Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Asignación de llamadas para ascensores con arquitectura Double Deck mediante algoritmos de Búsqueda Tabú

Autor: Alejandro Vázquez Ledesma
Tutor: Pablo Fabio Cortés Achedad

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

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Asignación de llamadas para ascensores con arquitectura Double Deck mediante algoritmos de Búsqueda Tabú

Autor:
Alejandro Vázquez Ledesma

Tutor:
Pablo Fabio Cortés Achedad
Catedrático de Universidad

Dep. Organización y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Asignación de llamadas para ascensores con arquitectura Double Deck mediante algoritmos de Búsqueda Tabú

Autor: Alejandro Vázquez Ledesma

Tutor: Pablo Fabio Cortés Achedad

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

Agradecimientos

Al Dr. Pablo Cortés por la orientación, ayuda y atención prestada durante la realización de este trabajo y con su rápida respuesta ante cualquier tipo de duda o problema.

A mi pareja y familia por su paciencia, confianza, ánimo y apoyo constante a lo largo del camino ayudándome a conseguir los objetivos y metas marcadas durante estos años de estudio.

A los que ya no están pero que aportaron su granito de arena.

Muchas Gracias

Resumen

El precio del suelo está siendo incrementado cada día ya que hay menos lugares para construir, por lo que el transporte vertical está adquiriendo una gran relevancia en la actualidad. Además, cada día la altura de los edificios que se construyen alrededor del mundo es mayor para aprovechar el espacio disponible. Este trabajo pretende minimizar el tiempo medio de espera de un pasajero frente a las puertas de un ascensor con arquitectura *Double Deck*. Se implementa un algoritmo de *Búsqueda Tabú* en C++ mediante el software *Microsoft Visual Studio* y se realizan simulaciones con el software *Elevate*. Se comparan los resultados obtenidos con otros algoritmos de asignación de llamadas a ascensores.

Abstract

The price of the floor is raising every year because there are less places to build. So, vertical transport is becoming very important in the last years due to this fact. Moreover, each day there are higher buildings around the world. This project aims to minimize the average waiting time that passengers wait in front of the doors of an elevator with a *Double Deck* architecture. It will be implemented a *Tabu Search* algorithm in C++ with a software called *Microsoft Visual Studio* and it will be simulate with a software called *Elevate*. The results will be compared with other algorithms.

Índice

Índice de figuras

Índice de tablas

1. OBJETO

Este Trabajo Fin de Grado tiene como objetivo el estudio de la asignación de llamadas a ascensores con arquitectura *Double Deck* mediante un algoritmo basado en la Búsqueda Tabú implementado en C++ con el software *Microsoft Visual Studio* para ser estudiado mediante la simulación con el software *Elevate 8.*

Se trata de encontrar aquella asignación de llamadas que proporcione un menor tiempo de espera de los usuarios en la puerta del ascensor de modo eficiente debido a que deberemos encontrar dicha asignación en un tiempo considerado como aceptable. Además, se va a estudiar también el tiempo medio de tránsito de los pasajeros que suben al ascensor.

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

En primer lugar, en la segunda sección se introduce al mundo del ascensor a través de una pequeña reseña histórica y la tipología de ascensores que se encuentran en la actualidad. En esta misma sección se describe el ascensor con arquitectura *Double Deck*, objeto de estudio en este trabajo.

En la tercera sección se detallan los tipos de patrones de tráfico de pasajeros que existen en los edificios y se explica el sistema de evaluación de la calidad del servicio que se va a emplear para comprobar la bondad del algoritmo que se programa.

La sección cuarta de este trabajo recoge las restricciones del problema que se estudia así como una explicación del algoritmo de Búsqueda Tabú. Además, en esta misma sección se detalla el diagrama de flujo y los pasos que sigue el algoritmo de Búsqueda Tabú que se implementa para la asignación de llamadas.

La quinta sección trata sobre los dos softwares que se emplean en este trabajo: *Elevate* y *Microsoft Visual Studio.* Se explican las características de cada uno de ellos por separado y se detalla el proceso de instalación de la interfaz de desarrollador que permite la conexión entre ambos para la realización de las simulaciones.

La sección sexta es probablemente la más importante ya que en ella se encuentra el objetivo principal de este trabajo. Se realiza una experimentación consistente en la simulación del funcionamiento del algoritmo implementado en edificios que varían tanto el número de plantas como el número de ascensores. Se obtienen los tiempos medios de espera y los tiempos medios de tránsito para cada combinación del número de plantas del edificio con el

número de ascensores que el edificio tiene. Una vez se han obtenido los resultados anteriores se comparan con los obtenidos a través de otros algoritmos de asignación de llamadas a ascensores (algoritmo genético, algoritmo genético modificado y *double deck destination control*) para comprobar el funcionamiento correcto, o no, del algoritmo de Búsqueda Tabú. Se realiza un análisis para cada edificio en el que se compara el valor del tiempo medio obtenido mediante cada uno de los algoritmos. Como análisis complementario, por ser un objetivo secundario, se compara el valor del tiempo medio de tránsito que se obtiene de cada uno de los algoritmos.

En la sección séptima se elabora un análisis de las conclusiones que se extraen de la experimentación y estudio del problema. Se detalla si el algoritmo que se implementa cumple con los requisitos.

Se tiene un primer anexo en el cual 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 del algoritmo en C++.

2. INTRODUCCIÓN

En esta sección se trata la historia de los sistemas de ascensores así como los distintos tipos de ascensores que se pueden encontrar. Se pone especial hincapié en los ascensores con arquitectura *Double Deck*.

2.1. Reseña histórica

Hay indicios de ascensores rudimentarios cuya fuerza motriz era la fuerza humana, de animales o del 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.

En 1853, Elisha Graves Otis muestra un ascensor con "freno de emergencia" que evitaba la caída libre de la cabina en caso de rotura del cable que la sostenía. Elisha demostró la eficacia de su sistema de emergencia instalando su ascensor en el edificio Crystal Palace de New York. En la figura 1 se muestra la escena (Un poco de historia, 2016).

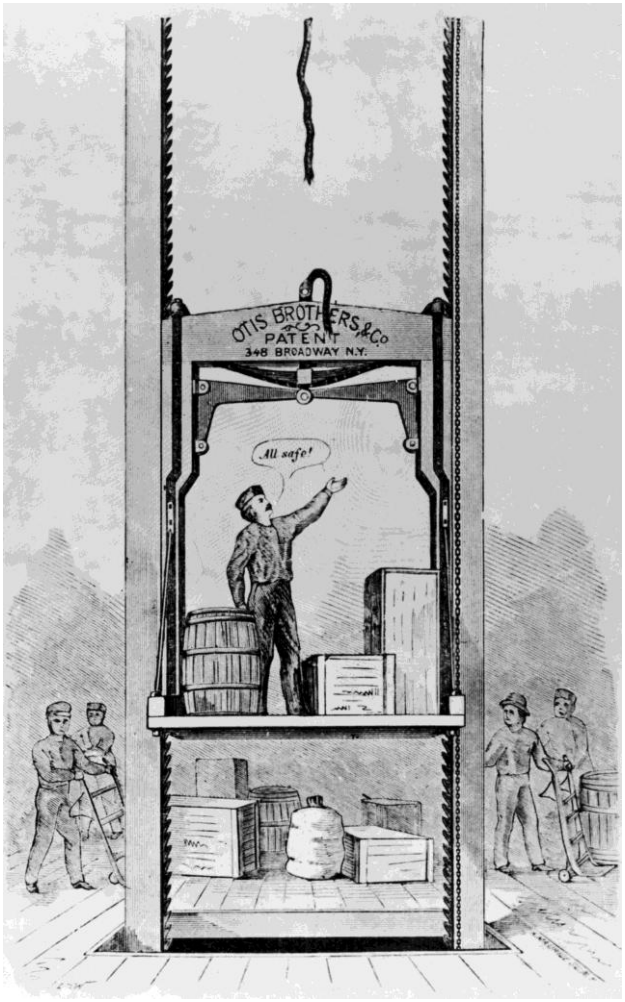


Figura 1: Elisha Graves muestra su "freno de emergencia"

Paralelamente, en Inglaterra, Frost y Stutt desarrollaron un ascensor con contrapeso que se accionaba mediante tracción cuya denominación fue "*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 en Broadway. Siendo esto un revulsivo importante ya que hasta dicha fecha, alojarse en niveles muy superiores de edificios era inaceptable ya que requería subir demasiadas escaleras con el 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.

Se muestra a continuación en la tabla 1 una comparativa de la carga y velocidad máxima de algunos ascensores a lo largo de la historia.

Tabla 1: Comparativa de carga y velocidad máxima en el tiempo

Año	Carga Máxima (kg)	Velocidad Máxima (m/min)
1857	450	12
1867	-	150
1889	675	30
1922	-	420

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 se 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.

Cabe destacar varias empresas que a lo largo de la historia han hecho evolucionar el mundo del ascensor, tales como *Mitsubishi Electrics*, *Otis* o *Schindler*. Aunque también encontramos otra serie de empresas de más reciente creación que favorecen la competitividad del sector en pos de favorecer la evolución del ascensor como *ThyssenKrupp*, *MP* u *Orona*. En las siguientes figuras se muestran los logotipos de las distintas empresas nombradas.

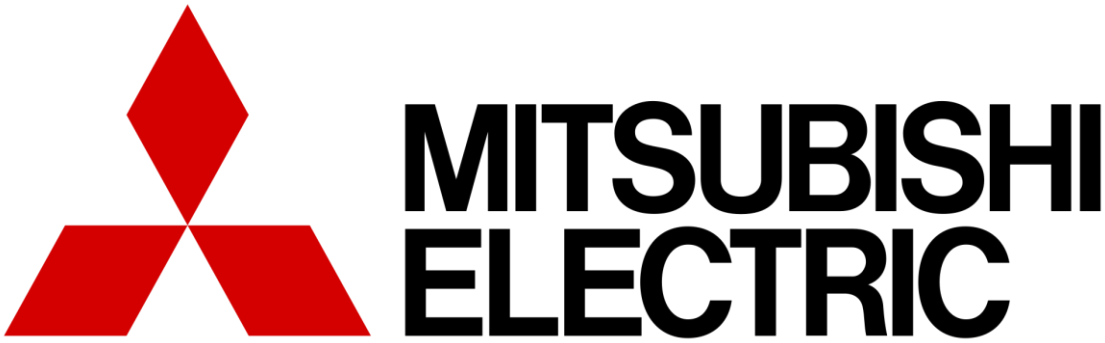


Figura 2: Logo Mitsubishi Electric

Figura 3: Logo Otis

Figura 4: Logo Schindler

Figura 5: Logo Thyssenkrupp

Figura 6: Logo MP

Figura 7: Logo Orona

2.2. Tipos de Ascensores

En este apartado se tratan los distintos tipos de ascensores que se encuentra en la actualidad. Se clasifican según su aspecto de diseño y según su aspecto técnico (Tipos de ascensores para las edificaciones, 2013).

2.2.1. Según su aspecto técnico

- **Ascensores autoportantes** (sin sala de máquinas):

  Este tipo de ascensor es muy común en lugares como viviendas unifamiliares, salones de fiesta, cines o cualquier otra instalación que deba prescindir de una sala de máquinas. La máquina de tracción en su conjunto completo se coloca en forma estructural dentro del mismo pasadizo y en su parte superior (cielo de la caja). Se muestra en la figura 8 este tipo de elevador.

Figura 8: Ascensor autoportante

- **Ascensores hidráulicos:**

  Son elevadores que se instalan en recorridos cortos, entre cuatro y cinco paradas. Su modo de funcionamiento no es a través de máquina de tracción sino que depende de una central oleodinámica que tiene en su interior una bomba sumergida en aceite para controlar el ascenso de la cabina. La sala de máquinas debe estar perfectamente dimensionada y habilitada para la instalación de la central y el control de maniobras.

  La instalación en el pasadizo requiere de un pistón colocado en el centro de una arcatina, que es la que eleva el pistón por medio de una polea de la cual los cables de tracción van unidos desde el punto fijo de la arcatina al punto fijo de la arcata. En la figura 9 se puede observar este tipo de elevador.

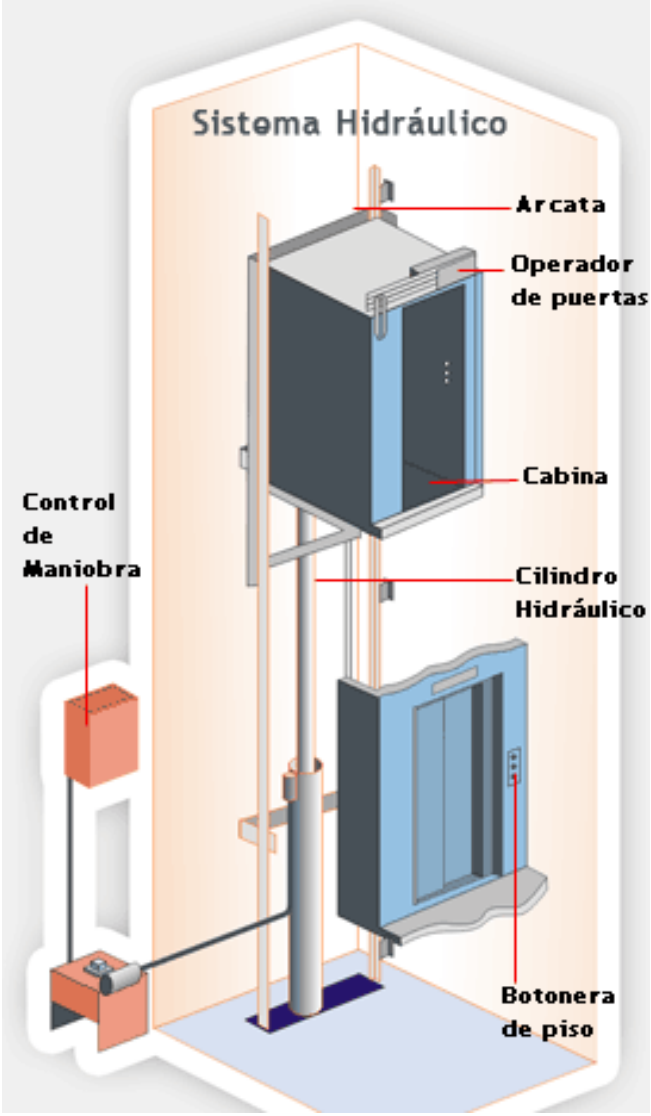


Figura 9: Ascensor hidráulico

- **Ascensores electromecánicos:**

  Son los ascensores más empleados en edificios de viviendas multifamiliares. Este tipo de elevadores, a diferencia del ascensor hidráulico, sí requiere de una máquina de tracción en la sala de máquinas que debe estar ubicada debajo o arriba de la instalación.

  La tracción se realiza con motor eléctrico, polea y máquina reductora. De la polea cuelga el cable de tracción que es arrastrado por fricción en su giro, mientras que la cabina es guiada en su trayecto por rieles.

  Una de las ventajas con las que cuenta este tipo de ascensor es que en caso de persona encerrada, se podrá intervenir manualmente accionando una manivela de freno, cortando previamente el suministro eléctrico. Se muestra en la figura 10 este tipo de ascensor (Ingesor, 2016).

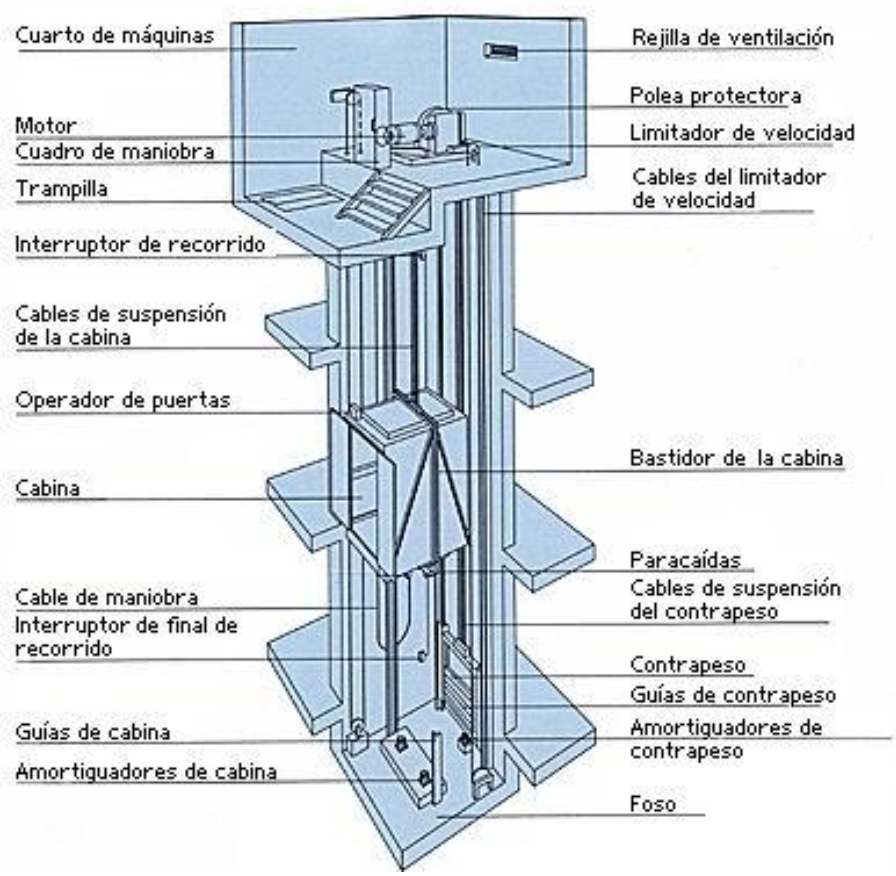


Figura 10: Ascensor electromecánico

2.2.2. Según su aspecto de diseño

- **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. Se muestra un ejemplo de ascensor panorámico en la figura 11 (Grupo de ascensores Enor, 2016).

