia, cuenta con espacios comerciales y estación de metro en la planta baja [26].	
Sydney Tower: 309 metros de altura y 17 plantas. Solo cuenta con 3 ascensores. Tiene una capacidad para 960 personas, cuenta con dos niveles de restaurantes, una cafetería, un observatorio y dos niveles de transmisión de telecomunicaciones [13].	
388 Greenwich Street: ubicado en Manhattan, New York. Junto con otro edificio vecino, es la sede de la empresa de servicios financieros Citigroup. También cuenta con un centro de fitness, restaurantes, un centro médico, un centro de conferencias. Es uno de los pocos en New York con ascensores Double Deck. Tiene 39 plantas y una altura de 151 metros [24].	
***20 Fenchurch Street*:** rascacielos de 160 metros de altura y 36 pisos, localizado en la City de Londres. Cuenta con un mirador y un jardín vertical en el piso superior [25].	

Torre Picasso: Es un rascacielos situado en Madrid, junto al Paseo de la Castellana. Cuenta con 45 plantas y 157 metros de altura. En ella se pueden encontrar oficinas, además de un helipuerto en la última planta y 5 plantas de sótano bajo nivel de calle [21].	

- **Ascensores *Twin***: *ThyssenKrupp Elevator* es el primer fabricante de ascensores en implementar la idea de dos cabinas viajando independientemente en un mismo hueco de ascensor. Gracias a un extraordinario trabajo de ingeniería y un avanzado sistema de control, con un concepto de alta seguridad, es posible que operen las dos cabinas de forma independiente, creándose inmensos beneficios potenciales para su uso en nuevas instalaciones y en modernizaciones de edificios.
  El corazón del sistema es un control de selección de destino, capaz de asignar de manera inteligente a cada ascensor las llamadas de los distintos pisos. Cuando un usuario llama a un ascensor desde el pasillo, antes de que el pasajero entre en el ascensor, recoge la información de la planta en la que está y de la planta a la que se dirige y le asigna el ascensor más adecuado para su trayecto. La principal ventaja de este sistema, es que incrementa la capacidad de transporte de los elevadores del edificio, utilizando un menor volumen de construcción y de espacio.
  Solución ideal para edificios entre 50 y 200 metros, *TWIN* también ofrece soluciones totalmente nuevas para el diseño de edificios y la disposición de los grupos de ascensores. Figura 10 [23].

Figura 10.- Ascensor TWIN

Se muestran en la Tabla 2 algunos edificios con ascensores de este tipo:

Tabla 2.- Edificios con asesores de arquitectura TWIN

***Moscow Federation Tower**:* será el rascacielos más alto de Europa (finalización en 2018). Actualmente se están equipando las dos torres del complejo de rascacielos Federation con 22 sistemas TWIN, entre otros. Con una velocidad de hasta 7 m/s se trata de los ascensores más rápidos del mundo de este exclusivo sistema de ascensor del grupo ThyssenKrupp Elevadores y toda una referencia para la capital rusa. La torre A de 93 plantas y una altura de 340m, cuenta con un ascensor convencional y 11 de tipo TWINs, con diferentes velocidades entre 4 y 7 m/s. Y la torre B, de 63 plantas y 242 metros de altura, cuenta con 7 ascensores convencionales y 11 ascensores TWINs [27].	
THYSSENKRUPP QUARTIER, ESSEN: La nueva sede central del grupo ThyssenKrupp AG está equipada con ascensores y escaleras mecánicas especialmente respetuosas con el medio ambiente. Tan solo tiene 50 metros de altura, pero cuenta con 2 sistemas de Panorama TWIN (total 4 cabinas), 14 ascensores convencionales para personas, 2 ascensores para bomberos y 2 montacargas [19].	 
***ROYAL LONDON HOSPITAL**:* El hospital en el centro de la capital británica fue puesto en funcionamiento en 2012. Con 20 plantas, tiene 37 ascensores convencionales, 5 de tipo TWIN, de entre ellos 3 como ascensores monta-camas (6 cabinas en total). Pueden alcanzar una velocidad de hasta 2,5 m/s [19].	

Se van a exponer los aspectos técnicos de los sistemas de transporte vertical, definiendo aquellos parámetros que se encuentran asociados a los procesos de dicho transporte.

En función del tipo de edificio, la afluencia de personas que circulan en él varía. Como es el caso de los edificios de oficinas, donde puede encontrarse el mayor tráfico de personas, y además donde se requiere que el tiempo empleado en el transporte de personas y/o mercancías sea el menor posible, pues no aporta valor al trabajo que se realiza. Debido a la importancia de este tipo de edificios, y con objeto de minimizar estos tiempos, estos serán los tipos de edificios analizados en este estudio.

Para evaluar el tráfico vertical se mide la calidad del servicio, atendiendo a los tiempos de espera medio de los pasajeros (AWT, *Average Waiting Time*) y el tiempo medio de tránsito (ATT, *Average Transit Time*).

2.2 PATRONES DE TRÁFICO

En la Figura 11 se puede observar un patrón de tráfico que se asemeja al comportamiento real de los flujos de pasajeros. En el eje de abscisas se observan las diferentes horas del día donde se produce transporte vertical, y en el de ordenadas la cantidad de tráfico ascendente y descendente en términos de porcentaje total de la población. Se representan los diversos patrones de tráfico con distintos colores [3].

Tradicionalmente se diferencian cuatro patrones de tráfico [1], según el instante del día al que correspondan:

- ***Uppeak*:** Se refiere al tramo horario coincidente con el acceso al edificio, con lo cual, se producen numerosas llamadas desde la planta baja del edificio con carácter ascendente con la finalidad de llegar al puesto de trabajo.

- ***Interfloor***: Se refiere a los tramos horarios en los que existen viajes entre plantas intermedias del edificio. Dentro de este patrón se encuentran otros dos específicos:
  - *Balanced Interfloor Traffic*: tráfico habitual de media mañana o media tarde.
  - *Unbalanced Interfloor Traffic:* tráfico no habitual de media mañana o media tarde. Suele ser debido a alguna reunión de personal.

- ***Lunchpeak***: Se refiere al tramo horario en el que los empleados abandonan el edificio para realizar el almuerzo. Está caracterizado por un intenso tráfico de llamadas descendentes a la hora de dejar el edificio y de llamadas ascendentes al volver del periodo de descanso.

- ***Downpeak****:* Se refiere al tramo horario coincidente con la hora de finalización del turno de trabajo. Se caracteriza por recibir un tráfico muy intenso de llamadas descendentes con destino final la planta baja del edificio.

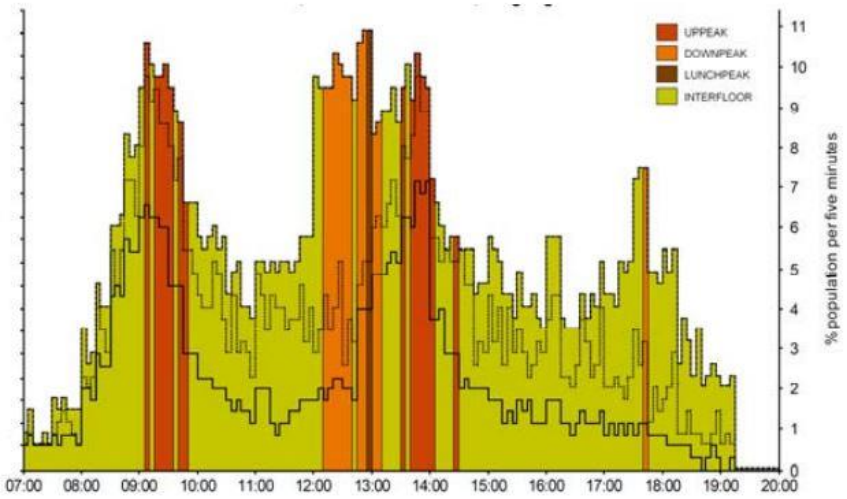


Figura 11.- Patrones de tráfico Siikonen

Puede decirse que el patrón *Uppeak* es uno de los más representativos respecto al número de llamadas de ascensores, ya que la hora de entrada al trabaja provoca gran cantidad de personas accediendo al edificio. Aun así, los horarios de trabajo son más flexibles y la prohibición de fumar dentro del mismo, están haciendo que el patrón *Interfloor* tome protagonismo, centrándose en é el estudio actual para tratar de ofrecer una mejor calidad de servicio.

2.3 EVALUACIÓN DE LA CALIDAD DEL SERVICIO

Aunque actualmente el tráfico *Interfloor* toma gran importancia, se considera que, si se consigue satisfacer la demanda durante el tramo *Uppeak*, se puede satisfacer la demanda de cualquier otro patrón.
Para medir el nivel de satisfacción, se van a tener en cuenta las siguientes Tablas 3 y 4:

Tabla 3.- Calidad del servicio según porcentaje de llamadas respondidas

Nivel de servicio	**Porcentaje de llamadas respondidas en**	
	30 segundos	60 segundos
Excelente	>75%	>98%
Bueno	>70%	>95%
Regular	>65%	>92%
Pobre/Inaceptable	<65%	<92%

Tabla 4.- Calidad del servicio según el tiempo empleado en responder las llamadas

Nivel de servicio	**Porcentaje de llamadas respondidas en**	
	50%	90%
Excelente	20 segundos	45 segundos
Bueno	22.5 segundos	50 segundos
Regular	25 segundos	55 segundos
Pobre/Inaceptable	>25 segundos	>55 segundos

Para medir la calidad del servicio en este trabajo se tendrá en cuenta el tiempo medio de espera que el pasajero se encuentra ante la puerta del ascensor desde que el ascensor ha sido llamado hasta que éste comienza a abrir sus puertas. Este tiempo se denomina con las siglas AWT (*Average Waiting Time*). Existen dos formas de calcular su valor, aunque cada una de ellas presenta ciertos problemas.

Este tiempo podría clasificarse de la siguiente forma en un Edificio de Oficinas, según la Figura 12:

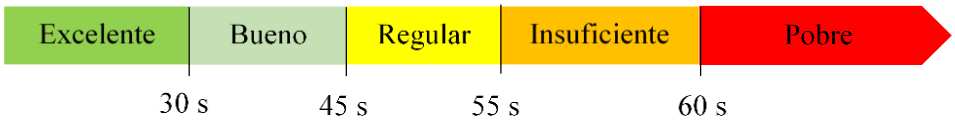


Figura 12.- Clasificación general según tiempos de espera

Otra variable importante a analizar es el tiempo medio de viaje, denominado ATT (*Average Transit Time)*, que es el tiempo transcurrido desde el momento en que se abren las puertas del ascensor y entra el pasajero, hasta que éste llega a su planta destino.

Además de estas dos variables, existen otras como:

- Tiempo Destino (DT), que se define como el tiempo desde que el pasajero registra su llamada hasta que la puerta del ascensor comienza a abrirse en la planta destino. Este podría decirse que es el tiempo de viaje total de un pasajero. Se puede ver la relación de esta variable con AWT y ATT en la Figura 13:

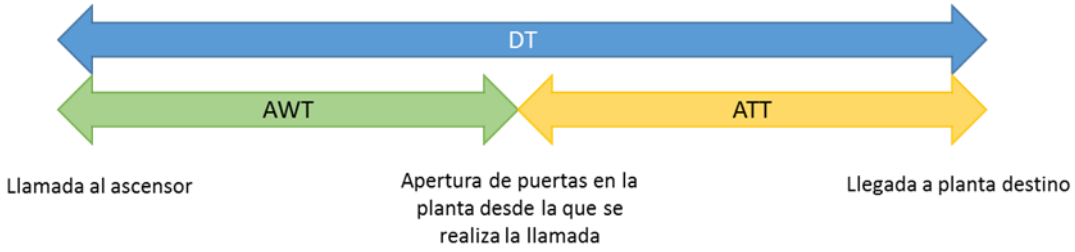


Figura 13.- Relación entre AWT, ATT y DT

- Número de paradas intermedias (IS), es el número de veces que el ascensor para con pasajeros entrando en él y otros llegando a su planta destino. Se pude clasificar la calidad del servicio según las paradas, como se muestra en la Figura 14:

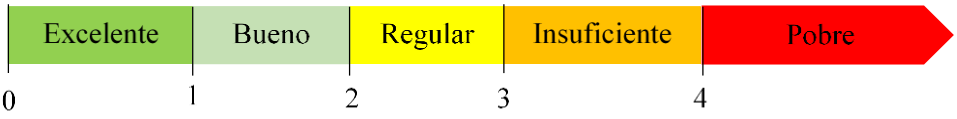


Figura 14.-Clasificación general según el número de paradas intermedias

2.4 DIMENSIONAMIENTO DEL EQUIPAMIENTO Y NÚMERO DE ASCENSORES

Los elementos de transporte vertical de un edificio deben responder perfectamente a las necesidades de este. Sobre todo, es importante tener en cuenta las "horas punta", donde el transporte vertical puede incrementar notablemente.

El conjunto de exigencias de los usuarios del edificio viene en mayor o menor grado definido como "calidad de servicio". Uno de los niveles de calidad más percibido por el usuario es el tiempo medio de espera que una persona debe esperar frente al ascensor para poder viajar en él, además del tiempo empleado hasta llegar a su destino.

Los técnicos de todo el mundo en materia de ascensores se han esforzado en dar una norma internacional que regule y limite los tiempos de espera, fijando unos valores máximos admisibles en cada caso, en función de las características específicas de cada edificio.

Se puede decir que un análisis de tráfico resulta bastante complicado debido a los múltiples parámetros a tener en cuenta, pero es imprescindible para definir una de las instalaciones fundamentales de un edificio, los ascensores [14].

Para realizar un estudio del tráfico, deben tenerse en cuenta parámetros muy distintos como:

- Tipo y uso del edificio
- Número de plantas
- Población por planta
- Distancia entre pisos y recorrido total
- Situación de la planta principal
- Velocidad del ascensor
- Número de pasajeros que se pueden desplazar en la cabina
- Tiempo entre paradas, ajustes o maniobras
- Duración de entrada y salida de pasajeros

En este caso, se va a analizar un edificio de oficinas estándar, en el cual, el tipo de tráfico más común es el Uppeak. El edifico tendrá las características que se detallan en la Tabla 5.

Tabla 5.- Características del edificio

Características del edificio	
Tipo de Uso	Oficinas
Población	50 personas/planta
Número máximo de personas en cabina	11
Distancia entre plantas	3,8 metros
Planta principal	Planta baja
Velocidad del ascensor	2,5 m/s
Tiempo de apertura/cierre de puertas	2 segundos
Tiempo de tránsito de pasajeros	5 segundos

Con todos estos datos, se puede hacer un análisis exhaustivo del número de ascensores que serán necesarios en un edificio, según las diferentes especificaciones.

El cálculo del número de ascensores necesarios en un edificio según sus especificaciones queda definido por la siguiente fórmula [10]:

$$n = \frac{0{,}0015 * h * N}{v * p}$$

siendo:

h: Altura total del edificio
N: Número total de personas en el edificio (Número de personas en una planta * Número de plantas)
v: Velocidad del ascensor en m/s
p: número de personas que pueden viajar en la cabina del ascensor
0,0015: Factor de proporcionalidad.

En función de esta fórmula, se obtienen los siguientes resultados mostrados en la Tabla 6:

Tabla 6.- Número de ascensores en función del número de plantas

Nº plantas	8	10	12	14	20	24	30	32	34
h	30,4	38	45,6	53,2	76	91,2	114	121,6	129,2
Población	400	500	600	700	1000	1200	1500	1600	1700
velocidad (m/s)	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5
Personas cabina	11	11	11	11	11	11	11	11	11
Factor	0,0015	0,0015	0,0015	0,0015	0,0015	0,0015	0,0015	0,0015	0,0015
Nº ascesnores	**1**	**2**	**2**	**3**	**5**	**6**	**10**	**11**	**12**

El software ELEVATE que se utilizará para realizar el análisis no permite la simulación con un número de ascensores superior a 12, por lo que el análisis queda restringido a esa cantidad. Pero, en el caso de que no existiese restricción alguna, la fórmula podría aplicarse para cualquier número de plantas de un edificio.

De dicha forma, se ha conseguido acotar mucho el rango de estudio, teniendo, para cada edificio con su correspondiente número de plantas, un número aproximado de ascensores necesarios.

Según la fórmula, se ha conseguido determinar el número aproximado de ascensores idóneos para cada edificio en función de sus plantas. Pero, para realizar un estudio más completo, se van a estudiar casos próximos a éstos, y que puedan darnos una visión de lo que verdaderamente ocurre en un edificio cuando se cambia la cantidad de ascensores de la que se dispone.

Es por lo que se ha optado a realizar el estudio de los casos que aparecen en la Tabla 7 marcados en naranja:

Tabla 7.- Casos de estudio

Nº Plantas / Nº Ascensores	8	10	12	14	20	24	30	32	34
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									

Una vez realizado el estudio, se podrá comprobar el comportamiento de cada uno de los métodos en cada situación, y determinar cuál es el más idóneo para realizar la asignación de ascensores a llamadas.

La importancia de la realización de un correcto estudio del número de ascensores necesarios para un edificio recae en la dificultad de que, una vez terminada la construcción, puede ser casi imposible corregir la situación. Por tanto, un mal dimensionamiento del número de ascensores implica deficiencias en el diseño del edificio.

3 SISTEMAS DE ASIGNACIÓN DE LLAMADAS

"La inteligencia consiste no solo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica"

-Aristóteles-

En este apartado, se va a describir exhaustivamente el problema a resolver, citando todas las posibles restricciones que éste puede tener. La resolución a este problema que se plantea no es trivial, por lo que será necesaria la utilización de diferentes metaheurísticas que permitan alcanzar, o al menos acercase, a la solución óptima del problema. Se detallarán estas metaheurísticas detalladamente, mostrando además los diagramas de flujo que siguen.

3.1 DESCRIPCIÓN DEL PROBLEMA

El problema a estudiar consiste en la asignación de llamadas realizadas desde las diferentes plantas de un edificio de oficinas a las que se debe asignar una cabina de un grupo de ascensores coordinados. Los pasajeros pueden realizar llamadas de subida y de bajada desde las distintas plantas del edificio. El número de llamadas que realizan los pasajeros dependerá del tramo horario, como bien se observó en los distintos patrones de tráfico del epígrafe anterior. Asimismo, los ascensores del edificio están diseñados mediante una arquitectura del tipo *Double Deck*.

El objetivo principal será asignar las distintas llamadas a ascensores minimizando el tiempo medio de espera de los pasajeros (AWT). Como objetivo secundario, se tratará de minimizar igualmente el tiempo medio de tránsito (ATT).

Para la resolución de este problema, se deben tener en cuenta ciertas restricciones, tanto generales como particulares, las cuales se detallan a continuación [5].

i. Restricciones generales

Se dividen en restricciones explícitas e implícitas.

- Restricciones generales explícitas:
  - Una cabina solo puede servir una llamada en cada instante.
  - El número máximo de pasajeros que se puede servir es la capacidad máxima (en kg) que la cabina puede soportar.
  - Un ascensor no puede parar en una planta a menos que un pasajero quiera subir o bajar del mismo.
  - Un ascensor no puede parar en una planta a recoger pasajeros si ya hay otro ascensor parado en la planta.

- Restricciones generales implícitas
  - Las llamadas realizadas dentro de un ascensor siempre se sirven secuencialmente en la dirección del mismo, es decir, no podrá saltarse ninguna planta de destino de un pasajero.
  - Un ascensor no puede cambiar de dirección mientras lleve pasajeros a bordo, es decir, no podrá cambiar de dirección hasta no servir a todos los pasajeros.
  - Teniendo la posibilidad de subir o bajar, el ascensor preferirá subir.

ii. Restricciones particulares

Estas restricciones dependen de las condiciones específicas del edificio de estudio. Las más habituales son:

- Disponer de, al menos, un ascensor para el transporte de mercancías o para el servicio de minusválidos.
- Servir con una cabina vacía o con pocos pasajeros algunas plantas específicas.
- Disponer de uno o varios ascensores fuera del grupo para viajes especiales.
- Reducir los tiempos de espera en algunas plantas con preferencia.

3.2 OPTIMIZACIÓN EN PROBLEMAS COMPLEJOS

Encontrar la solución óptima de un problema complejo del tipo NP-Duro puede resultar muy complicado e incluso imposible, por lo que es necesario el uso de métodos de resolución que proporcionen resultados que se aproximen al valor óptimo (o alcanzarlo a veces). A este tipo de técnicas se les denomina metaheurísticas.

Dentro de esta clasificación, se pueden encontrar metaheurísticas que se basan en la observación de la naturaleza para su semejanza posterior con el problema a optimizar. Entre ellas se encuentran los algoritmos genéticos o los algoritmos de colonias de hormigas. Otras metaheurísticas se basan en vecindades en las que explorar las posibles soluciones. Entre éstas se encuentran el algoritmo de recocido simulado y la búsqueda tabú".

En este caso, la asignación optima los ascensores de un grupo coordinado a las distintas llamadas de planta efectuadas, para conseguir minimizar el tiempo medio de espera de los pasajeros (AWT) y a la vez, minimizar el tiempo medio de viaje de todos los pasajeros, se considera un problema complejo del tipo NP-Duro.

En este proyecto se va emplear dos de los métodos nombrados anteriormente para resolver el problema de asignación de llamadas de ascensores, concretamente algoritmos genéticos y búsqueda tabú, y se podrán comparar según los resultados obtenidos. También se resolverá el problema por un método heurístico conocido por el acrónimo se sus siglas en inglés: el método ETA (*Estimated Time of Arrival*), comparándose sus resultados con los dos anteriores enfoques.

3.3 CÁLCULO DE LA FUNCIÓN OBJETIVO

La función objetivo devuelve el tiempo esperado para que el grupo de ascensores sirva todas las llamadas solicitadas desde planta y asignadas, englobando así tanto el tiempo de espera por parte del pasajero delante del ascensor, como el tiempo de tránsito dentro del ascensor hasta llegar a su destino. Obviamente, se trata de una estimación debido a la incapacidad de predecir el comportamiento futuro del pasajero. La llegada de pasajeros a los pisos es aleatoria y sus destinos son desconocidos.

Este procedimiento de estimación depende del estado del ascensor (si está subiendo, bajando o parado). Sin embargo, en cada caso se puede calcular el tiempo de función de cuatro valores, que definiremos como P1, P2, P3 y P4.

Para el caso de que el ascensor se encuentre subiendo o parado:

P1: Planta en la que se encuentra el ascensor en el momento que se realiza la simulación.

P2: Piso más alto al que hay que llevar pasajeros con sentido ascendente.

P3: Piso más bajo al que hay que llevar pasajeros con sentido descendente.

P4: Piso más alto entre el piso más bajo y P1 al que hay que llevar pasajeros con sentido ascendente.

$$Fitness = [(P2 - P1) + (P2 - P3) + (P4 - P3)] * tiempo\ estimado\ de\ viaje\ entre\ plantas$$

Se puede ver que se incluye el máximo viaje ascendente conocido, el máximo viaje descendente conocido. Hay que tener en cuenta que la ecuación no incluye los viajes destino de los pasajeros, ya que se desconocen hasta el momento que entran en el ascensor.

Para el caso de que el ascensor se encuentre descendiendo:

P1: Planta en la que se encuentra el ascensor en el momento que se realiza la simulación.

P2: Piso más bajo al que hay que llevar pasajeros con sentido descendente.

P3: Piso más alto al que hay que llevar pasajeros con sentido ascendente.

P4: Piso más bajo entre el piso más alto y P1 al que hay que llevar pasajeros con sentido descendente.

$$Fitness = [(P1 - P2) + (P3 - P2) + (P3 - P4)] * tiempo\ estimado\ de\ viaje\ entre\ plantas$$

En las siguientes figuras puede verse esquemáticamente la elección de los diferentes parámetros de cada una de las ecuaciones, en la Figura 15 para el caso del ascensor ascendiendo o parado y en la Figura 16 para el ascensor descendiendo [4]:

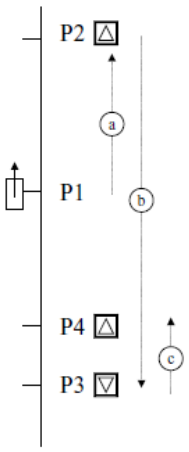


Figura 15.- Ascensor ascendiendo o parado

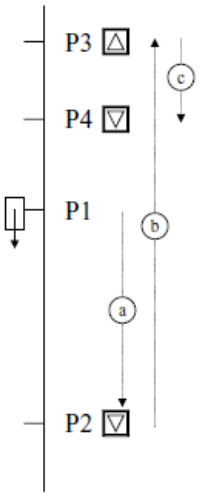


Figura 16.- Ascensor descendiendo

El diagrama de flujo que se sigue para el cálculo del tiempo óptimo es el mostrado en la Figura 17:

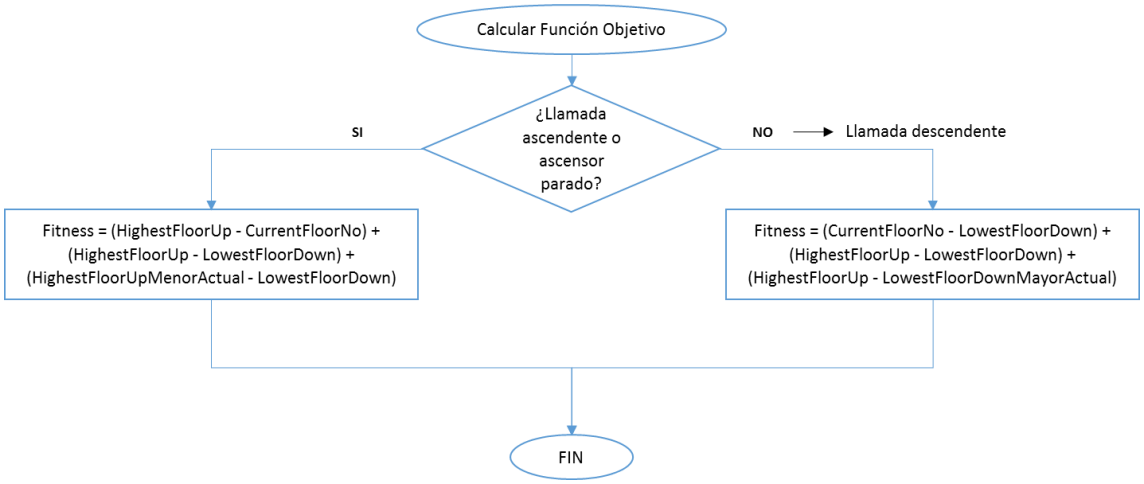


Figura 17.- Función objetivo

Adicionalmente, hay que tener en cuenta una serie de tiempos para calcular el tiempo total del proceso. Todos ellos están asociados a los ascensores e incluye el tiempo empleado por el ascensor durante el proceso de desaceleración, el tiempo de apertura de puertas, el tiempo empleado en la entrada y la salida de pasajeros en la cabina, el tiempo de cierre de puertas y el tiempo de aceleración hasta que el ascensor alcanza su velocidad nominal. Estos tiempos tomarán los siguientes valores:

- Tiempo de apertura de puertas = topen: 2 segundos
- Tiempo de cierre de puertas = tclose: 2 segundos
- Tiempo aceleración = tacel: 2 segundos
- Tiempo desaceleración = tdesacel: 2 segundos
- Tiempo entrada/salida de pasajeros = tp: 5 segundos

Una vez conocidos esos tiempos, tendrán que añadirse al valor del *fitness* por cada parada que realice el ascensor (que se puede conocer gracias al vector que asigna las llamadas a los ascensores).

Por tanto, la función completa que se va a optimizar sería:

$$FO = Fitness + (topen + tclose + tacel + tdesacel + tp) * Número\ de\ Paradas$$

La función objetivo a optimizar será la misma tanto en este método como en el algoritmo genético, que más tarde se expone. Para el caso del algoritmo Double Deck-ETA, ya tiene implementado su propia función objetivo.

3.4 ASIGNACIÓN DE ASCENSORES A LLAMADAS

Una vez ha finalizado el proceso de búsqueda de soluciones, se escoge aquella solución cuya función objetivo sea menos, es decir, haya conseguido minimizar el tiempo de viaje de los ascensores que se van a emplear.

Los ascensores que se van a emplear, asignados por la mejor solución obtenida, se encuentran asociados a cada una de las plantas del edificio a las que van a servir mediante una variable llamada P(j), que toma el valor "i" en caso de que la planta "j" del edificio vaya a ser servida por el ascensor "i". Se guarda la solución de ascensores en las variables "Allocate" y "Allocated" mediante un bucle de tamaño dos veces el número de plantas del edificio en el que si la planta "j" tiene llamada ascendente se guarda en la variable "Allocate" o si tiene llamada descendente en "Allocated", el valor de la variable P(j), es decir, se guarda el ascensor "i" que se asigna a cada planta. En la primera mitad del vector se almacenan las llamadas ascendentes y en la segunda las llamadas descendentes.

Los ascensores empleados tienen arquitectura Double Deck, por lo que la asignación de cabinas no es trivial. Una vez asignado un ascensor a una planta, mediante ciertas variables de Elevate hay que asignar la cabina superior a las llamadas de plantas pares y a la última del edificio ("l[Allocate]d.m_UpLandingCallsUpperCar" o "l[Allocate]d.m_UpLandingCallsUpperCar" igual a 1) y la cabina inferior a las plantas impares y a la primera del edificio ("l[Allocate}d.m_UpLandingCalls" o "l[Allocate]d.m_UpLandingCalls" igual a 1).

Además, si alguna de las cabinas excede el límite de peso permitido, se debe desasignar la llamada a dicho ascensor y obtener una nueva solución.

En la Figura 18 se muestra el diagrama de flujo seguido para la asignación de llamadas a ascensores con arquitectura *Double Deck*.

Esta asignación se realiza de igual manera tanto para el algoritmo de búsqueda tabú, como para el algoritmo genético. *Double Deck*-ETA tiene implementado su propio procedimiento de asignación de llamadas.

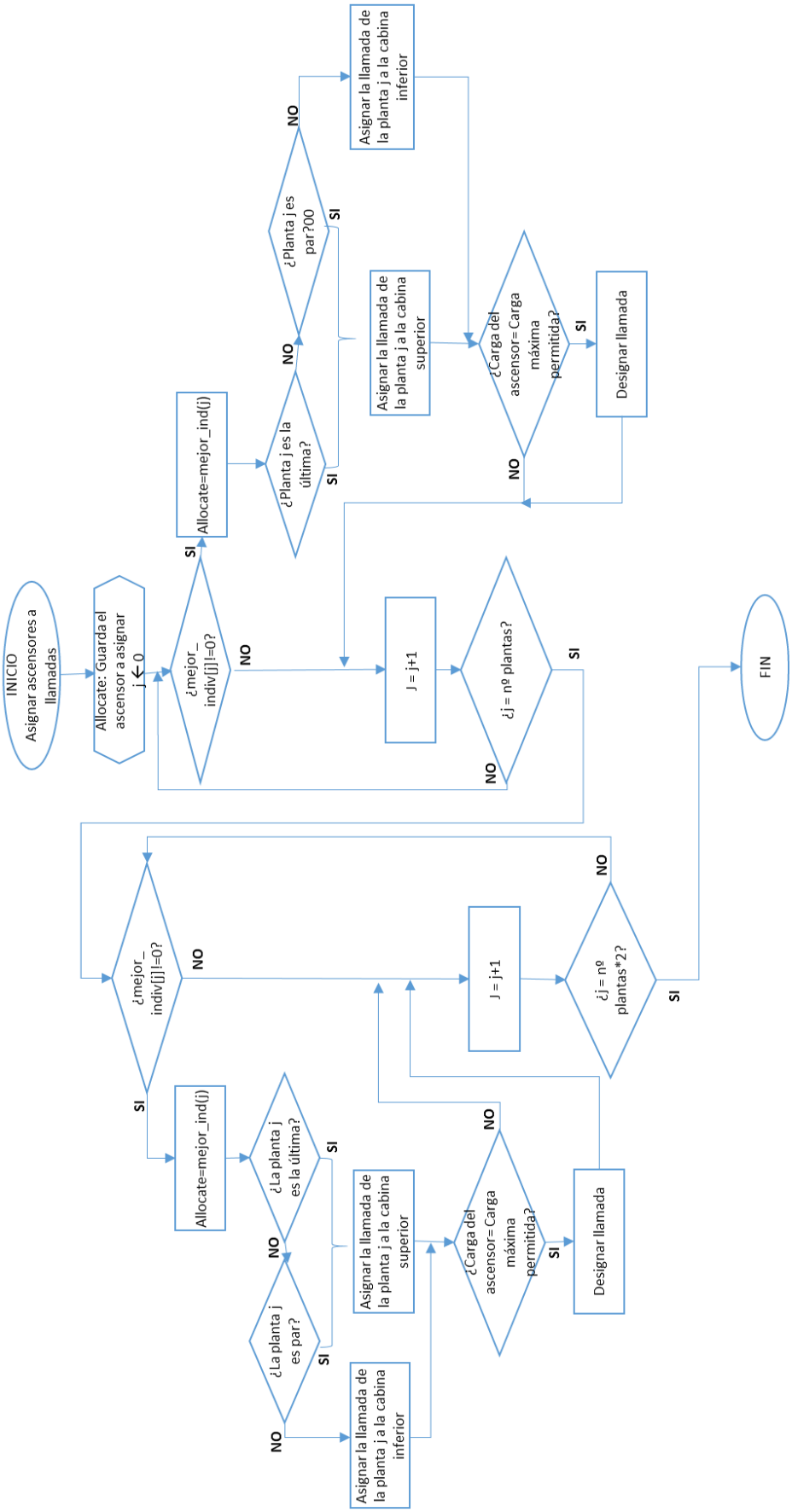


Figura 18.- Asignación de ascensores a llamadas según Búsqueda Tabú

A continuación, se detallan cada uno de los métodos empleados y los diferentes procedimientos a seguir para la correcta aplicación de los algoritmos.

3.5 BÚSQUEDA TABÚ

Proviene del inglés (*tabú search*), la búsqueda tabú es una metaheurística que guía un algoritmo heurístico de búsqueda local para explorar el espacio de soluciones más allá de la simple optimalidad local, según Fred Glover, su primer definidor [6].
Se basa en tres puntos principales:

1. Se usa memoria adaptativa diseñada para permitir evaluar la información de búsqueda histórica.
2. Se usa un mecanismo de memoria que restringe y libera el proceso de búsqueda.
3. Usa memorias en distintos lapsos de tiempo (corto, medio, largo), lo cual ayuda a guardar aquellas características que logran una buena solución, olvidando otras y permitiendo la diversificación. La memoria a corto plazo almacena los atributos de solución que han sido cambiados en el pasado reciente (memoria basada en hechos recientes).

Una solución que se encuentra en la lista tabú no implica que esa solución no sea válida, ya que el método define unos niveles de aspiración con los cuales se evalúan duchas soluciones, y si los resultados obtenidos son considerados como admisibles (por dar mejor solución que la anterior generada), son rescatados de la lista tabú.