Figura 11: Ascensor panorámico

- **Ascensores multicabina:**

  Los sistemas de ascensores multicabina son una apuesta de futuro cuyo objetivo principal es que el usuario que llama a un ascensor sea atendido en un tiempo muy breve, gracias a que en cada hueco de ascensor habrás 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 del sistema MULTI es crear un bucle continuo en el que las cabinas suben por un conducto y bajan por el contrario (La nueva era de los ascensores revoluciona la construcción de mediana y gran altura, 2016).

Se muestra a continuación, en la figura 12, el sistema de ascensores multicabina.

Figura 12: Sistema de ascensores multicabina

- **Ascensores Double Deck:**

  Este tipo de ascensores tienen la peculiaridad de tener dos cabinas solidarias, una sobre la otra, de modo que ambas recorren el hueco del ascensor conjuntamente. Gracias a este novedoso sistema, se consigue desplazar el doble de pasajeros por cada viaje realizado de modo que el tiempo de espera de cada usuario para recibir un ascensor es menor. No solo se consigue aumentar el confort del pasajero gracias a esperar menor tiempo sino que además, al realizarse menor número de viajes, el consumo energético es inferior. Se puede observar en las figuras 13 y 14 ascensores con este tipo de arquitectura.

Figura 13: Ascensor *Double Deck*

Figura 14: Sistema de ascensores *Double Deck*

Debido a las ventajosas características de este tipo de ascensores, se pueden encontrar en gran cantidad de edificios. A continuación se muestra una serie de edificios que cuentan con sistemas de ascensores con arquitectura *Double Deck*.

Torre Picasso

Esta torre se encuentra junto al Paseo de la Castellana de Madrid. Consta de 51 plantas repartidas en sus 157 metros de altura para labores de oficina. Cuenta además con sótano y helipuerto. Se muestra en la figura 15 (Torre Picasso, 2016).

Figura 15: Torre Picasso (Madrid)

The Shard

Se encuentra en la zona de Southwark de Londres. Consta de 72 plantas utilizables (de las 87 plantas que dispone) repartidas en los 309,6 metros de altura que lo hacen el edificio más alto de la Unión Europea y cuyo objetivo principal es el uso como oficinas además de disponer de restaurantes, apartamentos y un hotel. Se observa en la figura 16 (The Shard, 2016).

Figura 16: The Shard (izquierda) junto a Tower Bridge (Londres)

Citigroup Center

Se encuentra en la ciudad de New York y cuenta con 279 metros repartidos en sus 59 plantas. Actualmente alberga la sede central de *Citibank*. Se muestra en la figura 17.

Figura 17: Citigroup Center (New York)

Torre Burj Khalifa

Se encuentra en Dubai. Sus 828 metros de altura repartidos en 163 plantas hacen que este rascacielos sea el más alto del mundo desde el año 2010 tras arrebatar dicho puesto a la Torre *Taipei 101* de la República de China. La torre *Burj Khalifa* es usada como hotel, residencias de lujo, gimnasio y suites corporativas. Además, en la planta 124 se encuentra un mirador denominado "*At the top*". Cuenta con una serie de ascensores de arquitectura *Double Deck*, instalados por *Otis*, que transportan pasajeros a una velocidad de diez metros por segundo consiguiendo ser los ascensores de doble cabina más veloces del mundo. Se muestra en la figura 18 (Otis en la torre Burj Khalifa, 2016).

Figura 18: Torre Burj Khalifa (Dubai)

3. SISTEMAS DE TRANSPORTE VERTICAL

En esta sección se tratan los aspectos técnicos de los sistemas de transporte vertical. Se trata de definir los parámetros que se encuentran asociados a los procesos de transporte vertical.

La afluencia de personas que circulan en un edificio varía según sea dicho edificio residencial u oficinas. Debido a que es en los edificios de oficinas en los que mayor tráfico de personas se encuentra, siendo además el lugar en el que menor tiempo se debe perder en transportes de personas y/o mercancías ya que no aportan valor al trabajo que se realiza, es el tipo de edificio que se estudia a lo largo de este trabajo.

La gestión del tráfico vertical se mide en función de 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*).

3.1. Patrones de tráfico

Se observa en la figura 19, el patrón de tráfico que sigue un edificio de oficinas según la hora del día en la que se encuentre. En el eje de abscisas se observa la hora y en el eje de ordenadas la cantidad de tráfico ascendente y descendente en términos de porcentaje total de la población. Los distintos colores muestran los diversos patrones que se detectan (Cortés P. et al., 2009).

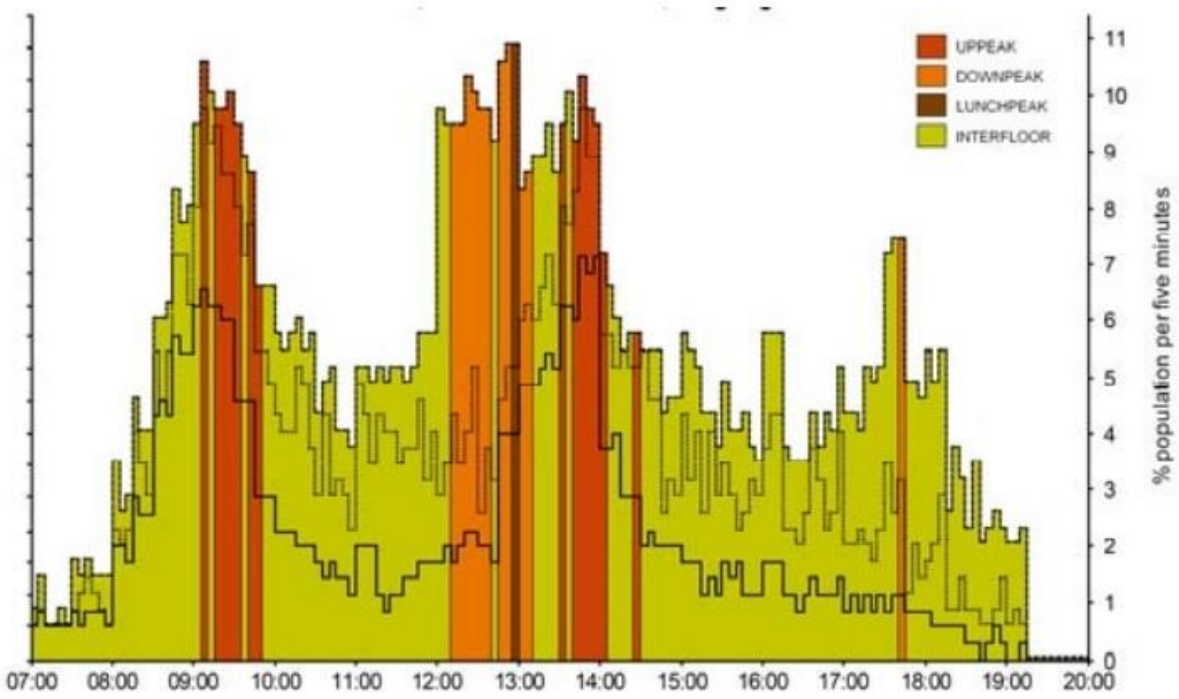


Figura 19: Patrón de tráfico Siikonen

La teoría clásica (Barney, 2003) establece unos patrones de tráfico según la hora del día, siendo estos:

- **Uppeak:**

  Se refiere al tramo horario coincidente con el acceso al edificio, y por tanto se producen numerosas llamadas desde la planta baja del edificio con carácter ascendente con el objeto de llegar al puesto de trabajo.

- **Interfloor:**

  Se refiere a los tramos horarios en los cuales existen viajes entre plantas intermedias del edificio. Se encuentran en este patrón dos tipos diferentes de patrones, aunque ambas se agrupan como *Interfloor*.

i. **Balanced Interfloor Traffic**, qué es el tráfico habitual de media mañana o media tarde.

ii. **Unbalanced Interfloor Traffic**, qué es el tráfico no habitual de media mañana o media tarde y que suele ser debido a alguna reunión de personal.

- **Lunchpeak:**

  Se refiere al tramo horario en el cual los empleados del edificio lo abandonan 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.

Se suele considerar el periodo *Uppeak* como el más severo en cuanto al número de llamadas a ascensores debido a que la hora de entrada al puesto de trabajo genera una gran masa de población que accede al edificio. Sin embargo, debido a la flexibilización de los horarios de trabajo y a la prohibición de fumar dentro del mismo, el tráfico *Interfloor* adquiere un gran protagonismo y es estudiado en la actualidad con el objeto de ofrecer una mejor calidad de servicio.

3.2. Evaluación de la calidad del servicio

A pesar de que el tráfico *Interfloor* ofrece una gran importancia, se puede considerar que si se consigue satisfacer la demanda durante el *Uppeak* se puede satisfacer la demanda de cualquier otro patrón. Se toman las siguientes tablas, 2 y 3, como referencia de satisfacción.

Tabla 2: Nivel de 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 3: Nivel de servicio en función del 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

En este trabajo, la calidad del servicio se mide a través del tiempo medio de espera que el pasajero se encuentra ante la puerta del ascensor desde que el ascensor ha sido llamado hasta que el ascensor comienza a abrir sus puertas. Este tiempo se denomina mediante las siglas **AWT**, cuyo significado es *Average Waiting Time.*

La problemática surge al contabilizar el tiempo medio de espera de los pasajeros sucesivos que llegan ante la puerta del ascensor tras haberse realizado ya la llamada del mismo. Una de las opciones para solventar dicho problema es contabilizar manualmente el tiempo mediante un cronómetro, pero es una opción poco factible. En todo caso, esto no es inconveniente para la realización del trabajo ya que se emplea el software *Elevate* que tiene en cuenta este tiempo.

Otra variable importante a analizar es el tiempo medio de viaje, denominado **ATT** (*Average Trip Time*), que es el tiempo que el pasajero emplea desde que las puertas del ascensor se abren ante él hasta que llega a su planta destino.

4. SISTEMAS DE ASIGNACIÓN DE LLAMADAS

Una vez conocidos los sistemas de ascensores y patrones de tráfico, se pasa a detallar en esta sección el sistema de asignación de llamadas que se emplea para la resolución del problema.

4.1. El problema

El problema que se va a tratar es el consistente en un edificio de oficinas, en el cual se encuentra una serie de ascensores con arquitectura *Double Deck*, y unos pasajeros que realizan llamadas tanto de subida como de bajada desde las distintas plantas del edificio. Cabe destacar que el número de llamadas a ascensores que realizan los pasajeros dependen del tramo horario como se observó en los distintos patrones de tráfico de la sección anterior.

El objetivo principal que se debe satisfacer es el de asignar las distintas llamadas a ascensores de modo que el tiempo medio de espera de los pasajeros (AWT) sea el menor posible. Además, se tiene un objetivo secundario, muy ligado al principal, que es que el tiempo medio de tránsito (ATT) también sea el menor posible.

Este problema cuenta con una serie de restricciones, tanto generales como particulares que se pasa a detallar a continuación (Cortés et al., 2003).

i. Restricciones generales

Se dividen las restricciones generales en explícitas e implícitas.

- **Restricciones generales explícitas:**
  - → Una cabina solo puede servir una llamada en cada instante.
  - → El número máximo de pasajeros que se pueden 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 esa 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

Las restricciones particulares dependen de las condiciones específicas del edificio. Se detallan algunas restricciones particulares que se encuentran habitualmente:

- → 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.

4.2. Optimización de recursos

La optimización se ha convertido en los últimos tiempos en una prioridad debido a que cada vez el coste de los recursos es mayor, además de que pueden llegar a ser escasos, por lo que es necesario conseguir una eficiencia capaz de lograr los mejores resultados con una cantidad de recursos mínima.

Es por ello por lo que las técnicas de optimización son necesarias y van evolucionando en el tiempo. Se trata de conseguir la mejor solución en un tiempo considerado como factible, esto es, si encontrar la mejor solución requiere de emplear varios años en su búsqueda no servirá este método.

Ya que encontrar la solución óptima de un problema puede llegar a ser muy difícil e incluso imposible, se emplean métodos de resolución que nos aproximan al valor óptimo (o incluso lo llegan a obtener a veces). A este tipo de técnicas se les conoce como metaheurísticas.

Se encuentran metaheurísticas que se basan en la observación de la naturaleza para su semejanza posterior con el problema a optimizar, entre ellas se encuentran el "Algoritmo genético" o el "Algoritmo de colonias de hormigas". Otras metaheurísticas se basan en vecindades para explorar las distintas posibles soluciones, entre las que se encuentran el "Algoritmo de recocido simulado" o el "Algoritmo de búsqueda Tabú".

4.3. Algoritmo de Búsqueda Tabú

Para la resolución del problema de asignación de llamadas a ascensores se emplea un algoritmo basado en la Búsqueda Tabú.

Proveniente 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 (Glover F., 1989).

La búsqueda tabú se basa en tres puntos principales. En primer lugar, se usa memoria adaptativa diseñada para permitir evaluar la información de búsqueda histórica. En segundo lugar, se emplea un mecanismo de memoria que restringe y libera el proceso de búsqueda. Y como último punto, el uso de memorias en distintos lapsos de tiempo (corto, medio y largo) ayuda a guardar aquellas características que logran una buena solución, olvidando otras y permitiendo al método la diversificación.

Es importante la distinción entre memoria a corto y largo plazo. La memoria a corto plazo almacena los atributos de solución que han sido cambiados en el pasado reciente por lo que también recibe el nombre de memoria basada en hechos recientes. Estos atributos

seleccionados recientemente se designan como "tabú-activos" y las soluciones que los contienen (o combinaciones particulares de estos atributos) pasan a formar parte de la lista tabú para evitar que pertenezcan a la futura vecindad y puedan llegar a ser revisitadas. El tiempo que un atributo permanece como "tabú-activo" se denomina tenencia tabú y se mide en número de iteraciones, es variable según los atributos y las combinaciones de estos. En cambio, la memoria a largo plazo emplea memoria basada en frecuencia de cambio de atributos con la cual se pretende potenciar la búsqueda de soluciones, aunque a veces no es necesario su empleo.

No siempre el que una solución se encuentre en la lista tabú indica que dicha solución no es válida, ya que el método define unos niveles de aspiración con los cuales se evalúan dichas soluciones en nuestra función objetivo y si los resultados obtenidos son considerados como admisibles, por dar mejor solución que la anterior generada, estos son rescatados de la lista tabú.

El algoritmo de búsqueda tabú emplea estrategias de intensificación y/o diversificación. Siendo la estrategia de intensificación aquella en la que la evolución de soluciones es favorecida por combinaciones de movimiento y características de soluciones que históricamente haya sido buena. Sin embargo, la estrategia de diversificación buscará explorar nuevas regiones a través del cambio de las reglas de elección de atributos. Se busca el empleo de atributos que no hayan sido usados frecuentemente.

4.4. Ventajas e inconvenientes de la Búsqueda Tabú

Uno de los inconvenientes de este método, y que es compartido con otras metaheurísticas, es la incapacidad para conocer la calidad de la solución obtenida debido al hecho de que se emplea esta técnica por no poder llegar al óptimo por lo que no podemos hacer una comparación del óptimo con la solución obtenida.

Debido a que la búsqueda tabú no es una técnica "rígida", esto es, debe confeccionarse para cada problema concreto, la forma del algoritmo depende del problema que se vaya a resolver. Esto puede llegar a ser un inconveniente debido al hecho de que se necesita conocer muy bien tanto la metódica de la técnica de Búsqueda Tabú, como del propio problema a optimizar, para poder adaptarlo y conseguir resolverlo. El solucionador del problema debe estar familiarizado con este, conociendo su naturaleza y la forma de las soluciones factibles para poder elaborar la configuración del vecindario, criterio de aspiración y lista tabú.

Una de las grandes ventajas de la Búsqueda Tabú es su capacidad de escapar de la optimalidad local gracias a vetar el acceso a ciertas soluciones pertenecientes a la lista tabú. Es por esto por lo que este algoritmo de búsqueda consigue explorar vecindades y logra evitar el carácter cíclico que tienen otros algoritmos. Es el empleo de una memoria basada en lista tabú la principal característica distintiva del resto de metaheurísticas. Además, la capacidad del "olvido estratégico" de soluciones tabú mediante los criterios de aspiración hacen que este método consiga explorar una gran cantidad de posibles buenas soluciones.

El método de Búsqueda Tabú permite ser empleado directamente a expresiones verbales o simbólicas, no teniendo la necesidad de transformación a expresiones matemáticas que podrían generar conflicto en su posible dificultad de expresión.

4.5. Búsqueda Tabú para la asignación de llamadas

Como se ha comentado, para la resolución de este problema se va a emplear un algoritmo de Búsqueda Tabú, cuyo diagrama de flujo se muestra a continuación.

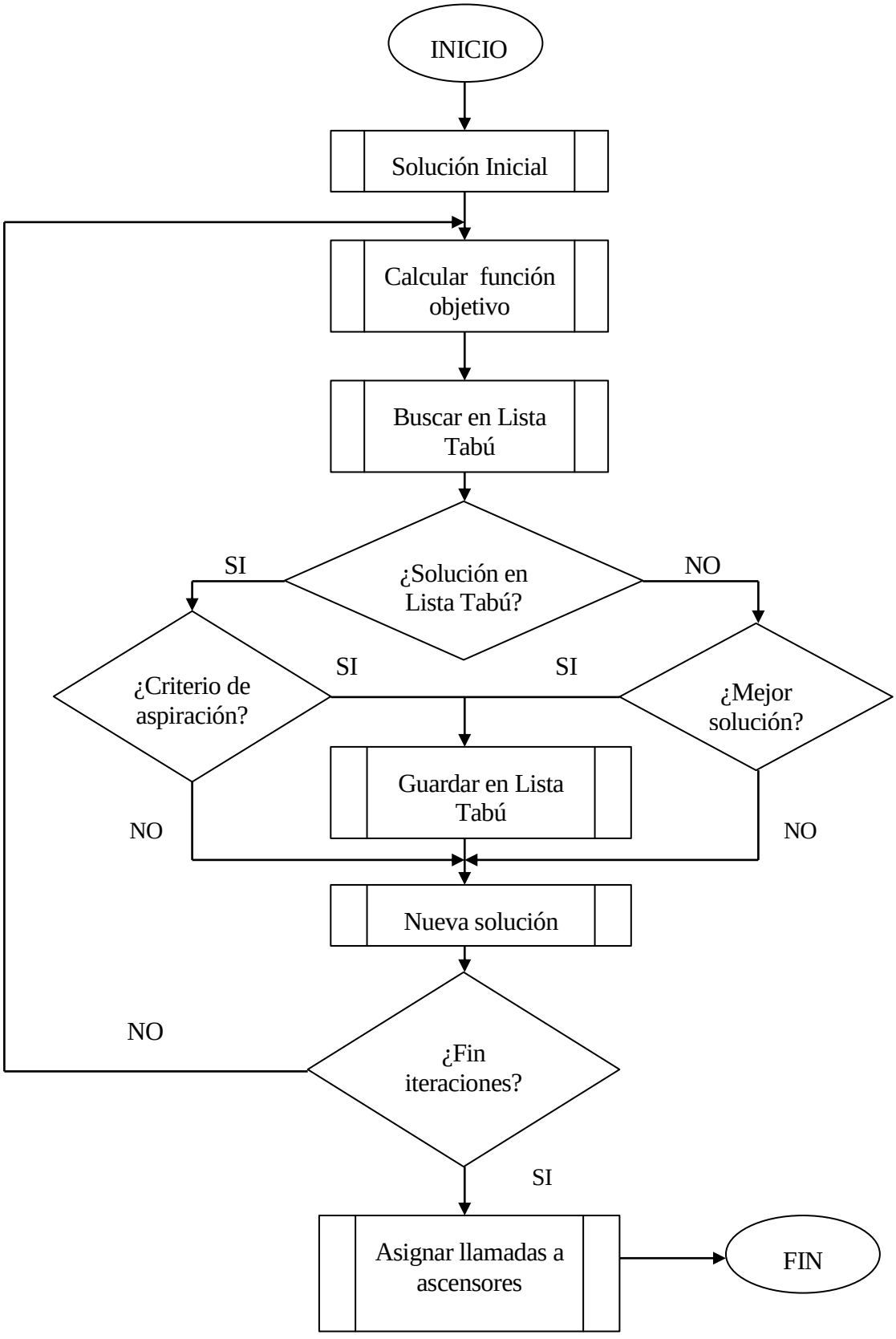


Figura 20: Diagrama de flujo principal

El algoritmo que se implementa en C++ recibe del software *Elevate* las características del edificio y de los ascensores así como del patrón de tráfico del edificio que se va a emplear. Una vez se procesa la información mediante el algoritmo programado se devuelven a *Elevate* los ascensores que deben ir a cada una de las plantas que reciben llamadas. Se detalla a continuación cada uno de los pasos que se llevan a cabo en el algoritmo de Búsqueda Tabú programado.

4.5.1. Solución inicial

Para iniciar el algoritmo, se busca una solución inicial de asignación de llamadas a ascensores en la que se rellena el vector que contiene los ascensores (1 si el ascensor "i" va a ser empleado y 0 si el ascensor "i" no será empleado) de forma totalmente aleatoria. Además, estos ascensores son asignados a plantas con llamadas, también de manera aleatoria.

4.5.2. Calcular la función objetivo

Se trata del cálculo de la suma de distancias de cada ascensor, de los ascensores que van a ser empleados, a las plantas de las cuales se reciben llamadas. De este modo, se consigue calcular la función objetivo para cada solución de asignación de llamadas a ascensores con el objetivo de poder comparar para buscar la menor distancia, y por tanto, la mejor asignación de llamadas a ascensores que minimiza el tiempo de espera de los pasajeros frente a los ascensores.

Se deben tener en cuenta en este apartado ciertas restricciones del modelo para un cálculo correcto de las distancias ya que se pueden dar varias casuísticas que se detallan a continuación:

1. Tal y como se detalla en el apartado 4.1 los ascensores que tienen una ruta definida (ascendente o descendente) no podrán variarla. Debido a esto, si existe una llamada ascendente y el ascensor se encuentra descendiendo, el cálculo de la distancia es la suma de la distancia que resta hasta llegar a la planta baja más la distancia desde esa planta baja hasta la planta que se ha recibido la llamada. De igual modo se procede si la llamada es descendente y el ascensor se encuentra ascendiendo, en cuyo caso la distancia se calcula como la distancia que resta hasta llegar a la última planta más la distancia hasta la planta de la que se recibe la llamada.

2. Si se tiene una llamada descendente de una planta y el ascensor correspondiente se encuentra descendiendo, se comprueba si es posible que pare para acoger dicha llamada. En caso de que el ascensor no pueda parar, el cálculo de la distancia es la suma de la distancia que resta hasta llegar a la planta baja más la distancia hasta llegar a la cima del edificio más la distancia hasta la planta de la que se recibe la llamada. De igual modo se actúa si la llamada es ascendente ya que la distancia se calcula como la suma de la distancia que resta hasta llegar a la planta alta más la distancia hasta llegar a la planta baja más la distancia que hay hasta la planta de la que se recibe la llamada.

Se muestra en la figura 21 el diagrama de flujo que se sigue para el cálculo de la distancia.

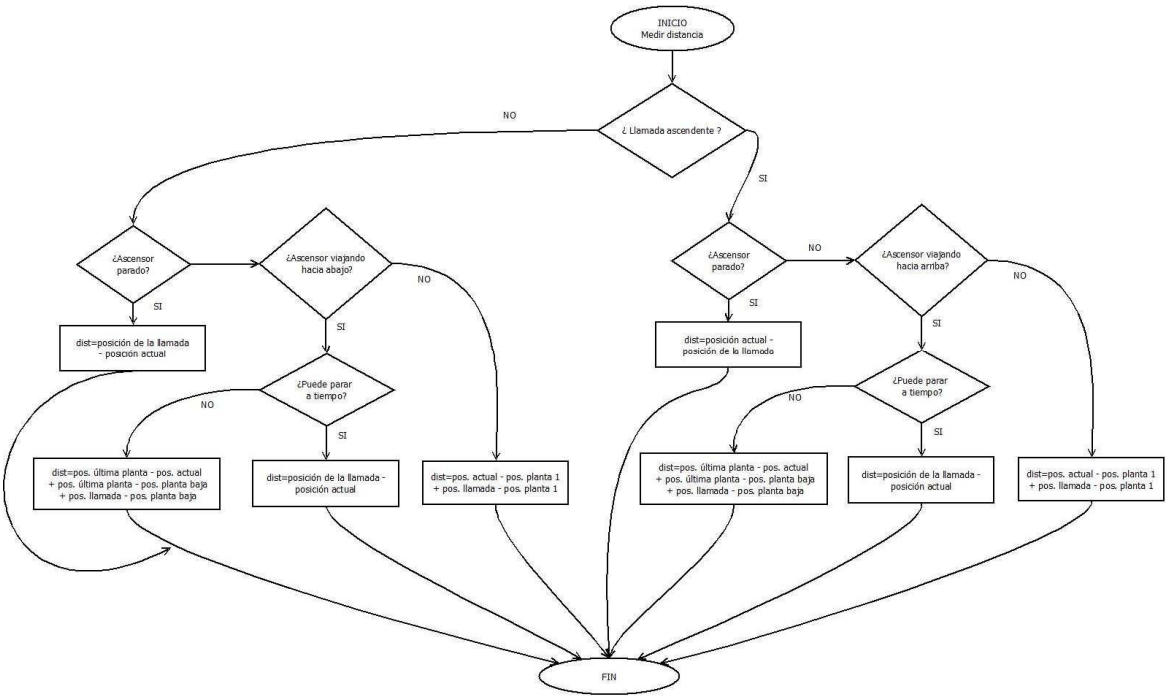


Figura 21: Diagrama de flujo del cálculo de la función objetivo

Para una mejor comprensión de este apartado, se van a ejemplificar varias posibles situaciones y el cálculo de la función objetivo correspondiente para cada caso:

"*Se tiene un edificio de 38 metros dividido en 10 plantas de 3,8 metros cada una. El edificio consta de 2 ascensores (A y B) con arquitectura Double Deck. Se da la situación de que en este instante se encuentra el ascensor A en la planta baja (parado) y el ascensor B en la planta 4 (ascendiendo).*

1. *Se recibe una llamada ascendente desde la planta 6.*

   -El ascensor A se encuentra a 3,8·6= 22,8 metros de distancia en este instante.

   -El ascensor B tiene 2 posibles casuísticas:

   a) *El ascensor aun puede parar en la planta 6 por lo que la distancia a la llamada es de 3,8·2= 7,6 metros.*

   b) *El ascensor no puede parar en la planta 6 por lo que la distancia a la llamada es de 3,8·6+3,8·10+3,8·4= 76 metros.*

2. *Se recibe una llamada ascendente desde la planta 3.*

   -El ascensor A se encuentra a 3,8·3= 11,4 metros de distancia de la llamada.

   -El ascensor B se encuentra a 3,8·6+3,8·10+3,8·3= 72,2 metros de distancia.

3. *Se recibe una llamada descendente desde la planta 4.*

   -El ascensor A se encuentra a 3,8·4= 15,2 metros de distancia.

   -El ascensor B se encuentra a 3,8·6+3,8·6= 45,6 metros de distancia de la llamada."

4.5.3. Buscar en la Lista Tabú

Una vez se tiene una solución propuesta con su función objetivo correspondiente se busca en la Lista Tabú dicha solución para comprobar si ya se ha visitado o no. En caso de que la solución propuesta se encuentre en la Lista Tabú, se comprueba si se cumple el criterio de aspiración, esto es, se acepta aquella solución cuya función objetivo se encuentra como máximo un 20 % por encima de la mejor función objetivo encontrada hasta el momento. En caso de que la solución propuesta no se encuentre en la Lista Tabú, si su función objetivo es mejor que la mejor función objetivo encontrada hasta ese punto, se acepta como posible solución al problema.

4.5.4. Guardar en la Lista Tabú

La Lista Tabú es una matriz en la cual se guarda por cada fila el vector de ascensores que se emplean y su función objetivo correspondiente. La capacidad de dicha lista viene dada por un término denominado *Tabú Ténere* y que es fijado al comienzo del uso del algoritmo. Cada vez que se va a incorporar una nueva solución a la Lista Tabú, se elimina la solución más antigua de la lista de forma que se sigue una tendencia *First In First Out* (FIFO).

4.5.5. Nueva solución

Para la creación de una nueva solución, se pueden dar tres casos:

1. Se genera una solución que se encuentra en la Lista Tabú y que no cumple el criterio de aspiración.
2. Se genera una solución que no se encuentra en la Lista Tabú pero no tiene una función objetivo mejor que la actual.
3. Se guarda la solución anterior en la Lista Tabú y se requiere de una nueva solución para seguir la búsqueda de una mejor solución.

El proceso para la creación de una nueva solución consiste en asignar al vector de ascensores que se emplean el valor contrario al que se tenía anteriormente si se cumple una verdad aleatoria, esto es, si anteriormente el ascensor "i" se empleaba (valor 1), si se cumple que un número aleatorio (entre 0 y 1) es mayor que 1/2, el ascensor "i" pasa a no emplearse (valor 0). De igual modo ocurre con aquellos ascensores cuyo valor actual es 0.

4.5.6. Asignación de llamadas a ascensores

Una vez han finalizado las iteraciones del proceso de búsqueda de soluciones, se escoge aquella solución cuya función objetivo es menor, es decir, aquella que consigue que la suma de las distancias desde los ascensores que se van a emplear hasta las plantas que tienen llamadas sea la menor posible, y se emplea para la asignación de llamadas a ascensores.

Los ascensores que se van a emplear, dados por la mejor solución obtenida, se encuentran asociados a cada una de las llamadas 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 sea servida por el ascensor *i* . Se guarda la solución de ascensores que se emplean en las variables *Allocate* y *Allocated* mediante un bucle de tamaño dos veces el número de plantas del edificio (*NoFloors·2*) en el que si la planta *j* tiene llamada, se guarda en la variable *Allocate* o Allocated (según sea la llamada ascendente o descendente) el valor de la variable *P(j)*, es decir, se guarda el ascensor *i* que se le asigna a esa planta. En la primera mitad del bucle se almacenan las llamadas ascendentes en la variable *Allocate* y en la segunda mitad las llamadas descendentes en la variable *Allocated*.

Debido a que los ascensores empleados tienen arquitectura *Double Deck* la asignación de cabinas no es trivial ya que una vez se ha asignado un ascensor a una planta hay que asignar con las variables de *Elevate* la cabina superior (*l[Allocate/d].m_UpLandingCallsUpperCar o l[Allocate/d].m_DownLandingCallsUpperCar* igual a 1) a las llamadas de plantas pares y la última del edificio y la cabina inferior (*m_UpLandingCalls* o m*_DownLandingCalls* igual a 1) a las llamadas de plantas impares y la primera del edificio.

En concreto, se realiza un bucle con el número de plantas (*NoFloors*) multiplicado por 2 ya que en la primera mitad se guardan las llamadas ascendentes y en la segunda mitad las llamadas descendentes. A cada planta con llamada, esto es, cuando las variables *m_UpLandingCalls* o *m_DownLandingCalls* toman el valor 1, se le ha asignado un ascensor que viene dado por las variables *Allocate* (llamadas ascendentes) y *Allocated* (llamadas descendentes) anteriormente citadas.

Una vez se tiene la planta con llamada y el ascensor que se le ha asignado se pregunta si la planta es la última del edificio. En caso afirmativo, se le asigna la cabina superior. En caso negativo, se pregunta si la llamada es par en cuyo caso se le asigna la cabina superior. Si la planta es impar o la planta baja del edificio se asigna la cabina inferior.

Además, en caso de que alguna de las cabinas del ascensor exceda el límite de peso permitido se debe desasignar la llamada a ese ascensor.

Se muestra en la figura 21 el diagrama de flujo que se emplea para la asignación de llamadas a ascensores con arquitectura *Double Deck.*

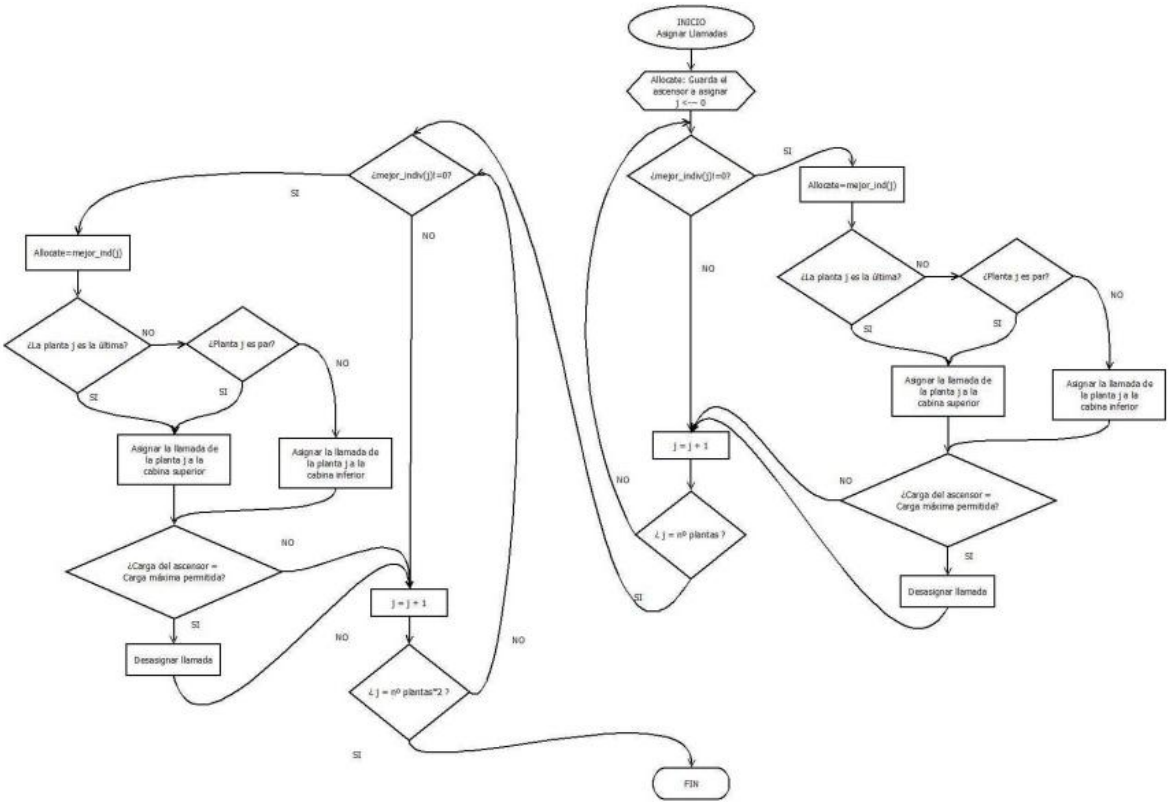


Figura 22: Asignación de llamadas a ascensores

5. IMPLEMENTACIÓN DEL SOFTWARE

Para poder implementar el algoritmo en primer lugar se deben instalar una serie de softwares. Uno de ellos es *Elevate 8*, software de simulación, desarrollado por Peters Research Ltd, en el cual se puede seleccionar el número, tipo y velocidad de ascensores en distintos tipos de edificios. Por otro lado, se tiene el software *Microsoft Visual Studio 2008* en el cual se programa el algoritmo de Búsqueda Tabú en lenguaje C++.

Además, se debe descargar un archivo desde la propia web de *Elevate* que permite obtener la interfaz de desarrollador, que es necesaria para la conexión entre *Elevate 8* y *Visual Studio 2008*.

Se recomienda encarecidamente la instalación de *Visual Studio* ***2008*** ya que además de ser el software recomendado por *Elevate*, se comprueba a través de prueba-error que es el único software que evoluciona favorablemente sin generar errores.

Se pasa a detallar ahora la instalación y características de ambos softwares.