Se emplean estrategias de intensificación y/o diversificación. La intensificación favorece la evolución de las soluciones gracias a las combinaciones de movimiento y características de soluciones que históricamente hayan sido buenas. Sin embargo, la estrategia de diversificación busca explorar nuevas regiones a través del cambio de reglas de elección de atributos. Se buscan atributos que no hayan sido usados frecuentemente.

Como todo algoritmo, presenta sus ventajas y sus inconvenientes. Entre los inconvenientes se puede encontrar el problema que presentan también otras metaheurísticas, que es la incapacidad para conocer la calidad de la solución obtenida, ya que el óptimo nunca será encontrado con esta técnica y, por tanto, no se puede comprar la solución obtenida con el óptimo. Además, al no tratarse de una técnica "rígida", es decir, debe confeccionarse para cada problema concreto. Esto puede llegar a ser un inconveniente debido al hecho de que se necesita conocer muy bien tanto la técnica de búsqueda tabú, como el propio problema a optimizar.

Dentro de las ventajas, se encuentra la capacidad de escapar de la optimalidad local. Consigue explorar vecindades y logra evitar el carácter cíclico de otros algoritmos. Además, tiene la capacidad del "olvido estratégico" de soluciones tabú mediante los criterios de aspiración. Otro punto a favor es que permite el empleo de expresiones verbales o simbólicas directamente.

El algoritmo que se implementa en C++, recibe de Elevate las características del edificio, los ascensores correspondientes y el patrón de tráfico a emplear. Una vez el algoritmo procesa la información, obtiene resultados asignando ascensores a cada una de las plantas de las cuales se reciben llamadas. Se detallan a continuación cada uno de los pasos que se llevan a cabo en este método.

A continuación, se muestra en la Figura 19 el diagrama de flujo seguido en el método de la búsqueda tabú:

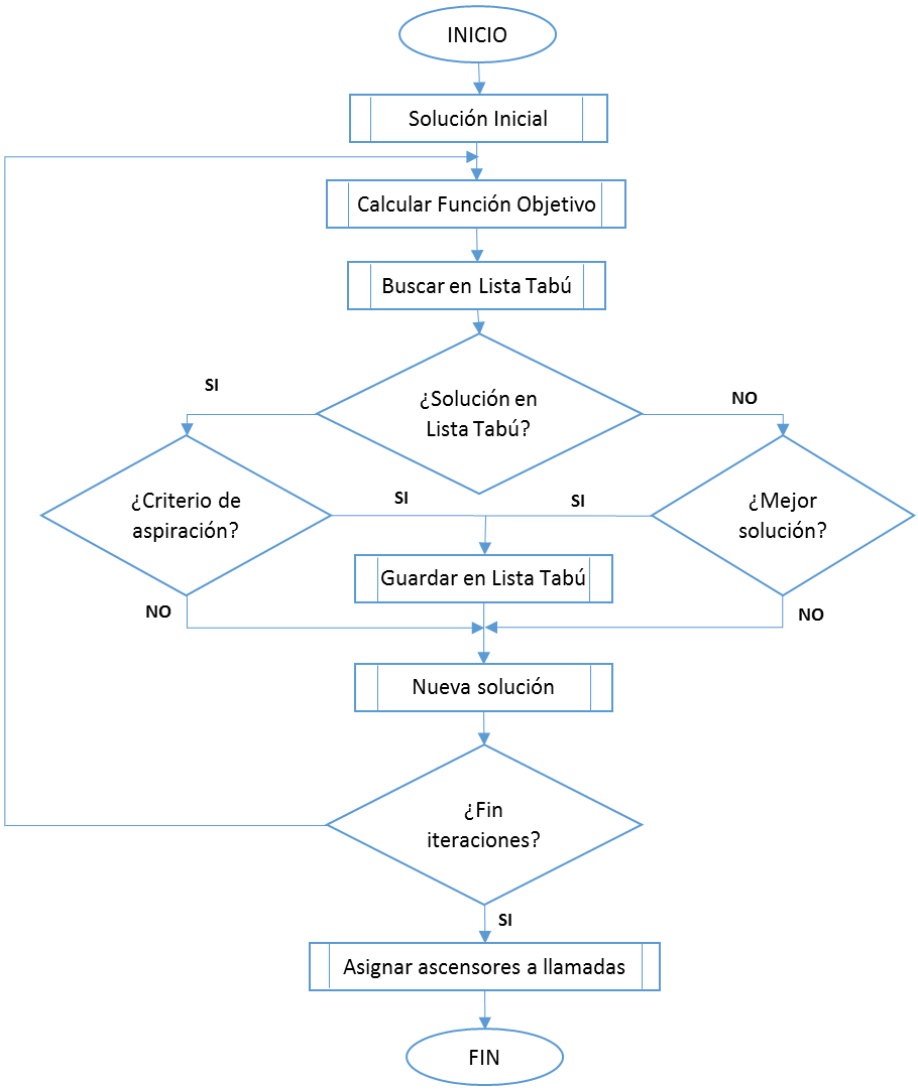


Figura 19.- Diagrama de flujo principal Búsqueda Tabú

3.5.1 Solución inicial

Para poder iniciar el algoritmo, es necesaria una solución inicial de asignación de llamadas a ascensores. Esta asignación se realiza en el vector P(i), cuya dimensión es el número de ascensores. En este caso, se realiza de forma completamente aleatoria, y ya una vez obtenida la solución de esta asignación, el método evoluciona proporcionando una mejor solución. Este vector se rellena de la siguiente forma:

$$P(i) \begin{cases} 1 \text{ en el caso de que el ascensor “i” sea asignado para servir alguna planta aleatoria} \\ 0 \text{ en el caso de que el ascensor “i” no sea asignado para servir alguna planta aleatoria} \end{cases}$$

3.5.2 Calcular la función objetivo

Para calcular el valor de la función objetivo, habría que aplicar lo detallado en el punto 3.3 de esta sección.

3.5.3 Buscar en la Lista Tabú

Para cada solución candidata a ser visitada, una vez se ha obtenido el resultado de la función objetivo, se busca en la Lista Tabú dicha solución para comprobar si ya "se ha visitado" o no.

En el caso de que se encuentre en la Lista:

1. Si la función objetivo está como máximo un 20% por encima de la mejor función objetivo encontrada hasta el momento, se acepta como posible solución.
2. Si la función objetivo está por encima del 20% de la mejor función objetivo encontrada hasta el momento, es descartada como posible solución.

En el caso de que la solución propuesta no se encuentre en la Lista Tabú:

1. Si su función objetivo es mejor que la mejor función objetivo encontrada hasta el momento, se acepta como posible solución.
2. Si su función objetivo es peor que la mejor función objetivo encontrada hasta el momento, es descartada como posible solución.

3.5.4 Guardar en la Lista Tabú

La Lista Tabú es una matriz compuesta por los vectores de ascensores que se emplean y su función objetivo correspondiente, ordenada por filas. El tamaño de dicha lista está designado por el término "Tabú Ténere", que es fijado al comiendo del algoritmo.

Cada nueva solución que se incorpora a la Lista Tabú, se elimina la solución más antigua, siguiendo una tendencia FIFO (Fist In, Fist Out)

3.5.5 Creación de una nueva solución

Esta situación puede darse en tres casos:

1. Se genera una solución que se encuentra en la Lista Tabú pero que no cumple el criterio de aspiración.
2. Se genera una solución que no se encuentra en la Lista Tabú pero no tiene valor de la función objetivo mejor que la actual.
3. Se guarda la solución anterior en la Lista Tabú y se requiere una nueva solución para seguir con la búsqueda de una mejor solución (tantas veces como iteraciones se definan en el algoritmo).

El procedimiento que se sigue para crear una nueva solución consiste en asignar al vector de asociación de ascensores el valor contrario al que tenían anteriormente si se cumple "una verdad aleatoria", es decir, si anteriormente el valor "i" se empleaba (valor 1), si se cumple que un número aleatorio entre 0 y 1 es mayor que ½, el ascensor "i" pasa a no emplearse (valor 0). De igual modo ocurre con aquellos ascensores que no se empleaban (valor 0).

g(i)

- Valor contrario al que tenía anteriormente si se cumple que un número aleatorio (entre 0 y 1) es mayor que ½.
- Mismo valor que tenía anteriormente si el número aleatorio (entre 0 y 1) es menor que ½.

3.5.6 Asignación de llamadas a ascensores

Para llevar a cabo este paso del método, se detalla el procedimiento a seguir en el apartado 3.4 de esta sección.

3.6 ALGORITMOS GENÉTICOS

Los algoritmos genéticos son una técnica de búsqueda basada en la teoría de la evolución de Darwin, basado en los mecanismos de selección natural y genética natural. Combina la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana. Básicamente, en cada generación se crea un conjunto de nuevos individuos utilizando solo aquellas partes adecuadas del progenitor.

Por tanto, se partería de una población compuesta por individuos y éstos a su vez por genes. Entre los individuos de la población se producen cruces que dan lugar a otros individuos. Cada individuo tiene asociado "una puntación" relacionado con la bondad de una solución factible al problema, es decir, el grado de efectividad de dicho individuo para sobrevivir. Una vez producido el cruce, uno de los tres individuos ("padre1", "padre2" e "hijo") debe salir de la población. Tendrá más posibilidades de salir el más débil, o lo que es lo mismo, el que presente una bondad de solución peor.

La novedad que incluyen los algoritmos genéticos es que explotan la información histórica para especular sobre nuevos puntos de búsqueda, esperando un resultado mejorado.

Algunas de las ventajas a mencionar de este método serían:

- No necesitan conocimientos específicos sobre el problema que intentan resolver.
- Se trata de un algoritmo muy versátil, es capaz de adaptarse a cualquier tipo de problema de optimización.
- Operan de forma simultánea con varias soluciones.

Por otro lado, sus limitaciones:

- Pueden tardar mucho en converger, o no converger, dependiendo en cierta medida de los parámetros que se utilicen (tamaño de la población, numero de generaciones…)
- No garantiza encontrar la solución óptima, aunque sí soluciones de un nivel aceptable en un tiempo competitivo con el resto de algoritmos de optimización.

El diagrama de flujo seguido por los algoritmos genéticos sería el que se representa en la siguiente Figura 20:

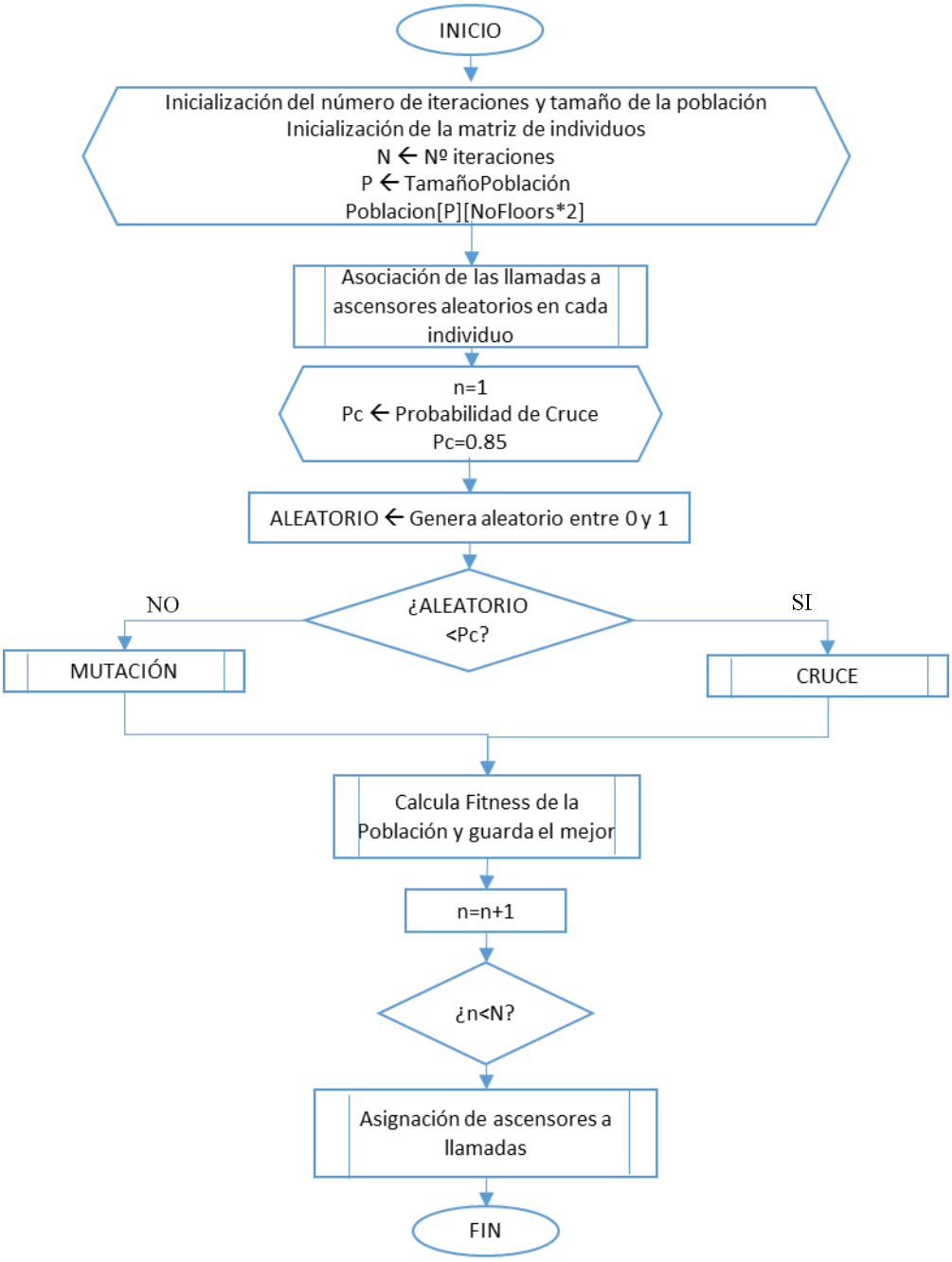


Figura 20.- Diagrama de Flujo Algoritmo Genético

3.6.1 Elementos de los Algoritmos Genéticos y su traducción biológica

En los algoritmos genéticos, el término *cromosoma* se refiere a un candidato a la solución del problema. Se podría seguir profundizando dentro de este término, subdividiéndolo en *genes* y *alelos* (corresponderían a los bits y los valores de estos bits, respectivamente), pero en este caso no se llega a tal nivel de detalle [17].

El *cruce*, que biológicamente se corresponde con el intercambio de genes entre cada par de cromosomas en cada padre para formar un gameto, en el algoritmo consiste en el intercambio de material genético entre dos cromosomas de dos padres, dando lugar así a un hijo.

La *mutación*, que biológicamente se correspondería con el cambio de algún nucleótido de padre a hijo, en el algoritmo se trata de una permutación en una posición en un lugar aleatorio, por otro valor también escogido aleatoriamente.

3.6.2 Solución inicial

Al tratarse de una analogía directa del comportamiento de organismos vivos, precisan su inicialización partiendo de una población inicial. Dicha población está compuesta por individuos, y estos a su vez por sus genes.

Cada individuo está compuesto por genes:

$$Nº\ genes = (Nº\ plantas\ del\ edificio) * 2$$

La primera mitad de los genes recogerán las llamadas ascendentes, y la segunda las descendentes.

3.6.3 Cruce de individuos

En primer lugar, se define la probabilidad de cruce, Pc, con valor 0,85. Una vez conocido esto, pueden darse dos situaciones, el cruce entre dos individuos con una probabilidad de Pc o mutación con probabilidad (1-Pc).

Para el caso de cruce, se seleccionan dos individuos al azar para que sean los padres, cuyo hijo heredará un gen de los padres con un 50% de probabilidad de cada uno. Una vez creado el hijo, se calcula el *fitness* de ambos padres y del hijo, y mediante la función "*Wheel Roulette*" se otorga a cada uno de ellos la probabilidad de salir de la población en función de su bondad de solución (cuanto peor sea su fitness, otorgará una probabilidad mayor de abandonar la población). Este procedimiento se observa en la Figura 21.

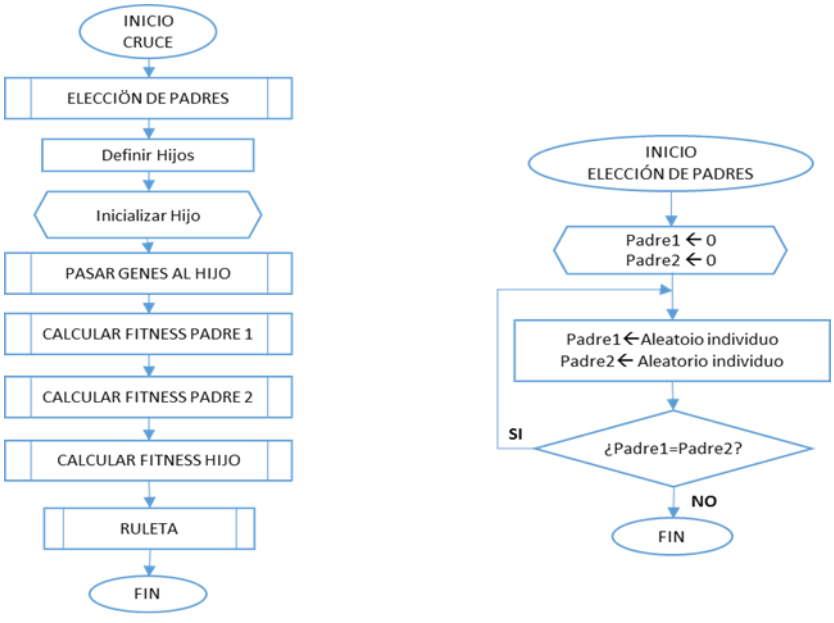


Figura 21.- Diagrama de Flujo del proceso de cruce

En la función "*Wheel Roulette*" se guardan en primer lugar los *fitness* de los padres y el hijo (fitpadre1, fitpadre2 y fithijo, respectivamente), normalizándolos. Si los elegidos para salir de la población son el padre1 o el padre2, el hijo los sustituye. En cambio, su el elegido para salir es el hijo, todo continúa igual. Esto se observa en la Figura 22.

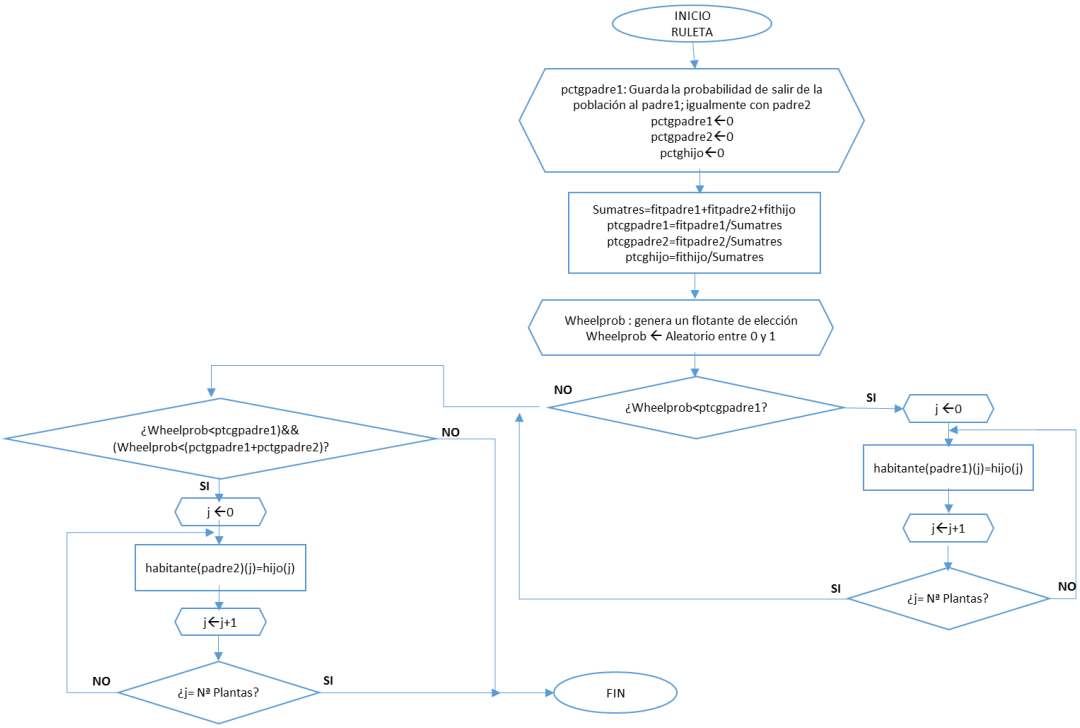


Figura 22.- Diagrama de Flujo del Proceso de Ruleta

En la Figura 23 se detalla el proceso de Pasar genes al hijo desde la cadena genética de los padres.

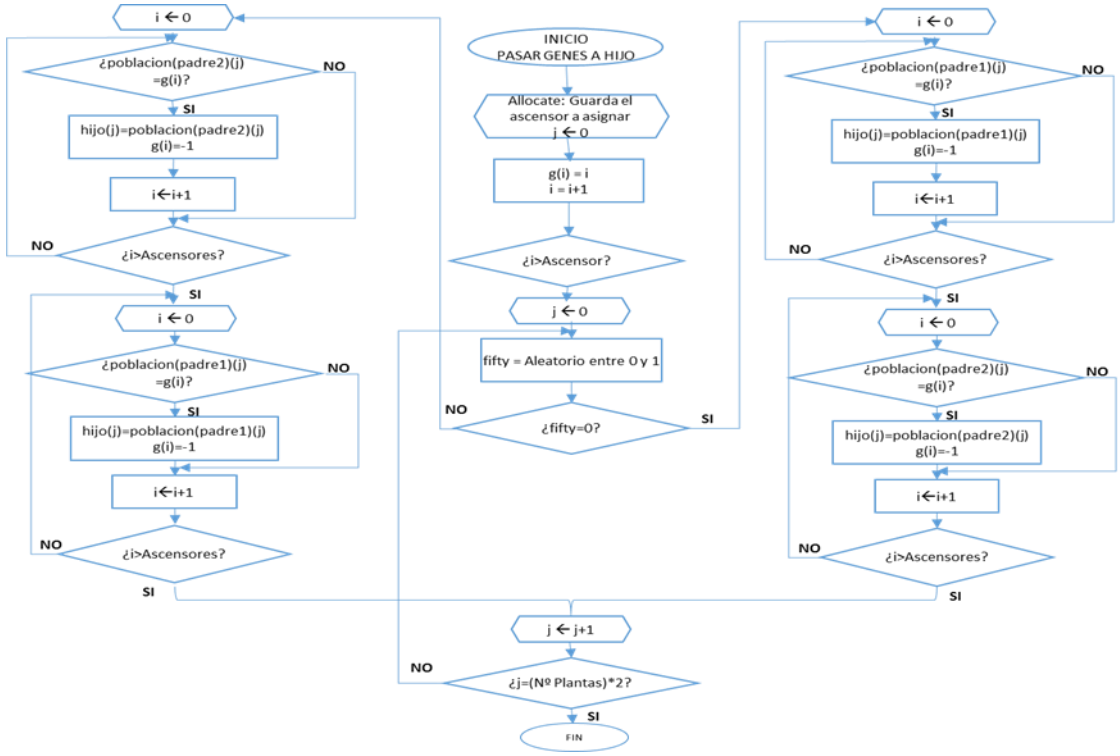


Figura 23.- Diagrama de Flujo del proceso de Pasar genes al hijo

3.6.4 Mutación de individuos

La probabilidad de mutación es mucho menos que la de cruce porque es la realidad de lo que ocurre en la población (siendo 1-0,85=0,15).

Cuando se da este caso se selecciona al azar uno de los individuos y se cambian la posición de dos llamadas asignadas.

A continuación, se presenta el diagrama de flujo del proceso de mutación en la Figura 24.

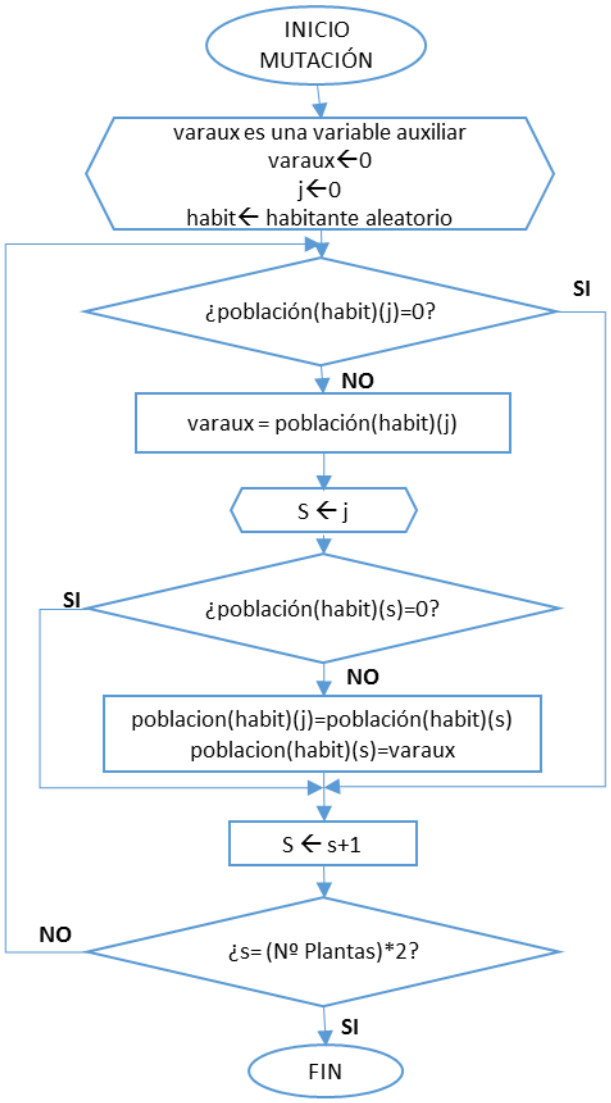


Figura 24.- Diagrama de Flujo del proceso de Mutación

3.6.5 Calcular la función objetivo

Para el algoritmo genético, se va a utilizar la misma función objetivo que en la búsqueda tabú, con la peculiaridad, que se calcula un fitness para cada uno de los padres y para el hijo de ellos, y en función de este *fitness* se estudiará su bondad a la solución del problema y se eliminará o mantendrá en la población.

El diagrama de flujo del cálculo de la función objetivo está ilustrado anteriormente en la Figura 17, en el punto 3.3 de esta sección, donde se redacta lo referente a la función objetivo.

La única diferencia es que, para este algoritmo, se calcula el *fitness* para cada uno de los padres y para el hijo, se elige la mejor solución (menor tiempo) para asignar ascensores a llamadas, y para la siguiente iteración, se deshecha el individuo con peor Fitness y se continúa el método con los dos individuos restantes. Si el peor Fitness es el de hijo, se mantienen los padres igual para la siguiente iteración; en el caso de que sea uno de los padres el que obtenga peor Fitness, el hijo ocupará su lugar para la siguiente iteración.

3.6.6 Asignación de llamadas a ascensores

El diagrama de flujo de la asignación de llamadas a ascensores está ilustrado anteriormente en la Figura 18, en el punto 3.4 de esta sección, donde se redacta lo referente a la función objetivo.

3.7 ALGORITMO HEURÍSTICO DOUBLE DECK - ETA

El algoritmo *Double Deck*-ETA (*Estimated time of arrival*) se basa en asignar el ascensor que tarda menos tiempo en llegar a la llamada realizada. Este algoritmo se encuentra implementado de forma interna en ELEVATE, y se utiliza concretamente para los ascensores de estructura *Double Deck* [16].

En la Figura 25 ilustra el funcionamiento de dicho algoritmo mediante un ejemplo:

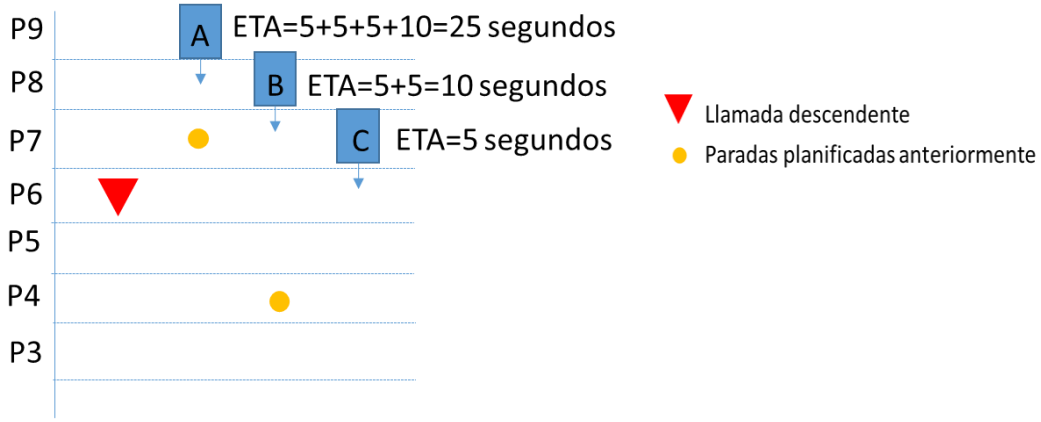


Figura 25.- Double Deck-ETA

Así, en el ejemplo de la Figura 25, se considera un tiempo de viaje entre plantas de 5 segundos y un tiempo de tránsito de pasajeros de 10 segundos (5 segundos entrada y 5 segundos salida). Según esto, el ascensor A, al encontrarse 3 plantas por arriba tardaría en servir la llamada (5+5+5) segundos de viaje de las 3 plantas más 10 segundos de tránsito de pasajeros en la planta 7, donde ya tenía una parada planificada. El ascensor B llegaría a la planta 6, 10 segundos. Y, por último, el ascensor C, solo le tomaría 5 segundos, al encontrarse tan solo una planta por arriba y sin ninguna parada planificada. Por tanto, el algoritmo optaría por elegir al ascensor C como el más idóneo para servir la llamada de la planta 6, consiguiendo así minimizar el tiempo estimado de llegada a la llamada.

4 IMPLEMENTACIÓN DEL SOFTWARE

"Innovar es encontrar nuevos o mejorados usos
a los recursos de que ya disponemos"

- Peter Drucker -

Para la implementación del algoritmo es necesario el uso de dos software. Uno de ellos es ELEVATE 8, software de simulación desarrollado por Peters Research Ltd, en el que se pueden seleccionar el número, tipo y velocidad de ascensores, al igual que las características específicas del edificio (pisos, altura de los pisos, número de personas…). Por otro lado, el software Microsoft Visual Studio 2008, en el cual es programado los algoritmos que se van a utilizar para resolver el problema en lenguaje C++.

Además, es necesario un archivo que permite obtener la interfaz de desarrollador, que se encuentra en la misma página web de ELEVATE. Sin esta interfaz sería imposible conectar ELEVATE 8 y Microsoft Visual Studio 2008.

Cabe destacar que la versión de Microsoft Visual Studio a descargar sea la de 2008 ya que, además de ser la recomendada por ELEVATE, se comprueba que es la única que no genera problemas de compatibilidad durante las pruebas.

A continuación, se detalla la instalación y características específicas de cada uno de los software.

4.1 ELEVATE 8

Tras la instalación de ELEVATE 8 y la inicialización de programa, se muestra una pantalla inicial que se puede observar en la Figura 26.

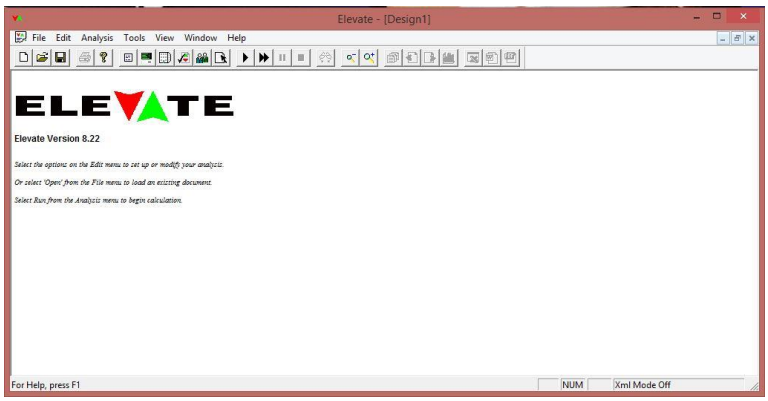


Figura 26.- Pantalla principal ELEVATE

4.1.1 Ventanas de configuración de ELEVATE

Para configurar las características de los ascensores y del edifico que se va a simular, hay que pinchar en el botón de Analysis Data, el cual se encuentra en la barra de herramientas de la pantalla inicial de Elevate.

- *Analysis Data*

  Se despliega una primera ventana en la cual se seleccionan los parámetros referentes al método y el tiempo de simulación.

  En esta ventana se debe seleccionar en *Analysis Type* la opción de *Simulation*, y en *Dispacher* la opción de *Custom*, para poder obtener los algoritmos de la interfaz de desarrollador. De igual forma, se puede seleccionar el número de intervalos de tiempo antes de mostrar los resultados de la simulación (*No o simulations to run for each configuration*) y la duración de estos intervalos. También se podrán realizar simulaciones que incluyan el modelo energético (*Energy model*), aunque en este caso no se utilizará. Se muestra esta pantalla en la Figura 27.

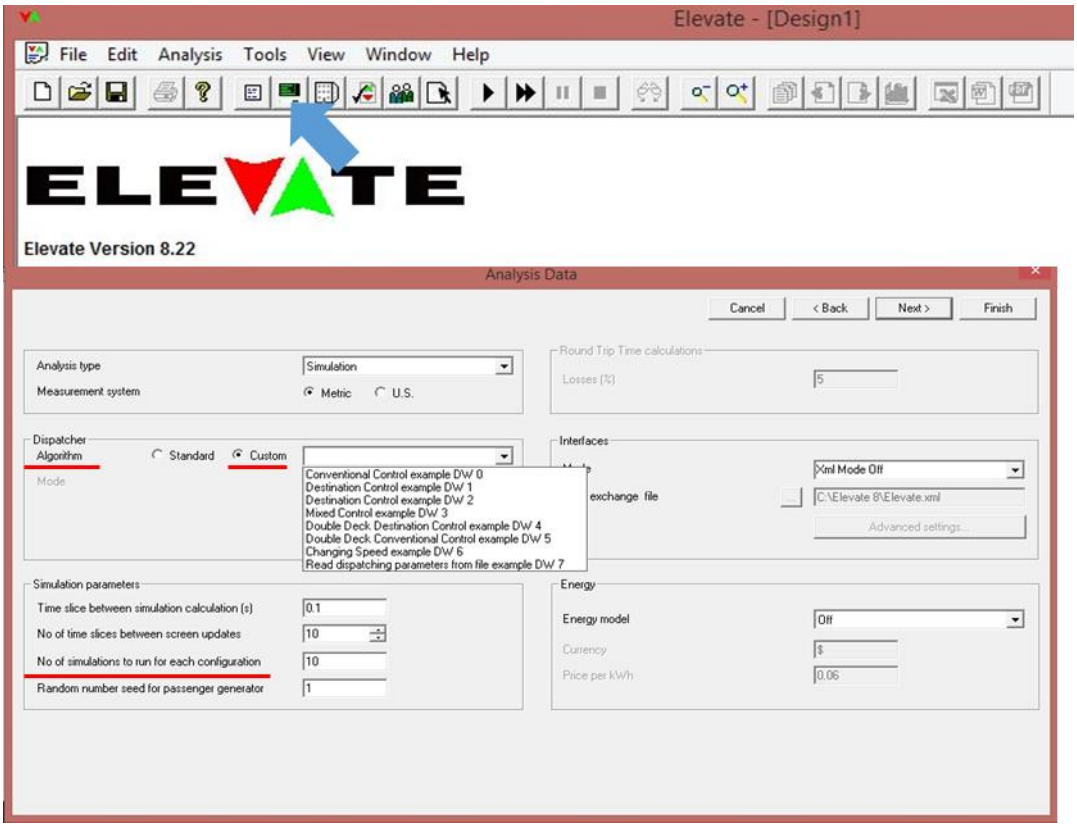


Figura 27.- Pantalla "Analysis Data" de ELVATE

- *Building Data*

  En esta ventana se seleccionan las características del edificio. Se podrá modificar el nombre de cada planta (*Floor Name*), altura de la misma (*Floor Level*), número de personas que hay en cada planta (*No of people*), área de cada planta (*Area*), área por persona (*Area/person*) y seleccionar la planta de entrada al edificio (*Entrance Floor*). También se podrá regular el porcentaje de absentismo que hay en el edificio (*Absenteesim*). Se muestra esta pantalla en la Figura 28.