5.1. Elevate 8

Una vez se instala *Elevate 8* y se abre, se muestra una pantalla inicial como la que se observa en la figura 22.

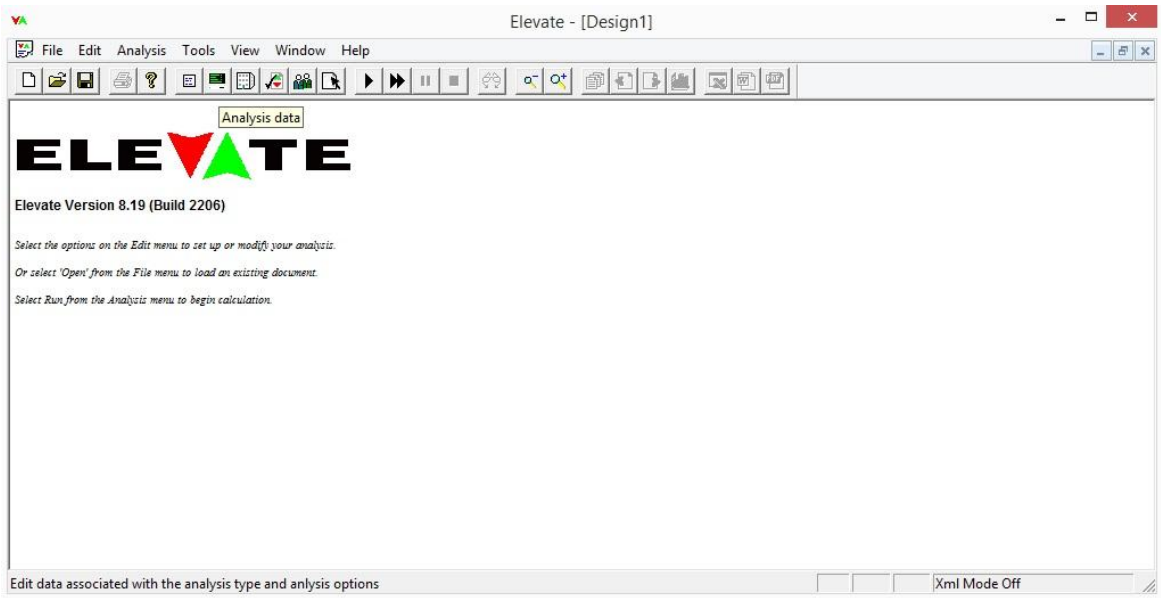


Figura 23: Ventana principal *Elevate*

5.1.1. Características del ascensor y edificio

Para configurar las características de los ascensores y edificio que se va a simular, se debe pulsar en el botón del software *Analysis Data*. Se despliega una primera ventana (se muestra en la figura 23) en la cual se debe seleccionar en *Analysis type* la opción *Simulation* y en *Dispatcher* la opción *Custom* si se quieren obtener los algoritmos de la interfaz de desarrollador. Además, en esta ventana se puede seleccionar el número de intervalos de tiempo antes de mostrar los resultados (*No of simulations to run for each configuration*) y la duración de dichos intervalos. Además, se pueden realizar simulaciones que incluyan el modelo energético (*Energy model*) aunque para este trabajo no se emplea.

Una vez se han realizado todos los cambios pertinentes en esta ventana, se avanza hacia otra serie de ventanas de configuración que se pasa a detallar.

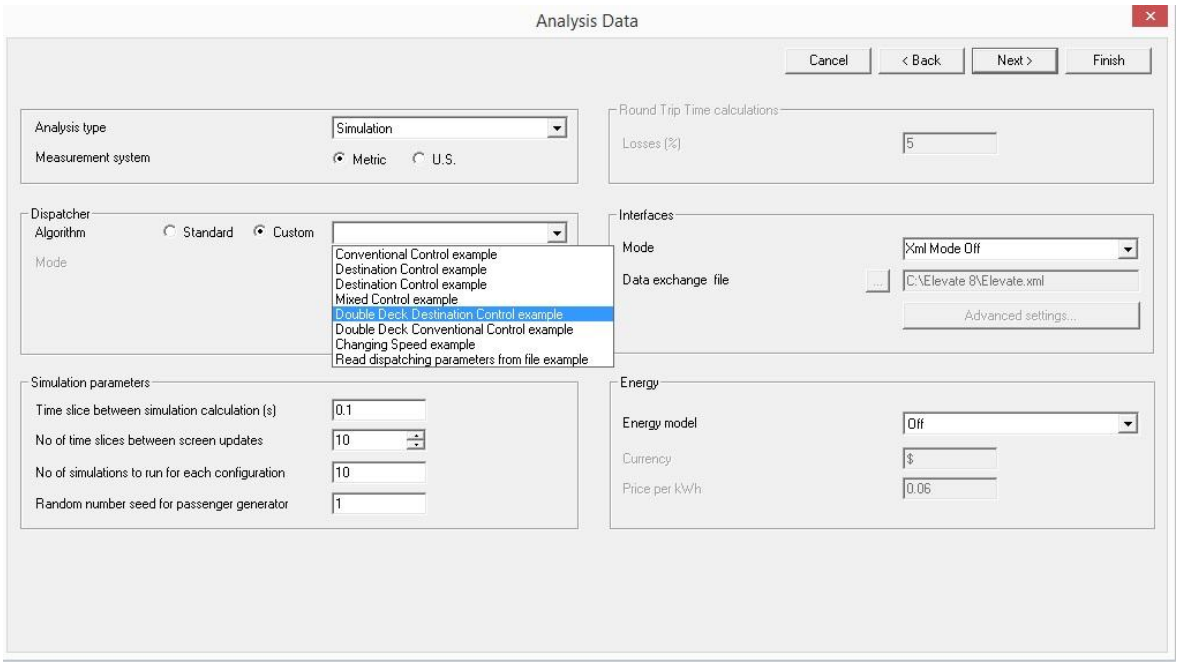


Figura 24: Ventana *Analysis Data*

Building Data:

En esta ventana que se muestra en la figura 24, se seleccionan las características del edificio tales como 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 selección de la planta entrada del edificio (*Entrance Floor*). Además, se puede regular el porcentaje de absentismo que hay en el edificio (*Absenteeism*).

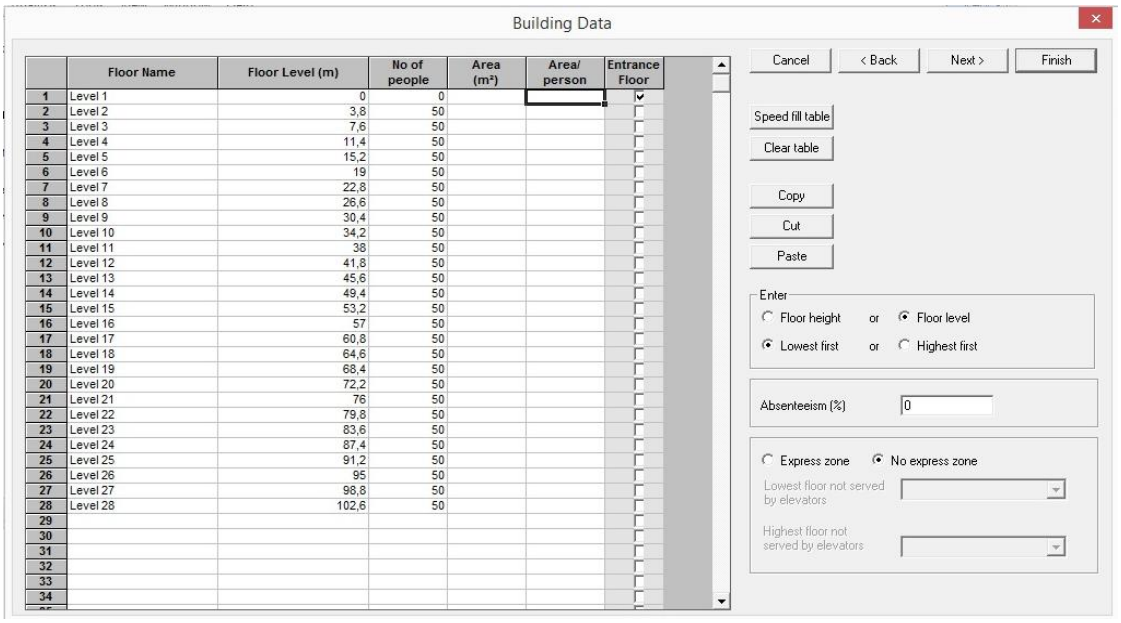


Figura 25: Ventana *Building Data*

Elevator Data:

En esta ventana, se seleccionan las características de los ascensores. Se pueden configurar en modo *Standard* en dónde se modifican las características de todos los ascensores a la vez o modo *Advanced* en cuyo caso se modifican las características de cada ascensor por independiente.

En el modo *standard* que se muestra en la figura 25 se puede seleccionar 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 segundo al cubo (*Jerk*), el tiempo de espera para comenzar el movimiento en segundos (*Start delay*) y la planta principal (*Home floor*).

En el modo *advanced* que se muestra en la figura 26 se pueden seleccionar los mismos datos anteriores con la salvedad de que en este caso se puede tener una configuración diferente para cada ascensor.

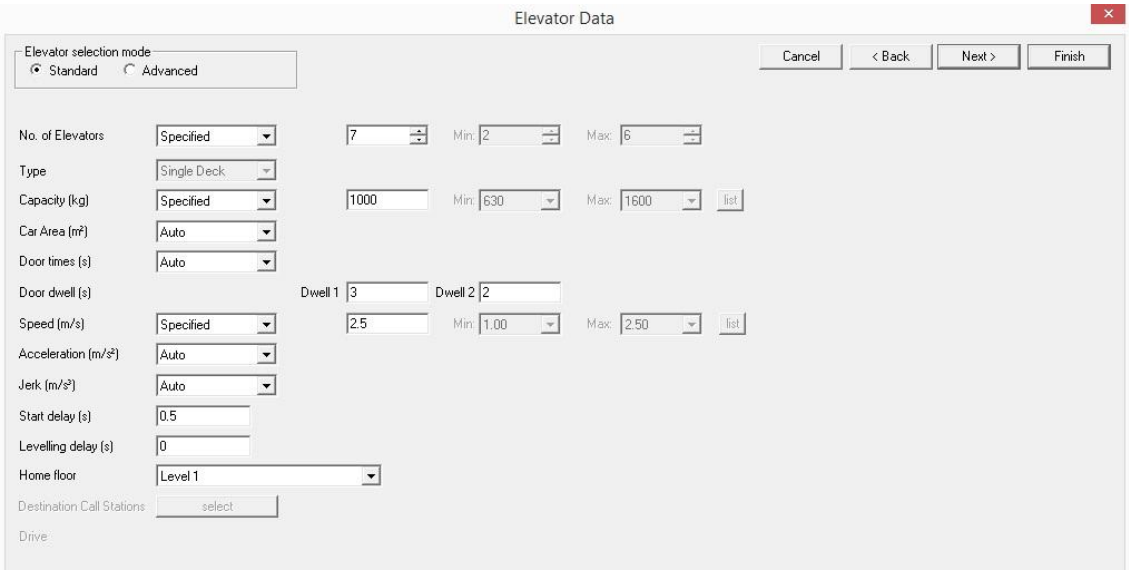


Figura 26: Ventana *standard Elevator Data*

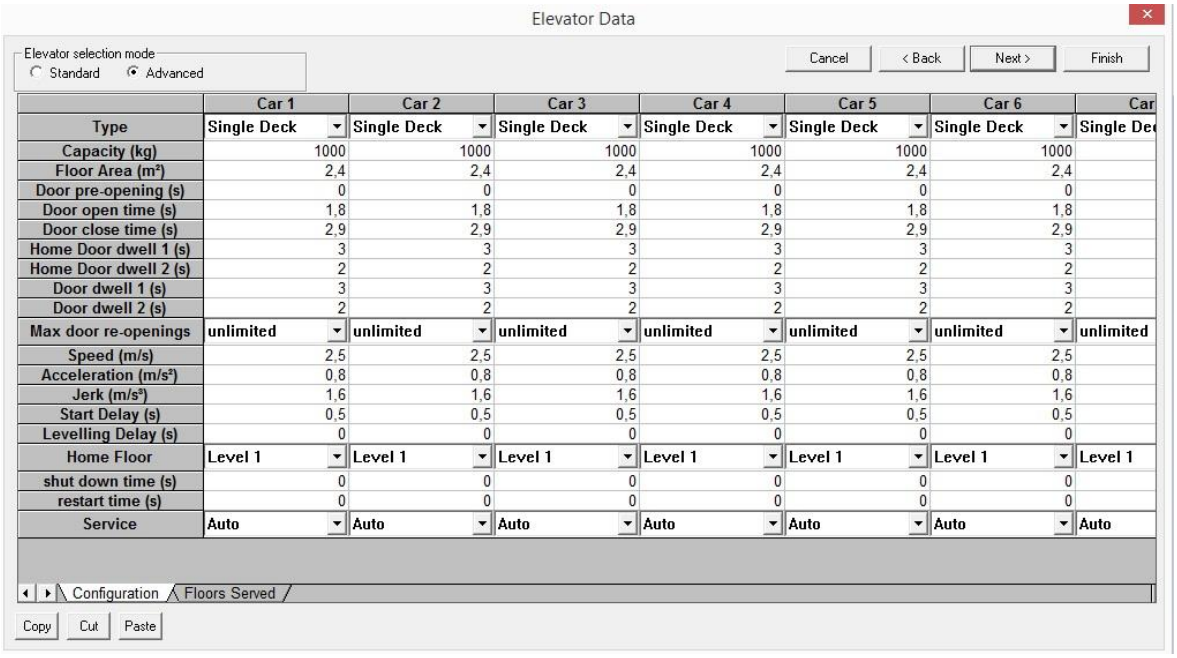


Figura 27: Ventana *advanced Elevator Data*

Passenger Data:

En esta ventana se seleccionan las características del tráfico de pasajeros del edificio. Al igual que en la ventana anterior, se cuenta con un modo *Standard* para la selección de características para toda la población por igual y un modo *Advanced* para una configuración personalizada a cada periodo de tiempo.

En el modo *standard* que se muestra en la figura 27 se puede seleccionar el tipo de tráfico que se quiere emplear según se tengan ascensores convencionales o con arquitectura *Double Deck* (*Arrangement*), 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 al edificio (*Incoming*), el porcentaje de personas que salen del edificio (*Outgoing*), el porcentaje de personas que se mueven entre 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 al ascensor en segundos (*Loading Time*), el tiempo que tardan los pasajeros en bajar del ascensor en segundos (*Unloading Time*), la masa de los pasajeros en kilogramos (*Passenger Mass*), el área que ocupa cada pasajero en metros cuadrados (*Passenger Area*), el porcentaje de factor de 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*).

En el modo *advanced* que se muestra en la figura 28 se pueden seleccionar los mismos datos anteriores con la salvedad de que se seleccionan según distintos periodos de tiempo.

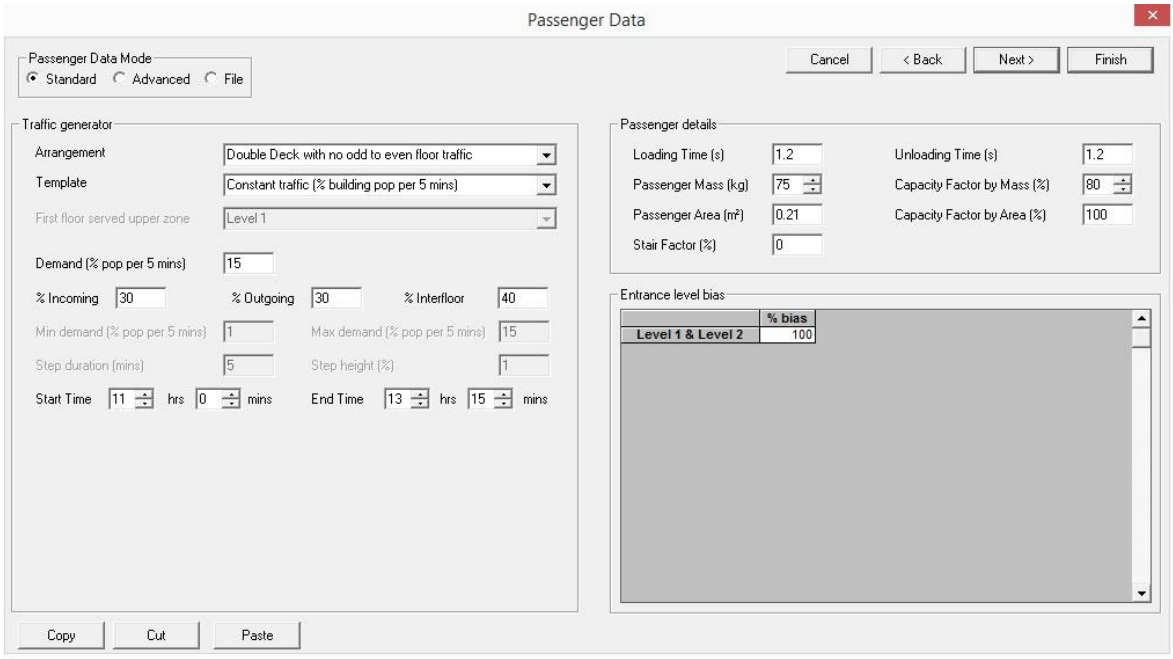


Figura 28: Ventana *standard Passenger Data*

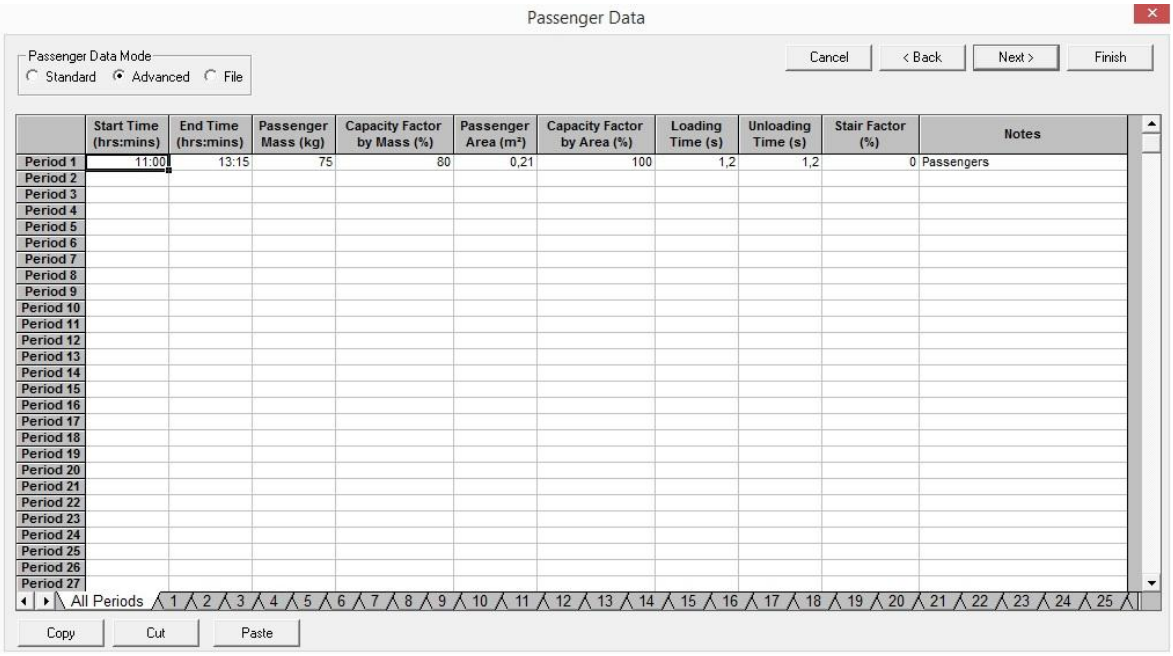


Figura 29: Ventana *advanced Passenger Data*

Se pueden seleccionar varios tipos 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. Al medio día tiene dos periodos de carácter moderado, uno *downpeak* y otro *uppeak*. Se muestra dicho patrón en la figura 29 (CIBSE Guide, 2000).