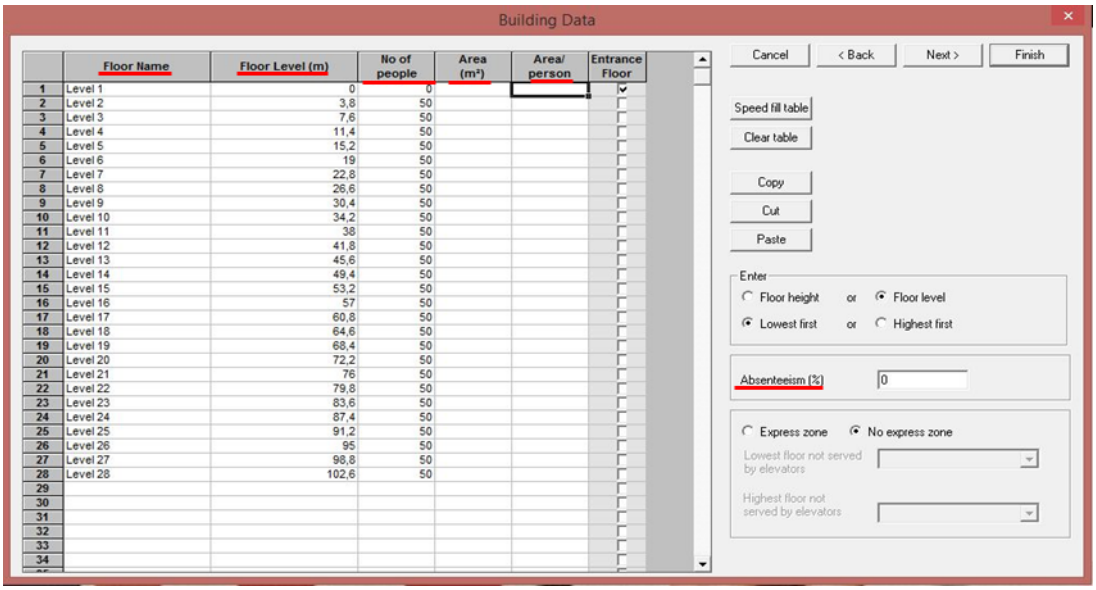


Figura 28.- Pantalla "Building Data" de Elevate

- *Elevator Data*

  En esta ventana se seleccionan las características de los ascensores. Se pueden configurar de dos modos: modo Standard, donde se modifican las características de todos los ascensores a la vez, o modo Advanced, en el cual se pueden modificar las características de cada ascensor de forma independiente.
  En ambos modos, los parámetros que se pueden variar son: el número de ascensores (*No. of Elevators)*, el tipo de ascensor (*Type*), la capacidad en kilogramos (*Capacity*), el área del ascensor en metros cuadrados (*Car Area*), el tiempo de apertura de las puertas del ascensor en segundos (*Door times*), la velocidad del ascensor en metros por segundo (*Speed*), la aceleración del ascensor en metros por segundo al cuadrado (*Acceleration*), la sobreaceleración en metros por segundos al cubo (*Jerk*), el tiempo de espera para comenzar el movimiento en segundos (*Start delay*) y la planta principal (*Home Floor*). Se muestran dichas pantallas en la Figura 29.

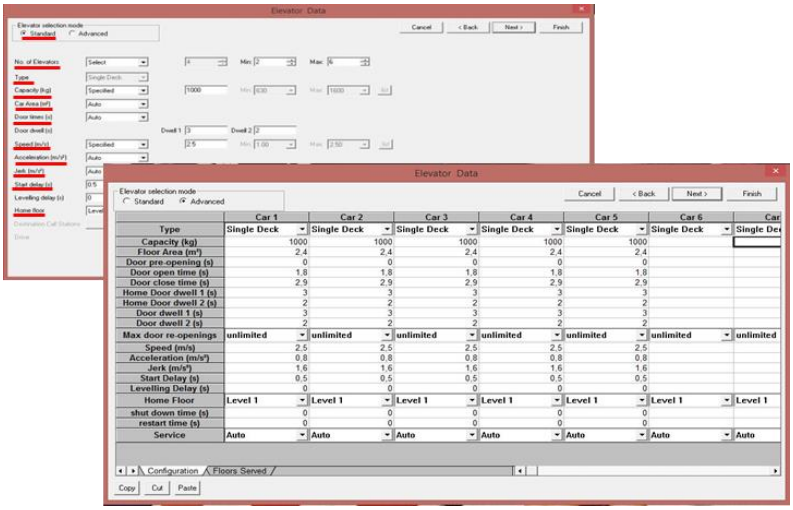


Figura 29.- Pantalla "Elevator Data" de Elevate

- *Passenger Data*

  En esta ventana se seleccionan las características referentes al tráfico de pasajeros en el edifico. Al igual que anteriormente, se cuenta con un modo *Standard* y otro *Advanced*. La principal diferencia entre ambos modos es que en el *Standard* se seleccionan las características para toda la población por igual y en el *Advanced* se podrá personalizar para cada periodo de tiempo. En ambos modos se pueden varias los siguientes parámetros: Tipo de tráfico en función del tipo de ascensor (Arrangemente), el tipo de patrón de tráfico que se requiere para la simulación (*Template*), la semilla aleatoria que se emplea (*Demand*), el porcentaje de personas que entran en el edificio (*Incoming*), el porcentaje de personas que dejan el edificio (*Outgoing*), el porcentaje de personas que se mueven entre las plantas del edificio (Interfloor), la hora de inicio de la simulación (*Start Time*), la hora de finalización de la simulación (*End Time*), el tiempo que tardan los pasajeros en subir/bajar del ascensor en segundos (Loading/Unloading time), la masa de pasajeros en kilogramos (Passenger Area), el porcentaje de factor de la capacidad por masa (*Capacity Factor by Mass*), el porcentaje de factor de capacidad por área (*Capacity Factor by Area*) y el porcentaje de uso de las escaleras del edificio (*Stair Factor*). Se muestran estas pantallas en la Figura 30.

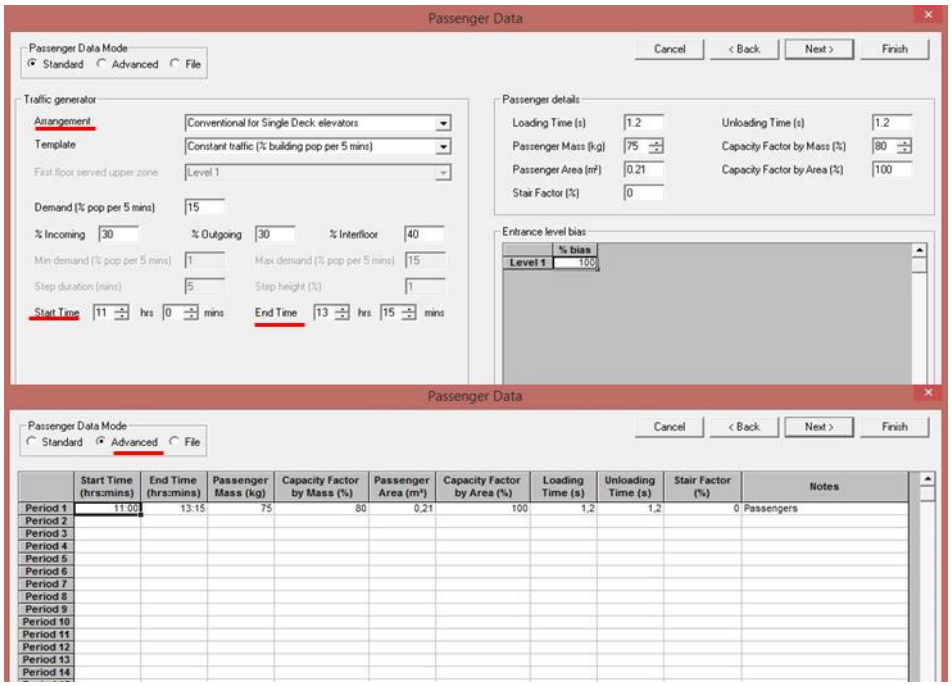


Figura 30.-Pantalla "Passenger Data" de Elevate

En estas ventanas se puede seleccionar el tipo de tráfico de pasajeros, siendo los principales:

- CIBSE

  Se caracteriza por tener un periodo *uppeak* abrupto al inicio de la jornada y un periodo *downpeak* fuerte al final de la jornada. A medio día tiene dos periodos de carácter moderado, uno downpeak y otro uppeak. Se muestra este patrón en la siguiente Figura 31 [2].

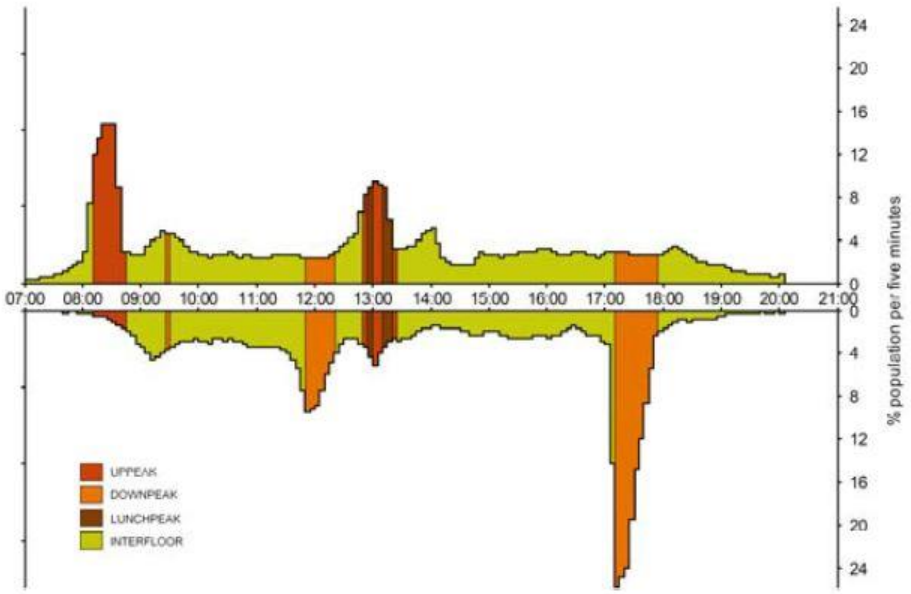


Figura 31.-Tráfico tipo CIBSE

- Strackosh
  Se trata de un modelo menos común, con curvas suaves, máximos pequeños y periodos ligeros *uppeak* y *downpeak* durante toda la mañana y la tarde. Se muestra este patrón en la siguiente Figura 32 [3].

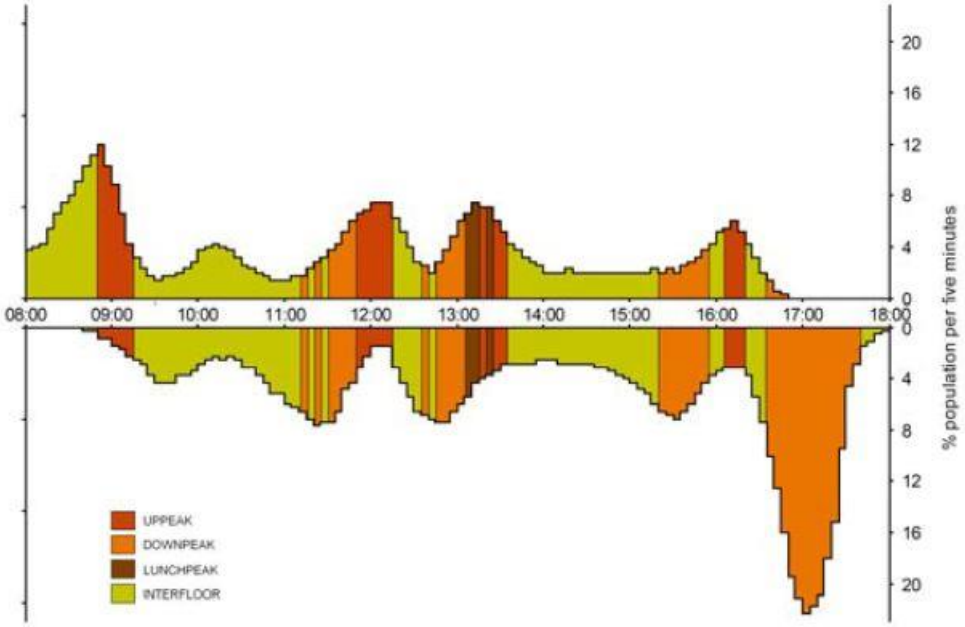


Figura 32.- Tráfico tipo STRACKOSH

- Siikomen
  Se caracteriza por ser el patrón de tráfico de pasajeros que más se aproxima al comportamiento real de flujo de pasajeros. Se muestra este patrón en la siguiente Figura 33 [3].

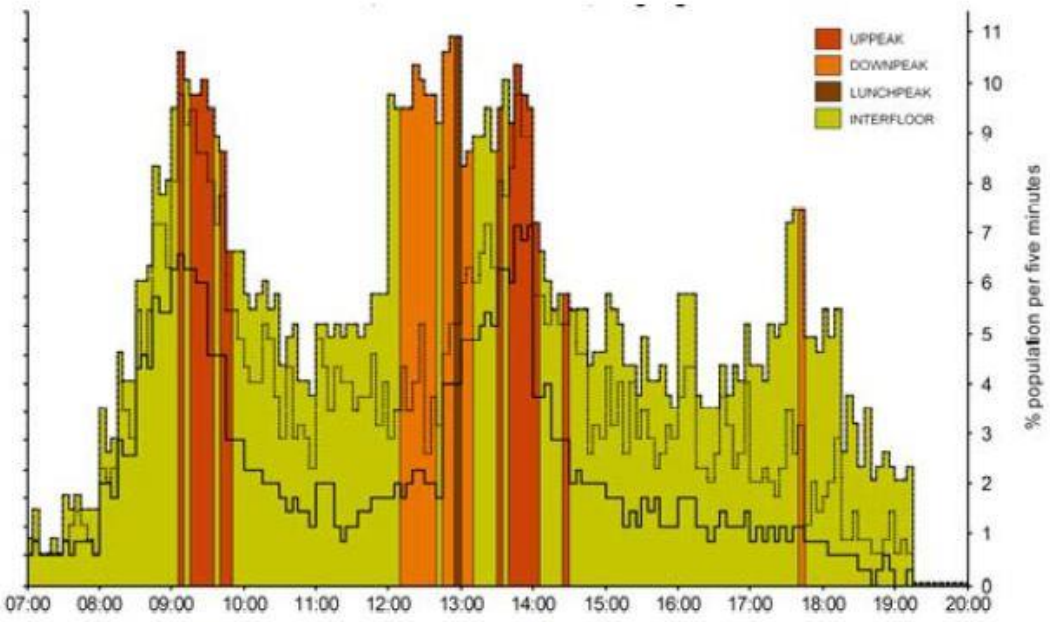


Figura 33.- Tráfico tipo "SIIKOMEN"

- Report options

  En esta ventana se seleccionan los tipos de análisis que se requiere que se desplieguen automáticamente al finalizar la simulación. En este caso, se seleccionan las opciones que ofrecen los tiempos medios de espera (*Distribution of Passenger Waiting Times*), de tránsito (Distribution of Passenger Transit Times) y de llegada al destino (*Distribution of Time to Destination*). Además, es posible seleccionar otras opciones como la demanda de pasajeros (*Passenger Demand*), la actividad total de pasajeros (Total Passenger Activity), la longitud de las colas de espera (*Queue Lenghts*) o el consumo energético de los ascensores durante la simulación (*Energy Consumption*). Se muestra esta pantalla en la Figura 34.

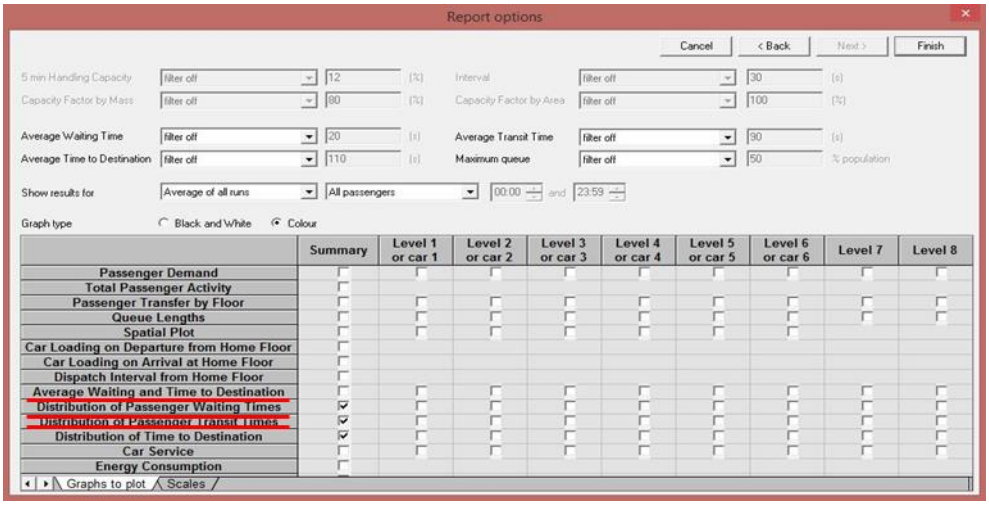


Figura 34.- Pantalla "Report Options" de Elevate

- Simulación

Una vez concluida la configuración de las anteriores pestañas, se puede visualizar una simulación del edificio y ascensores que se hayan definido anteriormente. En esta ventana puede observarse las diferentes plantas del edificio, los ascensores, las llamadas que se realizan desde cada planta, ya sean ascendentes (flechas color verde) o descendentes (flechas color rojo), y los correspondientes destinos de los pasajeros (puntos azules). Todos estos elementos se muestran en la siguiente Figura 35

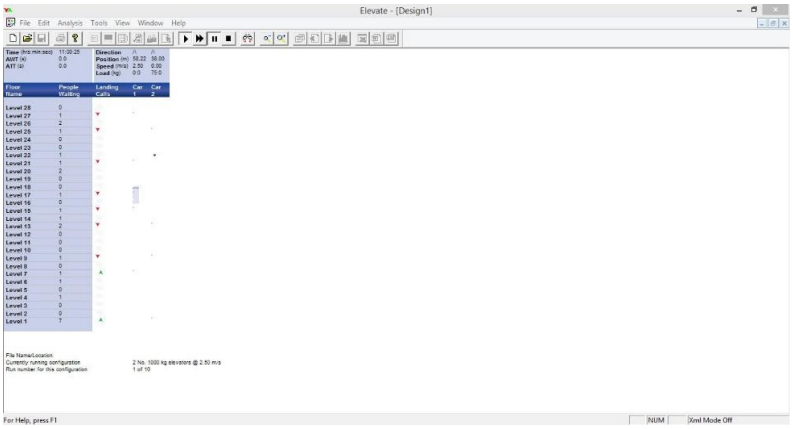


Figura 35.- Modo simulación Elevate 8

- Resultados del software

Una vez finalizada la simulación, ELEVATE muestra los resultados de los análisis que se hayan seleccionado en la ventana de *Report Options*. Se puede ver la pantalla de resultados obtenidos en la Figura 36.

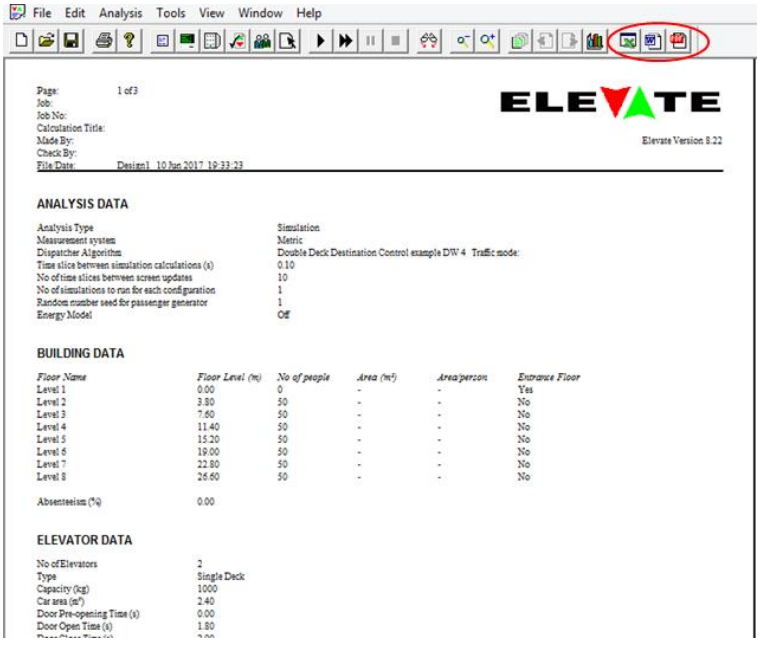


Figura 36.- Resultados Elevate 8

Estos resultados también pueden obtenerse como archivos en formato Excel, Word o PDF, seleccionando los botones marcados en rojo en la Figura 36.

4.2 MICROSOFT VISUAL STUDIO 2008

Para el desarrollo del algoritmo y posteriormente la conexión entre el algoritmo y ELEVATE, se requiere la instalación de este programa, concretamente de la versión 2008, lo cual no resulta trivial de encontrar.

4.2.1 Interfaz de desarrollador

Al descargar el archivo necesario para la instalación de la interfaz de desarrollador de la web de Peter Research Ltd. hay que seguir una serie de pasos, que se detallan en la carpeta *How to Use*. Entre los pasos a realizar, hay uno que es fundamental para la correcta conexión entre ambos software. Se trata de generar el archivo "DIspatch_00.dll" en una carpeta que no exija permisos de administrador, para evitar problemas de depuración. Se muestran en la Figura 37.

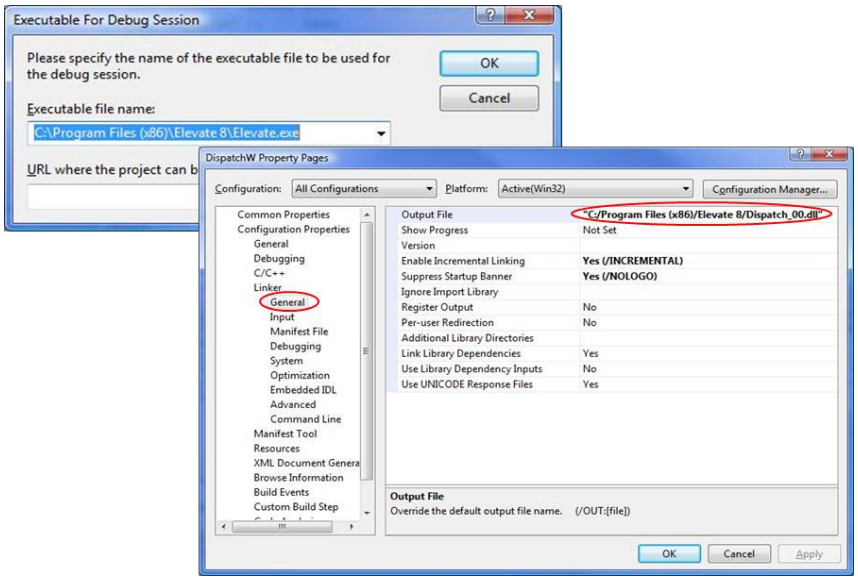


Figura 37.- Sincronizar Elevate 8 con Microsoft Visual Studio

4.2.2 Desarrollo en Microsoft Visual Studio

El lenguaje de programación que se emplea para la programación del algoritmo es C++. A continuación, se muestra en la Figura 38 parte del algoritmo desarrollado en Microsoft Visual Studio.

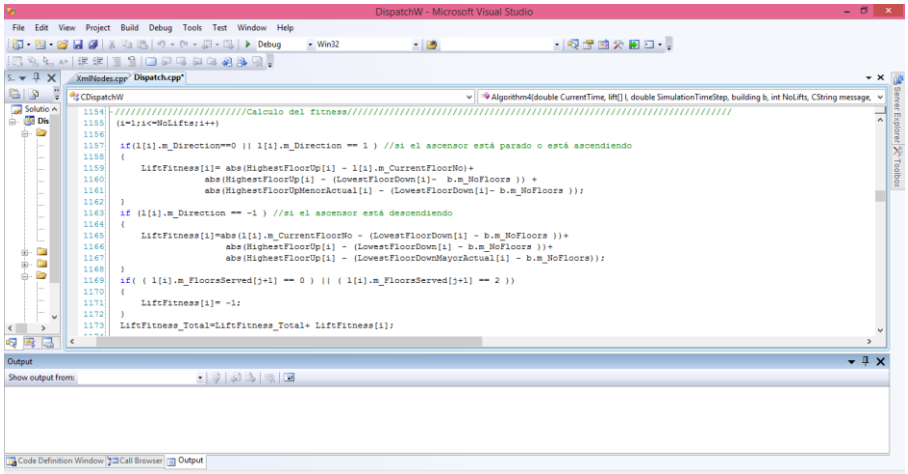


Figura 38.- Funcionamiento Microsoft Visual Studio

5 EXPERIMENTACIÓN

"La diferencia entre ganar y perder a menudo consiste en no abandonar".

- Walt Disney -

En la fase de experimentación se podrá comprobar la eficacia del algoritmo programado con respecto al tiempo medio de espera (AWT) y al tiempo medio de tránsito (ATT) de los pasajeros.

Se cuenta con una serie de algoritmos de asignación de llamadas, que se han explicado anteriormente, con los cuales se llevará a cabo la fase de experimentación y se compararán los resultados obtenidos. Así, se podrá comprobar qué algoritmo es el más efectivo en cada uno de los casos estudiados.

5.1 CARACTERÍSTICAS DE LA EXPERIMENTACIÓN

Para que la comparativa de algoritmos tenga validez, las especificaciones de los edificios y ascensores de las simulaciones para todos los casos deben ser idénticas, y así poder eliminar posibles distorsiones en los resultados.

5.1.1 Edificios

Como se presentó anteriormente, se van a estudiar edificios con diferentes alturas para observar el comportamiento de los diferentes algoritmos, pero a la vez, debe haber características comunes en todos ellos. En la Tabla 8 se muestran las características de los edificios estudiados:

Tabla 8.- Características del edificio

Características	Edificio
Número de plantas	X
Separación entre plantas	3,8 metros
Altura del edificio	3,8*X
Personas por planta	50
Población del edificio	50*X

Para el número de plantas, se tomarán los valores presentados en el Estudio de Casos anteriormente en el apartado 2.4 de este proyecto.

Como se ha señalado, los ascensores empleados son de arquitectura *Double Deck*, lo que condiciona a que los edificios tengan que tener un número de plantas par. Esto es debido a que este tipo de ascensores requiere ir siempre de plantas pares a pares y de impares a impares, a no ser que se establezcan dos plantas auxiliares (sin población) en la parte inferior y superior del edificio, que permitan que ascensor pueda acceder a la primera planta con la cabina superior y a la última planta con la cabina inferior.

5.1.2 Patrón de tráfico

El tráfico que se va a emplear para la experimentación es constante (*Constant Traffic (% Building pop per 5 mins*)). Se ha realizado la experimentación para tráfico *Interfloor* ya que este incorpora deslazamientos menos previsibles dentro del edificio permitiendo un análisis más general de los algoritmos. Con este patrón, para realizar las simulaciones se supondrá que un 30% de pasajeros que entran al edificio, un 30% de pasajeros que salen del edificio y un 40% de pasajeros que realizan movimientos entre plantas. Además, se supone el uso nulo de las escaleras en el edificio.

5.1.3 Características técnicas de los ascensores

El número de ascensores empleados varía en función del número de plantas del edificio, pero entre ellos tienen las mismas características, igual que en el caso de las plantas, para evitar distorsionar los resultados. Las características de los ascensores se muestran en la Tabla 9.

Tabla 9.- Características del ascensor

Características	
Capacidad de los ascensores	1000 Kg
Velocidad de los ascensores	2,5 m/s
Aceleración de los ascensores	0,8 m/s2
Retraso de inicio de marcha	0,5 s
Tiempo de apertura de puertas	2 s
Tiempo de cierre de puertas	2 s
Tiempo de carga y descarga	5 s
Factor de capacidad	80%

5.2 RESULTADOS Y DISCUSIONES

A continuación, se van a exponer los resultados obtenidos durante las simulaciones y se van a comparar los tiempos obtenidos por los distintos algoritmos utilizados, con la intención de poder determinar cuál de ellos es el que consigue optimizar los tiempos de espera y tiempos de tránsito, por separado, y también en su conjunto (tiempo total de viaje).

5.2.1 Análisis de los tiempos medios de espera

En cada uno de los casos que se han estudiado, se han hecho simulaciones de 2 horas de duración, especificando en cada caso las plantas del edificio y los ascensores utilizados. Los resultados obtenidos según los tiempos medios de espera (AWT) pueden verse en la Tabla 10.

Tabla 10.- Tiempos medios de Espera obtenidos para los tres métodos (seg)

AWT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	**1**	48,8	49,2	48,1
	2	26	23	21,3
10	**2**	41,2	37,2	40,1
	3	28,1	21,8	17,4
12	**2**	62,6	54,2	54,8
	3	39,3	28,4	28,6
14	**2**	86,5	95,2	76
	3	53,5	50,1	36,7
	4	37	24,3	23
20	**4**	91,6	55,5	54,3
	5	56,2	44,1	40,6
	6	49,4	26,7	26,1
24	**5**	91,8	59,8	64,9
	6	73	49,5	40,4
	7	57	36,2	32
30	**9**	76,4	43,6	35,9
	10	56,1	34,2	28,9
	11	54	30,4	24,1
32	**10**	68,2	38,7	33,8
	11	66,5	37,6	27,9
	12	61,4	32,3	25,5
34	**11**	74,7	43,7	28,6
	12	72,2	37,6	27

Para cada simulación, además de los tiempos medios de espera, se obtiene una gráfica en la que se muestra la evolución de éstos. A continuación, se muestra una de estas gráficas en la Figura 39 (concretamente la de la simulación de búsqueda tabú para un edificio de 24 plantas y 6 ascensores). El resto de gráficas obtenidas se muestran en el Anexo A de este documento.

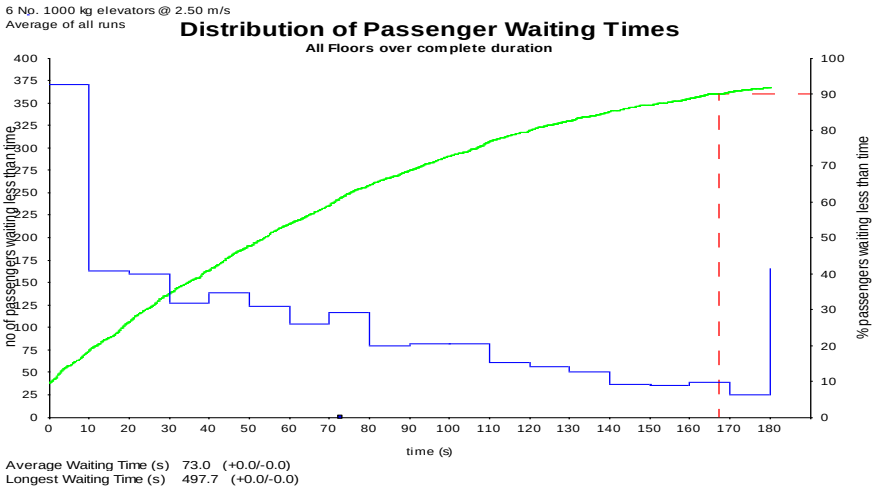


Figura 39.- Gráfica para el tiempo medio de espera con el algoritmo de búsqueda tabú

En el eje de abscisas de la izquierda se representa el número de personas que esperan menos que el tiempo del eje de ordenadas. En el eje de la derecha se representa la cantidad de personas anterior en forma de porcentaje.

La línea verde indica la evolución del tiempo de espera de los pasajeros.

La línea azul representa la cantidad de personas que esperan menos tiempo que el marcado en el eje del tiempo.

La línea azul representa el porcentaje de personas que esperan menos tiempo que el marcado en el eje del tiempo.

En este caso se observa que el 90% de las personas esperan menos de 170 segundos. Alrededor de 275 personas esperan menos de 10 segundos. Hay 25 personas que esperan 125 segundos. El tiempo medio de espera es de 79 segundos con un error de ± 0,6 segundos. Además, el mayor tiempo de espera que se ha dado en esta simulación es de 497,7 segundos con un error de 35,8 segundos.

5.2.2 Comparativa tiempos medios de espera

En la Tabla 8 del apartado anterior se mostraban los tiempos medios de espera obtenidos para cada uno de los métodos por separado.

En este apartado, para poder estudiar cada uno de los métodos en mayor profundidad, se comparan individualmente con respecto a los demás, calculando la diferencia de tiempos entre ellos. La comparativa con respecto a la Búsqueda Tabú se encuentra en la Tabla 11, con respecto al algoritmo genético en la Tabla 12 y a *Double Deck*-ETA en la Tabla 13.

Tabla 11.- Comparativa de AWT (seg) respecto a Búsqueda Tabú

AWT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	1	**48,8**	0,4	-0,7
	2	**26**	-3	-4,7
10	2	**41,2**	-4	-1,1
	3	**28,1**	-6,3	-10,7
12	2	**62,6**	-8,4	-7,8
	3	**39,3**	-10,9	-10,7
14	2	**86,5**	8,7	-10,5
	3	**53,5**	-3,4	-16,8
	4	**37**	-12,7	-14
20	4	**91,6**	-36,1	-37,3
	5	**56,2**	-12,1	-15,6
	6	**49,4**	-22,7	-23,3
24	5	**91,8**	-32	-26,9
	6	**73**	-23,5	-32,6
	7	**57**	-20,8	-25
30	9	**76,4**	-32,8	-40,5
	10	**56,1**	-21,9	-27,2
	11	**54**	-23,6	-29,9
32	10	**68,2**	-29,5	-34,4
	11	**66,5**	-28,9	-38,6
	12	**61,4**	-29,1	-35,9
34	11	**74,7**	-31	-46,1
	12	**72,2**	-34,6	-45,2

Tabla 12.- Comparativa AWT (seg) respecto a Algoritmo Genético

AWT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	1	-0,4	**49,2**	-1,1
	2	3	**23**	-1,7
10	2	4	**37,2**	2,9
	3	6,3	**21,8**	-4,4
12	2	8,4	**54,2**	0,6
	3	10,9	**28,4**	0,2
14	2	-8,7	**95,2**	-19,2
	3	3,4	**50,1**	-13,4
	4	12,7	**24,3**	-1,3
20	4	36,1	**55,5**	-1,2
	5	12,1	**44,1**	-3,5
	6	22,7	**26,7**	-0,6
24	5	32	**59,8**	5,1
	6	23,5	**49,5**	-9,1
	7	20,8	**36,2**	-4,2
30	9	32,8	**43,6**	-7,7
	10	21,9	**34,2**	-5,3
	11	23,6	**30,4**	-6,3
32	10	29,5	**38,7**	-4,9
	11	28,9	**37,6**	-9,7
	12	29,1	**32,3**	-6,8
34	11	31	**43,7**	-15,1
	12	34,6	**37,6**	-10,6

Tabla 13.- Comparativa AWT (seg) respecto a Double Deck-ETA

AWT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	1	0,7	**1,1**	**48,1**
	2	4,7	1,7	**21,3**
10	2	1,1	-2,9	**40,1**
	3	10,7	4,4	**17,4**
12	2	7,8	-0,6	**54,8**
	3	10,7	-0,2	**28,6**
14	2	10,5	19,2	**76**
	3	16,8	13,4	**36,7**
	4	14	1,3	**23**
20	4	37,3	1,2	**54,3**
	5	15,6	3,5	**40,6**
	6	23,3	0,6	**26,1**
24	5	26,9	-5,1	**64,9**
	6	32,6	9,1	**40,4**
	7	25	4,2	**32**
30	9	40,5	7,7	**35,9**
	10	27,2	5,3	**28,9**
	11	29,9	6,3	**24,1**
32	10	34,4	4,9	**33,8**
	11	38,6	9,7	**27,9**
	12	35,9	6,8	**25,5**
34	11	46,1	15,1	**28,6**
	12	45,2	10,6	**27**

Según estas tablas, se puede observar que el método de búsqueda tabú obtiene unos tiempos medios de espera (AWT) mayores que los otros dos métodos. Incluso, a medida que va aumentando el número de plantas, estos tiempos tienden a ser peores.

El algoritmo genético mejora los tiempos de la búsqueda tabú, pero sus resultados siguen siendo un poco peores (solo unos segundos) que los del método *Double Deck*-ETA.

Por tanto, el método que optimiza los tiempos medios de espera es el *Double Deck*-ETA, seguido muy de cerca por el algoritmo genético.

El motivo por el que la técnica heurística proporcione resultados mejores que las metaheurísticas puede ser debido a que, en los algoritmos genéticos y búsqueda tabú, se acepten ocasionalmente malos movimientos, es decir, que la nueva solución no sea mejor que la inmediatamente anterior, unido al periodo asumible de computación para proporcionar una solución en tiempo real al sistema de ascensores. Es posible que esto explique por qué en este caso las metaheurísticas no consiguen aproximarse al óptimo del problema durante la simulación [18].

Otra explicación de que el método de búsqueda tabú, que es el que más se aleja de los tiempos obtenidos por *Double Deck*-ETA, sea el que obtenga peores resultados podría ser que, este algoritmo, debe en cada iteración recorrer la Lista Tabú en busca de zonas del espacio de soluciones que se hayan visitados para buscar zonas nuevas. Esto implica que se ha de emplear un tiempo computacional importante [7].

En la siguiente Figura 40 puede verse la evolución de cada uno de los métodos del tiempo medio de espera a medida que se van aumentando el número de plantas de edificio, con sus respectivos ascensores.

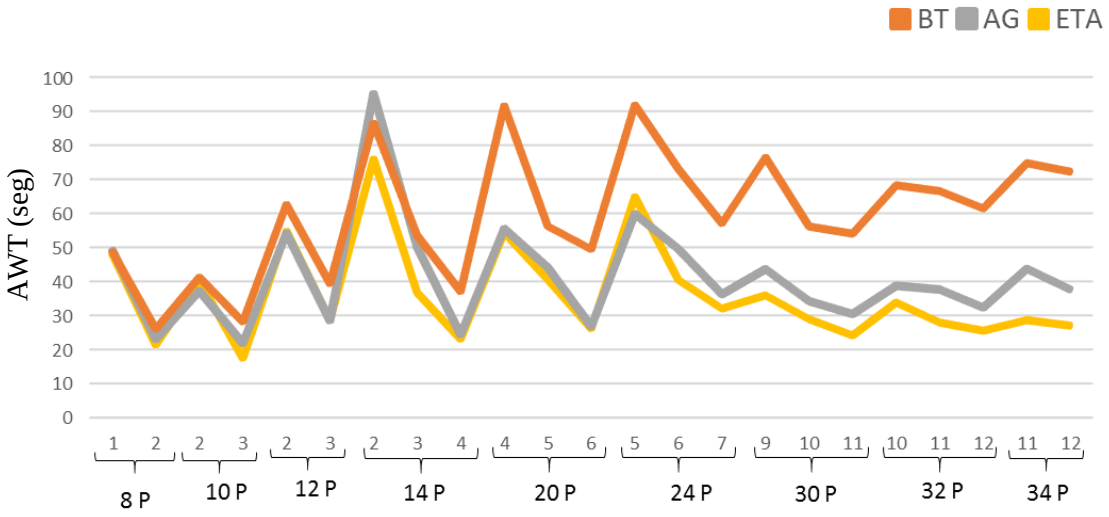


Figura 40.- Comparación AWT

5.2.3 Análisis de los tiempos medios de tránsito

Se analizan igualmente los tiempos medios de tránsito. De igual modo que para los tiempos medios de espera, se intentan minimizar.

Esta variable es igual de importante que el AWT, ya que de nada serviría que el tiempo de espera del pasajero frente al ascensor sea pequeño, si luego el tiempo que tarda en llegar a su destino es muy alto.

Se muestra en la Tabla 14 los resultados para el caso del ATT, para cada situación y cada algoritmo.

Tabla 14.- Tiempos medios de tránsito obtenidos para los tres métodos (seg)

ATT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	**1**	36,2	35,6	37,9
	2	29	28,9	29
10	**2**	38,8	37,9	39,5
	3	33,7	33	33
12	**2**	49,6	49,8	51,7
	3	42,5	41,6	43,3
14	**2**	59,8	62,3	64,3
	3	52,3	52,5	54,5
	4	46	48,4	47,3
20	**4**	73	74,9	78
	5	65,4	67,6	69,4
	6	61	63	64,5
24	**5**	84	87,7	91,7
	6	76,8	81,9	82,8
	7	70,6	74,7	77
30	**9**	82,9	90,5	95,2
	10	80	86,6	88,4
	11	77,3	81,9	82,9
32	**10**	87,8	98,8	98,1
	11	85,2	92,1	92,6
	12	81,2	86,7	88,6
34	**11**	92,4	97,2	99,2
	12	93,9	95	95,6

En las tablas 15, 16 y 17 se comparan entre sí los distintos algoritmos empleados, al igual que en las Tablas 11, 12 y 13 en el caso del AWT.

Tabla 15.- Comparativa ATT (seg) respecto a Búsqueda Tabú

ATT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	1	**36,2**	-0,6	1,7
	2	**29**	-0,1	0
10	2	**38,8**	-0,9	0,7
	3	**33,7**	-0,7	-0,7
12	2	**49,6**	0,2	2,1
	3	**42,5**	-0,9	0,8
14	2	**59,8**	2,5	4,5
	3	**52,3**	0,2	2,2
	4	**46**	2,4	1,3
20	4	**73**	1,9	5
	5	**65,4**	2,2	4
	6	**61**	2	3,5
24	5	**84**	3,7	7,7
	6	**76,8**	5,1	6
	7	**70,6**	4,1	6,4
30	9	**82,9**	7,6	12,3
	10	**80**	6,6	8,4
	11	**77,3**	4,6	5,6
32	10	**87,8**	11	10,3
	11	**85,2**	6,9	7,4
	12	**81,2**	5,5	7,4
34	11	**92,4**	4,8	6,8
	12	**93,9**	1,1	1,7

Tabla 16.- Comparativa ATT (seg) respecto a Algoritmo Genético

ATT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	1	0,6	**35,6**	2,3
	2	0,1	**28,9**	0,1
10	2	0,9	**37,9**	1,6
	3	0,7	**33**	0
12	2	-0,2	**49,8**	1,9
	3	0,9	**41,6**	1,7
14	2	-2,5	**62,3**	2
	3	-0,2	**52,5**	2
	4	-2,4	**48,4**	-1,1
20	4	-1,9	**74,9**	3,1
	5	-2,2	**67,6**	1,8
	6	-2	**63**	1,5
24	5	-3,7	**87,7**	4
	6	-5,1	**81,9**	0,9
	7	-4,1	**74,7**	2,3
30	9	-7,6	**90,5**	4,7
	10	-6,6	**86,6**	1,8
	11	-4,6	**81,9**	1
32	10	-11	**98,8**	-0,7
	11	-6,9	**92,1**	0,5
	12	-5,5	**86,7**	1,9
34	11	-4,8	**97,2**	2
	12	-1,1	**95**	0,6

Tabla 17.- Comparativa ATT (seg) respecto a Double Deck-ETA

ATT				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	1	-1,7	-2,3	**37,9**
	2	0	-0,1	**29**
10	2	-0,7	-1,6	**39,5**
	3	0,7	0	**33**
12	2	-2,1	-1,9	**51,7**
	3	-0,8	-1,7	**43,3**
14	2	-4,5	-2	**64,3**
	3	-2,2	-2	**54,5**
	4	-1,3	1,1	**47,3**
20	4	-5	-3,1	**78**
	5	-4	-1,8	**69,4**
	6	-3,5	-1,5	**64,5**
24	5	-7,7	-4	**91,7**
	6	-6	-0,9	**82,8**
	7	-6,4	-2,3	**77**
30	9	-12,3	-4,7	**95,2**
	10	-8,4	-1,8	**88,4**
	11	-5,6	-1	**82,9**
32	10	-10,3	0,7	**98,1**
	11	-7,4	-0,5	**92,6**
	12	-7,4	-1,9	**88,6**
34	11	-6,8	-2	**99,2**
	12	-1,7	-0,6	**95,6**

Como puede verse para el caso de los tiempos medios de tránsito, se trata del algoritmo de búsqueda tabú el que consigue mejores resultados. El algoritmo genético obtiene resultados muy similares, solo con algunos segundos de diferencia, pero se puede observar una leve tendencia a empeorar a media que aumenta el número de plantas. Lo mismo ocurre con *Double Deck*-ETA.

Se podría decir que, para minimizar los tiempos de tránsito de los pasajeros, el mejor método es la búsqueda tabú, aunque los tres métodos obtienen resultados muy parecidos, como puede verse en la Tabla 16.

En la siguiente Figura 41 puede observarse la evolución de cada uno de los métodos del tiempo medio de espera a medida que se van aumentando el número de plantas de edificio, con sus respectivos ascensores. Los tres métodos consiguen resultados muy parejos, pero se observa que es la búsqueda tabú la que consigue menores tiempos, ganado distancia a medida que aumenta la altura del edificio.

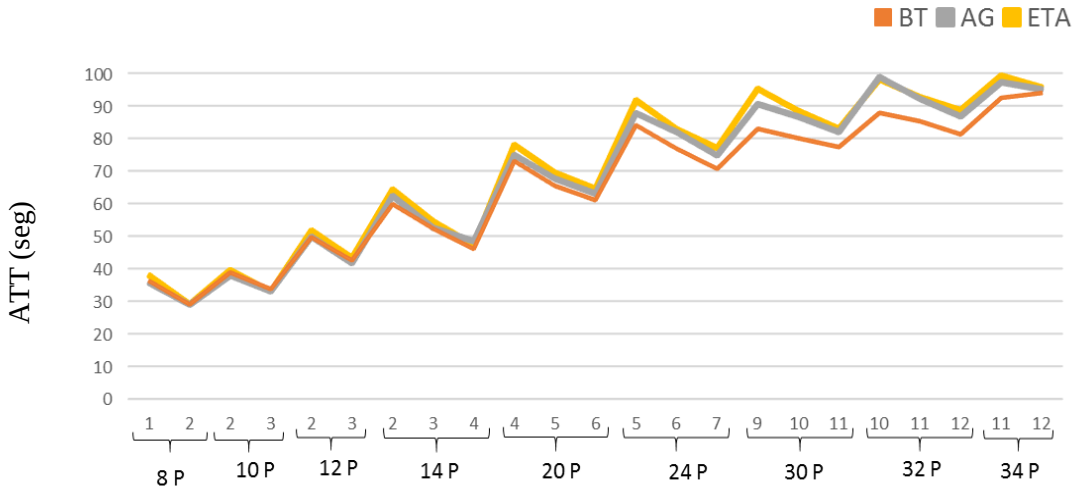


Figura 41.-Comparativa de los Tiempos medios de Tránsito

Puede parecer extraño que, para el caso de la optimización de los tiempos de tránsito, sea la búsqueda tabú el método que obtiene mejores resultados, cuando en el caso anterior de optimizar el tiempo de espera era el menos adecuado. Esto puede deberse a que el método requiere tiempo para realizar la asignación de llamadas a ascensores (lo que aumenta el AWT), pero una vez que están asignadas, optimiza el recorrido a seguir por el ascensor, consiguiendo así que tiempo que tarda el pasajero en llegar a su destino sea mínimo.

5.2.4 Análisis de detalle según el edificio

Una vez visto el comportamiento general de cada uno de los algoritmos, se realiza un análisis detallado de cada uno de los edificios estudiados según los tres métodos.

- Edificio de 8 plantas. Figura 42.

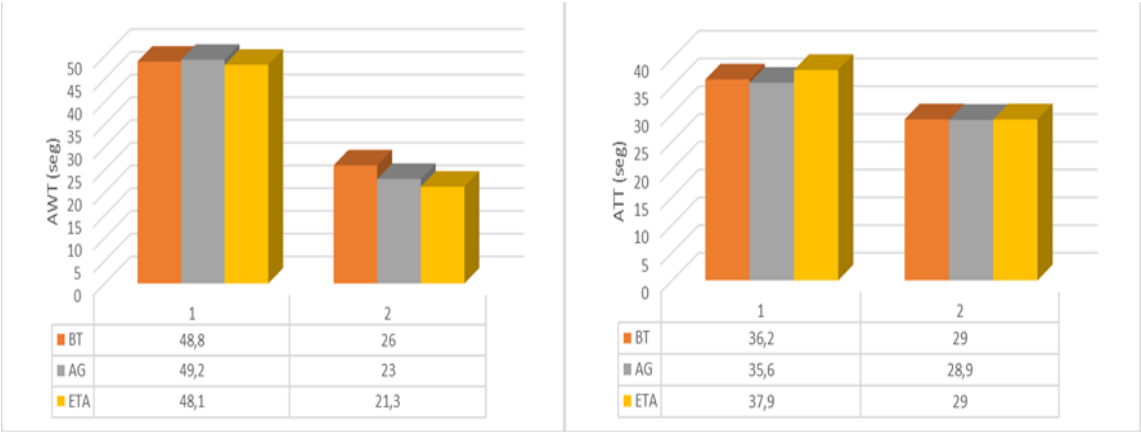


Figura 42.-AWT y ATT para un edificio de 8 plantas

Para un edifico de 8 plantas, se calculó anteriormente que el número óptimo de ascensores es 1.

Si se observan los resultados, se puede ver que, obviamente, para el caso de 2 ascensores se obtienen tiempos menores (como era de esperar pues si se aumenta la capacidad, disminuye la ocupación), pero para un solo ascensor, se obtienen resultados aceptables para la calidad del servicio de los pasajeros, tal como puede verse en la Figura 42.

Respecto a los algoritmos, en esta situación, todos los algoritmos proporcionan resultados muy parejos, por lo que sería indiferente la utilización de uno u otro de ellos.

- Edificio de 10 plantas. Figura 43.

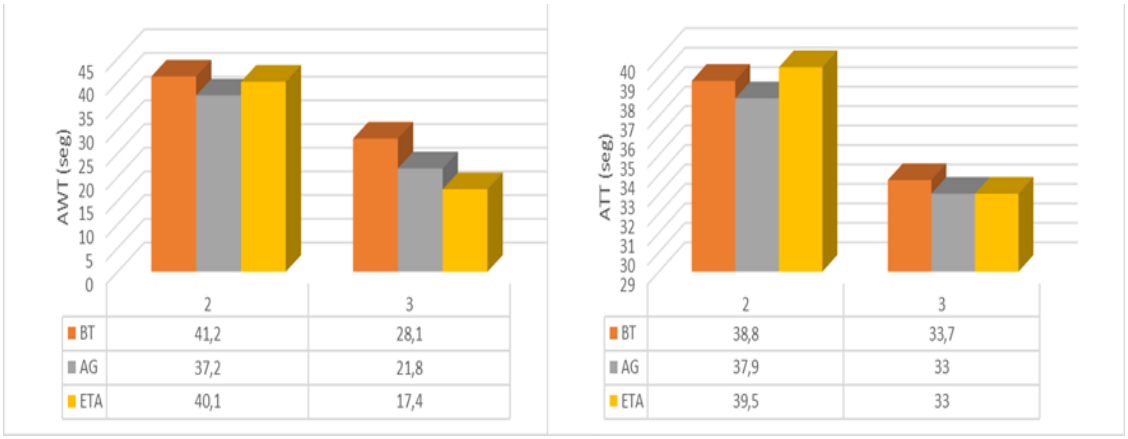


Figura 43.- AWT y ATT para un edificio de 10 plantas

En el caso de un edificio de 10 plantas, son 2 los ascensores que optimizan los recursos. Según las gráficas de la Figura 43, de nuevo vuelve a ocurrir igual que en el caso anterior: aumenta el número de ascensores y disminuye el tiempo, pero los resultados obtenidos para 2 ascensores son considerados aceptables.

En este caso de 2 ascensores, sería el Algoritmo Genético el que minimiza tanto el tiempo medio de espera y como el tiempo medio de tránsito.

- Edificio de 12 plantas. Figura 44.

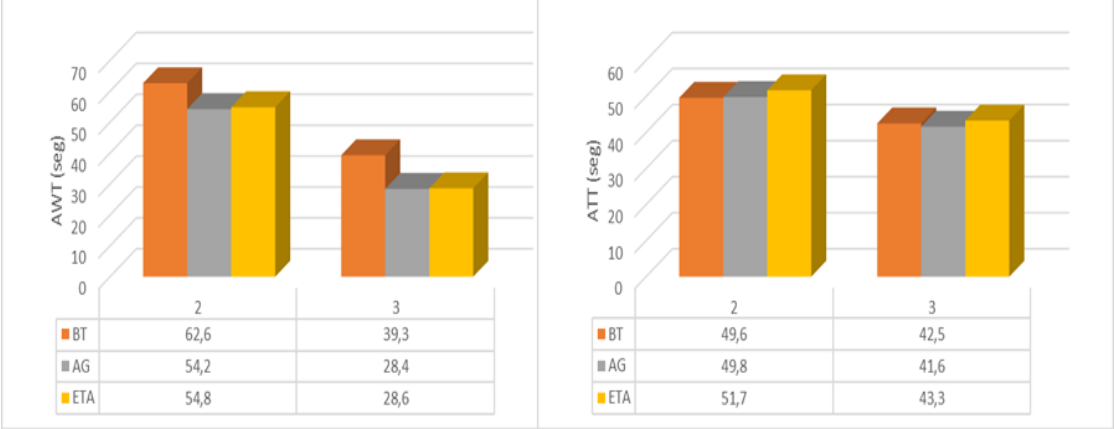


Figura 44.- AWT y ATT para un edificio de 12 plantas

En la situación de un edificio de 12 plantas, se determinó que el número óptimo de ascensores era 2, pero según se puede observar en la Figura 44, que los tiempos obtenidos por los tres métodos, no se ajustan a la calidad de servicio esperada por los pasajeros.

Por lo que sería conveniente en este caso replantearse la situación y optar por incluir un ascensor más en el edificio. Pues, aunque suponga mayor coste, la calidad de servicio ofrecida por el edificio aumentaría notablemente.

En cuanto a los algoritmos de simulación, el algoritmo genético es el que consigue menores tiempos medios de espera; mientras que, para los tiempos medios de tránsito, se obtienen mejores resultados con la búsqueda tabú o el algoritmo genético.

- Edificio de 14 plantas. Figura 45.

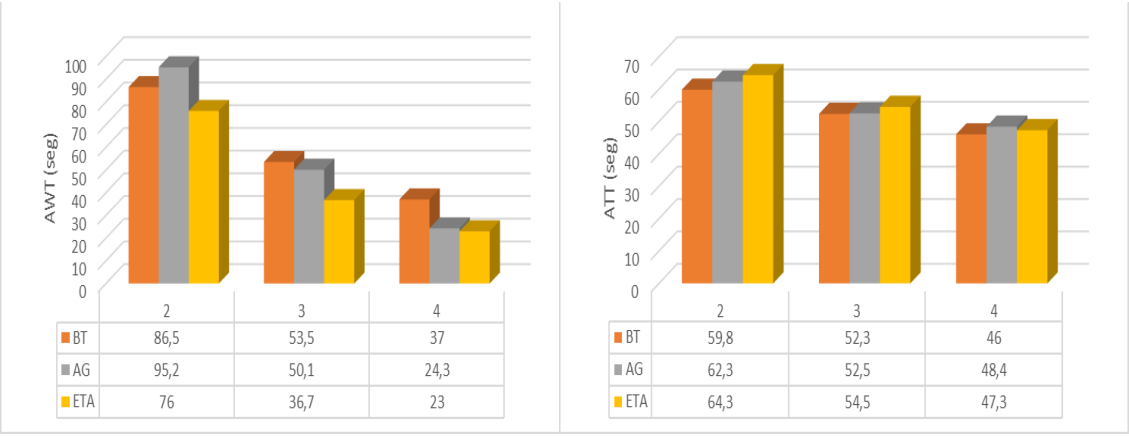


Figura 45.- AWT y ATT para un edificio de 14 plantas

En este caso, el edificio optimizaría la calidad del servicio con 3 ascensores, y según las gráficas de la Figura 45, sería correcta dicha suposición. Puede verse que con el algoritmo *Double Deck*-ETA, se consigue un tiempo medio de espera mucho menor que el considerado como aceptable para una buena calidad del servicio.

Para minimizar el tiempo de tránsito, es el algoritmo de búsqueda tabú, seguido muy de cerca por el algoritmo genético, el que consigue mejores resultados.

- Edificio de 20 plantas. Figura 46.

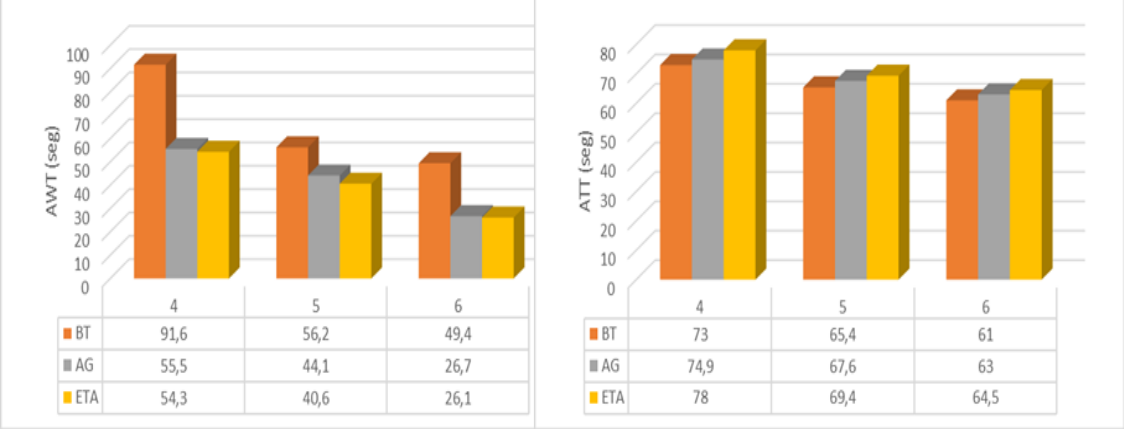


Figura 46.- AWT y ATT para un edificio de 20 plantas

Para el caso de un edificio con 20 plantas, se había establecido como número óptimo de ascensores, 5. En las gráficas de la Figura 46 puede verse que los tiempos obtenidos para dichos ascensores se ajustan a las exigencias de los pasajeros; pero si se colocase un ascensor más, estos tiempos descenderían notablemente (casi a la mitad) se los obtenidos con 5 ascensores con el Algoritmo Genético y el algoritmo de *Double Deck*-ETA.

Los nombrados anteriormente, son los que consiguen unos tiempos medios de espera más bajos; mientras que, para obtener un tiempo medio de tránsito pequeño, la mejor opción sería optar por el algoritmo de Búsqueda Tabú.

- Edificio de 24 plantas. Figura 47.

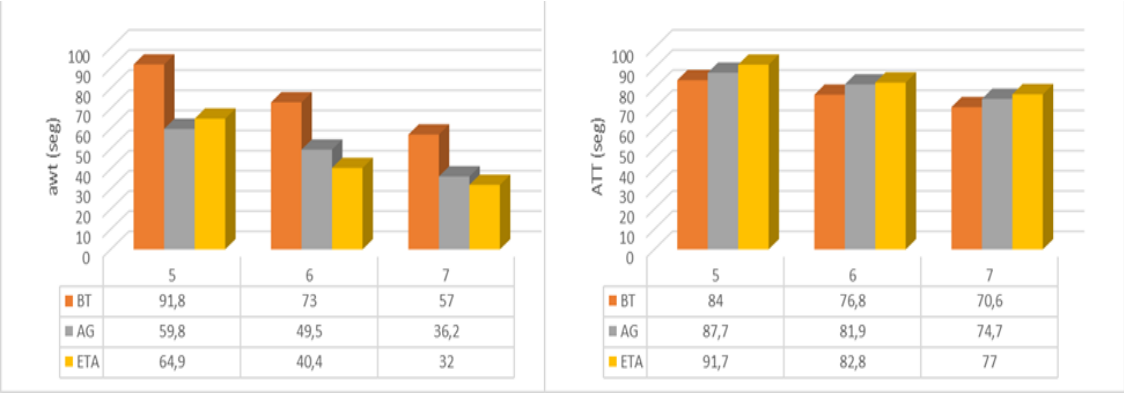


Figura 47.- AWT y ATT para un edificio de 24 plantas

Esta situación es idéntica a la anterior, pero en este caso, el número óptimo de ascensores era 6. Puede observarse en la Figura 47, que, si fueran 7 los ascensores del edificio, los tiempos medios de espera mejorarían para el caso del algoritmo genético y para *Double Deck*-ETA.

De la misma forma, es la búsqueda tabú el algoritmo que consigue mejores tiempos de tránsito.

- Edificio de 30 plantas. Figura 48.

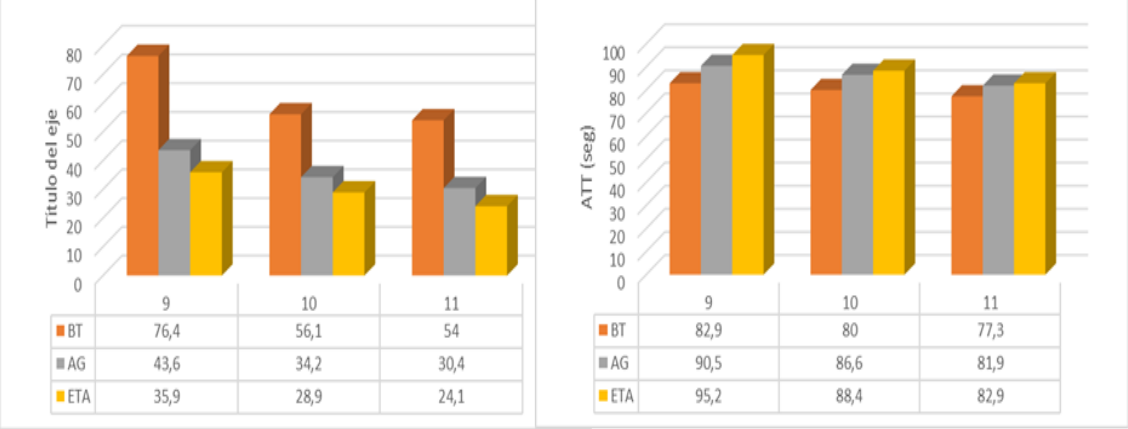


Figura 48.- AWT y ATT para un edificio de 30 plantas

Para un edificio de 30 plantas, se puede observar en la Figura 48 que con 10 ascensores se obtienen los mejores tiempos con los algoritmos de *Double Deck*-ETA y el algoritmo genético. Aunque se aumentase el número de ascensores, los tiempos no descienden suficiente como para asumir el coste que supondría la instalación de otro ascensor.

Es de nuevo la Búsqueda Tabú el algoritmo que consigue los mejores tiempos medios de tránsito, aunque con muy poca diferencia con respecto a los otros dos algoritmos.

- Edificio de 32 plantas. Figura 49.

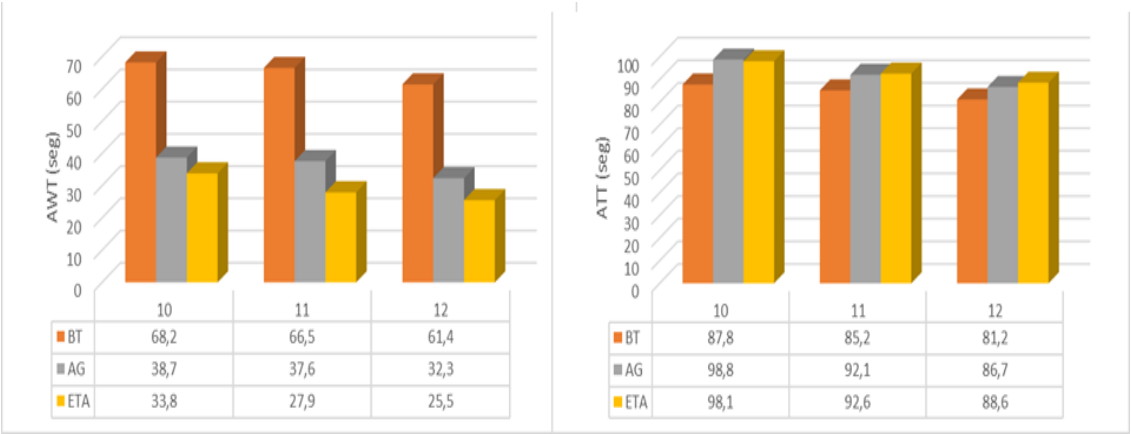


Figura 49.- AWT y ATT para un edificio de 32 plantas

Esta situación es muy parecida a la anterior del edificio con 30 plantas. Se optimizan los tiempos medios de espera con el número de ascensores calculado, 11, en los casos del algoritmo genético y *Double Deck*-ETA.

Los menores tiempos de tránsito son conseguidos por la búsqueda tabú.

- Edificio de 34 plantas (12). Figura 50.

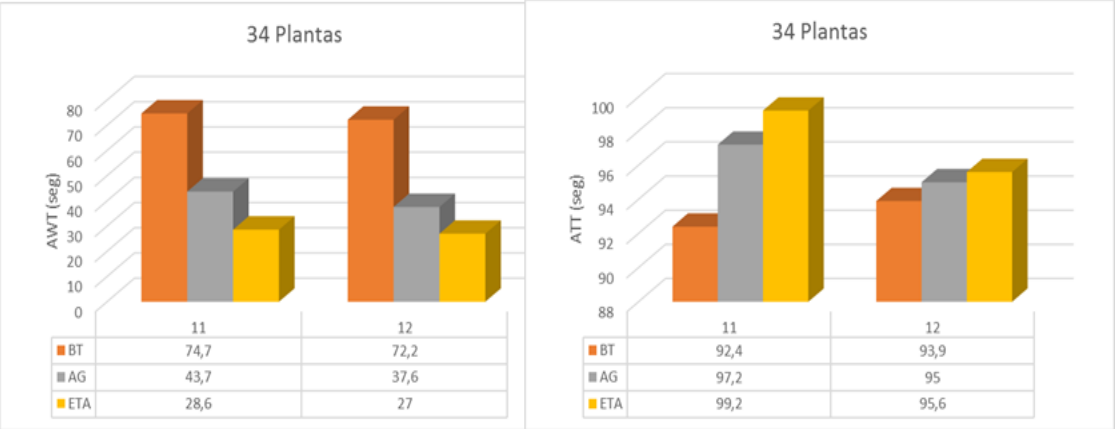


Figura 50.- AWT y ATT para un edificio de 34 plantas

Para el caso del edificio de 34 plantas mostrado en la Figura 50, se puede ver que la variación entre 11 ascensores y 12 es mínima. Inicialmente, el número óptimo de ascensores es 12, pero se comprueba que con 11 también se alcanzan tiempos aceptables por la calidad del servicio requerida.

Igualmente son el algoritmo genético y *Double Deck*-ETA los métodos que consiguen minimizar el AWT y la búsqueda tabú optimiza el ATT.

Tras el análisis de cada edificio, se ha comprobado que, para minimizar los tiempos medios de espera de los pasajeros (AWT), los algoritmos más idóneos son el algoritmo genético y *Double Deck*-ETA, obteniendo ambos resultados muy parecidos en los edificios de menos plantas, y a medida que éstas van aumentando, sus valores se van distanciando un poco más, pero sin salirse del rango admisible.

El algoritmo de búsqueda tabú no obtiene muy buenos resultados a la hora de minimizar los tiempos de espera; en cambio, es el mejor para minimizar los tiempos de tránsito (ATT). En los edificios con un número de plantas menor, la diferencia de los tiempos de tránsito entre los tres algoritmos no es tan notoria como en los edificios de mayor número de plantas. Es en estos últimos casos en los que el algoritmo de búsqueda tabú destaca, obteniendo los mejores resultados.

5.2.5 Análisis del tiempo total de viaje

Anteriormente, se han estudiado el tiempo de espera y el tiempo de tránsito de pasajeros por separado. Pero además de estas dos variables, es importante tener en cuenta en tiempo total de viaje de los pasajeros en el edificio.

El tiempo total de viaje es la suma del tiempo medio de espera de los pasajeros frente al ascensor más el tiempo medio de tránsito hasta llegar a su destino.

$$Tiempo\ Total = AWT + ATT$$

En la Tabla 18 se muestra el tiempo total para cada uno de los casos anteriormente estudiados, siguiendo la misma distribución de plantas y ascensores.