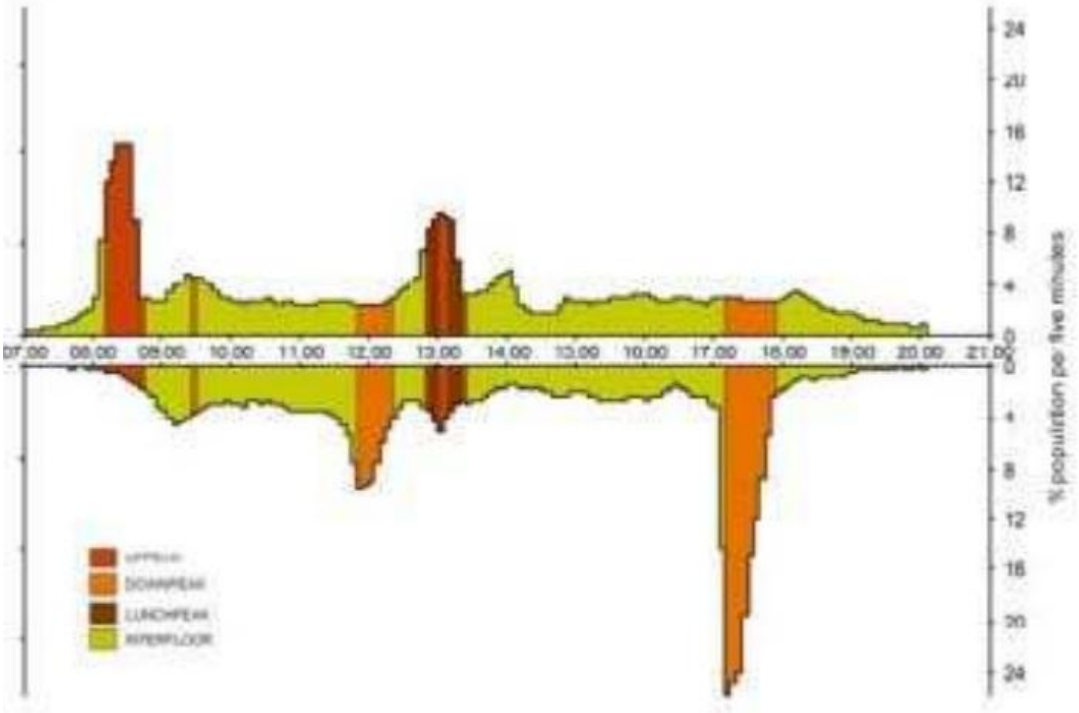


Figura 30: Patrón de tráfico CIBSE

- **Strackosh:**

  Se caracteriza por ser un modelo mucho más atípico, con curvas suaves, máximos pequeños y periodos ligeros *uppeak* y *downpeak* durante toda la mañana y tarde. Se muestra en la figura 30 (Cortés P. et al., 2009).

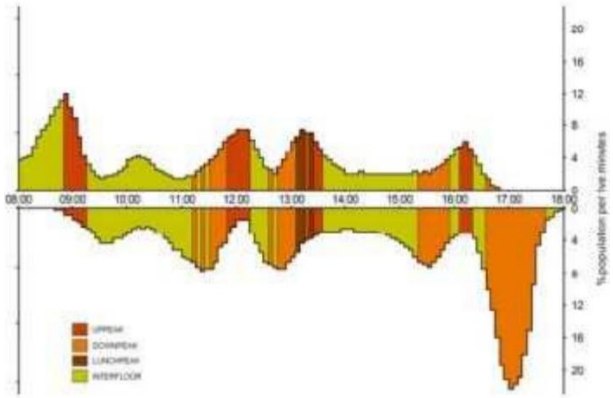


Figura 31: Patrón de tráfico *Strackosh*

- **Siikonen:**

  Se caracteriza por ser el patrón de tráfico de pasajeros que más se aproxima al comportamiento real de flujo de pasajeros. Se muestra en la figura 31 (Cortés P. et al., 2009).

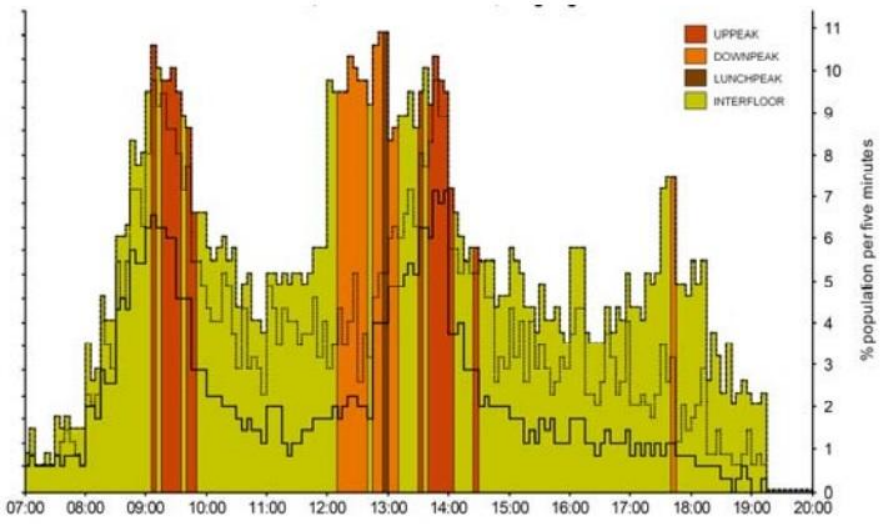


Figura 32: Patrón de tráfico *Siikonen*

Report options:

En esta ventana que se puede observar en la figura 32, se seleccionan los tipos de análisis que se quiere que se desplieguen automáticamente al finalizar la simulación. Para la simulación que se lleva a cabo 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 los 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*).

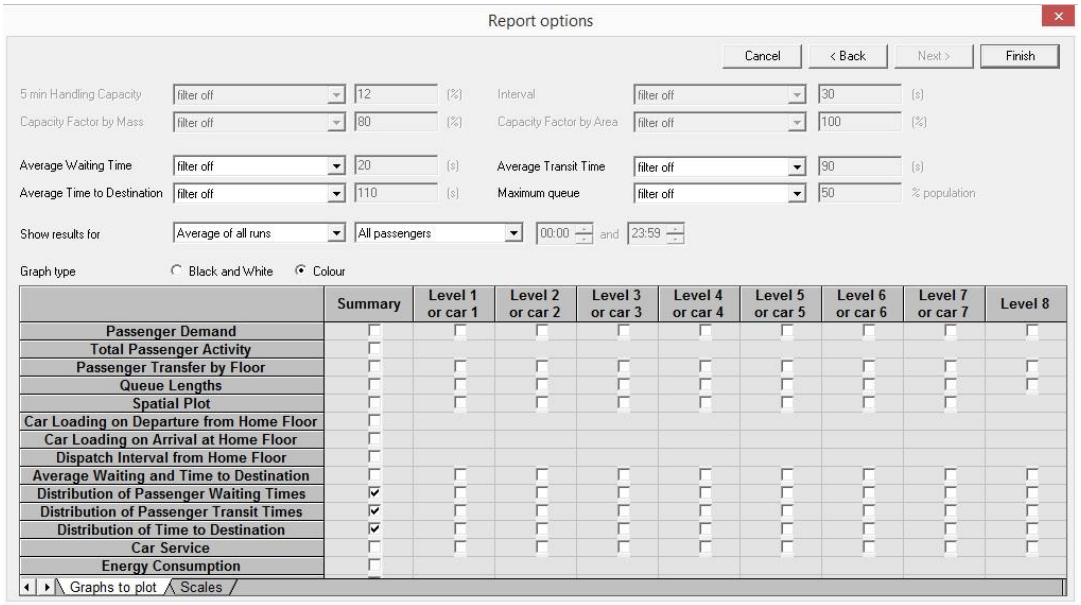


Figura 33: Ventana *Report Options*

Simulación

Una vez se ha configurado lo anterior, se puede visualizar una simulación del edificio y ascensores. En esta ventana se encuentran las diferentes plantas del edificio, los ascensores con sus respectivas puertas, flechas de color verde y rojo para las llamadas que se realizan de cada planta (ascendentes y descendentes respectivamente) y puntos azules que indican los destinos de los pasajeros. Se muestra dicha ventana en la figura 33.

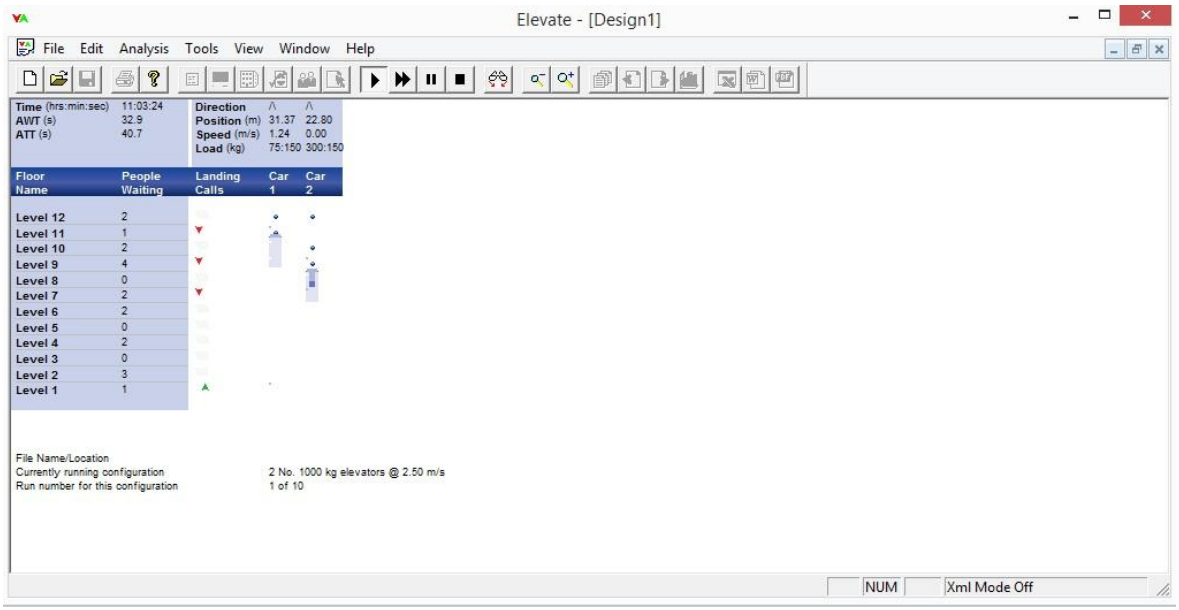


Figura 34: Ventana de simulación de *Elevate*

Resultados del software

Finalizada la simulación, *Elevate* ofrece los resultados que se han seleccionado en la ventana *Report Options.* Para este trabajo son de especial utilidad los resultados obtenidos del tiempo medio de espera y tiempo medio de tránsito.

5.2. Microsoft Visual Studio 2008

Se requiere la instalación de este programa ya que es el recomendado por *Elevate* para el desarrollo del algoritmo a través de la interfaz de desarrollador. La instalación de dicha interfaz no es trivial debido a que se requiere exactamente la versión 2008 del software para el correcto funcionamiento de la conexión entre ambos softwares a través de la interfaz.

5.2.1. Interfaz de desarrollador

Al descargar el archivo necesario para la instalación de la interfaz de desarrollador de la web de Peters Research Ltd. hay que seguir una serie de pasos detallados en una carpeta llamada *How to Use*. Hay un aspecto a destacar, y es que a la hora de la configuración del vinculador siguiendo los pasos *Propiedades* → *Todas las configuraciones* → *Vinculador* se debe generar el archivo "Dispatch_00.dll" en una carpeta que no exija permisos de administrador para evitar problemas de depuración. Se muestra en la figura 34 la ventana de propiedades de *Visual Studio* en la cual habrá que realizar los cambios pertinentes para conseguir la interfaz de desarrollador.

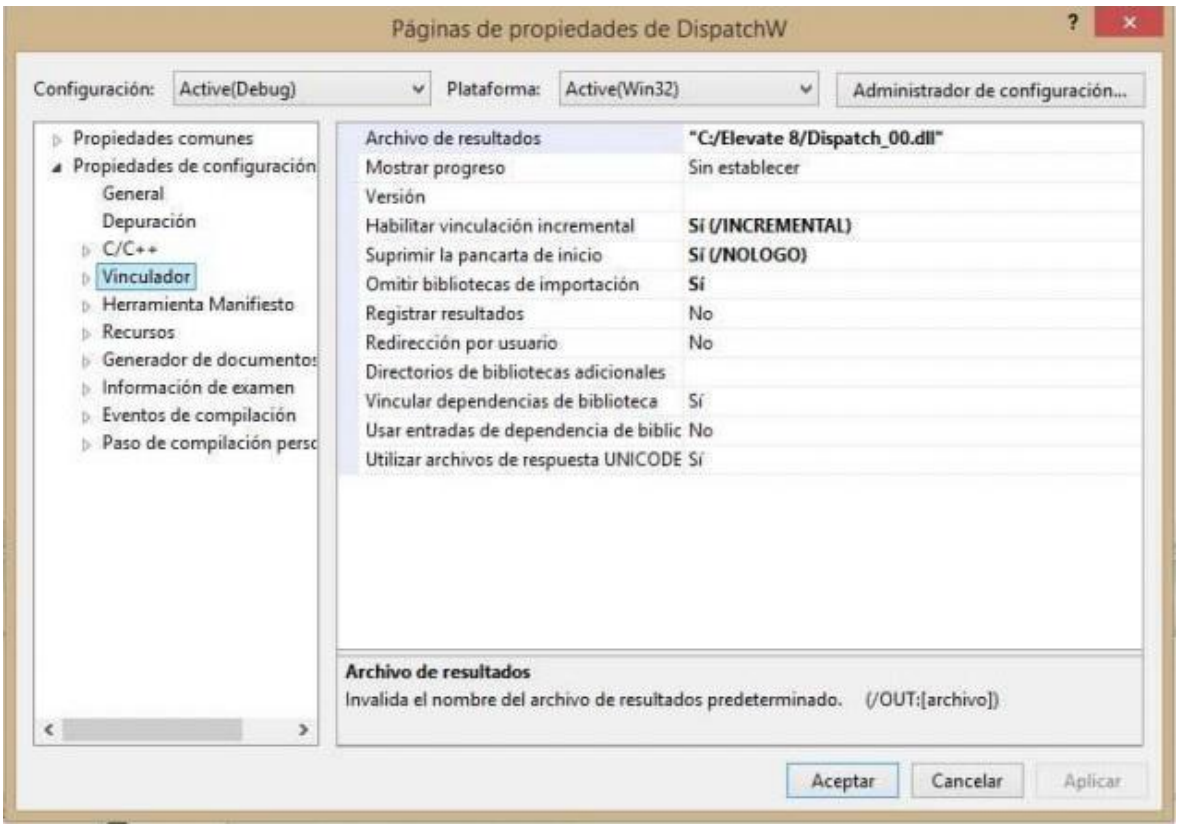


Figura 35: Ventana de propiedades de *Microsoft Visual Studio*

5.2.2. Desarrollo en Microsoft Visual Studio

El lenguaje de programación que se emplea para la programación del algoritmo es C++. Se muestra en la figura 35 parte del algoritmo desarrollado en *Microsoft Visual Studio*.