Tabla 18.-Tiempos totales de viaje (seg)

TIEMPO TOTAL DE VIAJE				
Número de Plantas	Número de Ascensores	BT	AG	ETA
8	**1**	85	84,8	86
	2	55	51,9	50,3
10	**2**	80	75,1	79,6
	3	61,8	54,8	50,4
12	**2**	112,2	104	106,5
	3	81,8	70	71,9
14	**2**	146,3	157,5	140,3
	3	105,8	102,6	91,2
	4	83	72,7	70,3
20	**4**	164,6	130,4	132,3
	5	121,6	111,7	110
	6	110,4	89,7	90,6
24	**5**	175,8	147,5	156,6
	6	149,8	131,4	123,2
	7	127,6	110,9	109
30	**9**	159,3	134,1	131,1
	10	136,1	120,8	117,3
	11	131,3	112,3	107
32	**10**	156	137,5	131,9
	11	151,7	129,7	120,5
	12	142,6	119	114,1
34	**11**	167,1	140,9	127,8
	12	166,1	132,6	122,6

En la Figura 51 se puede ver además la evolución del tiempo total de viaje. A medida que aumenta el número de plantas del edificio y a su vez el número de ascensores, el tiempo total de viaje se incremente, como es lógico.

Pueden observarse también varios picos en casos como 14 plantas y 7 ascensores, 20 plantas y 10 ascensores y 24 plantas y 13 ascensores. Estos tiempos totales de viaje tan elevados nos demuestran que el número de ascensores para esos edificios debe ser como mínimo el calculado en el punto 4 de este proyecto.

Respecto a los algoritmos usados, los tres obtienen resultados muy similares para los tiempos totales de viaje en casos de edificios de pocas plantas. A medida que aumenta el número de plantas, el algoritmo de búsqueda tabú comienza a generar resultados que estarían fuera del rango admisible de calidad del servicio. Esto es debido a los altos tiempos medios de espera que se generan con este método a medida que aumentan las plantas del edificio. Aunque los tiempos de tránsito sí sean menores que el resto de algoritmos, la diferencia tan notable en los tiempos de espera, al sumarse a los bajos tiempos de tránsito, es lo que causa estos elevados tiempos totales de viaje.

Mientras, el algoritmo genético y *Double Deck*-ETA, mantienen resultados bastante parecidos para toda variedad de edificios, siendo el algoritmo genético el que obtiene los mejores resultados para las situaciones óptimas de ascensores.

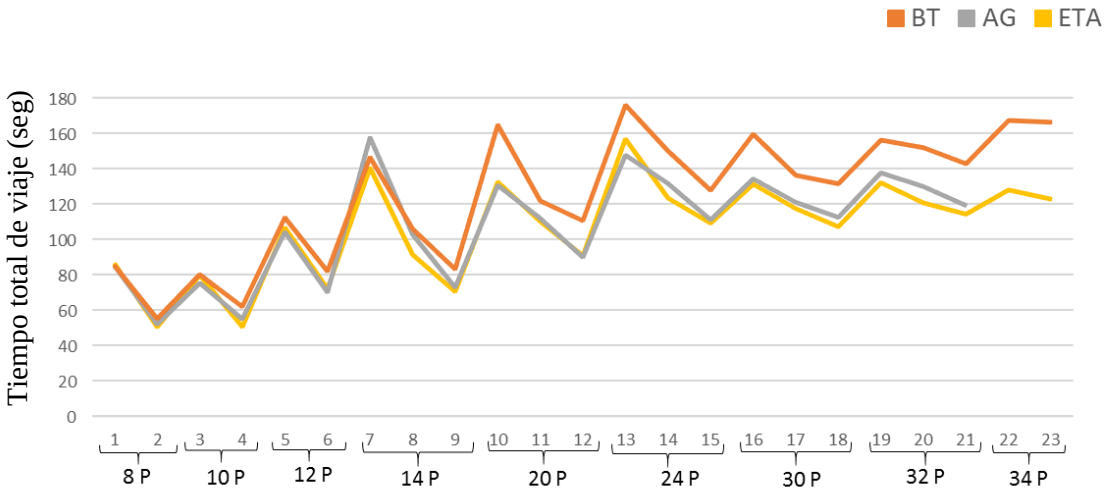


Figura 51.- Tiempo total de viaje (seg)

6 CONCLUSIONES

«La conclusión final es que sabemos muy poco y sin embargo, es asombroso lo mucho que conocemos. Y más asombroso todavía es que un conocimiento tan pequeño nos pueda dar tanto poder».

- Bertrand Russell -

En la actualidad, el alto precio del suelo hace que se pretenda aprovechar lo máximo posible. Es por ello por lo que los edificios que se construyen, cada vez cuentan con más plantas. De aquí surge la importancia del tráfico vertical, y la necesidad de optimización de los recursos disponibles (ascensores).

Es entonces cuando surgen los ascensores de arquitectura Double Deck (y más recientemente, los TWINS), los cuales permiten la movilidad de una mayor población por cada viaje que el ascensor realiza. Sin embargo, se requiere de un algoritmo de asignación de ascensores a llamadas lo suficientemente eficiente como para que se sirvan todas las llamadas del edificio de manera que el tiempo de espera de los pasajeros frente al ascensor y el tiempo que estos tardan en llegar a su destino sea el menor posible.

Los algoritmos desarrollados para la asignación de ascensores a llamadas han sido la búsqueda tabú, los algoritmos genéticos y *Double Deck*-ETA, con los cuales se han estudiado distintas situaciones en busca de una buena solución al problema.

Se ha implementado el algoritmo en lenguaje C++ de programación mediante el software Microsoft Visual Studio 2008. Además, se emplea el software ELEVATE 8 para realizar las simulaciones para cada una de las situaciones de estudio. Se requiere la previa instalación de la interfaz de desarrollador para la conexión entre ambos programas.

Para las simulaciones realizadas se han empleado 9 edificios con diferente número de plantas (8, 10, 12, 14, 20, 24, 30, 32 y 34), a los cuales se les ha calculado el número óptimo de ascensores que cumplirían con la calidad de servicio exigida. Se han elaborado simulaciones para estas combinaciones y para las situaciones de más o menos ascensores del número óptimo, para poder corroborar que el 'Estudio de casos' realizado es correcto. Se emplean para estas simulaciones ascensores de arquitectura Double Deck.

Se han estudiado varios tipos de algoritmos de asignación de llamadas a ascensores para analizar los tiempos medios de espera y los tiempos medios de tránsito de los pasajeros.

Para los edificios con pocos ascensores, en cuanto a tiempos de espera se refiere, los tres algoritmos obtienen buenos resultados que se ajustan a la calidad del servicio requerida.

En cambio, a medida que aumenta el número de plantas, es el algoritmo *Double Deck*-ETA el que consigue los mejores tiempos medios de espera, seguido muy de cerca por el algoritmo genético

Respecto a los tiempos de tránsito, es el algoritmo de búsqueda tabú el que obtiene mejores resultados para todas las casuísticas de edificios, siendo más notables las diferencias a medida que aumenta el número de plantas del edificio.

En cuanto a los tiempos totales de viaje, que resulta ser la suma del tiempo de espera frente al ascensor más el tiempo que el pasajero tarda en llegar a su destino una vez ha ingresado en el ascensor, son de nuevo el algoritmo genético y *Double Deck*-ETA los algoritmos que consiguen minimizar estos tiempos. En edificios con pocas plantas, las diferencias entre los resultados obtenidos con estos algoritmos y con la búsqueda tabú, no son tan altas como a medida que va creciendo el número de plantas de los edificios.

A pesar de que los tres algoritmos se encuentran programados para optimizar los tiempos medios de espera de los pasajeros, es solo en dos de los casos en los que se consigue de manera notable. Es de resaltar que, aunque estén programados para dicha finalidad, consiguen minimizar igualmente los tiempos de tránsito (esta vez para los tres algoritmos).

Se demuestra que, en la mayoría de los casos, el pasajero se encuentra más cómodo una vez ha entrado en el ascensor, prestándole menos importancia a la duración del trayecto hacia el destino. Es por ello por lo que se trata de promover la minimización del AWT frente al ATT. Por lo que respecta en este aspecto, los algoritmos con mejor comportamiento fueron los algoritmos genéticos y *Double Deck*-ETA.

Puede parecer contradictorio que una heurística como *Double Deck*-ETA consiga mejores resultados que una metaheurística como la búsqueda tabú. Esto puede ser debido a lo comentado en la sección anterior. Es posible que se acepten ocasionalmente malos movimientos, es decir, que la nueva solución no sea mejor que la inmediatamente anterior, uniéndose ello al tiempo de que se dispone para computar la solución en tiempo real (el cual es pequeño). Puede deberse a esto por lo que en este caso las metaheurísticas no consiguen aproximarse al óptimo del problema en las situaciones de estudio.

En cambio, si lo que se desea es minimizar el tiempo que el pasajero pasa dentro del ascensor hasta llegar a su destino, el algoritmo más correcto a implementar sería la Búsqueda Tabú. Resulta extraño que el mismo método que ha obtenido los peores resultados en la optimización de uno de los tiempos, consiga los mejores en el otro caso. Esto puede ser debido, tal como se expuso en la sección anterior, a un requerimiento alto de tiempo por parte de este método para realizar la asignación de llamadas a ascensores. Pero una vez realiza la asignación, sea el método que más rápido realice el trayecto requerido por el usuario que entra en el ascensor.

En la optimización del tiempo total de viaje, es obvio que los tiempos resultantes en el caso de la búsqueda tabú se ven distorsionados por los altos tiempos obtenidos por este método para la espera por parte de los pasajeros del ascensor. Es por lo que, para este caso, sería el algoritmo genético el método más apropiado para la obtención de resultados optimizados.

7 BIBLIOGRAFÍA

[1] Barney G. (2003). Elevator traffic handbook / Spon Press.

[2] CIBSE Guide D (2000). Transportation systems in buildings, pp 3-2.

[3] Cortés P, Fernández J., Delgado M. (2009). Controlador basado en lógica difusa para la detección del patrón de tráfico en sistemas de transporte vertical. XIII Congreso de Ingeniería de Organización. Barcelona-Terrasa, 2-4 Septiembre 2009.

[4] Cortés P., Larrañeta J.. (2003). Genetic algorithm for controllers in elevator groups. pp. 164.

[5] Cortés P., Larrañeta J., Onieva L., Muñuzuri J., Guadix J. (2003). Algoritmos para la Asignación de Llamadas en Sistemas de Tráfico Vertical Selectivo en Bajada. V Congreso de Ingeniería de Organización. Valladolid, 4-5 Septiembre 2003 pp. 1-4.

[6] Glover F. (1989). Tabu Search- Part I. ORSA Journal on Computing Vol. I, No. 3, Summer 1989.

[7] Ledesma, A. (2016). Asignación de llamadas para ascensores con arquitectura Double Deck mediante algoritmos de Búsqueda Tabú. Sevilla. Junio 2016 pp. 34-41.

[8] Ascensor panorámico. (2006). http://www.enor.es/catalogo_otea_ES.pdf
Último acceso: 27/06/2017

[9] Ascensores multicabina. (2016). http://www.urban-hub.com/es/ideas/la-nueva-era-de-los-ascensores-revoluciona-la-construccion-de-mediana-y-gran-altura/
Último acceso: 27/06/2017

[10] Cálculo de ascensores según especificaciones. (2003).
http://www3.amb.cat/normaurb/EDIFICACIO/2-2-5-3.pdf
Último acceso: 27/06/2017

[11] Double Decker Elevators On The Rise. (2014).
http://www.elevatordesigninfo.com/double-decker-elevators-on-the-rise
Último acceso: 27/06/2017

[12] Hub, U. (2016). Idea Nueva era de los ascensores. http://www.urban-hub.com/es/ideas/la-nueva-era-de-los-ascensores-revoluciona-la-construccion-de-mediana-y-gran-altura/
Último acceso: 27/06/2017

[13] Megaconstrucciones. (2014). http://megaconstrucciones.net/?construccion=torre-sidney
Último acceso: 27/06/2017

[14] Parámetros que determinan el número de ascensores de un edificio. (2010). http://www.astarlifts.com/blog/ascensores-elevadores/que-determina-el-numero-de-ascensores-que-necesita-un-edificio
Último acceso: 27/06/2017

[15] Patricio Barros, A. B. (2001). Libros Maravillosos. Capítulo 17: http://www.librosmaravillosos.com/estoyaexistioenlaantiguedad/capitulo17.html
Último acceso: 27/06/2017

[16] Peters, R. D. (2014). Elevator Dispatching. http://www.peters-research.com/index.php/support/articles-and-papers/151-elevator-dispatching
Último acceso: 27/06/2017

[17] Rodríguez-Piñero, P. T. (s.f.). Introducción a los algoritmos genéticos y sus aplicaciones. https://www.uv.es/asepuma/X/J24C.pdf
Último acceso: 27/06/2017

[18] Sánchez, Á. G. (2006). Técnicas metaheurísticas. http://www.iol.etsii.upm.es/arch/metaheuristicas.pdf
Último acceso: 27/06/2017

[19] ThyssenKrupp. (2017). ELI-Elevator Information. http://www.thyssenkrupp-elevator-eli.com/nc/es/unternehmen/referencias-csc-new-installations/essen.html
Último acceso: 27/06/2017

[20] Tipos de ascensores para edificaciones. Arquigrafico. Arquitectura, Ingeniería y Edificación. (2013). http://www.t3mascensores.com/tipos-de-ascensores-para-las-edificaciones/
Último acceso: 27/06/2017

[21] Torre Picasso. (2012). http://www.elmundo.es/elmundo/2011/12/29/economia/1325177138.html
Último acceso: 27/06/2017

[22] Un poco de historia. (2016). http://www.silcon.com.ar/un_poco_de_historia.htm
Último acceso: 27/06/2017

[23] TWIN. http://liftpages.ru/news/item/Lifty-Bashni-Federatsiya
Último acceso: 27/06/2017

[24] Wikipedia. (2016). Greenwich Street. https://es.wikipedia.org/wiki/388_Greenwich_Street
Último acceso: 27/06/2017

[25] Wikipedia. (2016). Frenchurh Street. https://en.wikipedia.org/wiki/20_Fenchurch_Street
Último acceso: 27/06/2017

[26] Wikipedia. (2016). Torre de Shanghai.
https://es.wikipedia.org/wiki/Torre_de_Shangh%C3%A1i
Último acceso: 27/06/2017

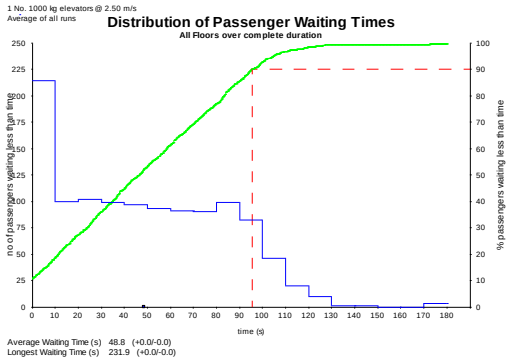
[27] Willis, C. (2017). World Towers above 380m.
http://skyscraper.org/EXHIBITIONS/SUPERTALL/federation.php
Último acceso: 27/06/2017

ANEXO A: GRÁFICAS EXPERIMENTALES

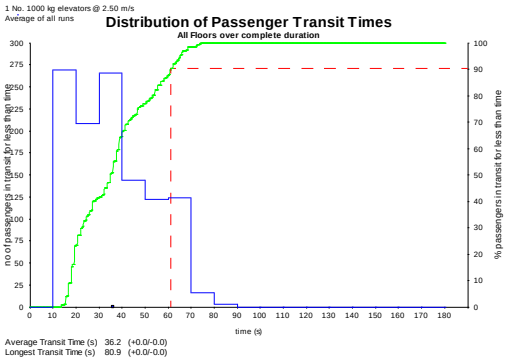
En este anexo se muestran todas las gráficas del tiempo media de espera de los pasajeros frente al ascensor y el tiempo medio de tránsito que ELEVATE 8 proporciona para cada una de las simulaciones realizadas. Se muestran las gráficas agrupadas según el algoritmo utilizado en cada caso para la asignación de ascensores a llamadas. Además, se refleja en cada gráfica el número de plantas del edificio y el número de ascensores para cada caso.

- **<u>Búsqueda Tabú</u>**

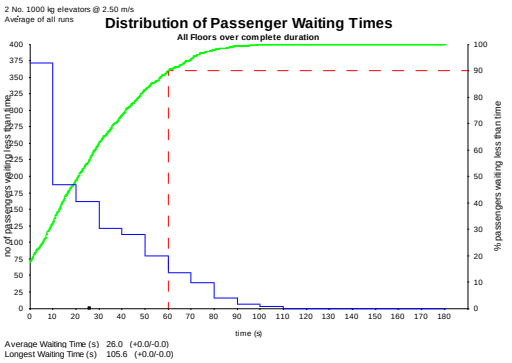
AWT 8P 1A



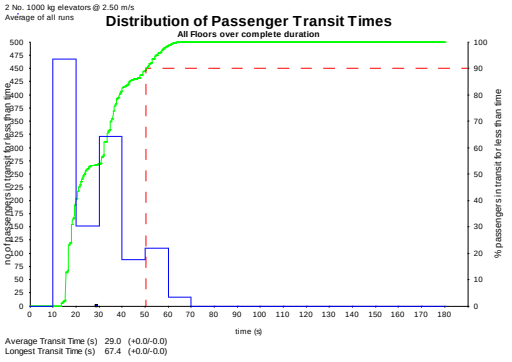
ATT 8P 1A



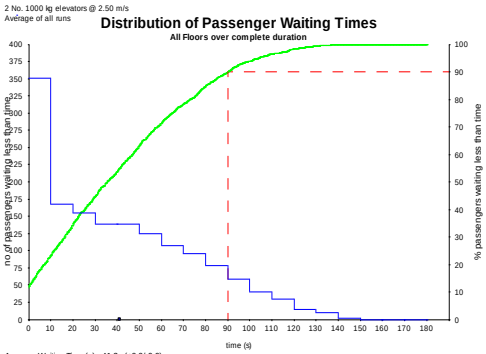
AWT 8P 2A



ATT 8P 2A

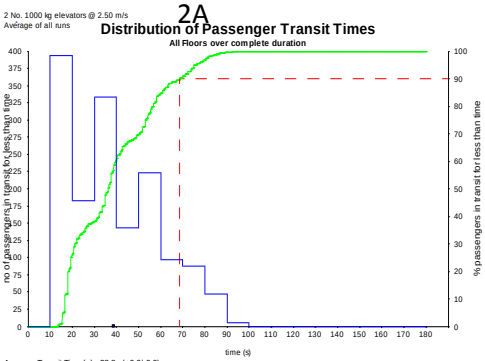


AWT 10P 2A



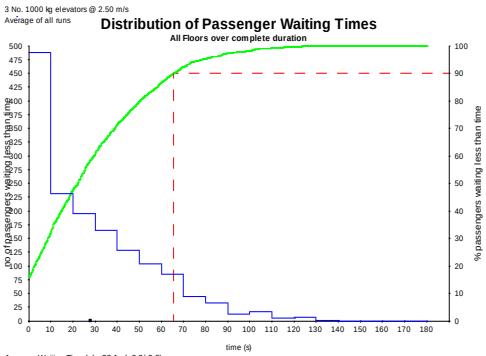
Average Waiting Time (s) 41.2 (+0.0/-0.0)
Longest Waiting Time (s) 145.9 (+0.0/-0.0)

ATT 10P 2A



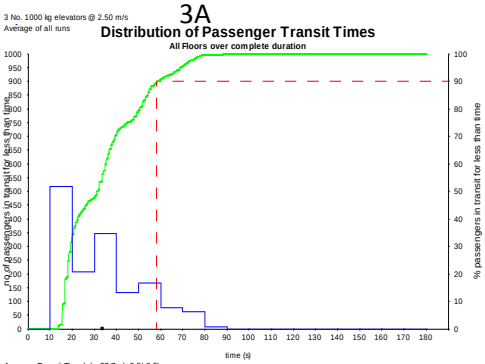
Average Transit Time (s) 38.8 (+0.0/-0.0)
Longest Transit Time (s) 94.7 (+0.0/-0.0)

AWT 10P 3A



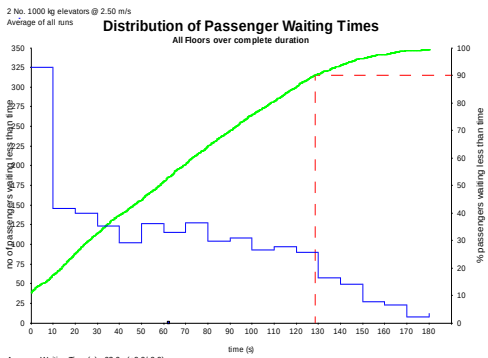
Average Waiting Time (s) 28.1 (+0.0/-0.0)
Longest Waiting Time (s) 130.3 (+0.0/-0.0)

ATT 10P 3A



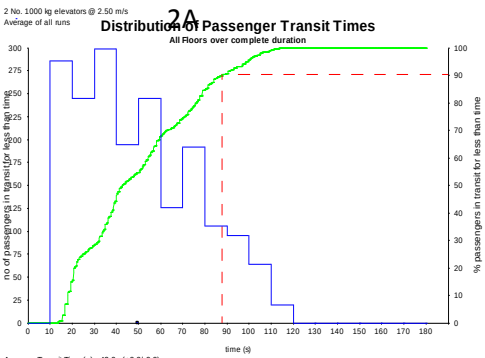
Average Transit Time (s) 33.7 (+0.0/-0.0)
Longest Transit Time (s) 89.6 (+0.0/-0.0)

AWT 12P 2A



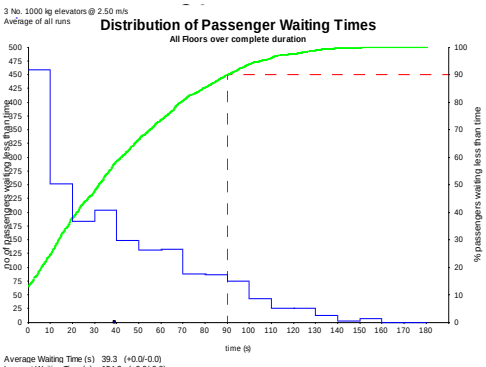
Average Waiting Time (s) 62.6 (+0.0/-0.0)
Longest Waiting Time (s) 281.6 (+0.0/-0.0)

ATT 12P 2A



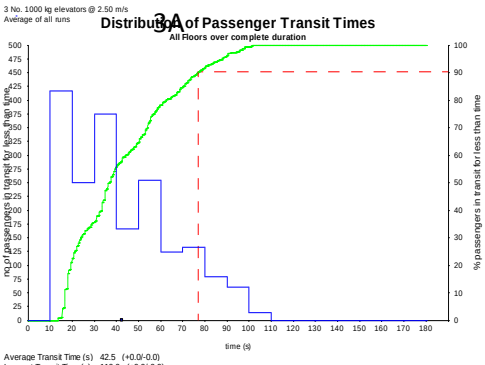
Average Transit Time (s) 49.6 (+0.0/-0.0)
Longest Transit Time (s) 117.8 (+0.0/-0.0)

AWT 12P



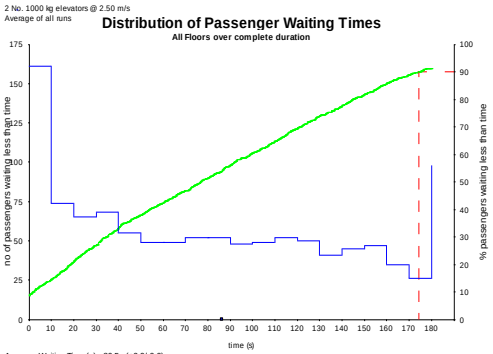
Average Waiting Time (s) 39.3 (+0.0/-0.0)
Longest Waiting Time (s) 154.2 (+0.0/-0.0)

ATT 12P 3A



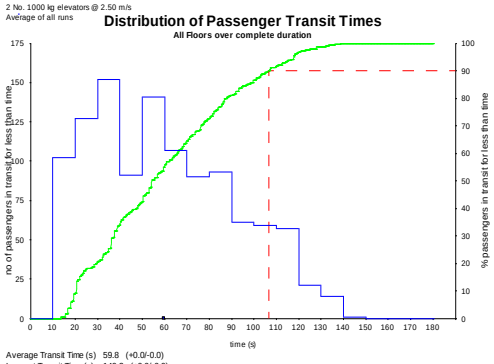
Average Transit Time (s) 42.5 (+0.0/-0.0)
Longest Transit Time (s) 110.0 (+0.0/-0.0)

AWT 14P 2A



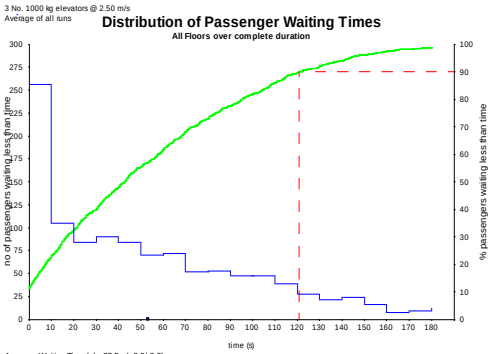
Average Waiting Time (s) 86.5 (+0.0/-0.0)
Longest Waiting Time (s) 438.6 (+0.0/-0.0)

ATT 14P 2A



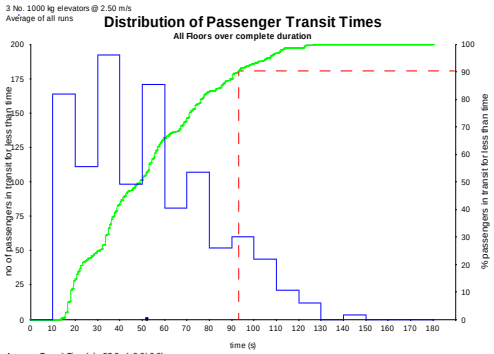
Average Transit Time (s) 59.8 (+0.0/-0.0)
Longest Transit Time (s) 140.2 (+0.0/-0.0)

AWT 14P 3A



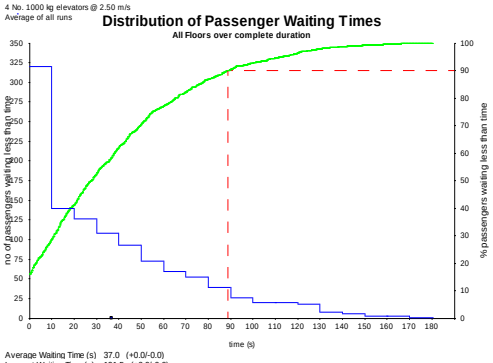
Average Waiting Time (s) 53.5 (+0.0/-0.0)
Longest Waiting Time (s) 425.9 (+0.0/-0.0)

ATT 14P 3A



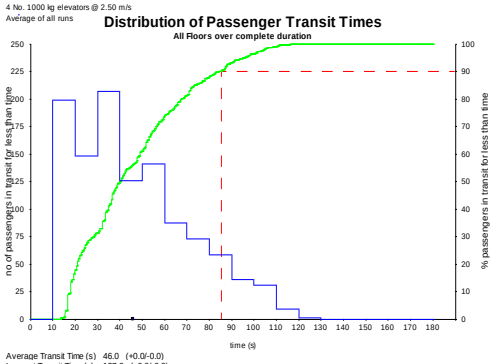
Average Transit Time (s) 52.3 (+0.0/-0.0)
Longest Transit Time (s) 141.1 (+0.0/-0.0)

AWT 14P 4A



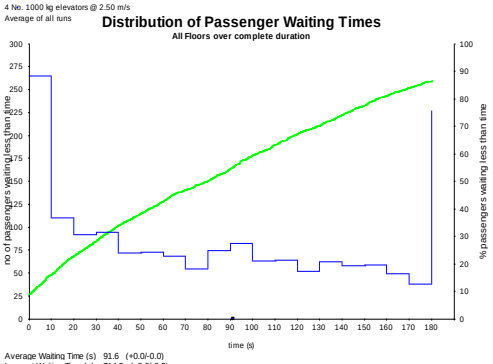
Average Waiting Time (s) 37.0 (+0.0/-0.0)
Longest Waiting Time (s) 191.5 (+0.0/-0.0)

ATT 14P 4A



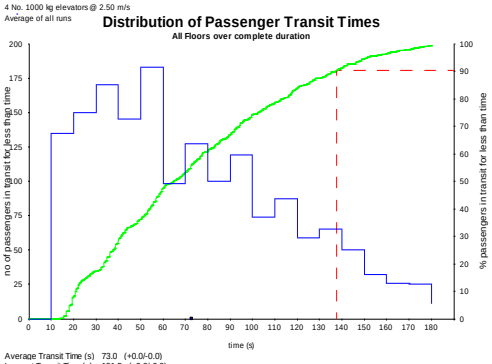
Average Transit Time (s) 46.0 (+0.0/-0.0)
Longest Transit Time (s) 127.6 (+0.0/-0.0)

AWT 20P 4A



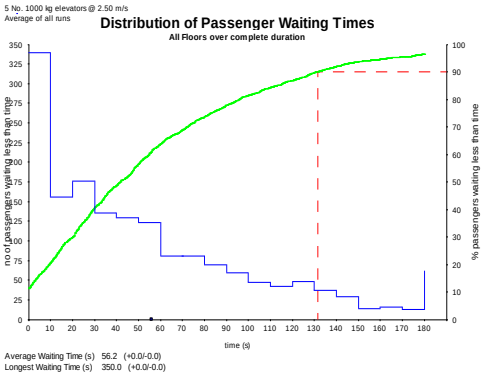
Average Waiting Time (s) 91.6 (+0.0/-0.0)
Longest Waiting Time (s) 714.5 (+0.0/-0.0)

ATT 20P 4A



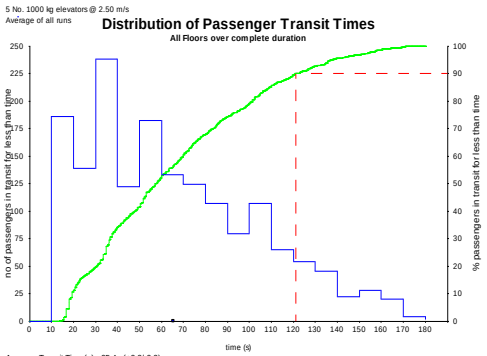
Average Transit Time (s) 73.0 (+0.0/-0.0)
Longest Transit Time (s) 191.5 (+0.0/-0.0)

AWT 20P 5A



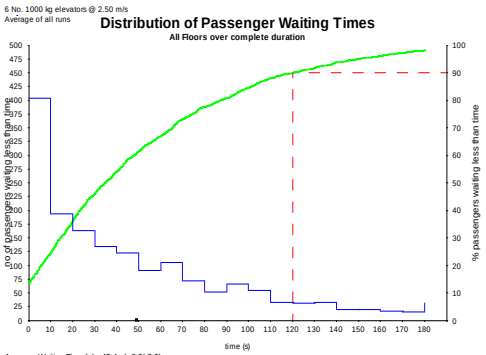
Average Waiting Time (s) 56.2 (+0.0/-0.0)
Longest Waiting Time (s) 350.0 (+0.0/-0.0)

ATT 20P 5A



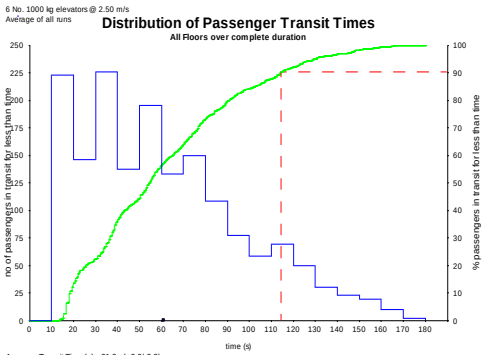
Average Transit Time (s) 65.4 (+0.0/-0.0)
Longest Transit Time (s) 183.7 (+0.0/-0.0)

AWT 20P 6A



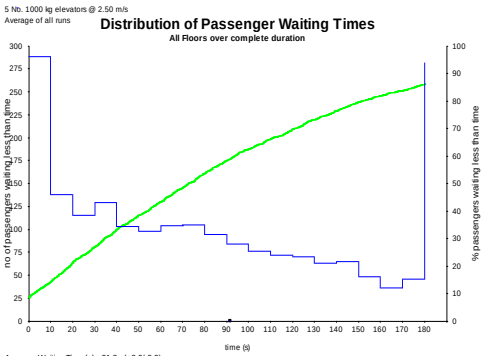
Average Waiting Time (s) 49.4 (+0.0/-0.0)
Longest Waiting Time (s) 352.2 (+0.0/-0.0)

ATT 20P 6A



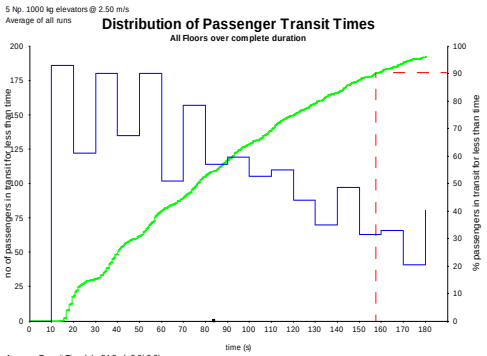
Average Transit Time (s) 61.0 (+0.0/-0.0)
Longest Transit Time (s) 177.2 (+0.0/-0.0)

AWT 24P 5A



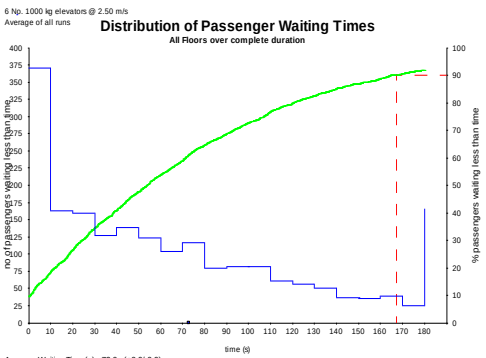
Average Waiting Time (s) 91.8 (+0.0/-0.0)
Longest Waiting Time (s) 550.7 (+0.0/-0.0)

ATT 24P 5A



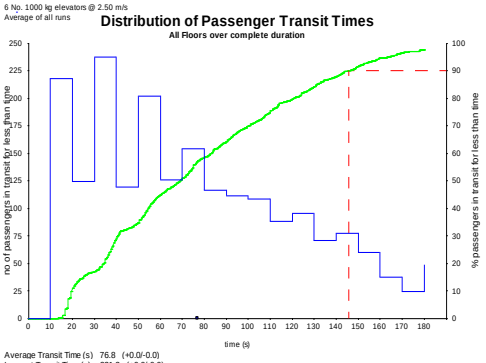
Average Transit Time (s) 84.0 (+0.0/-0.0)
Longest Transit Time (s) 227.4 (+0.0/-0.0)

AWT 24P 6A



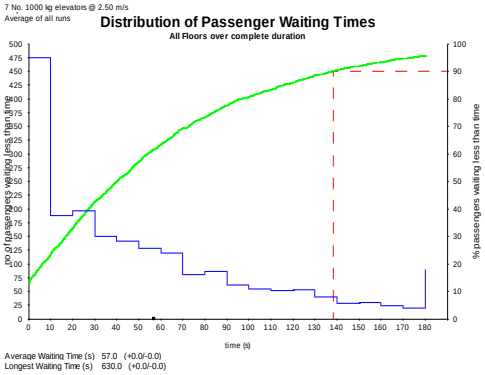
Average Waiting Time (s) 73.0 (+0.0/-0.0)
Longest Waiting Time (s) 497.7 (+0.0/-0.0)