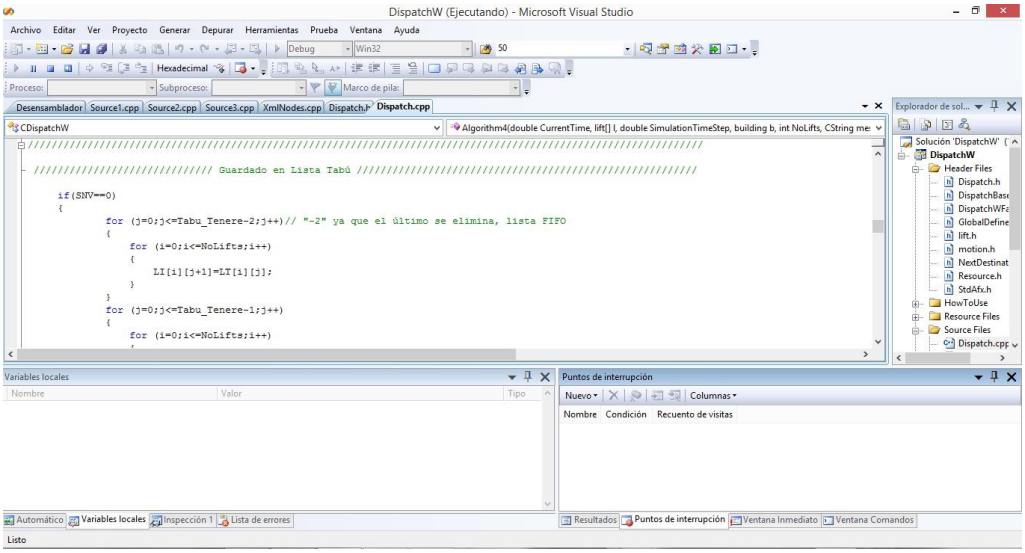


Figura 36: Extracto del algoritmo programado en Visual Studio

A través del empleo de puntos de rotura en el código se consigue depurar para observar que el algoritmo programado realiza la función requerida además de poder extraer los valores que toman las variables del problema en cada instante.

6. EXPERIMENTACIÓN

Se pasa a experimentar con el modelo de simulación mediante *Elevate* con el objetivo de comprobar la bondad del algoritmo programado con respecto al tiempo medio de espera (AWT) de los pasajeros.

6.1. Algoritmos de comparación

Para realizar dicha experimentación, se cuenta con una serie de algoritmos de asignación de llamadas con los cuales se comparan los resultados obtenidos para poder comprobar el funcionamiento correcto, o no, del algoritmo de Búsqueda Tabú. Los algoritmos que se van a emplear para comparar son:

- **Double Deck Destination Control:**

  Este algoritmo se encuentra implementado de forma interna en *Elevate*. Su funcionamiento consiste en el cálculo de los tiempos medios de espera y de tránsito de cada posible asignación. La asignación de llamadas a ascensores se realiza a través de una función de coste que minimiza el tiempo de espera y tránsito.

  En primer lugar, se recorre mediante un bucle todo el edificio en busca de llamadas. Cuando se tiene una llamada desde alguna planta del edificio (*m_UpLandingCalls* o *m_DownLandingCalls* igual a 1), se calcula la distancia desde cada uno de los ascensores disponibles, es decir, aquellos ascensores que toman el valor 1 en el vector de ascensores, hacia esa llamada. Una vez se ha calculado la distancia desde cada uno de ellos siguiendo el mismo tipo de función que se emplea en el apartado 4.5.2, se busca aquel ascensor que ha obtenido la menor distancia y se asigna a esa llamada de modo que el ascensor *i* se encuentra asignado a la planta *j*. Asignado el ascensor a la planta, se emplea la misma función del apartado 4.5.6 para asignar la cabina superior o inferior a la llamada. Cabe destacar que una vez se ha enviado un ascensor a una planta, este ascensor no para en otras plantas para recoger nuevos pasajeros hasta llevar al destino correspondiente a los pasajeros que ha recogido en la asignación primera, se consigue de este modo minimizar también el tiempo medio de tránsito.

Se puede observar en la figura 36 el diagrama de flujo de dicho algoritmo.

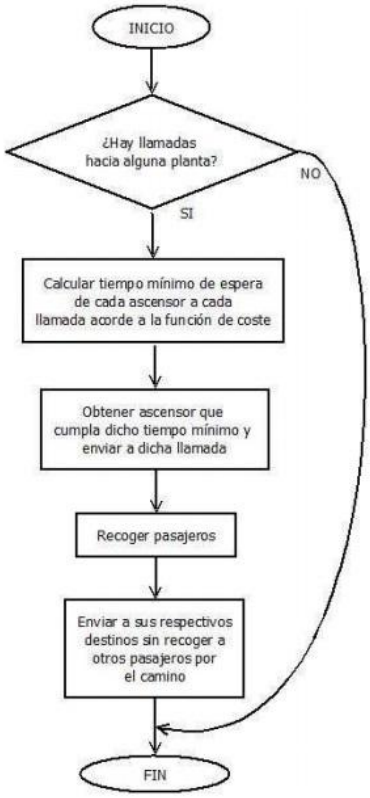


Figura 37: Diagrama de flujo del algoritmo *Destination Control*

- **Algoritmo Genético:**

  Es un método adaptativo que se basa en el proceso genético de los organismos vivos. Se parte de una población inicial, compuesta por individuos con sus respectivos genes, que evoluciona bien a través de la mutación bien a través del cruce para dar lugar a mejores individuos. El algoritmo genético aplicado a la asignación de llamadas a ascensores calcula el fitness de cada individuo, esto es, la suma de las distancias de cada ascensor a sus respectivas plantas, y a través de la evolución de la población se logra el mejor fitness que da lugar a la mejor asignación de llamadas a ascensores.

  Cuando se reciben llamadas desde las plantas del edificio (*m_UpLandingCalls* o *m_DownLandingCalls* igual a 1), se genera una población inicial de asignaciones de ascensores, es decir, se rellena una matriz en la que por cada fila se encuentra el número de ascensores disponibles que se rellenan con 1 y 0 en función de si el ascensor es empleado o no respectivamente. La población es evaluada en función de la distancia que hay entre los ascensores y las llamadas siguiendo la función que se detalla en el apartado 4.5.2.

  Una vez se ha evaluado toda la población, esto es, una vez se ha calculado la distancia total resultante de asignar esa población a las llamadas, se selecciona el mejor individuo, es decir, aquella fila de la matriz que ha conseguido que la distancia sea menor, para reproducirlo en pos de encontrar una nueva mejor solución. En función de una probabilidad, la reproducción se realiza vía mutación o vía cruce siendo la probabilidad de cruce del 85% y de mutación del 15%.

  En el caso de que se produzca un cruce, se seleccionan 2 de los mejores individuos de la

población (son los "padres") y se combinan sus características de asignación de llamadas a ascensores (se consigue un "hijo"), es decir, se realiza una combinación de ascensores empleados por los padres para dar lugar a un hijo que combina los genes (ascensores empleados) de los padres. Se consigue de este modo una posible mejora de la distancia de llamadas a ascensores.

En el caso de que se produzca una mutación, se selecciona el mejor individuo y se realizan combinaciones de sus genes, es decir, se intercambia la asignación de ascensores a llamadas. *A modo de ejemplo, si el ascensor 1 se encuentra asignado a la llamada de la planta 5 ascendente y el ascensor 3 se encuentra asignado a la llamada de la planta 7 descendente, para realizar la mutación se intercambian ambas asignaciones de modo que tras ella el ascensor 1 queda asignado a la llamada de la planta 7 descendente y el ascensor 3 a la llamada de la planta 5 ascendente.*

Una vez se realiza la reproducción se obtienen nuevos individuos, esto es, nuevos valores de 1 y 0 para el vector de ascensores, que vuelven a ser evaluados mediante el cálculo de distancias. Si se alcanza el criterio de optimización que viene dado por la consecución de mejorar la mejor distancia conseguida hasta el momento, el algoritmo devuelve el mejor individuo. En caso contrario se vuelve a seleccionar el mejor individuo o los mejores individuos (en función de si se realiza mutación o cruce, respectivamente) para volver a reproducirlo. El proceso se realiza iterativamente hasta encontrar el mejor individuo y por tanto la mejor asignación de llamadas a ascensores.

Se muestra en la figura 37 el diagrama de flujo que emplea este algoritmo (Mesa J., 2015).

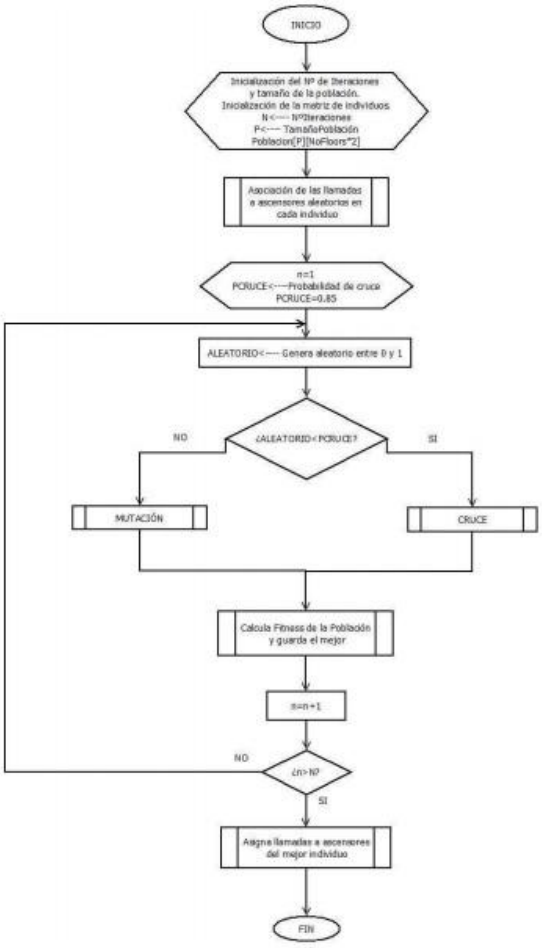


Figura 38: Diagrama de flujo del algoritmo genético

- **Algoritmo Genético Modificado:**

  Este algoritmo sigue el mismo diseño que el anterior con la diferencia de que en este caso se desasignan llamadas de forma que si un ascensor ya se encuentra atendiendo una llamada, este no para a servir otras plantas hasta no terminar con el servicio actual. Esta variación se realiza con el objetivo de mejorar el tiempo de tránsito de los pasajeros.

6.2. Características de la experimentación

Debido a que se va a realizar una comparación con los algoritmos citados anteriormente, las características de edificios y ascensores del experimento a realizar deben ser iguales para que los resultados obtenidos se puedan comparar sin pérdida de veracidad.

6.2.1. Edificios

Se van a emplear una serie de edificios diferentes que permitan observar el comportamiento de cada algoritmo cuando la altura del edificio se incrementa. Las características de los edificios se muestran en la tabla 4, en la cual la variable "x" toma los valores: 12, 20, 24, 28, 32, 36 y 40.

Tabla 4: Características del edificio

	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

Ya que se emplean ascensores con arquitectura *Double Deck*, los edificios que se emplean en la experimentación cuentan con plantas pares 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 alguna) en la parte superior e inferior del edificio que permitan que el ascensor pueda acceder a la primera planta con la cabina superior y a la última planta con la cabina inferior.

6.2.2. Patrón de tráfico

El tráfico que se va a emplear para la experimentación es constante (*Constant Traffic (% Building pop per 5 mins)*). Debido a que el patrón de tráfico *Interfloor* ha adquirido un gran protagonismo en la actualidad debido a la flexibilización de los horarios y la prohibición de fumar dentro del edificio se cuenta con que en el edificio hay 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. Se va a suponer también que el uso de las escaleras en el edificio es nulo.

Además, hay una semilla aleatoria (porcentaje del número de pasajeros en cada planta) que se varía según el número de ascensores del edificio con el objetivo de mantener un flujo de pasajeros regular. Dicha semilla toma los valores:

Demand (% pop per 5 mins): 12,5 15 17,5 20 22,5

6.2.3. Ascensores

El número de ascensores que se emplean varía según el número de plantas del edificio. Las características de los ascensores se muestran en la tabla 5.

Tabla 5: Características del ascensor

Capacidad de los ascensores	1000 kilogramos
Velocidad de los ascensores	2,5 metros/segundos
Aceleración de los ascensores	0,8 metros/$segundos^2$
Retraso de inicio de marcha	0,5 segundos
Tiempo de apertura de puertas	1,8 segundos
Tiempo de cierre de puertas	2,9 segundos
Tiempo de carga y descarga	1,2 segundos
Factor de capacidad	80%

6.3. Resultados de la experimentación

Para cada edificio se han realizado simulaciones de 2 horas de duración para cada número de ascensores. *Elevate* devuelve el tiempo medio de espera de los pasajeros del edificio. Los resultados obtenidos se muestran en la tabla 6.

Tabla 6: Tiempos medios de espera de la experimentación

Número Plantas	Número Ascensores	Algoritmo Genético	Genético Modificado	Destination Control	Búsqueda Tabú
12	2	44,3	48,6	45,5	53
	3	30	35,5	32,7	31,1
	4	22,2	24,9	24,5	21,3
	5	16,5	18,9	19,7	15,8
	7	11,3	12,4	13,6	11,5
20	4	51,1	53	52	57,6
	5	36,8	43,1	46	46,1
	7	23,8	32	33,9	30,1
	8	21,8	26,3	30,9	21,8
	11	15,4	18,3	22,1	17,2
24	5	51,3	51,6	55,5	60,7
	7	32,2	38,4	43,7	39,8
	8	28,8	34,6	42,1	30,9
	10	23,5	29,1	34,8	25,8
	13	17,1	21,7	26,8	18,8
28	6	57	63,4	62,7	62,6
	9	30,9	37	44,8	40,4
	11	24,9	31,5	38,2	29
	13	19,7	28,4	34,2	24
	16	18,2	22,2	28,8	19,1
32	6	94,5	102,8	77,8	108,7
	10	36,8	43,4	50,9	40,7
	12	30,3	37,9	44,2	31,8
	14	23,6	33,1	41,2	28
	17	20,9	28,2	34,8	27
36	7	89,4	105,4	78,5	107,8
	11	39,3	56,5	55,6	51,5
	14	31,7	44,3	47,3	38
	16	27,6	36,6	43,4	30,9
	19	23,7	29,7	38,8	24,2
40	7	218,9	238	97,2	139,7
	13	43,2	52,6	55,6	45,7
	16	31,8	42,3	49	40
	19	27,8	37,4	44,2	31,1
	21	26,2	34,9	41,8	27,8

Además, para cada simulación de las realizadas se ha obtenido una gráfica en la que se muestra la evolución del tiempo medio de espera de los pasajeros. Se muestra en la figura 38 la gráfica obtenida para una simulación del algoritmo de Búsqueda Tabú para un edificio con 40 plantas, 21 ascensores y semilla aleatoria de 22,5 % pop. El resto de gráficas obtenidas se muestran en el Anexo A de este documento.

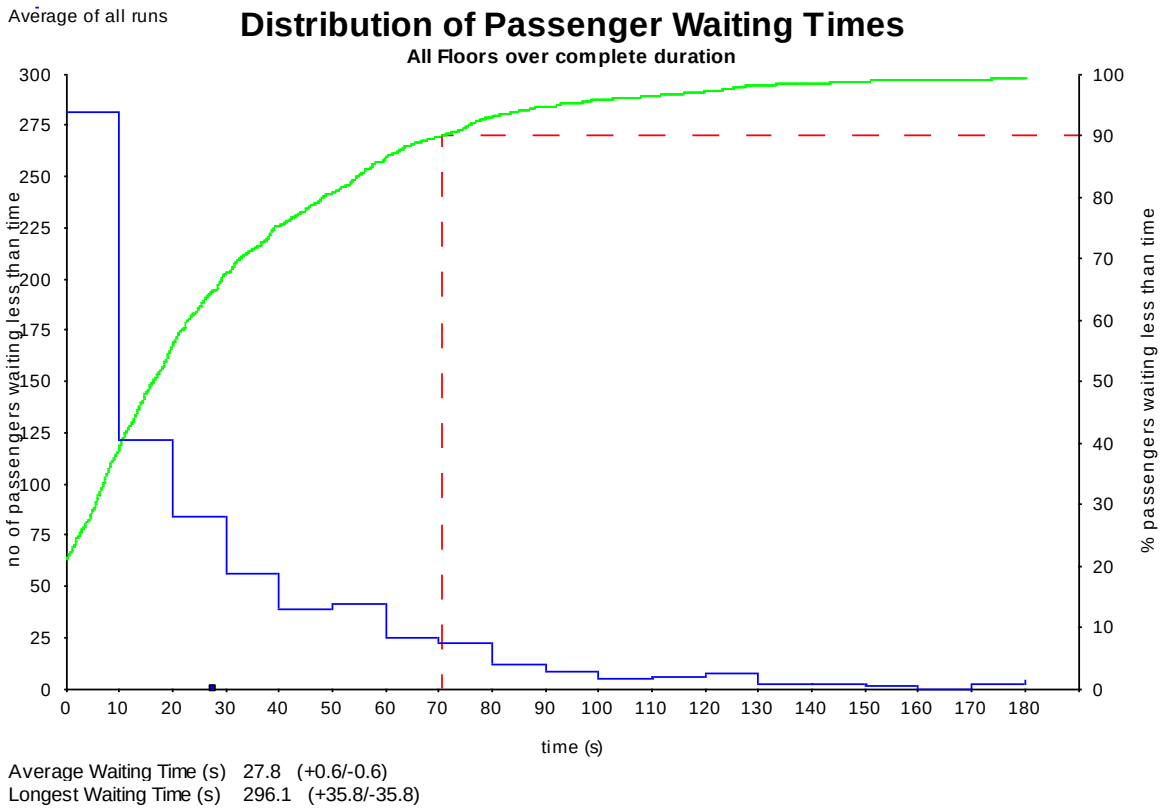


Figura 39: Gráfica para el tiempo medio de espera

Se observa en el lado izquierdo de la gráfica, en el eje de abscisas, el número de personas que esperan menos tiempo que el tiempo que se encuentra en el eje de ordenadas. En el lado derecho, encontramos la cantidad de personas anterior dada en porcentaje.

La línea verde de la gráfica indica la evolución del tiempo de espera de los pasajeros, mientras que las líneas azul y roja indican respectivamente la cantidad y porcentaje de personas que esperan menos tiempo que el marcado en el eje del tiempo.