ATT 24P 6A



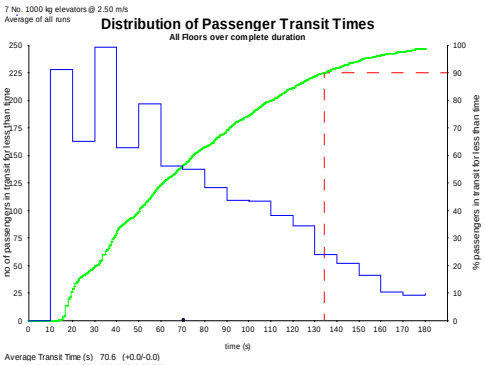
Average Transit Time (s) 76.8 (+0.0/-0.0)
Longest Transit Time (s) 221.2 (+0.0/-0.0)

AWT 24P 7A



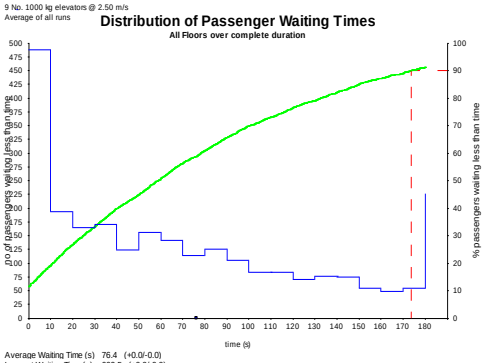
Average Waiting Time (s) 57.0 (+0.0/-0.0)
Longest Waiting Time (s) 630.0 (+0.0/-0.0)

ATT 24P 7A



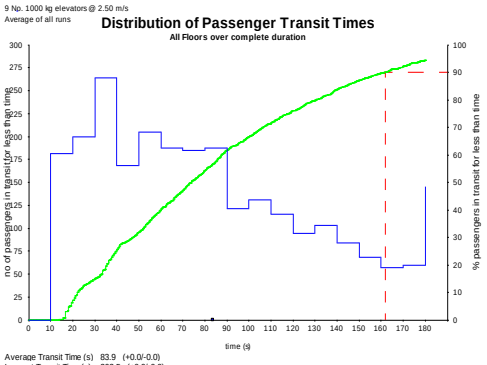
Average Transit Time (s) 70.6 (+0.0/-0.0)
Longest Transit Time (s) 209.5 (+0.0/-0.0)

AWT 30P 9A



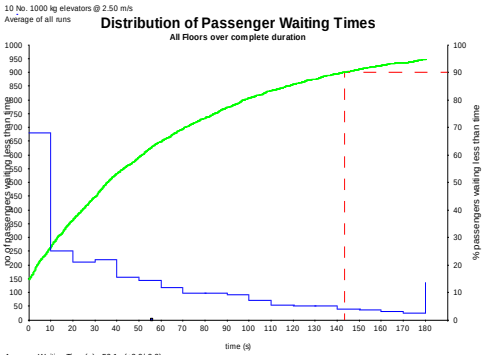
Average Waiting Time (s) 76.4 (+0.0/-0.0)
Longest Waiting Time (s) 669.5 (+0.0/-0.0)

ATT 30P 9A



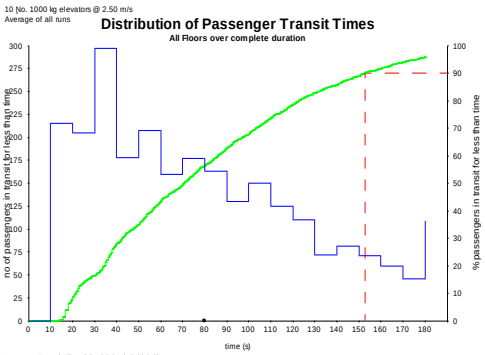
Average Transit Time (s) 83.9 (+0.0/-0.0)
Longest Transit Time (s) 262.5 (+0.0/-0.0)

AWT 30P 10A



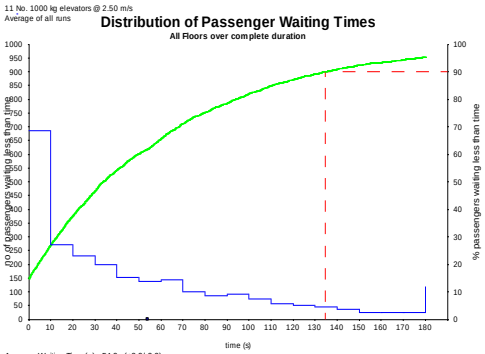
Average Waiting Time (s) 56.1 (+0.0/-0.0)
Longest Waiting Time (s) 437.1 (+0.0/-0.0)

ATT 30P 10A



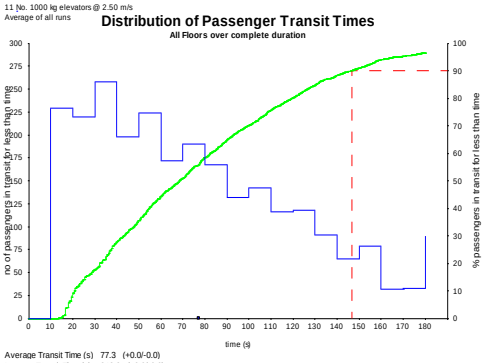
Average Transit Time (s) 80.0 (+0.0/-0.0)
Longest Transit Time (s) 269.3 (+0.0/-0.0)

AWT 30P 11A



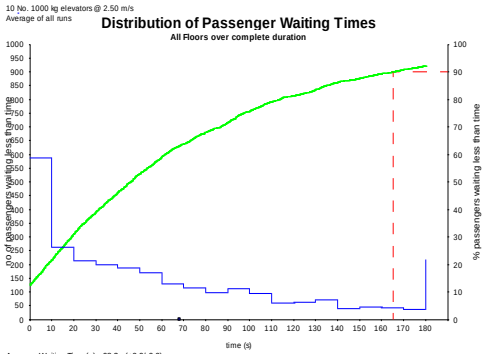
Average Waiting Time (s) 54.0 (+0.0/-0.0)
Longest Waiting Time (s) 373.9 (+0.0/-0.0)

ATT 30P 11A



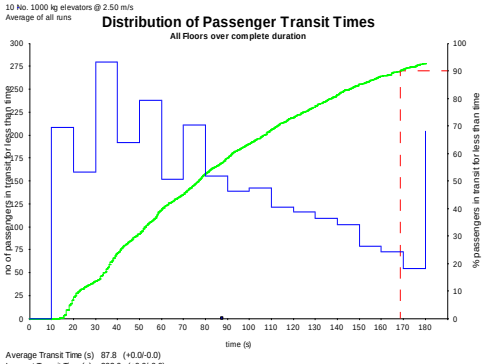
Average Transit Time (s) 77.3 (+0.0/-0.0)
Longest Transit Time (s) 248.8 (+0.0/-0.0)

AWT 32P 10A



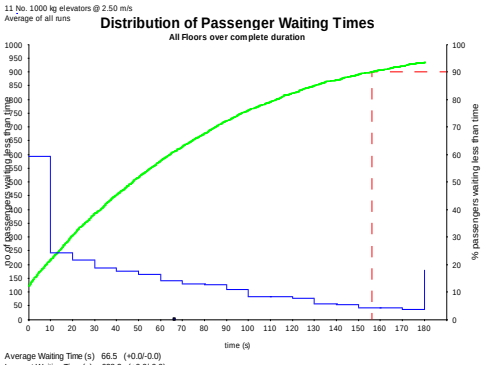
Average Waiting Time (s) 68.2 (+0.0/-0.0)
Longest Waiting Time (s) 741.3 (+0.0/-0.0)

ATT 32P 10A



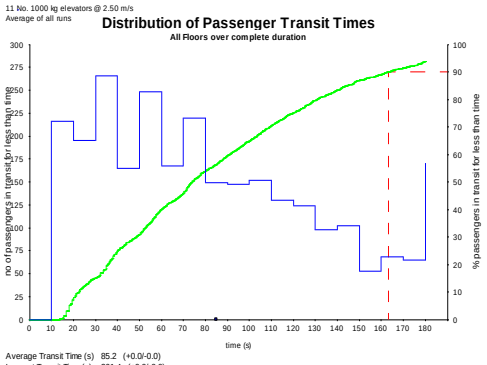
Average Transit Time (s) 87.8 (+0.0/-0.0)
Longest Transit Time (s) 302.9 (+0.0/-0.0)

AWT 32P 11A



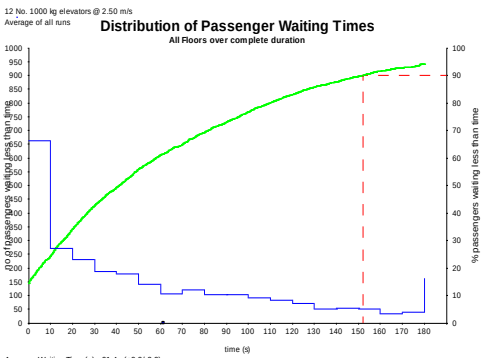
Average Waiting Time (s) 66.5 (+0.0/-0.0)
Longest Waiting Time (s) 628.2 (+0.0/-0.0)

ATT 32P 11A



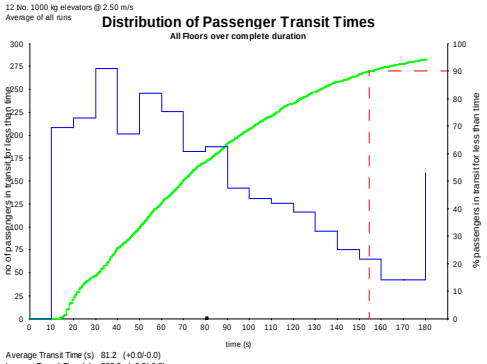
Average Transit Time (s) 85.2 (+0.0/-0.0)
Longest Transit Time (s) 261.4 (+0.0/-0.0)

AWT 32P 12A



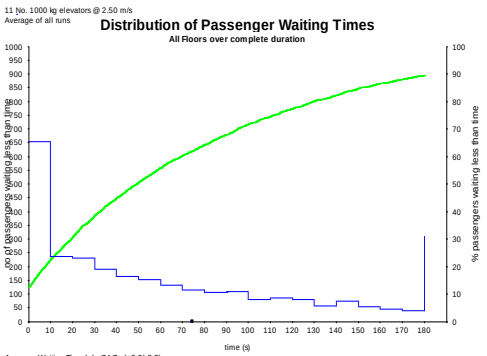
Average Waiting Time (s) 61.4 (+0.0/-0.0)
Longest Waiting Time (s) 475.2 (+0.0/-0.0)

ATT 32P 12A



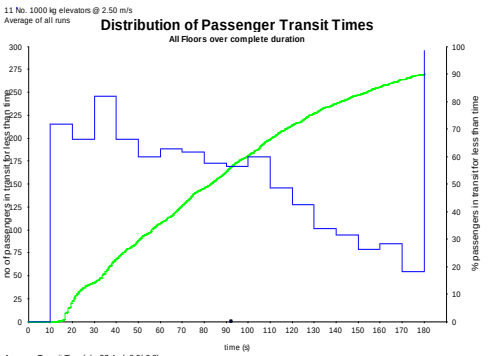
Average Transit Time (s) 81.2 (+0.0/-0.0)
Longest Transit Time (s) 269.9 (+0.0/-0.0)

AWT 34P 11A



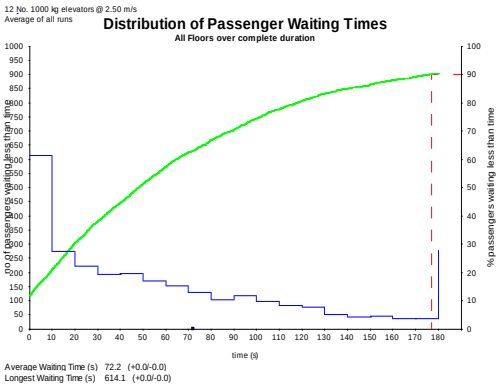
Average Waiting Time (s) 74.7 (+0.0/-0.0)
Longest Waiting Time (s) 626.6 (+0.0/-0.0)

ATT 34P 11A



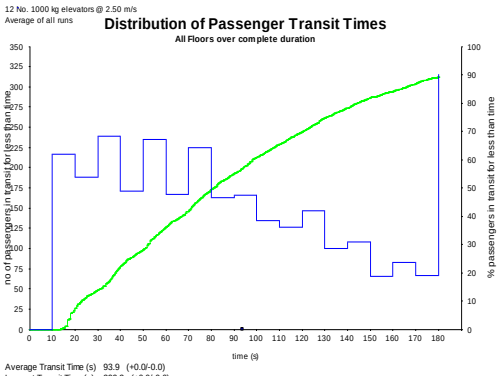
Average Transit Time (s) 92.4 (+0.0/-0.0)
Longest Transit Time (s) 289.1 (+0.0/-0.0)

AWT 34P 12A



Average Waiting Time (s) 72.2 (+0.0/-0.0)
Longest Waiting Time (s) 614.1 (+0.0/-0.0)

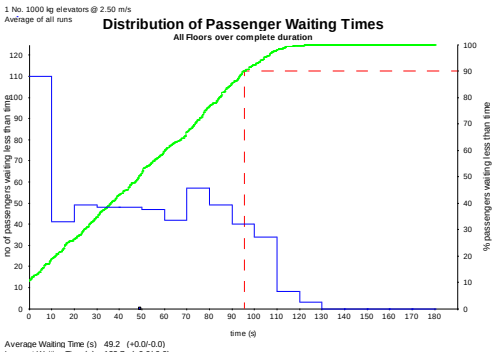
ATT 34P 12A



Average Transit Time (s) 93.9 (+0.0/-0.0)
Longest Transit Time (s) 299.2 (+0.0/-0.0)

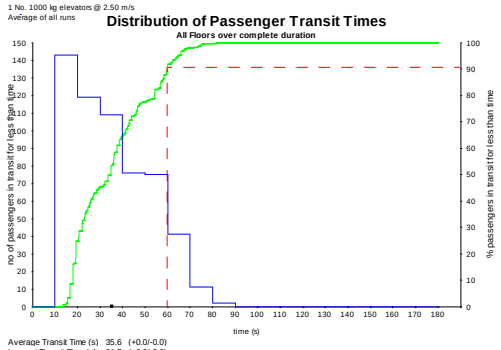
- **<u>Algoritmo Genético</u>**

AWT 8P 1A



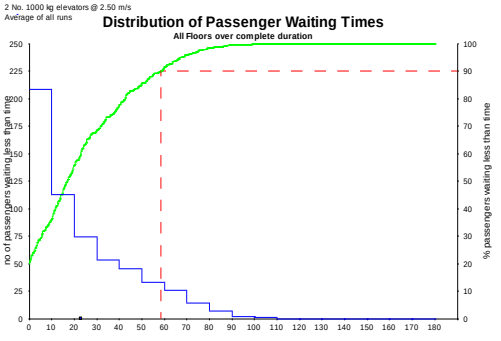
Average Waiting Time (s) 49.2 (+0.0/-0.0)
Longest Waiting Time (s) 122.7 (+0.0/-0.0)

ATT 8P 2A



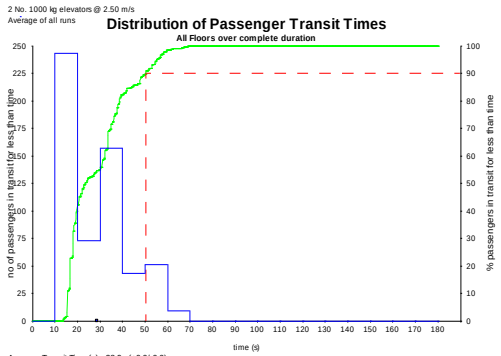
Average Transit Time (s) 35.6 (+0.0/-0.0)
Longest Transit Time (s) 81.7 (+0.0/-0.0)

AWT 8P 2A



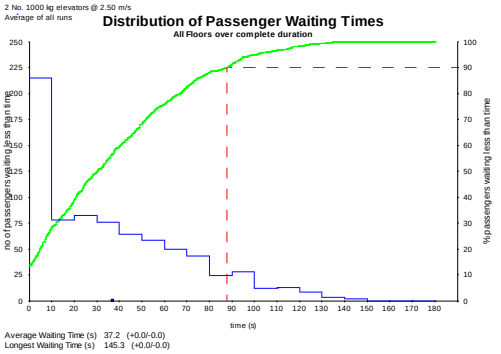
Average Waiting Time (s) 23.0 (+0.0/-0.0)
Longest Waiting Time (s) 109.9 (+0.0/-0.0)

ATT 8P 1A



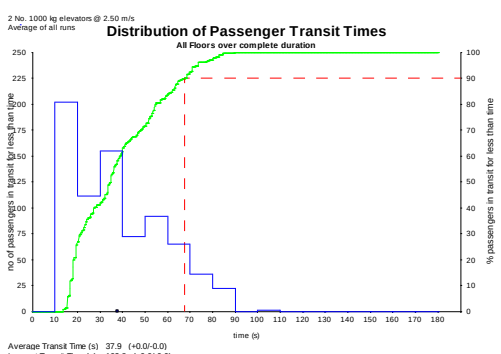
Average Transit Time (s) 28.9 (+0.0/-0.0)
Longest Transit Time (s) 68.9 (+0.0/-0.0)

AWT 10P 2A



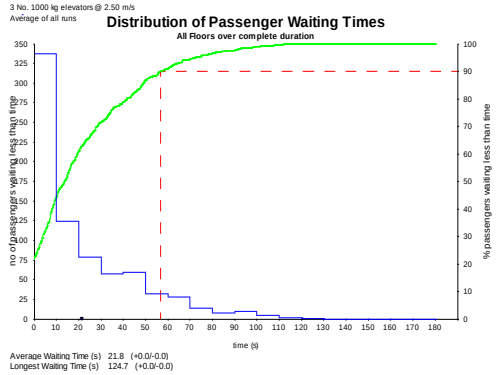
Average Waiting Time (s) 37.2 (+0.0/-0.0)
Longest Waiting Time (s) 145.3 (+0.0/-0.0)

ATT 10P 2A



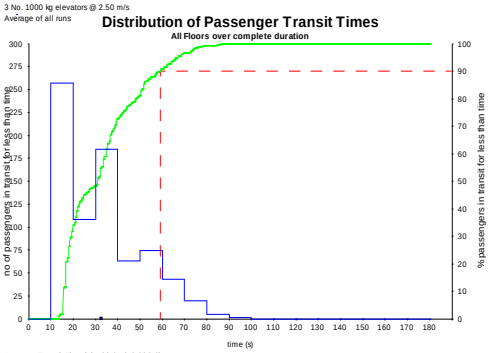
Average Transit Time (s) 37.9 (+0.0/-0.0)
Longest Transit Time (s) 102.8 (+0.0/-0.0)

AWT 10P 3A



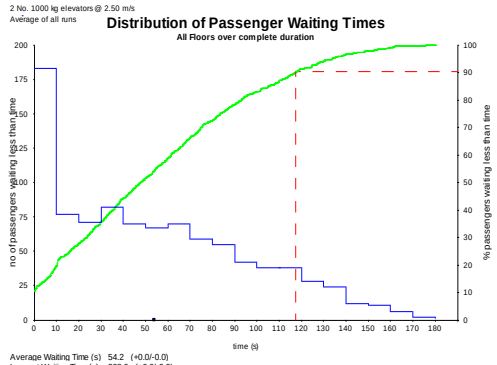
Average Waiting Time (s) 21.8 (+0.0/-0.0)
Longest Waiting Time (s) 124.7 (+0.0/-0.0)

ATT 10P 3A



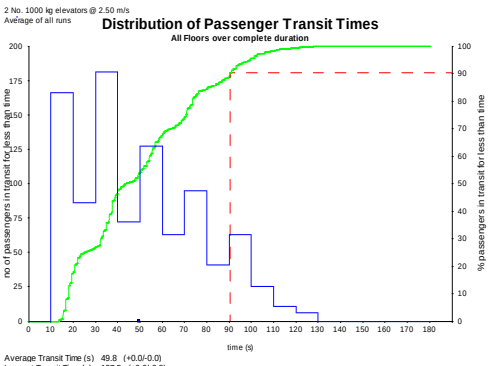
Average Transit Time (s) 33.0 (+0.0/-0.0)
Longest Transit Time (s) 95.9 (+0.0/-0.0)

AWT 12P 2A



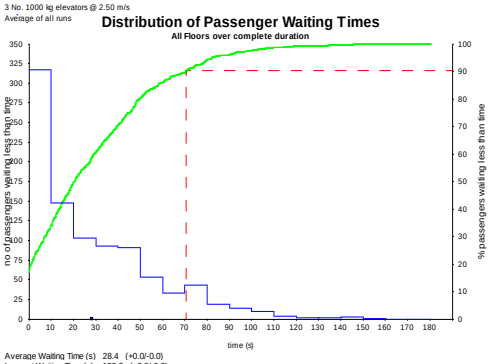
Average Waiting Time (s) 54.2 (+0.0/-0.0)
Longest Waiting Time (s) 208.6 (+0.0/-0.0)

ATT 12P 2A



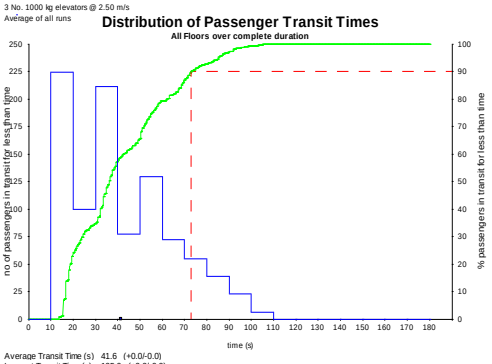
Average Transit Time (s) 49.8 (+0.0/-0.0)
Longest Transit Time (s) 127.5 (+0.0/-0.0)

AWT 12P 3A



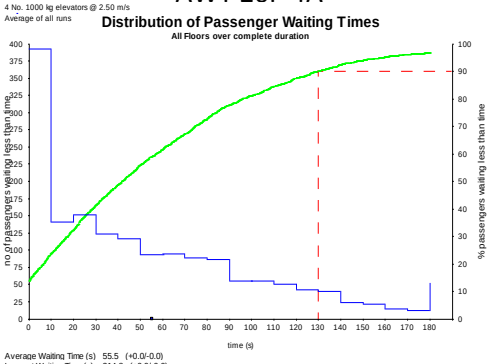
Average Waiting Time (s) 28.4 (+0.0/-0.0)
Longest Waiting Time (s) 153.3 (+0.0/-0.0)

ATT 12P 3A



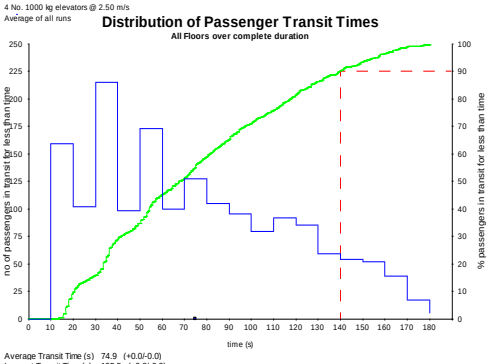
Average Transit Time (s) 41.6 (+0.0/-0.0)
Longest Transit Time (s) 105.0 (+0.0/-0.0)

AWT 20P 4A



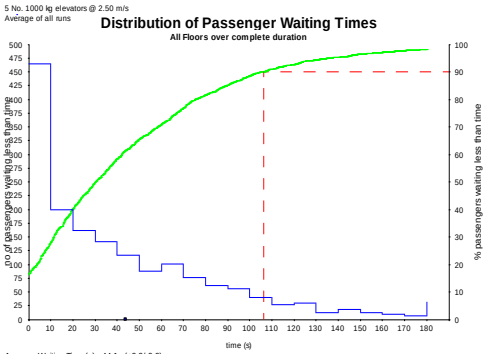
Average Waiting Time (s) 55.5 (+0.0/-0.0)
Longest Waiting Time (s) 314.8 (+0.0/-0.0)

ATT 20P 4A



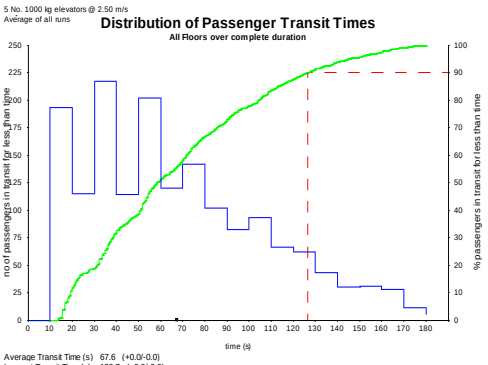
Average Transit Time (s) 74.9 (+0.0/-0.0)
Longest Transit Time (s) 195.5 (+0.0/-0.0)

AWT 20P 5A



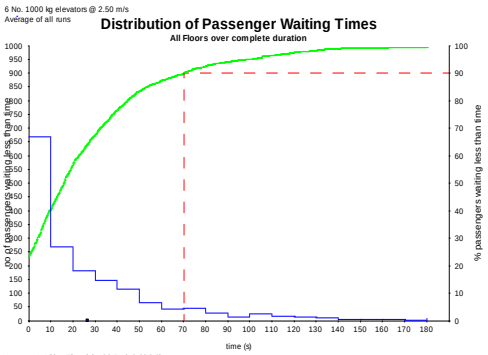
Average Waiting Time (s) 44.1 (+0.0/-0.0)
Longest Waiting Time (s) 284.6 (+0.0/-0.0)

AWT 20P 5A



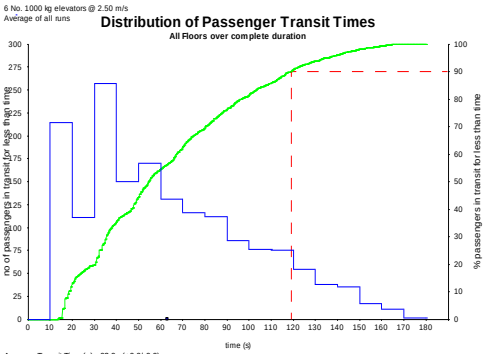
Average Transit Time (s) 67.6 (+0.0/-0.0)
Longest Transit Time (s) 190.7 (+0.0/-0.0)

ATT 20P 6A



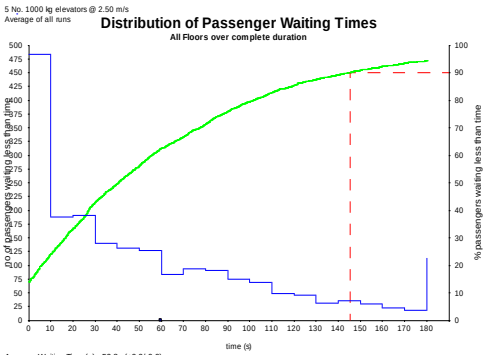
Average Waiting Time (s) 26.7 (+0.0/-0.0)
Longest Waiting Time (s) 239.6 (+0.0/-0.0)

ATT 20P 6A



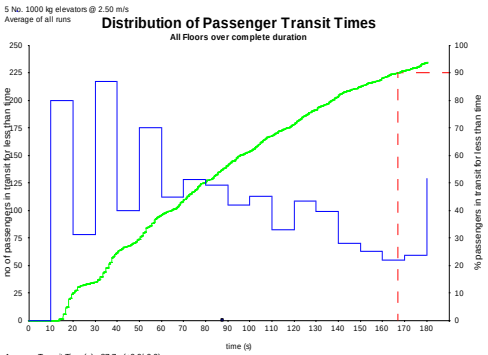
Average Transit Time (s) 63.0 (+0.0/-0.0)
Longest Transit Time (s) 185.1 (+0.0/-0.0)

AWT 24P 5A



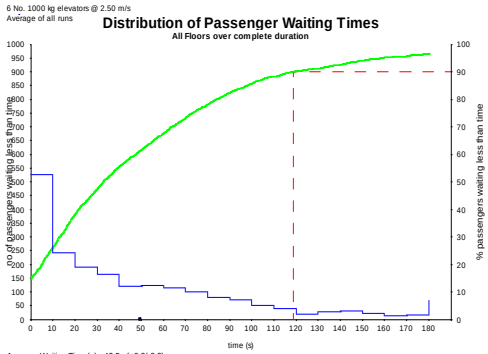
Average Waiting Time (s) 59.8 (+0.0/-0.0)
Longest Waiting Time (s) 559.8 (+0.0/-0.0)

ATT 24P 5A



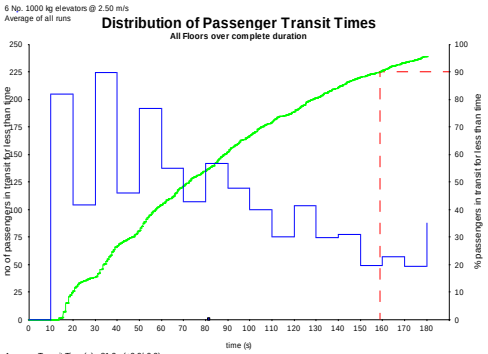
Average Transit Time (s) 87.7 (+0.0/-0.0)
Longest Transit Time (s) 236.5 (+0.0/-0.0)

AWT 24P 6A



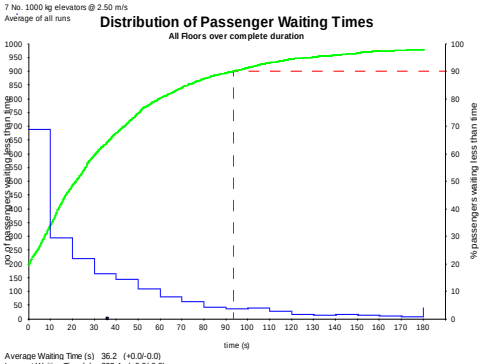
Average Waiting Time (s) 49.5 (+0.0/-0.0)
Longest Waiting Time (s) 324.7 (+0.0/-0.0)

ATT 24P 6A



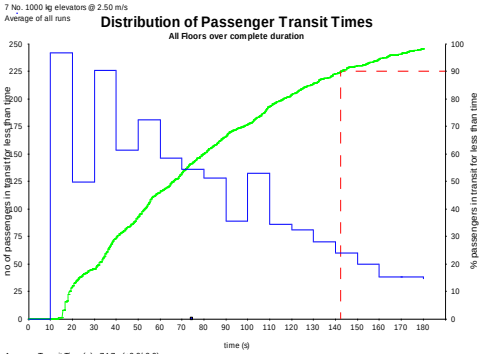
Average Transit Time (s) 81.9 (+0.0/-0.0)
Longest Transit Time (s) 229.0 (+0.0/-0.0)

AWT 24P 7A



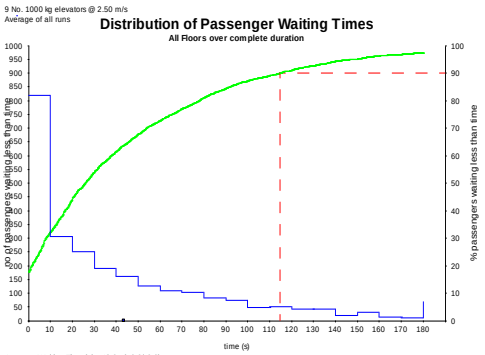
Average Waiting Time (s) 36.2 (+0.0/-0.0)
Longest Waiting Time (s) 293.4 (+0.0/-0.0)

ATT 24P 7A



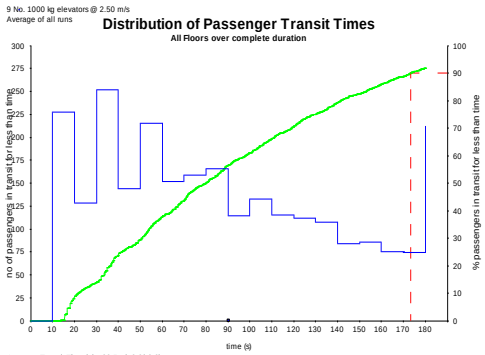
Average Transit Time (s) 74.7 (+0.0/-0.0)
Longest Transit Time (s) 224.2 (+0.0/-0.0)

AWT 30P 9A



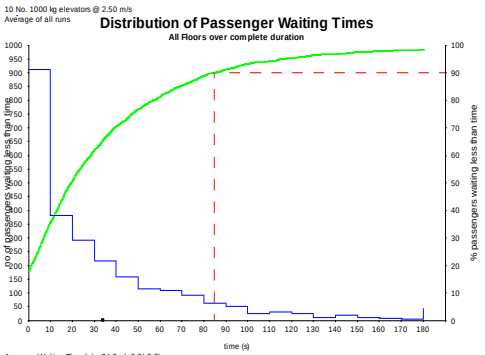
Average Waiting Time (s) 43.6 (+0.0/-0.0)
Longest Waiting Time (s) 312.1 (+0.0/-0.0)

ATT 30P 9A



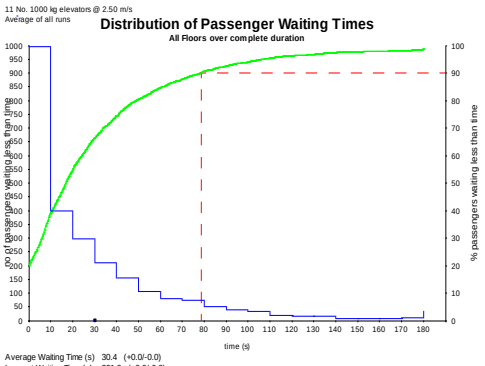
Average Transit Time (s) 90.5 (+0.0/-0.0)
Longest Transit Time (s) 285.7 (+0.0/-0.0)

AWT 30P 10A



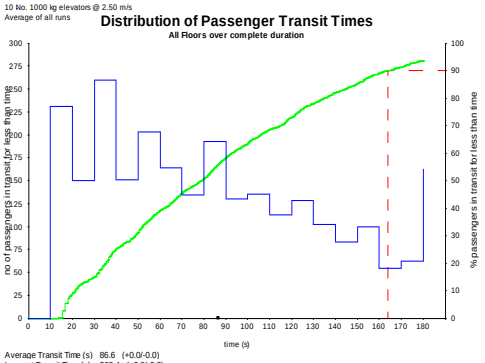
Average Waiting Time (s) 34.2 (+0.0/-0.0)
Longest Waiting Time (s) 425.0 (+0.0/-0.0)

AWT 30P 11A



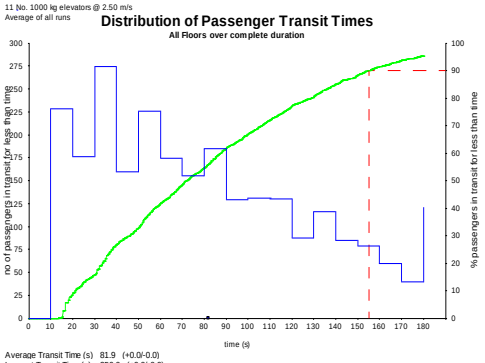
Average Waiting Time (s) 30.4 (+0.0/-0.0)
Longest Waiting Time (s) 321.9 (+0.0/-0.0)

ATT 30P 10A



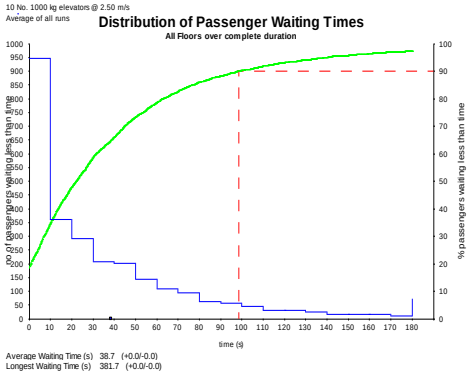
Average Transit Time (s) 86.6 (+0.0/-0.0)
Longest Transit Time (s) 260.4 (+0.0/-0.0)

ATT 30P 11A

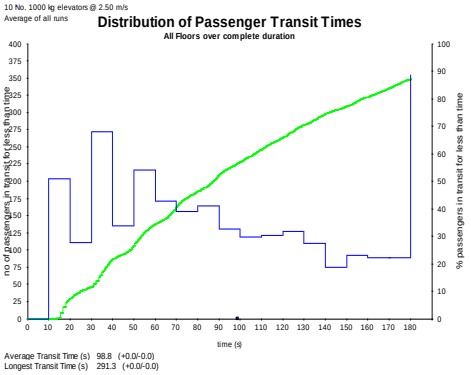


Average Transit Time (s) 81.9 (+0.0/-0.0)
Longest Transit Time (s) 256.0 (+0.0/-0.0)

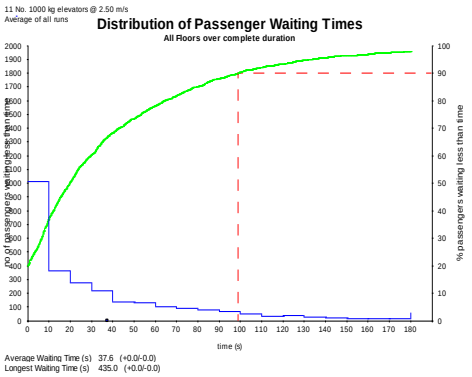
AWT32P 10A



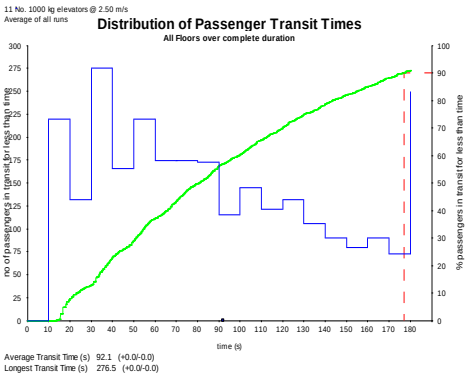
ATT 32P 10A



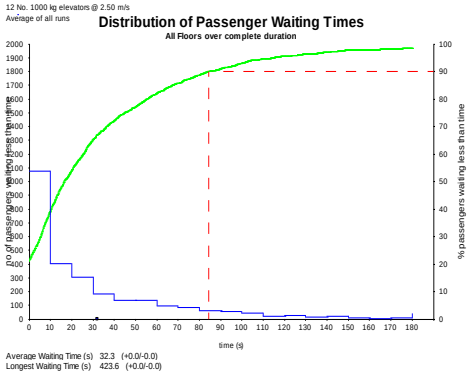
AWT 32P 11A



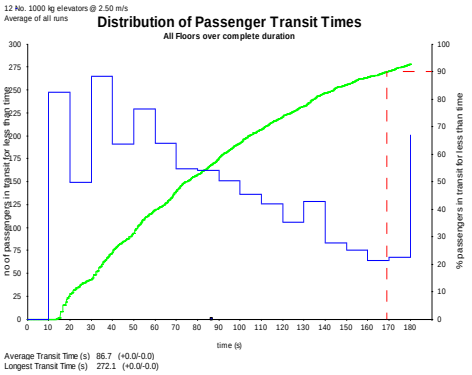
ATT 32P 11A



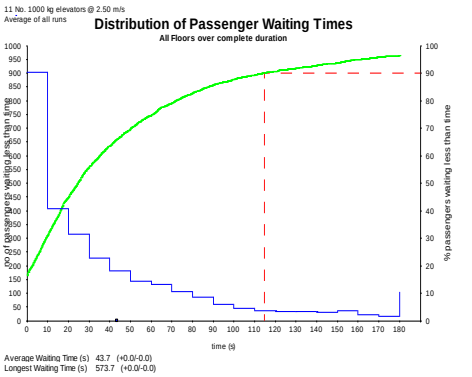
AWT 32P 12A



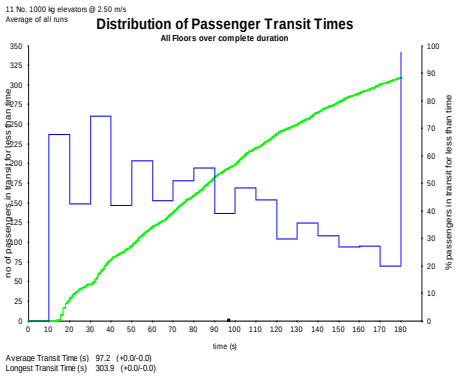
ATT 32P 12A



AWT 34P 11A

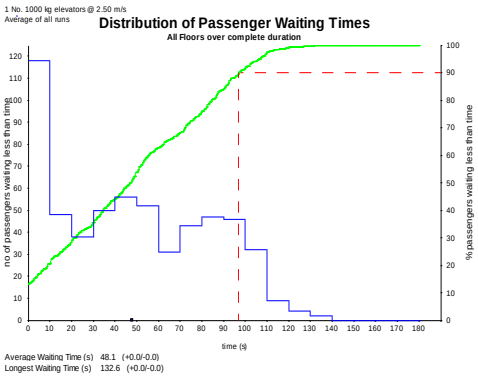


ATT 34P 11A

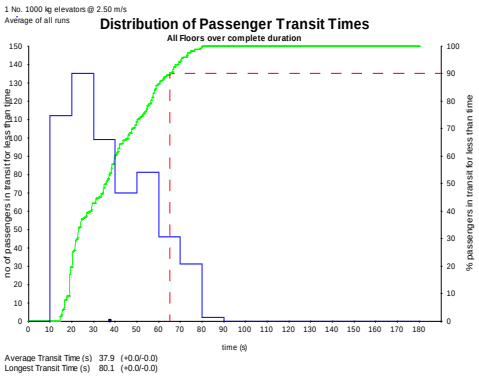


- **Double Deck -ETA**

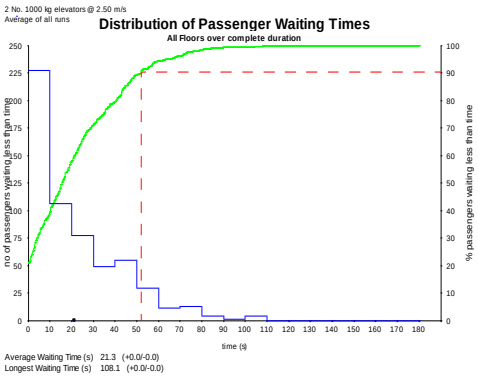
AWT 8P 1A



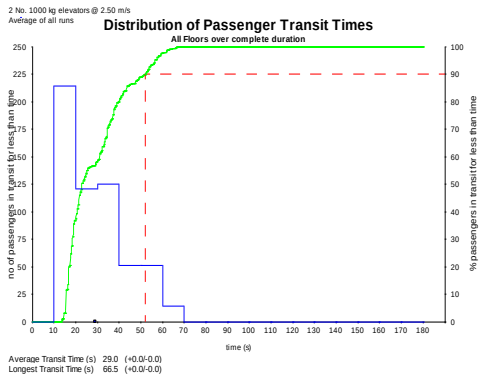
ATT 8P 1A



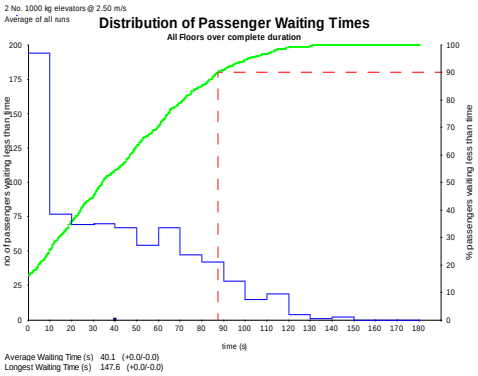
AWT 8P 2A



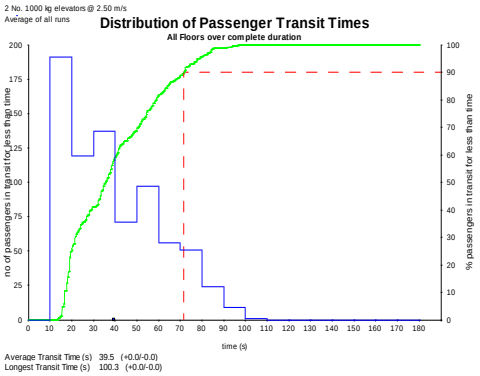
ATT 8P 2A



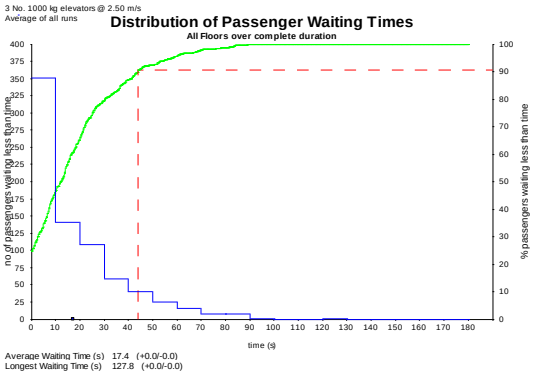
AWT 10P 2A



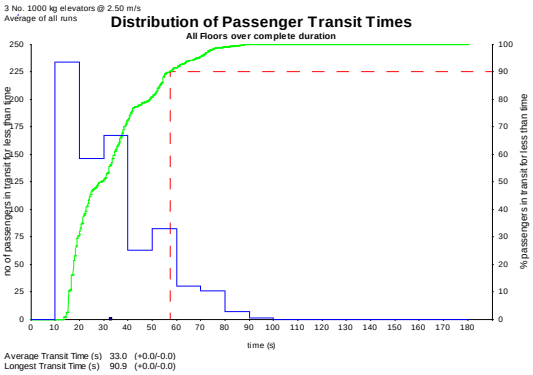
ATT 10P 2A



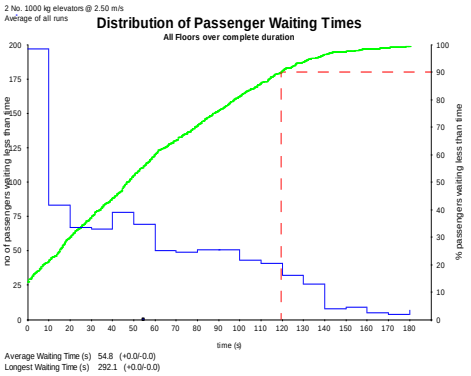
AWT 10P 3A



ATT 10P 3A

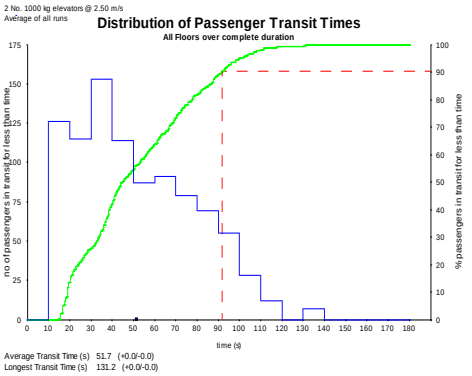


AWT 12P 2A



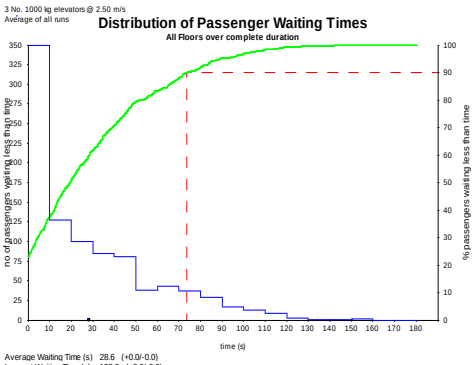
Average Waiting Time (s) 54.8 (+0.0/-0.0)
Longest Waiting Time (s) 292.1 (+0.0/-0.0)

ATT 12P 2A



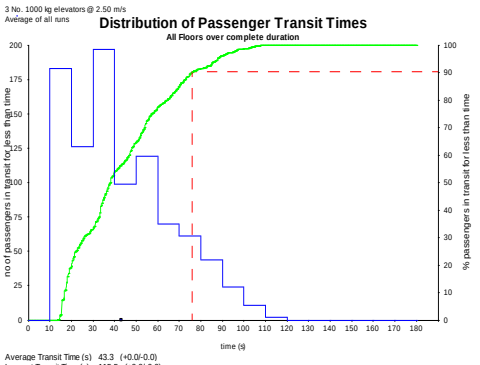
Average Transit Time (s) 51.7 (+0.0/-0.0)
Longest Transit Time (s) 131.2 (+0.0/-0.0)

AWT 12P 3A



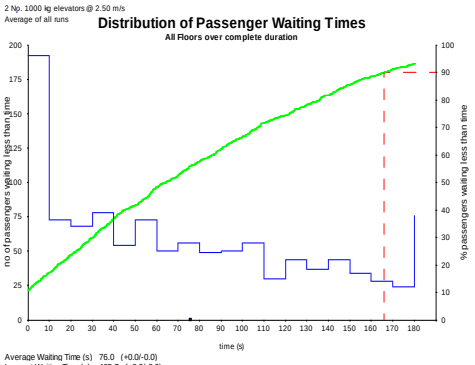
Average Waiting Time (s) 28.6 (+0.0/-0.0)
Longest Waiting Time (s) 158.6 (+0.0/-0.0)

ATT 12P 3A



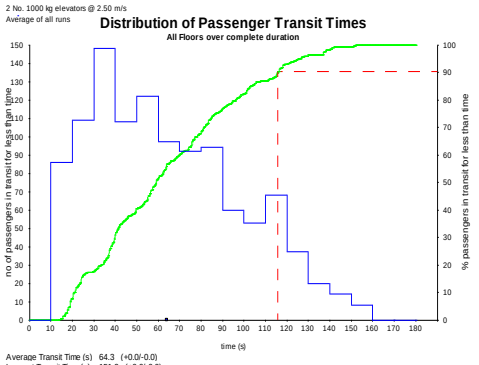
Average Transit Time (s) 43.3 (+0.0/-0.0)
Longest Transit Time (s) 115.5 (+0.0/-0.0)

AWT 14P 2A



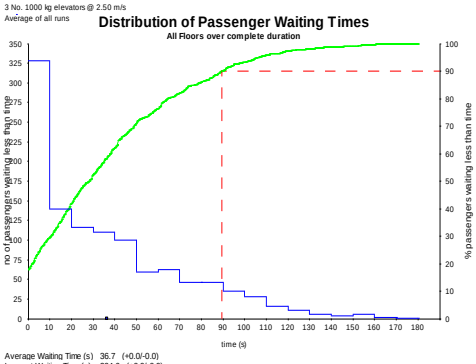
Average Waiting Time (s) 76.0 (+0.0/-0.0)
Longest Waiting Time (s) 465.7 (+0.0/-0.0)

ATT 14P 2A



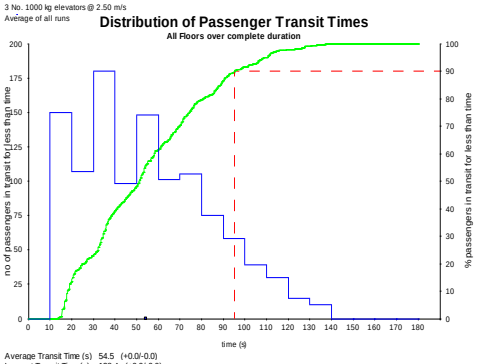
Average Transit Time (s) 64.3 (+0.0/-0.0)
Longest Transit Time (s) 151.9 (+0.0/-0.0)

AWT 14P 3A



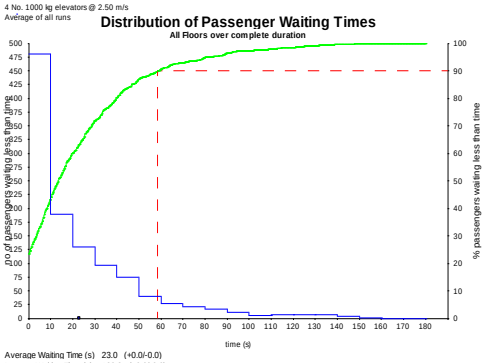
Average Waiting Time (s) 36.7 (+0.0/-0.0)
Longest Waiting Time (s) 204.6 (+0.0/-0.0)

ATT 14P 3A



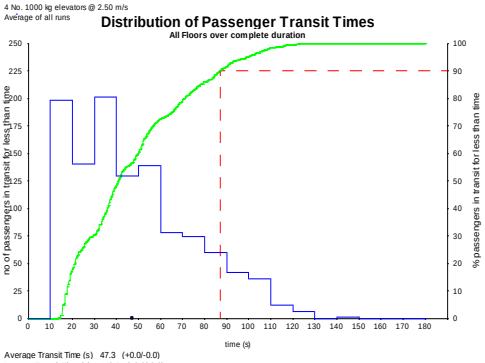
Average Transit Time (s) 54.5 (+0.0/-0.0)
Longest Transit Time (s) 138.4 (+0.0/-0.0)

AWT 14P 4A



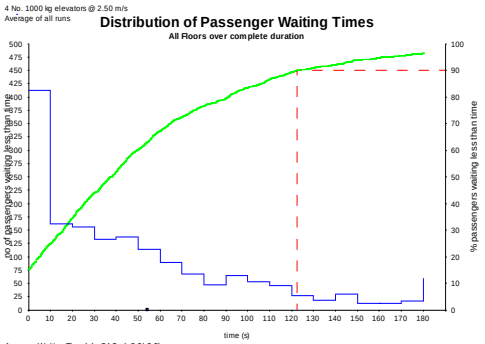
Average Waiting Time (s) 23.0 (+0.0/-0.0)
Longest Waiting Time (s) 182.3 (+0.0/-0.0)

ATT 14P 4A



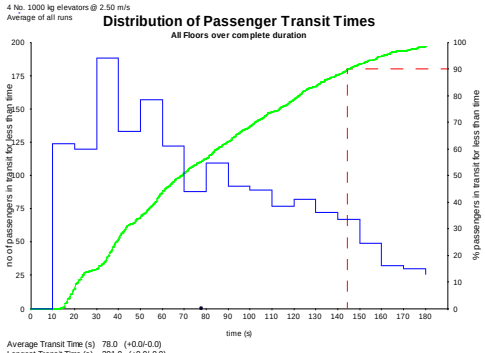
Average Transit Time (s) 47.3 (+0.0/-0.0)
Longest Transit Time (s) 143.5 (+0.0/-0.0)

AWT 20P 4A



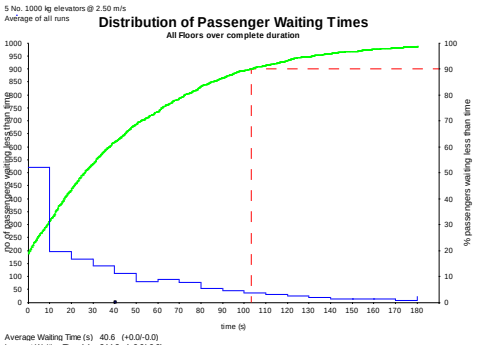
Average Waiting Time (s) 54.3 (+0.0/-0.0)
Longest Waiting Time (s) 748.5 (+0.0/-0.0)

ATT 20P 4A



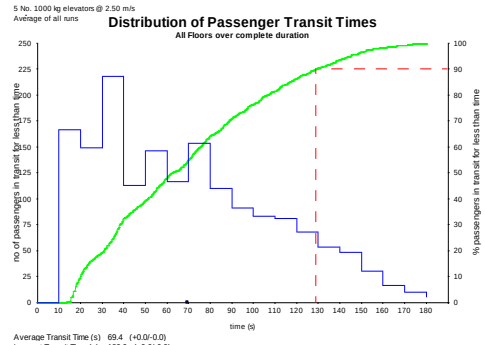
Average Transit Time (s) 78.0 (+0.0/-0.0)
Longest Transit Time (s) 201.9 (+0.0/-0.0)

AWT 20P 5A



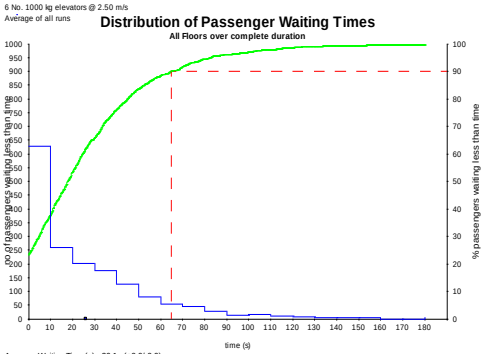
Average Waiting Time (s) 40.6 (+0.0/-0.0)
Longest Waiting Time (s) 244.2 (+0.0/-0.0)

ATT 20P 5A



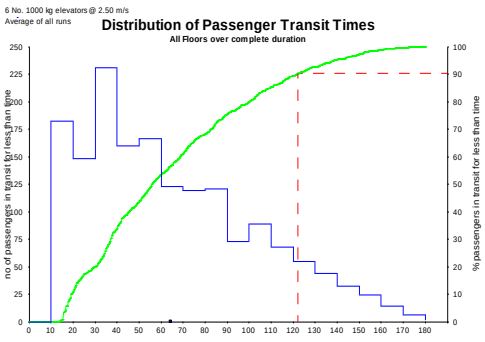
Average Transit Time (s) 69.4 (+0.0/-0.0)
Longest Transit Time (s) 189.9 (+0.0/-0.0)

AWT 20P 6A



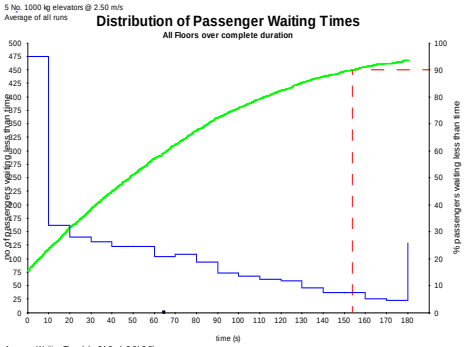
Average Waiting Time (s) 26.1 (+0.0/-0.0)
Longest Waiting Time (s) 290.6 (+0.0/-0.0)

ATT 20P 6A



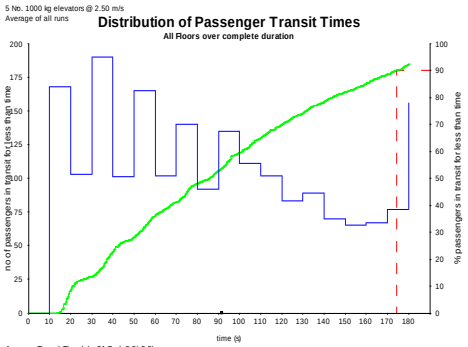
Average Transit Time (s) 64.5 (+0.0/-0.0)
Longest Transit Time (s) 191.4 (+0.0/-0.0)

AWT 24P 5A



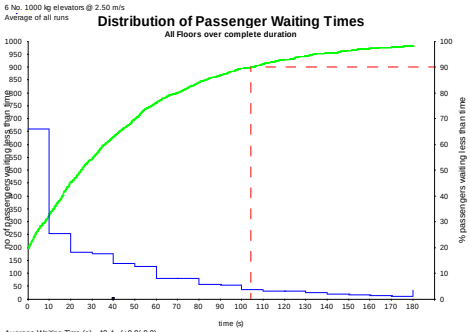
Average Waiting Time (s) 64.9 (+0.0/-0.0)
Longest Waiting Time (s) 552.2 (+0.0/-0.0)

ATT 24P 5A



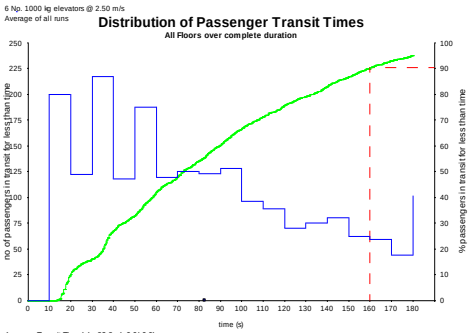
Average Transit Time (s) 91.7 (+0.0/-0.0)
Longest Transit Time (s) 229.7 (+0.0/-0.0)

AWT 24P 6A



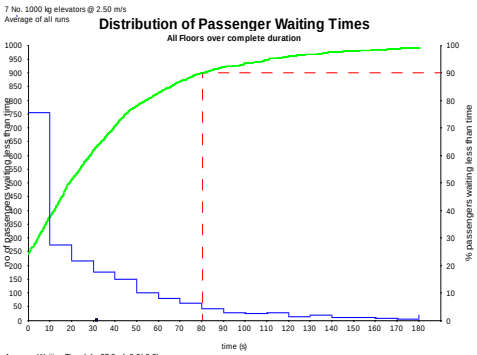
Average Waiting Time (s) 40.4 (+0.0/-0.0)
Longest Waiting Time (s) 387.4 (+0.0/-0.0)

ATT 24P 6A



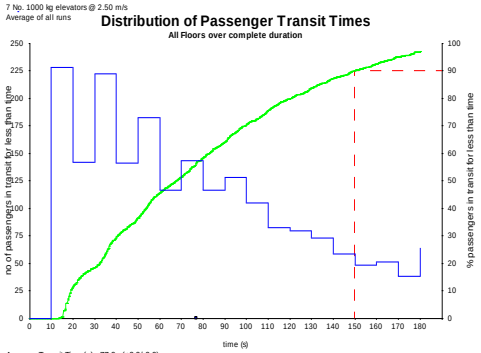
Average Transit Time (s) 82.8 (+0.0/-0.0)
Longest Transit Time (s) 232.2 (+0.0/-0.0)

AWT 24P 7A



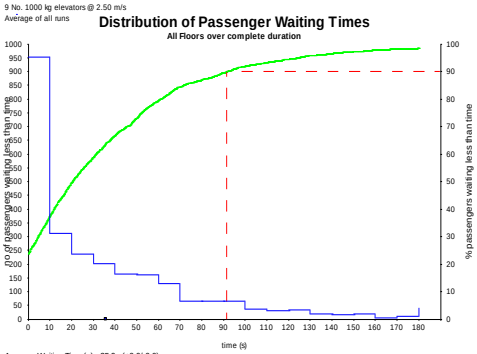
Average Waiting Time (s) 32.0 (+0.0/-0.0)
Longest Waiting Time (s) 315.6 (+0.0/-0.0)

ATT 24P 7A



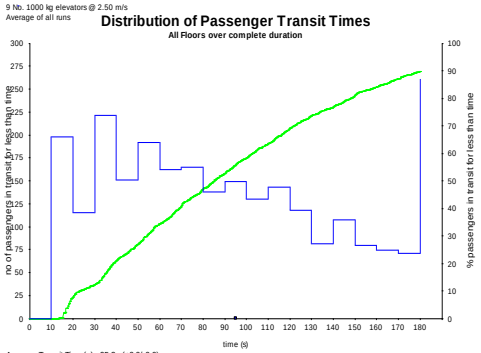
Average Transit Time (s) 77.0 (+0.0/-0.0)
Longest Transit Time (s) 227.9 (+0.0/-0.0)

AWT 30P 9A



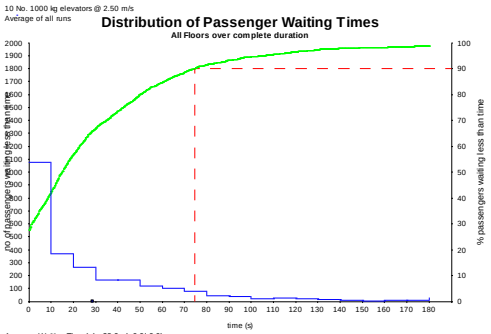
Average Waiting Time (s) 35.9 (+0.0/-0.0)
Longest Waiting Time (s) 385.4 (+0.0/-0.0)

ATT 30P 9A



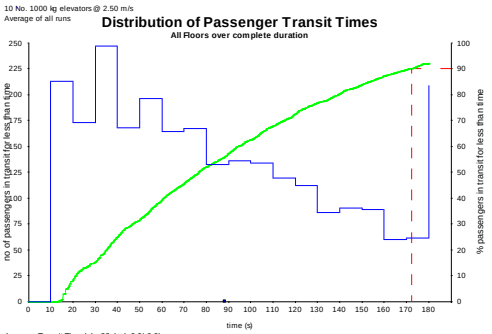
Average Transit Time (s) 95.2 (+0.0/-0.0)
Longest Transit Time (s) 301.0 (+0.0/-0.0)

AWT 30P 10A



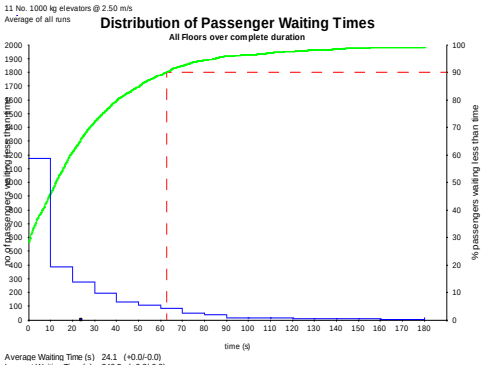
Average Waiting Time (s) 28.9 (+0.0/-0.0)
Longest Waiting Time (s) 303.1 (+0.0/-0.0)

ATT 30P 10A



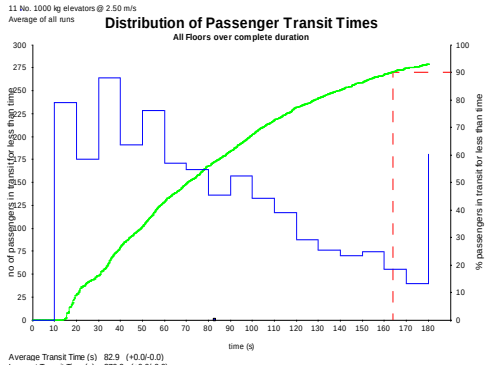
Average Transit Time (s) 88.4 (+0.0/-0.0)
Longest Transit Time (s) 278.0 (+0.0/-0.0)

AWT 30P 11A



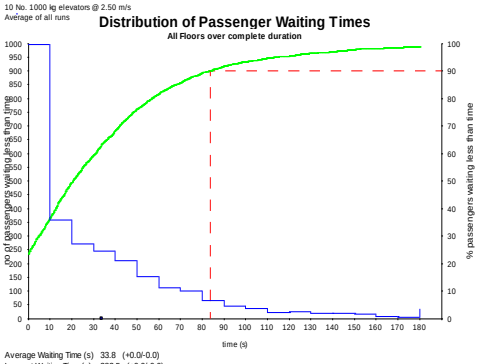
Average Waiting Time (s) 24.1 (+0.0/-0.0)
Longest Waiting Time (s) 346.5 (+0.0/-0.0)

ATT 30P 11A



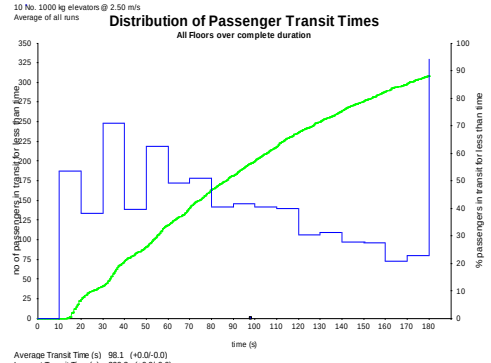
Average Transit Time (s) 82.9 (+0.0/-0.0)
Longest Transit Time (s) 270.9 (+0.0/-0.0)

AWT 32P 10A



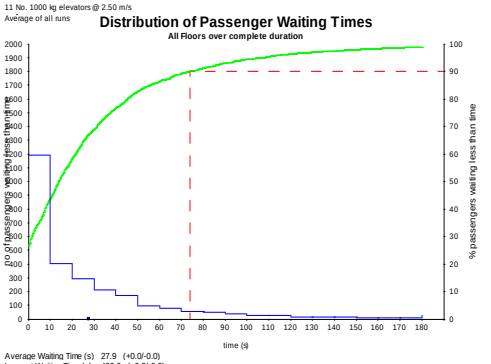
Average Waiting Time (s) 33.8 (+0.0/-0.0)
Longest Waiting Time (s) 383.5 (+0.0/-0.0)

ATT 32P 10A



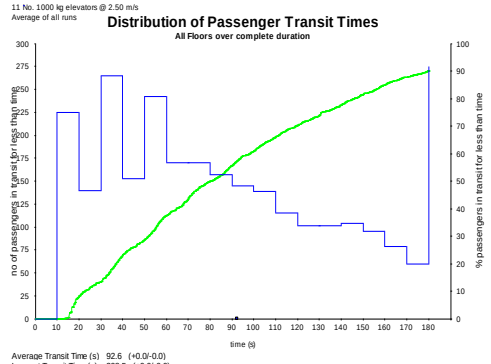
Average Transit Time (s) 98.1 (+0.0/-0.0)
Longest Transit Time (s) 290.3 (+0.0/-0.0)

AWT 32P 11A



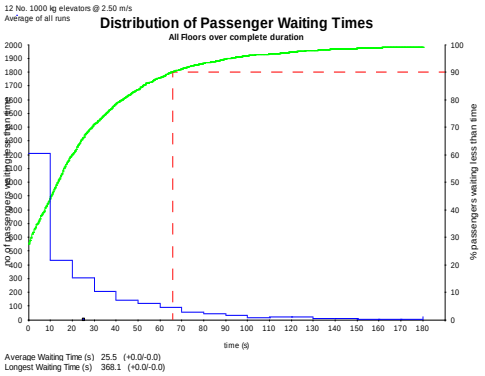
Average Waiting Time (s) 27.9 (+0.0/-0.0)
Longest Waiting Time (s) 406.6 (+0.0/-0.0)

ATT 32P 11A



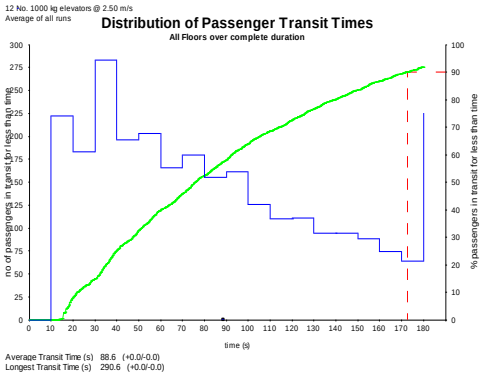
Average Transit Time (s) 92.6 (+0.0/-0.0)
Longest Transit Time (s) 308.5 (+0.0/-0.0)

AWT 32P 12A



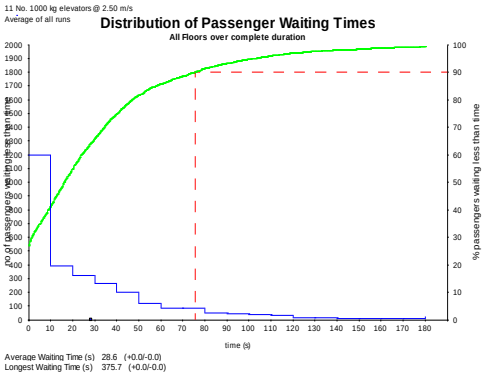
Average Waiting Time (s) 25.5 (+0.0/-0.0)
Longest Waiting Time (s) 368.1 (+0.0/-0.0)

ATT 32P 12A



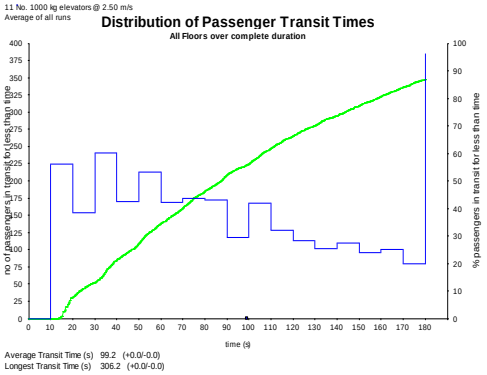
Average Transit Time (s) 88.6 (+0.0/-0.0)
Longest Transit Time (s) 290.6 (+0.0/-0.0)

AWT 34P 11A



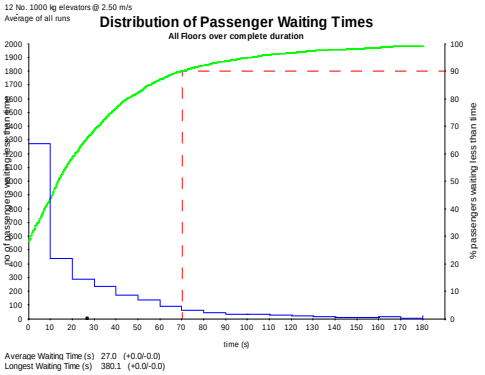
Average Waiting Time (s) 28.6 (+0.0/-0.0)
Longest Waiting Time (s) 375.7 (+0.0/-0.0)

ATT 34P 11A



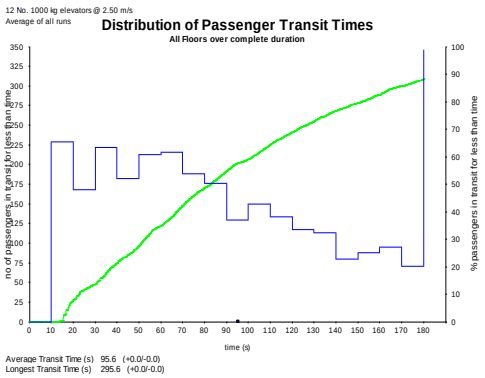
Average Transit Time (s) 99.2 (+0.0/-0.0)
Longest Transit Time (s) 306.2 (+0.0/-0.0)

AWT 34P 12A



Average Waiting Time (s) 27.0 (+0.0/-0.0)
Longest Waiting Time (s) 380.1 (+0.0/-0.0)

ATT 34P 12A



Average Transit Time (s) 95.6 (+0.0/-0.0)
Longest Transit Time (s) 295.6 (+0.0/-0.0)

ANEXO B: VARIABLES DE OBJETO

La implementación de los algoritmos empleados en este trabajo se ha realizado a través de programación C++ orientada a objetos. En este Anexo B se incluyen los objetos que se han empleado en dicha programación.