En este caso concreto se observa que el 90% de las personas esperan menos de aproximadamente unos 70 segundos. Alrededor de 275 personas esperan menos de 10 segundos. Hay 125 personas que esperan menos de 20 segundos, aproximadamente unas 80 personas menos de 30 segundos, unas 50 personas menos de 40 segundos. Entre 25 y 50 personas esperan entre unos 40 y 70 segundos. Se encuentran picos de espera de entre 80 y 296,1 segundos para un rango de personas entre 1 y 25. El tiempo medio de espera es de 27,8 segundos con un error de ± 0,6 segundos. Además, el mayor tiempo de espera que se ha dado en esta simulación concreta es de 296,1 segundos con un error de ± 35,8 segundos.

6.4. Diferencia de tiempos medios de espera

Se muestra a continuación en la tabla 7, una comparativa de los resultados obtenidos con el algoritmo de Búsqueda Tabú con respecto a los otros 3 algoritmos. Se muestra la diferencia de resultados que se ha obtenido con el algoritmo de Búsqueda Tabú comparado con el valor obtenido con los otros algoritmos. Se expone, además, el porcentaje que ha mejorado el AWT, o empeorado, dado por el algoritmo de Búsqueda Tabú con respecto a los tiempos obtenidos con los otros algoritmos junto al valor de la diferencia. En dicha tabla se observa en color rojo aquellas ocasiones en las cuales el algoritmo de Búsqueda Tabú ha obtenido peor tiempo medio de espera y en verde cuando se ha obtenido mejor tiempo.

"*A modo de ejemplo aclaratorio se tiene que para el caso de 12 plantas con 2 ascensores los tiempos medios de espera que se obtienen para los distintos algoritmos son:*

Algoritmo Genético	*Genético Modificado*	*Destination Control*	*Búsqueda Tabú*
44,3 s	*48,6 s*	*45,5 s*	*53 s*

Por lo que los tiempos de diferencia de los 3 primeros algoritmos con respecto a la Búsqueda Tabú son:

Algoritmo Genético	*Genético Modificado*	*Destination Control*
-8,7 s	*-4,4 s*	*-7,5 s*

Se tiene por tanto que los porcentajes de empeoramiento, en este caso concreto, son de:

Algoritmo Genético	*Genético Modificado*	*Destination Control*
-19,64%	*-9,05%*	*-16,48%*

Se puede decir para este caso de edificio con 12 plantas y 2 ascensores que el algoritmo de Búsqueda Tabú ofrece un AWT un 16,48% peor que el algoritmo Destination Control, un 9,05% peor que el algoritmo genético modificado y un 19,64% peor que el algoritmo genético."

Tabla 7: Comparativa de los tiempos medios de espera

Número Plantas	Número Ascensores	Ag-Bt (s)	Ag-Bt %	AgM-Bt (s)	AgM-Bt %	Dc-Bt (s)	Dc-Bt %
12	2	-8,7	-19,64	-4,40	-9,05	-7,50	-16,48
	3	-1,1	-3,67	4,40	12,39	1,60	4,89
	4	0,9	4,05	3,60	14,46	3,20	13,06
	5	0,7	4,24	3,10	16,40	3,90	19,80
	7	-0,2	-1,77	0,90	7,26	2,10	15,44
20	4	-6,5	-12,72	-4,60	-8,68	-5,60	-10,77
	5	-9,3	-25,27	-3,00	-6,96	-0,10	-0,22
	7	-6,3	-26,47	1,90	5,94	3,80	11,21
	8	0	-	4,50	17,11	9,10	29,45
	11	-1,8	-11,69	1,10	6,01	4,90	22,17
24	5	-9,4	-18,32	-9,10	-17,64	-5,20	-9,37
	7	-7,6	-23,60	-1,40	-3,65	3,90	8,92
	8	-2,1	-7,29	3,70	10,69	11,20	26,60
	10	-2,3	-9,79	3,30	11,34	9,00	25,86
	13	-1,7	-9,94	2,90	13,36	8,00	29,85
28	6	-5,6	-9,82	0,80	1,26	0,10	0,16
	9	-9,5	-30,74	-3,40	-9,19	4,40	9,82
	11	-4,1	-16,47	2,50	7,94	9,20	24,08
	13	-4,3	-21,83	4,40	15,49	10,20	29,82
	16	-0,9	-4,95	3,10	13,96	9,70	33,68
32	6	-14,2	-15,03	-5,90	-5,74	-30,90	-39,72
	10	-3,9	-10,60	2,70	6,22	10,20	20,04
	12	-1,5	-4,95	6,10	16,09	12,40	28,05
	14	-4,4	-18,64	5,10	15,41	13,20	32,04
	17	-6,1	-29,19	1,20	4,26	7,80	22,41
36	7	-18,4	-20,58	-2,40	-2,28	-29,30	- 37,32
	11	-12,2	-31,04	5,00	8,85	4,10	7,37
	14	-6,3	-19,87	6,30	14,22	9,30	19,66
	16	-3,3	-11,96	5,70	15,57	12,50	28,80
	19	-0,5	-2,11	5,50	18,52	14,60	37,63
40	7	79,2	36,18	98,30	41,30	-42,50	-43,72
	13	-2,5	-5,79	6,90	13,12	9,90	17,81
	16	-8,2	-25,79	2,30	5,44	9,00	18,37
	19	-3,3	-11,87	6,30	16,84	13,10	29,64
	21	-1,6	-6,11	7,10	20,34	14,00	33,49

Se puede observar que el algoritmo de Búsqueda Tabú obtiene mejores resultados en gran parte de las simulaciones cuando lo comparamos con el algoritmo *Destination Control* que implementa *Elevate* o con el algoritmo genético modificado. En cambio, cuando el edificio tiene pocos ascensores, el algoritmo de Búsqueda Tabú obtiene peores resultados.

Si lo comparamos con el algoritmo genético, se obtienen peores resultados en parte de las simulaciones realizadas, aunque sí es cierto que la diferencia de resultados entre ambas es despreciable en muchos casos ya que se tienen diferencias de pocos segundos.

Se muestra a continuación en la figuras 39 y 40 unas gráficas en las cuales se observa la evolución de cada uno de los algoritmos empleados cuando aumenta el número de ascensores. Se observa que cuando los edificios tienen un número pequeño de ascensores, el AWT es grande y que comienza a disminuir una vez el número de ascensores aumenta. Además, se observa un gran pico de tiempo medio de espera para la casuística del edificio de 40 plantas con 7 ascensores.

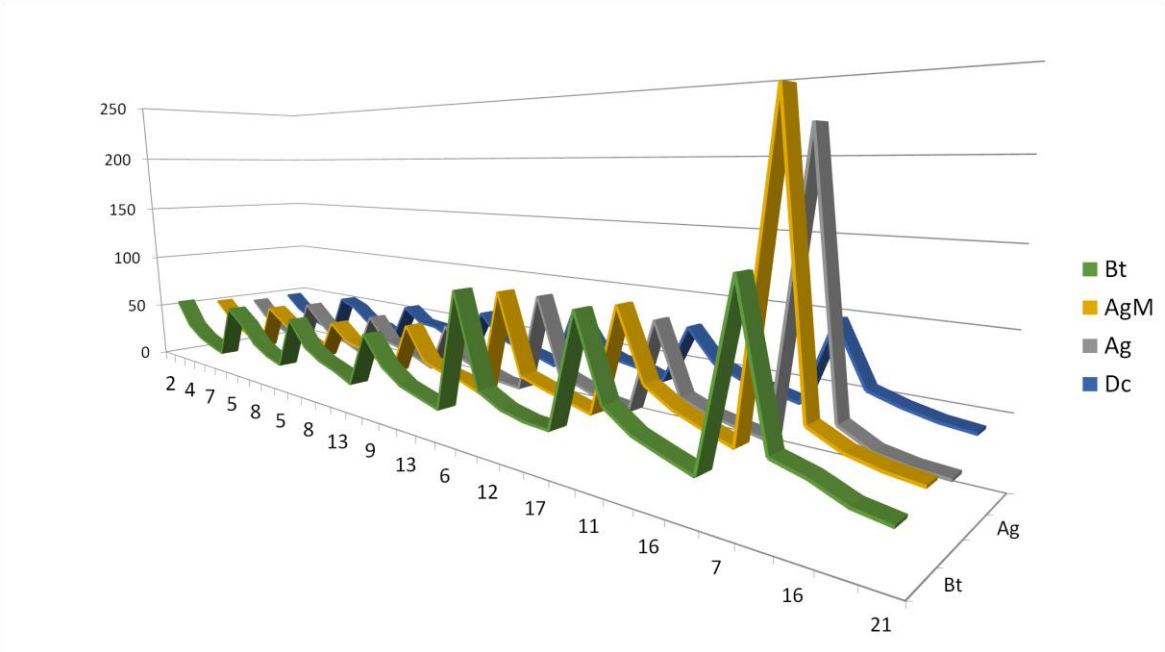


Figura 40: Gráfico 3D para el AWT

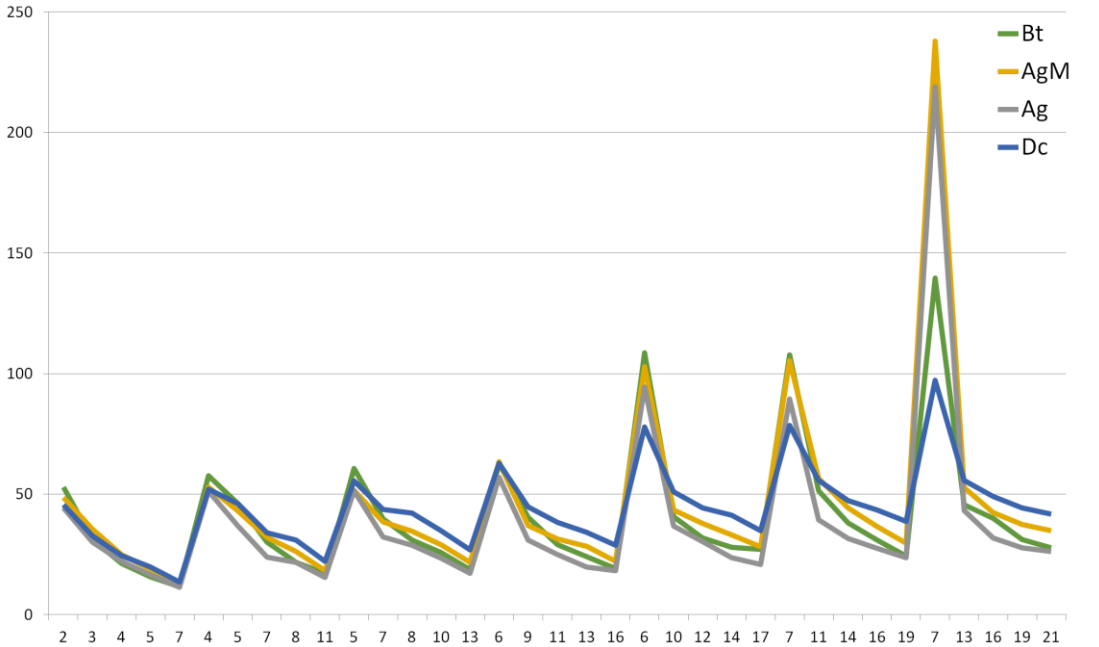


Figura 41: Gráfico 2D para el AWT

Se han realizado una serie de gráficas de superficie mediante el software *Matlab* en las que se representa en el eje vertical el tiempo medio de espera y en los ejes horizontales el número de ascensores y número de plantas para cada uno de los algoritmos de estudio. Se muestran en las figuras 42, 43, 44,45 y 46 que representan diferentes vistas del gráfico 3D.

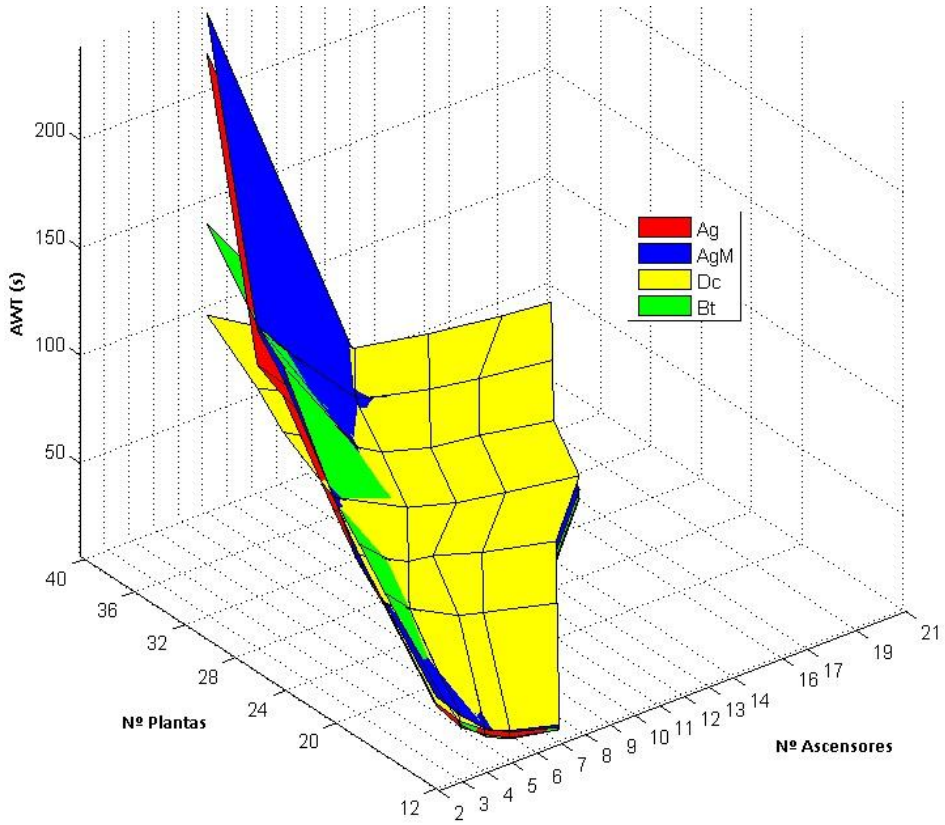


Figura 42: Gráfica 3D del AWT (vista general 1)

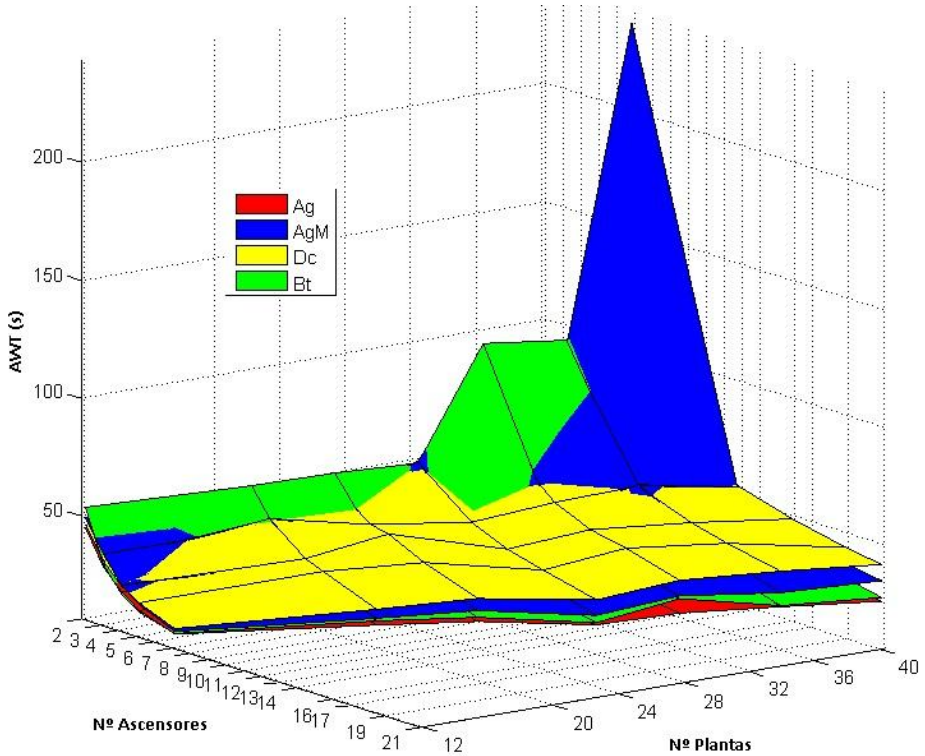


Figura 43: Gráfica 3D del AWT (vista general 2)

De las figuras 42 y 43 se observa que los algoritmos que predominan en cuanto a un mayor tiempo de espera de pasajeros son el algoritmo *Destination Control* y el algoritmo genético modificado debido a que son aquellos que tienen predominancia de color. Además, se puede observar que cuando hay pocos ascensores, la Búsqueda Tabú ofrece también tiempos altos tal y como se ha comentado anteriormente.