El primer objeto con el que se trabaja es "lift.h" en el cual se encuentran las especificaciones del ascensor. Entre dichas especificaciones se encuentran el estado de las puertas del ascensor, la capacidad total del ascensor, la capacidad actual, su velocidad y la dirección hacia la que se desplaza el ascensor. Además, como se trabaja con ascensores Double Deck, cada una de estas características dependen de si la cabina a la que van asociadas es la superior o la inferior.

El segundo objeto es "Dispatch.h", donde se encuentran las variables y características del edificio y de los pasajeros.

A continuación, se muestra el código de ambos objetos.

- **Objeto "lift.h"**

```
/*
-----------------------------------------------------------------
-------------
Filename   lift.h

Copyright  (c) Peters Research Ltd

SVN variables, Updated by SVN @ each commit Id, Revision
surrounded by '$'
if the SVN file properties SVN:Keywords are set
Revision
This keyword describes the last known revision in which this file
changed in the repository
$Id: lift.h 2184 2014-10-27 15:17:59Z Jim.Nickerson $
$Revision: 2184 $

-----------------------------------------------------------------
-------------
*/
//
#ifndef _INC_LIFT                    //check flag to avoid re-
definition of class
#define _INC_LIFT
//
#include "stdafx.h"
//
#if _MSC_VER > 1000
#pragma once
#endif
//
#include "arraysize.h"
#include "building.h"
#include "DestinationCall.h"
//
#define MAX_DESTINATION_CALL_LIFT  1000
```

```
//
//
class lift
{
      public:
      //
      enum _tag_LIFT_TYPE
      {     SINGLE_DECK = 0,      // IDSX_LIFT_TYPE_0
            DOUBLE_DECK = 1,      // IDSX_LIFT_TYPE_1
            TWO_CARS_PER_SHAFT = 2,    // IDSX_LIFT_TYPE_2
            TWO_D = 3,
            LAST_LIFT_TYPE             //  used  to  fill  string
arrays from string table
      };
      //
      enum _tag_CALL_CANCELLATION
      {
            ARRIVAL=0,
            SLOWDOWN=1
      };
      //
      enum _kW_STATES
      {                          // the energy consumed by the
drive in each of these load conditions
            DRIVE_OFF,
            DRIVE_ON,
            DRIVE_UP_0,
            DRIVE_UP_25,
            DRIVE_UP_50,
            DRIVE_UP_75,
            DRIVE_UP_100,
            DRIVE_DOWN_0,
            DRIVE_DOWN_25,
            DRIVE_DOWN_50,
            DRIVE_DOWN_75,
            DRIVE_DOWN_100,
            DRIVE_LAST_STATE
      };
      //
      enum _Axis
      {
            AXIS_X,
            AXIS_Y,
            AXIS_Z
      };
      //
      double m_Acceleration;                //rated          lift
acceleration (m/s/s) [elive static data] XMLS_1_ACCELERATION_MSS
      double m_AccelerationMultiply;  //not currently used, but
allows dispatcher to change the rated acceleration for a single
trip.  Set to 1.0 by default.
      int  m_ActualQuickestStopFloor;   //actual (as opposed to
ideal) quickest stop floor.  Not normally used in Elevate, but
when class used in real systems
```

```
      double m_ActualStoppingDistance; //actual (as opposed to
ideal) stopping distance.  Not normally used in Elevate, but when
class used in real systems
      int m_AlgorithmType;             //algorithm            type
CONVENTIONAL, DESTINATION or MIXED using global Elevate definition
      bool m_Available;                    //normally true, can
be off due to motor generator
      double m_Capacity;                   //nominal        lift
capacity (kg) [elive static data] XMLS_1_CAPACITY_KG
      int m_CarCall[MAX_FLOORS];       //car calls registered (1
registered, 0 not) [elive dynamic data] XMLS_1_REGISTERED_CALLS
      int m_CarCallCancellation;       //defines when lift class
cancels car calls
                                                 //options
ARRIVAL (default) or SLOWDOWN
                                                 //m_CarCall   is
set when value is 1.  If SLOWDOWN option selected then value is
changed to 2 on slowdown
      double m_CarCallDwellTime;       //not used
      double m_CarCallDwellTimePostPersonExit; //not used
      int m_CarCallUpperCar[MAX_FLOORS]; //car calls registered (1
registered, 0 not) [elive dynamic data]
      int m_CarService;                    //indicates   current
service state car is in (i.e. AUTOMATIC, etc) [elive dynamic data]
XMLS_1_CAR_SERVICE
      double m_CounterweightProportion; //used in conjunction with
m_VelocityMultiply to model ThyssenKrupp VMAX.
      double m_CurrentAcceleration;    //current      acceleration
(m/s/s)
      double m_CurrentArea;            //current area taken in car
by passengers (m²)
      double m_CurrentAreaUpperCar;    //current  area  taken  in
upper car by passengers (m²)
      double m_CurrentDistance;        //distance   travelled   on
current trip (m)
      int m_CurrentFloorNo;            //current   floor   number
(where 1 is lowest floor), NONE when travelling [elive dynamic
data] XMLS_1_FLOOR
      double m_CurrentJerk;            //current jerk (m/s/s/s)
      double m_CurrentLoad;            //current  car  load  (kg)
[elive dynamic data] XMLS_1_CURRENT_LOADKG
      double m_CurrentLoadUpperCar;    //current  car  load  upper
car (kg) [elive dynamic data]
      double m_CurrentPosition;        //current position (m above
reference)
      double m_CurrentTime;            //current   time   (s   past
reference)
      double m_CurrentVelocity;        //current velocity (m/s)
      bool m_DeleteDestinationCallAtDestination; //chooses if the
lift class deletes the call when the lift arrives at destination,
                                                 //or     changes
state to PENDING_RESET and saves the time the car arrives
      int DestinationByDispatcher;     //Normally the lift class
will decide what call to serve next based on the
                                                 //car  calls  and
allocated landing calls
```

```
                                                    //set       this
variable to 1 for the lift to travel to m_RequestedDestination
      int m_DestinationFloor;               //current destination
floor no  [elive dynamic data]
      double m_DestinationPosition;    //current   destination   (m
above reference)
      double m_DestinationTime;        //arrival time next planned
stop (s past reference)
      int m_Direction;                 //direction of travel (-1
down, 0 neither, 1 up) [elive dynamic data] XMLS_1_DIRECTION
      int m_DoorBeams;                 //flag   for   operation   of
door beams representing passenger
                                                    //transfer    (1
beams broken, 0 clear) [elive dynamic data]
      int m_DoorBeamsRear;             //flag   for   operation   of
door beams rear doors representing passenger
                                                    //transfer    (1
beams broken, 0 clear) [elive dynamic data]
      int m_DoorBeamsUpperCar;         //flag   for   operation   of
door beams upper car representing passenger
                                                    //transfer    (1
beams broken, 0 clear) [elive dynamic data]
      int m_DoorBeamsRearUpperCar;     //flag   for   operation   of
door beams upper car rear doors representing passenger
                                                    //transfer    (1
beams broken, 0 clear) [elive dynamic data]
      double m_DoorClose;                       //door   closing   time
(s) [elive static data]    XMLS_1_DOOR_CLOSE
      double m_DoorDwell1;             //door dwell time 1 (s)
                                                    //corresponds to
time doors will wait until
                                                    //closing     if
beam not broken
      bool m_DoorDwell1Expired;        //true  if  last  time  doors
closed, no one had entered/exited the lift
      bool m_DoorDwell1ExpiredRear;    //true  if  last  time  rear
doors closed, no one had entered/exited the lift
      bool m_DoorDwell1ExpiredRearUpperCar; //true  if  last  time
upper car rear doors closed, no one had entered/exited the lift
      bool m_DoorDwell1ExpiredUpperCar;          //true  if  last
time upper car doors closed, no one had entered/exited the lift
      double m_DoorDwell2;             //door dwell time 2 (s)
                                                    //corresponds to
time doors will wait until
                                                    //closing  after
beams have been broken/cleared
      int m_DoorDwellMode;             //not used
      bool m_DoorHoldOpen;             //set to true to hold doors
open, overrides m_DoorDwell2
      bool m_DoorHoldOpenRear;         //set to true to hold rear
doors open, overrides m_DoorDwell2
      double m_DoorOpen;                        //door  open  time  (s)
[elive static data]   XMLS_1_DOOR_OPEN
      double m_DoorPreOpen;            //door   pre-opening   (s)
[elive static data]   XMLS_1_DOOR_PRE_OPEN
```

```
     double m_DoorsStart;            //time    doors    started
opening/closing (s past reference)
     double m_DoorsStartRear;        //time  rear  doors  started
opening/closing (s past reference)
     double m_DoorsStartRearUpperCar;//time rear doors upper car
started opening/closing (s past reference)
     double m_DoorsStartUpperCar;    //time   doors   upper   car
started opening/closing (s past reference)
     int m_DoorStatus;                    //Door Status (1 fully
open,  2  closing,3  fully  closed,  4  opening,  5  nudging)  [elive
dynamic data] XMLS_1_DOOR_STATUS
     int m_DoorStatusCombined;       //used  by  dispatcher  to
determine one value combining front and rear door status
     int  m_DoorStatusCombinedUpperCar;//used  by  dispatcher  to
determine one value combining front and rear door status upper
car
     int m_DoorStatusRear;           //Door   dtatus   (1   fully
open,  2  closing,3  fully  closed,  4  opening,  5  nudging)  [elive
dynamic data] XMLS_1_DOOR_STATUS
     int m_DoorStatusRearUpperCar;   //Rear  door  status  upper
car (1 fully open, 2 closing,3 fully closed, 4 opening, 5 nudging)
[elive dynamic data] XMLS_1_DOOR_STATUS
     int m_DoorStatusUpperCar;       //Door status upper car (1
fully open, 2 closing,3 fully closed, 4 opening, 5 nudging) [elive
dynamic data] XMLS_1_DOOR_STATUS
     int  m_DownLandingCalls[MAX_FLOORS]; //down  landing  calls
allocated  to  lift  by  dispatcher  (1  registered,  0  not)  [elive
dynamic data] XMLS_1_DOWN_LANDING_CALL
     int   m_DownLandingCallsCombined[MAX_FLOORS];   //used   by
dispatcher to determine one value combining front and rear status
     int m_DownLandingCallsRear[MAX_FLOORS]; //rear down landing
calls allocated to lift by dispatcher[elive dynamic data]
     int m_DownLandingCallsRearUpperCar[MAX_FLOORS]; //rear down
landing  calls  upper  car  allocated  to  lift  by  dispatcher  [elive
dynamic data]
     int  m_DownLandingCallsUpperCar[MAX_FLOORS]; //down  landing
calls allocated to lift by dispatcher (1 registered, 0 not) [elive
dynamic data]
     bool m_EndTravelNoCall;              //defines what happens
if the call a car is traveling to is removed
                                                  //(e.g.       by
dispatcher).  False by default.  If true, then car will stop
                                                  //at  next  floor
(unless it has other calls to travel to).
     double m_FloorArea;                  //floor  area  of  car
(m²)
     double  m_FloorPositions[MAX_FLOORS]; //positions  of  floors
in building  [elive static data]
                                                  //(m       above
reference, can be negative)
                                                  //array  element
[0] not used, start with
                                                  //lowest   floor
at m_FloorPositions[1]
     int  m_FloorsServed[MAX_FLOORS];  //indicates  whether  lift
serves floor [elive static data] XMLS_1_FLOORS_SERVED
```

```
                                                    //used      when
lifts in a group do not serve all floors
                                                    //for   low/high
rise groups, run separate simulations
                                                    //may be changed
dynamically by dispatcher
                                                    //0 not served
                                                    //1   if   front
doors
                                                    //2    if    rear
doors
                                                    //3 if front and
rear doors
     bool  m_FrontDoors[MAX_FLOORS];   //true  if  front  doors  on
this landing
     int m_FrontDoorsOpenCount;      //counter              for
m_MaxDoorReOpen
     int     m_FrontDoorsOpenCountUpperCar;     //counter     for
m_MaxDoorReOpen, upper car
     bool m_FrontDoorsUpper[MAX_FLOORS]; //true if front doors on
this landing
     bool m_FrontLocks[MAX_FLOORS];  //true if car not allowed to
access for security reasons (even if served) [elive dynamic data]
     bool  m_FrontLocksUpper[MAX_FLOORS]; //ditto  for  upper  car
[elive dynamic data]
     bool m_FullByVision;            //true   if   vision   device
says  lift  if  full  -  for  use  in  real  systems  where  there  is  a
volumetric detection device
     int m_FutureCarCalls[MAX_FLOORS];//not used
     int m_Home;                                    //home
floor/default parking position [elive static data] XMLS_1_HOME
     double m_HomeDoorDwell1;        //alternative   door   dwell
time for home floor (s)
     double m_HomeDoorDwell2;        //alternative   door   dwell
time for home floor (s)
     double m_HomePeakLandingCallDwellTime; //not used
     double m_HorizontalPosition;    //horizontal   position   in
the shaft, e.g. [elive static data]
                                                    //1  -  shaft 1
                                                    //2  -  shaft 2
                                                    //1.5 - half way
between shaft 1 and 2
                                                    //user   of   lift
class must check that lifts do not crash!
     int m_Index;                           //index number of this
lift car [elive static data - allow string] XMLS_1_CARID
                                                    //note  that  the
Elevate lift array is 1 index.  l[0] is not used or initialized.
     double m_Jerk;                         //rated   lift   jerk
(m/s/s/s) [elive static data] XMLS_1_JERK_MSSS
     double m_JourneyStart;                 //time   lift   journey
started (s past reference)
     double m_kW[ DRIVE_LAST_STATE ]; //energy consumption, this
specifies the energy consumed while in each of these load states
     int m_LandingCallCancellation;  //defines  when  lift  class
cancel hall calls
```

```
                                                    //options
ARRIVAL (default) or SLOWDOWN
                                                    //Landing  calls
are set when value is 1.  If SLOWDOWN option selected then value
is changed to 2 on slowdown
      double m_LandingCallDwellTime;  //not used
      double m_LandingCallDwellTimePostCarCall; //not used
      double m_LevellingDelay;        //levelling delay (s)
      int m_MaxDoorReOpen;            //maximum  number  of  time
doors are allowed to re-open once car calls have been registered
                                                    //set  to  -1  for
unlimited
      double m_MaxVelocityMultiply;   //allows   dispatcher   to
change the rated acceleration for a single trip. Also use in
ThyssenKrupp VMAX to overspeed the lift.
      double m_MGRestartTime;              //time  it  takes  for
motor generator set to re-start (s)
      bool m_MGSet;                        //true  if  this  lift
has motor generator
      double m_MGShutDownAfterTime;   //time  after  which  motor
generator set will shut down (s)
      double m_MotorStartDelay;       //motor start up delay (s)
      int m_NoFloors;                      //no   of   floors   in
building [elive static data]    XMLS_1_NO_FLOORS
      int m_ParkCall[MAX_FLOORS];          //parking calls, like
landing  call,  but  placed  by  dispatcher[elive  dynamic  data]
XMLS_1_PARK_CALL
                                                    //lift  does  not
open doors on arrival
      int m_ParkOpenCall[MAX_FLOORS]; //as  parking  calls,  but
lift parks with doors open
      bool m_PeakMode;                //not used
      bool m_PersonTransferred;       //not used
      int m_PreDirection;                  //direction of travel
once we reach current destination [elive dynamic data]
      double m_QuickestStopPosition;  //next  possible  stop  lift
can make (m above reference)
      bool m_RearDoors[MAX_FLOORS];   //true if rear doors on this
landing
      int m_RearDoorsOpenCount;       //not used - for future
      int m_RearDoorsOpenCountUpperCar; //not used - for future
      bool m_RearDoorsUpper[MAX_FLOORS]; //true if rear doors on
this landing
      bool m_RearLocks[MAX_FLOORS];   //true if car not allowed to
access  rear  doors  for  security  reasons  (even  if  served)[elive
dynamic data]
      bool m_RearLocksUpper[MAX_FLOORS]; //true if car not allowed
to  access  rear  doors  upper  car  for  security  reasons  (even  if
served)[elive dynamic data]
      int m_ReasonForStopping;        //not used
      bool m_RequestAvailability;          //set to true to start
up MG set
      int m_RequestedDestination;          //Specific
destination requested by dispatcher
      bool m_SlowDown;                //true   if   the   lift   has
started slowing coming into a stop
```

```
      int m_StartFloor;                    //floor   no   current
journey started
      double m_StartPosition;              //position    current
journey started (m above reference)
      double m_TimeBeganStartUp;      //time we requested the MG
set to be turned on
      double m_TimeLastTrip;               //time last trip, used
to see if to turn off MG set
      double m_TimerT1;                    //time timer T1 began
(s past reference), -1 if not in use
      double m_TimerT1Rear;           //time timer T1 began for
rear doors (s past reference), -1 if not in use
      double m_TimerT1RearUpperCar;   //time timer T1 began for
rear doors upper car (s past reference), -1 if not in use
      double m_TimerT1UpperCar;       //time timer T1 began for
upper car (s past reference), -1 if not in use
      double m_TimerT2;                    //time timer T2 began
(s past reference), -1 if not in use
      double m_TimerT2Rear;           //time timer T2 began for
rear doors (s past reference), -1 if not in use
      double m_TimerT2RearUpperCar;   //time timer T2 began for
rear doors upper car (s past reference), -1 if not in use
      double m_TimerT2UpperCar;       //time timer T2 began for
upper car (s past reference), -1 if not in use
      int m_TravelStatus;                  //current      travel
status,   (1   traveling,   0   at   floor)   [elive   dynamic   data]
XMLS_1_TRAVEL_STATUS
      int m_Type;                                  //type   of   lift
(single, double deck, etc.), see _tag_LIFT_TYPE
      int    m_UpLandingCalls[MAX_FLOORS];//up   landing   calls
allocated to lift by dispatcher (1 registered, 0 not) [elive
dynamic data] XMLS_1_UP_LANDING_CALL
      int    m_UpLandingCallsCombined[MAX_FLOORS];    //used    by
dispatcher to determine one value combining front and rear status
      int  m_UpLandingCallsRear[MAX_FLOORS]; //rear  up  landing
calls  [elive dynamic data]
      int   m_UpLandingCallsRearUpperCar[MAX_FLOORS];  //rear   up
landing calls upper car [elive dynamic data]
      int  m_UpLandingCallsUpperCar[MAX_FLOORS]; //landing  calls
upper car [elive dynamic data]
      double m_Velocity;                   //rated lift velocity
(m/s) [elive static data]  XMLS_1_VELOCITY_MS
      double m_VelocityMultiply;      //allows   dispatcher   to
change the rated acceleration for a single trip.  Set to 1.0 by
default.
                                                 //Used        by
ThyssenKrupp VMAX feature.
      //
      //
      //new for multicar
      int m_HomeShaft;
      double m_VelocityXAxis;
      double m_AccelerationXAxis;
      double m_JerkXAxis;
      double m_LoadConnectionDelayXAxis;
      double m_UnLoadConnectionDelayXAxis;
```

```
      //
      int m_CurrentShaft;
      int m_NextShaft;
      int m_DirectionX;
      double m_CurrentPositionX;
      int m_DestinationShaft;
      double m_DestinationPositionX;
      double m_StartPositionX;
      double m_DestinationTimeX;
      double m_JourneyStartX;
      double m_QuickestStopPositionX;
      double m_CurrentDistanceX;
      //
      bool m_CarHold;
      //
      public:
      //constructors
      lift();
      //destructor
      ~lift() {};
      //member functions
      int ChangeJourney(int floor);   //change    journey,    new
destination, "floor"
                                                  //use       this
function    having    checked    this    is       possible    using
QuickestFloorStopFloor();
                                                  //returns   1   if
OK, -1 if not possible to change journey
      void   DestinationCallUpdate(int   floor,   DestinationCall
m_DestCalls[MAX_DESTINATION_CALLS],building  b);  //updates  the
status of the destination calls
      int FloorAt();                        //return  floor  no  if
not traveling
      int FloorNo(double position);   //returns   floor   no   at
position
      bool GetCarCall(int nFloor,int nDeck); //returns if there is
a car call registered for a given floor and deck
      int GetDoorStatus(int nSide, int nDeck); //returns the door
status of the given deck and side
      bool    GetFloorsForTripServed(int    nArrivalFloor,    int
nDestinationFloor, int nArrivalSide, int nDestinationSide);
                                                  //returns if the
lift can serve a trip
      bool GetLandingCall(int nDirection, int nFloor, int nDeck,
int nSide);
                                                  //returns   true
or false depending on if there is a landing call allocated to the
lift for the specified
                                                  //lift
direction, deck and side
      int GetNoDecks();                     //returns  the  number
of decks, currently limited but related routines being developped
                                                  //for   n   decks
for future use
      bool GetParkCall(int nFloor);   //returns  if  there  is  a
parking call for the given floor
```

```
      bool GetParkOpenCall(int nFloor); //returns if there is a
park open call for the given floor
      int HighestFloorServed();       //Highest floor served by
the lift
      int LowestFloorServed();        //Lowest floor served by
the lift
      bool NoCalls();                       //true if lift has no
calls at all (up, down, car, parking, etc.)
      bool    NoDestinationCalls(int    LiftNo,    DestinationCall
DestCalls[MAX_DESTINATION_CALLS]);
                                                  //true if there
are no outstanding destination calls to serve
      bool OpenDoorsNewCallAtLanding(building b); //checks to see
if new call registered while lift at landing and re-opens doors
if required
      bool OpenDoorsNewCallAtLandingRear(); //checks to see if new
call registered rear side while lift at landing and re-opens doors
if required
      bool OpenDoorsNewCallAtLandingUpperCar(); //checks to see if
new call upper car registered while lift at landing and re-opens
doors if required
      bool  OpenDoorsNewCallAtLandingUpperCarRear(); //checks to
see if new call registered upper car rear doors while lift at
landing and re-opens doors if required
      double QuickestStopDistance(int Axis);  //stopping distance
if started slowing down immediately
      int QuickestFloorStopFloor();   //next stop lift could make
(floor no)
      double QuickestStopPosition();  //next stop lift could make
(m above reference)
      double QuickestFloorStopTime(); //time of next stop lift
could make at floor (s after midnight day 1)
      double  QuickestFloorStopPosition(); //next stop at floor
lift could make (position)
      double QuickestStopTime();      //time of next stop lift
could make (s after midnight day 1)
      void RemoveLandingCall(int direction, int floor); //removes
landing call - called by class when lift arrives at landing.
                                                  //May   also   be
called by dispatcher when re-allocating call to another lift
      void Reset(building b, int index = 0);     //sets lift to
home position,cancels all calls, etc.
      void ResetDoorDwellTimers();    //Reset door dwell timers -
use when door opening complete, or if
                                                  //necessary   to
re-start the door time out process (e.g. new destination
                                                  //based control
system allocation while lift is at landing, to avoid passenger
missing lift.
      void  ResetDoorDwellTimersRear();//Reset door dwell timers
rear doors
      void  ResetDoorDwellTimersUpperCar();  //Reset door dwell
upper car timers
      void ResetDoorDwellTimersRearUpperCar(); //Reset door dwell
timers rear car upper lift
```

```
      void                          SetDestination(DestinationCall
m_DestCalls[MAX_DESTINATION_CALLS],    building    b);    //set
destination/direction travel
      int StartJourney(int floor, building b);   //start journey,
destination "floor" N.B. this sets the direction of the lift there
is no need to use SetDirection();
                                                 //returns  1  if
successful, -1 if failed e.g. because doors open
      double TimeSinceStartedJourney();//Calculates how long lift
has been travelling on its current trip
      void Update(double CurrentTime,int Index, DestinationCall
m_DestCalls[MAX_DESTINATION_CALLS], building b);
                                                 //this  function
updates the status of the lift (position, speed, door operation,
etc.)
      void UpdateDestination(double CurrentTime, DestinationCall
DestCalls[MAX_DESTINATION_CALLS]);
                                                 //check      for
calls allocated to lift and sets destination

      //
      private:
      void AdvancedCallCancellation();
      void EndTravelDeallocatedCalls();
      //
};
//
#endif
```

- **Objeto "Dispach.h"**

```
// Dispatch.h: interface for the CDispatchW class.
//
// $Id: Dispatch.h 2184 2014-10-27 15:17:59Z Jim.Nickerson $
// $Revision: 2184 $
// 12/01/2010 jimn add #if defined(DLL_USE_XML_PARAMETERS)
//////////////////////////////////////////////////////////////////
//////

#if
!defined(AFX_DISPATCH_H__163502DF_E3E1_4DA5_BB84_9BEF27C83AB0__I
NCLUDED_)
#define
AFX_DISPATCH_H__163502DF_E3E1_4DA5_BB84_9BEF27C83AB0__INCLUDED_
//
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
//
#include "DispatchBase.h"
//
#if defined(DLL_USE_XML_PARAMETERS)   //   if   used   defined   in
stdafx.h, added to return the Xml file name to Elevate
```

```
      #include "../Xml/CXml/Xml.h"    // required if using xml
parameters, make sure the path is correct for your implementation
      using namespace ElevateXml;          // required if using
xml parameters, this namespace is referenced
#endif
//
class CDispatchW : public CDispatchBase
{
public:
      CDispatchW();
      virtual ~CDispatchW();
      //
      // Public member functions
public:
      virtual void Update(double CurrentTime,
                  lift l[MAX_LIFTS],
                     double SimulationTimeStep,
                  building b,
                     int NoLifts,
                     CString message,
                     CString &mode,
                     CArray<person*, person*> &PersonArray,
                     int NoPassengers,
                     int NoTrafficPeriods,
                  double m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                     double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],
                double m_StartTime[MAX_TRAFFIC_PERIODS],
                     double m_EndTime[MAX_TRAFFIC_PERIODS],
                     int XMLmode,
                     DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                     CString Document,
                     bool DestinationButtons[MAX_FLOORS],
                     int UserSelectedMode);
      //
      virtual  void  Reset(building  b,  bool  ResetHistory,  int
version);
      virtual int GetNoOfAlgorithms();
      virtual CString GetAlgorithmName(int nAlgorithm);
      virtual int GetUpLandingCall(int floor, int side);
      virtual int GetDownLandingCall(int floor, int side);
      virtual void SetUpLandingCall(int floor, int val, int side);
      virtual  void  SetDownLandingCall(int  floor,  int  val,  int
side);
      virtual void SetAlgorithmNo(int val);
      virtual int GetAlgorithmNo();
      virtual int GetAlgorithmType();
      virtual CString GetDispatcherOptions();
      virtual  CString  GetAlgorithmModeName(int  nAlgorithm,  int
nMode);
      virtual int GetNoPeakModes(int nAlgorithm);
      virtual   void   SetDispatcherOptionsUserSelections(CString
DispatcherOptionsUserSelections);
      //
      // Data members
```

```
      //about the dispatcher algorithms
      int m_NoOfAlgorithms;                  //the    number    of
algorithms defined by the developer
                                                    //in   this
DLL (maximum MAX_USER_ALGORITHMS)
      //
      int m_DispatcherPeakMode;              //peak mode dispatcher
is currently using
                                                    //NORMAL,
UP_PEAK, DOWN_PEAK
      //
      CString m_AlgorithmName[MAX_USER_ALGORITHMS];
                                                    //the names
of the algorithms - these will be
                                                    //added  to
drop down box in Simulation Data
      //
      int m_NumberOfPeakModes[MAX_USER_ALGORITHMS];

      //determines how many peak modes, e.g. normal, up peak, down
peak, etc.
      //
      //the number of peak modes available with for algorithm
      CString
      m_PeakModeNames[MAX_PEAK_MODES][MAX_USER_ALGORITHMS];

                                                    //not used
      int m_Algorithm;                       //the  number  of  the
dispatcher algorithm selected by the user

      int m_AlgorithmType[MAX_USER_ALGORITHMS];
                                                    //the  type
of the algorithm
                                                    //0       -
conventional
                                                    //1       -
destinations registered at landings
      //
      //about the calls registered with the dispatcher
      int m_UpLandingCalls[MAX_FLOORS];     //up calls registered
with the dispatcher (1 registered, 0 not)
      int  m_UpLandingCallsRear[MAX_FLOORS];   //rear  up  landing
calls
      int m_DownLandingCalls[MAX_FLOORS];   //down           calls
registered with the dispatcher (1 registered, 0 not)
      int m_DownLandingCallsRear[MAX_FLOORS];     //rear     down
landing calls
      //
      //The  following  variables  provided  so  that  you  can  store
data between
      //calls to Update(double CurrentTime,lift l...).
      int m_User1[1000];
      double m_User2[1000];
      //
      bool m_RequestSimulationData;   //flag   to   indicate   that
simulation data is being requested
```

```
                                                    //by  dispatcher
which is not normally available to in real life
                                                    //set  this  flay
to true if you want to access data from the
                                                    //person   class
and the traffic arrival rate/destination matrix
      //
      CString m_DispatcherParameters;
      CString m_DispatcherOptions;
      CString m_DispatcherOptionsUserSelections;
      //
      CStdioFile m_File;
      //
      void Algorithm0(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
      void Algorithm1(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
      void Algorithm2(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
```

```
     void Algorithm3(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
     void Algorithm4(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
     void Algorithm5(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
     void Algorithm6(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
```

```
     void Algorithm7(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
     void Algorithm8(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
     void Algorithm9(double CurrentTime,lift l[MAX_LIFTS], double
SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
     void   Algorithm10(double   CurrentTime,lift   l[MAX_LIFTS],
double SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                  Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
```

```
      void   Algorithm11(double   CurrentTime,lift   l[MAX_LIFTS],
double SimulationTimeStep,building b,int NoLifts,
                              CString     message,CString     &mode,
CArray<person*,person*> &PersonArray, int NoPassengers,
                              int      NoTrafficPeriods,      double
m_u[MAX_TRAFFIC_PERIODS][MAX_FLOORS],
                              double
m_d[MAX_TRAFFIC_PERIODS][MAX_FLOORS][MAX_FLOORS],double
m_StartTime[MAX_TRAFFIC_PERIODS],
                              double
m_EndTime[MAX_TRAFFIC_PERIODS],DestinationCall
DestCalls[MAX_DESTINATION_CALLS],
                              CString                   Document,bool
DestinationButtons[MAX_FLOORS],int UserSelectedMode);
      //
      bool GetDispatcherOptionsValueBool(CString VariableName);
      double              GetDispatcherOptionsValueNumber(CString
VariableName);
      double GetDispatcherOptionsValueTime(CString VariableName);
      CString              GetDispatcherOptionsValueCombo(CString
VariableName);
      //
      void SetValue(CString VariableName, double VariableValue);
      void  SetValue(CString  VariableName,  int  index1,  double
VariableValue);
      void SetValue(CString VariableName, int index1, int index2,
double VariableValue);
      double GetValue(CString VariableName);
      double GetValue(CString VariableName, int index1);
      double  GetValue(CString  VariableName,  int  index1,  int
index2);
      CString NumberText(double val);
      CString NumberText(int val);
#if defined(DLL_USE_XML_PARAMETERS)
      void SetDispatcherDllOptions( CXmlNode DllOptions, CString
csAlgorithmName ); // if supports Xml Parameters
      void  GetOptionsFromXml(  CString  csAlgorithmName  );  //
extract the Option Value names and values from the Xml file for
this algorithm
#endif
protected:
};
//
#endif                                                              //
!defined(AFX_DISPATCH_H__163502DF_E3E1_4DA5_BB84_9BEF27C83AB0__I
NCLUDED_)
```