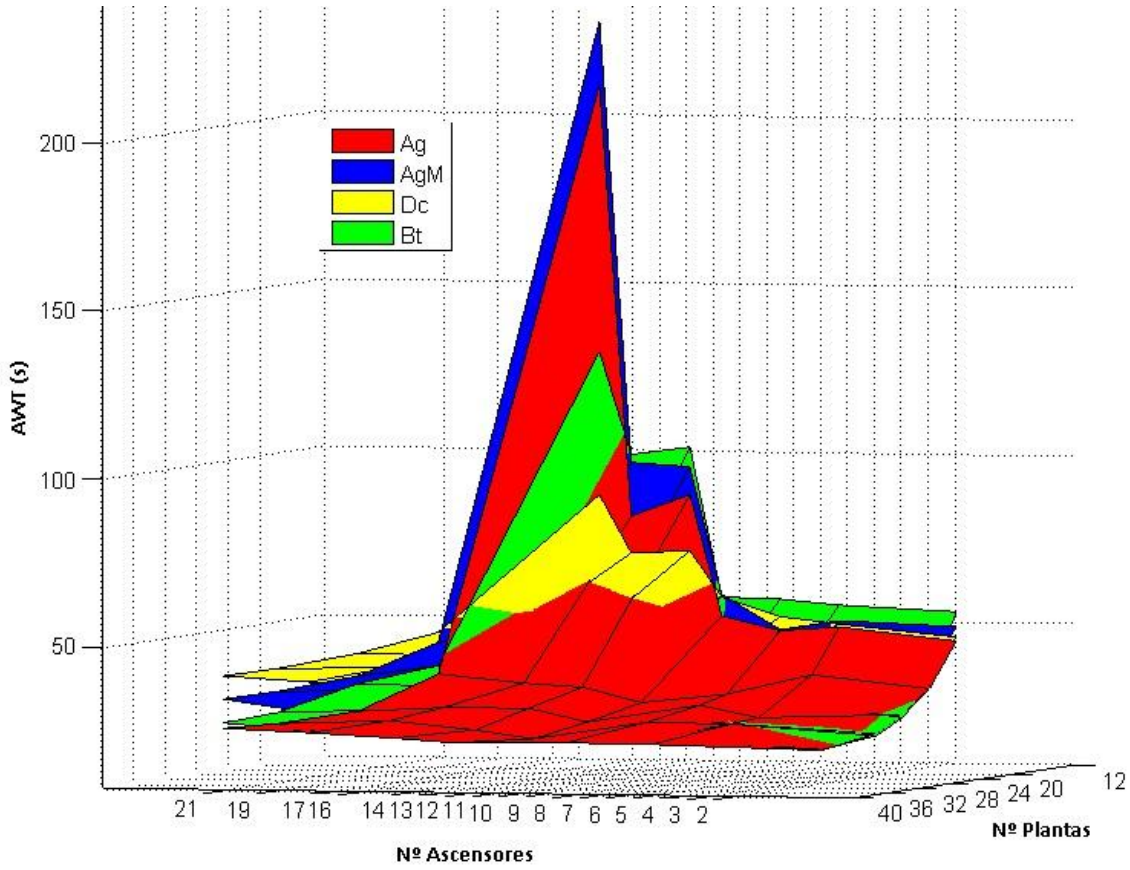


Figura 44: Gráfica 3D del AWT (vista trasera)

En la figura 44 se corrobora que en el pico de tiempo que se produce cuando el edificio tiene 40 plantas y 7 ascensores, los algoritmos genético y genético modificado son los que obtienen peores tiempos medios de espera.

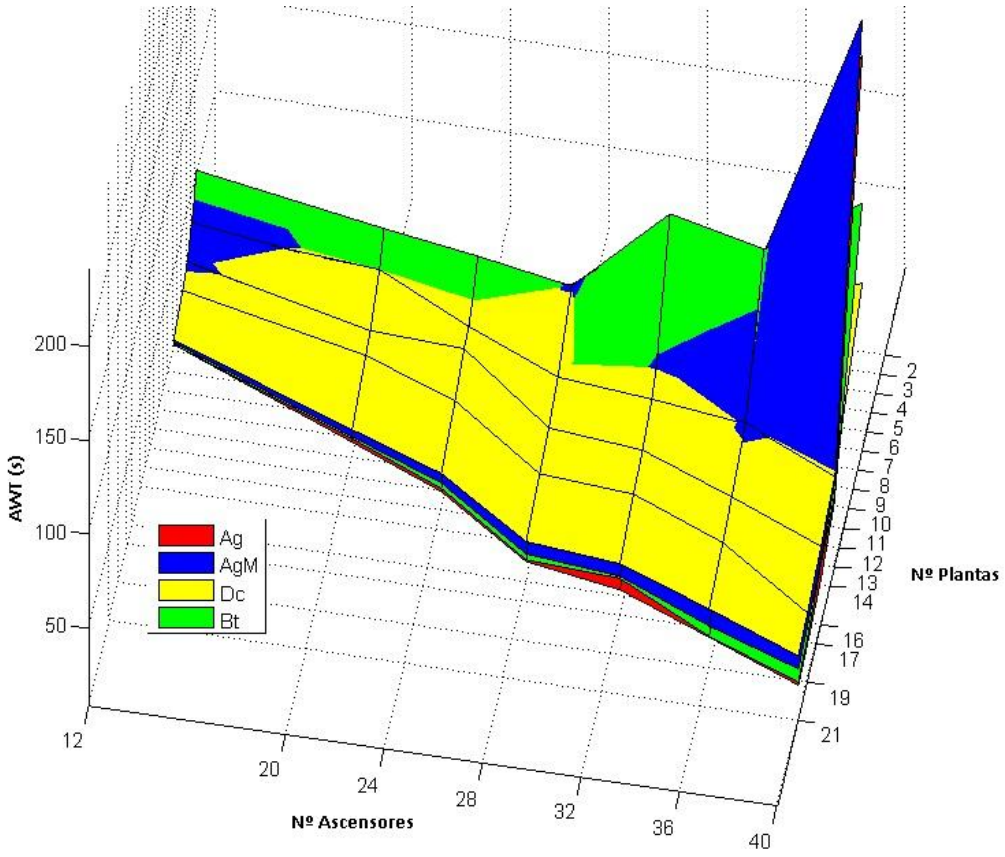


Figura 45: Gráfica 3D del AWT (vista superior)

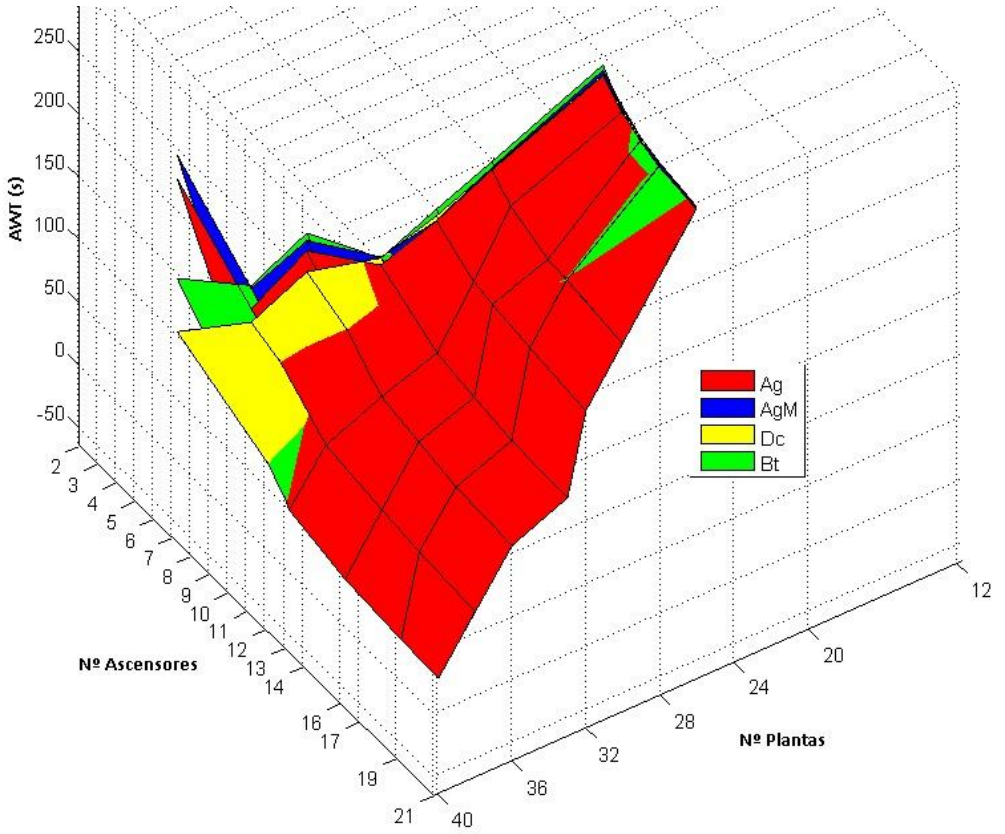


Figura 46: Gráfica 3D del AWT (vista inferior)

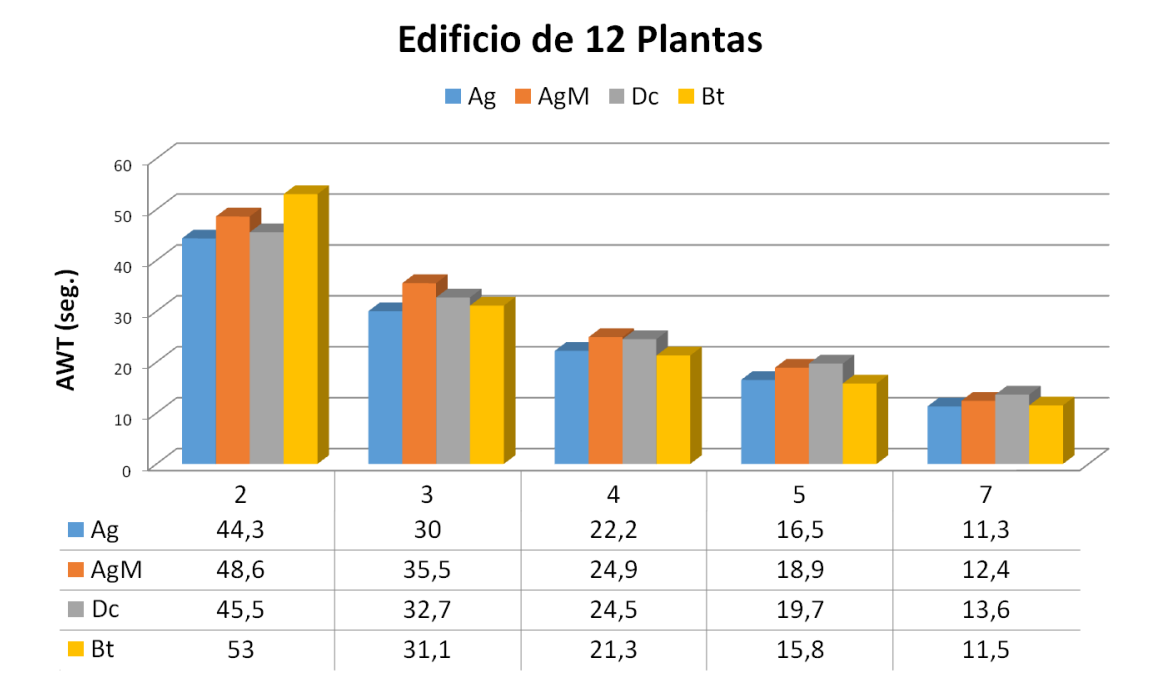
En la figura 45 se corrobora que los algoritmos que ofrecen peores tiempos de espera son el algoritmo *Destination Control* y algoritmo genético modificado debido a la predominancia de color.

En la figura 46 se corrobora que el algoritmo que genera mejores tiempos de espera es el algoritmo genético debido a la predominancia de color. Cabe decir que a pesar de que se produce este hecho, el algoritmo de Búsqueda Tabú se encuentra muy cerca de la curva de superfie del algoritmo genético.

6.5. Análisis de detalle de los tiempos medios de espera

A pesar de que ya se ha obtenido un resultado general del comportamiento de los algoritmos estudiados, se pasa a realizar un análisis para cada uno de los edificios que se han estudiado. En el eje de ordenadas se encuentra el número de ascensores en orden creciente y en el eje de abscisas el tiempo medio de espera (AWT).

12 Plantas

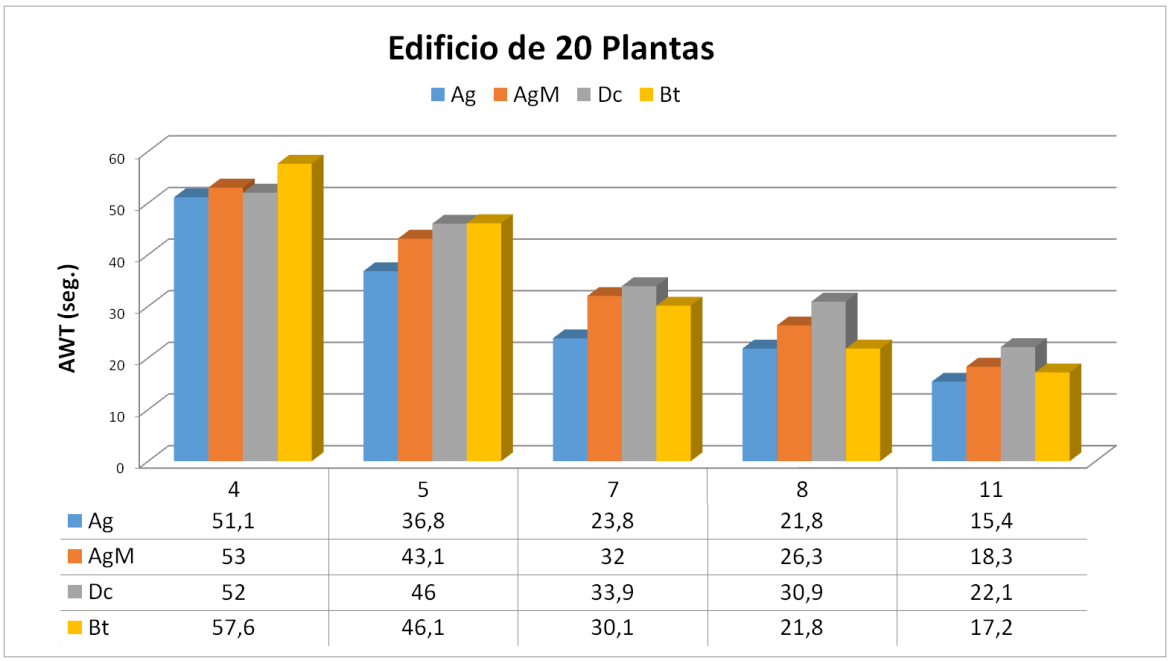


	2	3	4	5	7
Ag	44,3	30	22,2	16,5	11,3
AgM	48,6	35,5	24,9	18,9	12,4
Dc	45,5	32,7	24,5	19,7	13,6
Bt	53	31,1	21,3	15,8	11,5

Figura 47: AWT para edificio de 12 plantas

Para un edificio con 12 plantas se observa en la figura 39 que el tiempo medio de espera no tiene un algoritmo que predomine en cuanto a los resultados que obtiene. Se observa que el algoritmo genético ofrece mejores tiempos para el edificio que tiene 2 y 3 ascensores. Para el edificio con 4 y 5 ascensores, los mejores tiempos los ofrece el algoritmo de Búsqueda Tabú. En el edificio que tiene 7 plantas se observa que tanto el algoritmo genético como el algoritmo de Búsqueda Tabú ofrecen prácticamente el mismo tiempo medio de espera.

20 Plantas

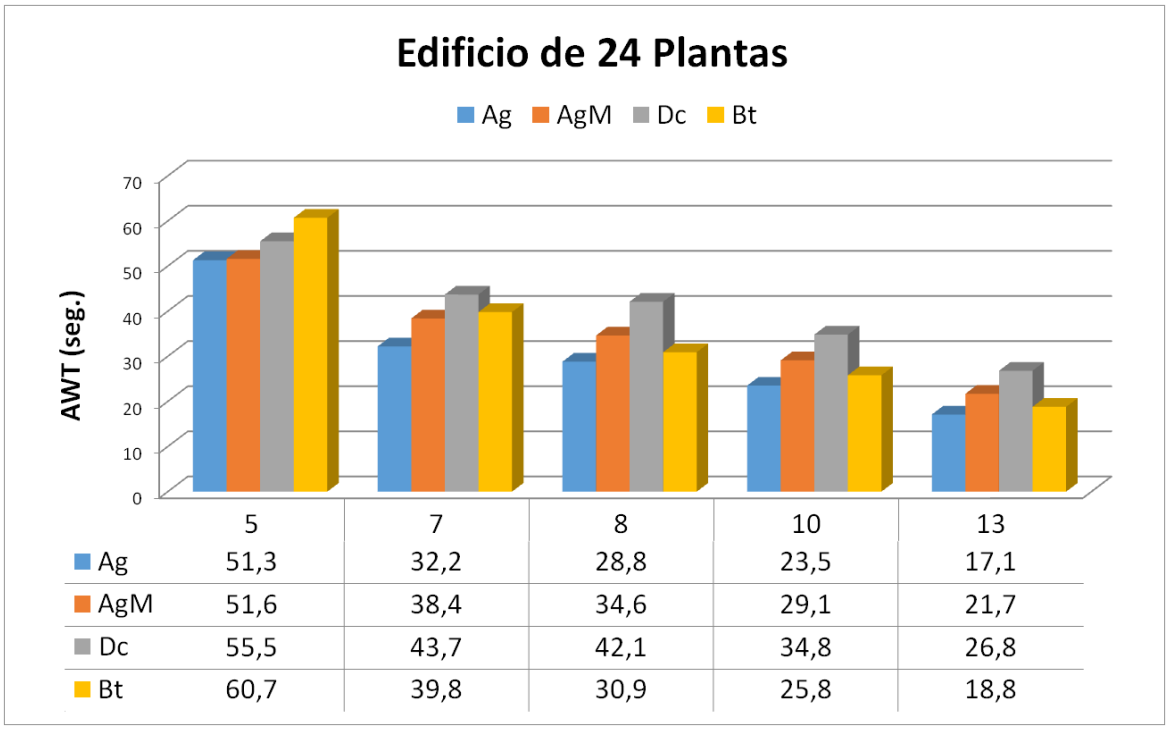


	4	5	7	8	11
Ag	51,1	36,8	23,8	21,8	15,4
AgM	53	43,1	32	26,3	18,3
Dc	52	46	33,9	30,9	22,1
Bt	57,6	46,1	30,1	21,8	17,2

Figura 48: AWT para edificio de 20 plantas

Para un edificio de 20 plantas se observa en la figura 40 que el algoritmo genético consigue obtener los mejores tiempos de espera para todo número de ascensores. Sin embargo, el algoritmo de Búsqueda Tabú demuestra que obtiene tiempos muy parecidos cuando el edificio tiene 8 y 11 ascensores.

24 Plantas

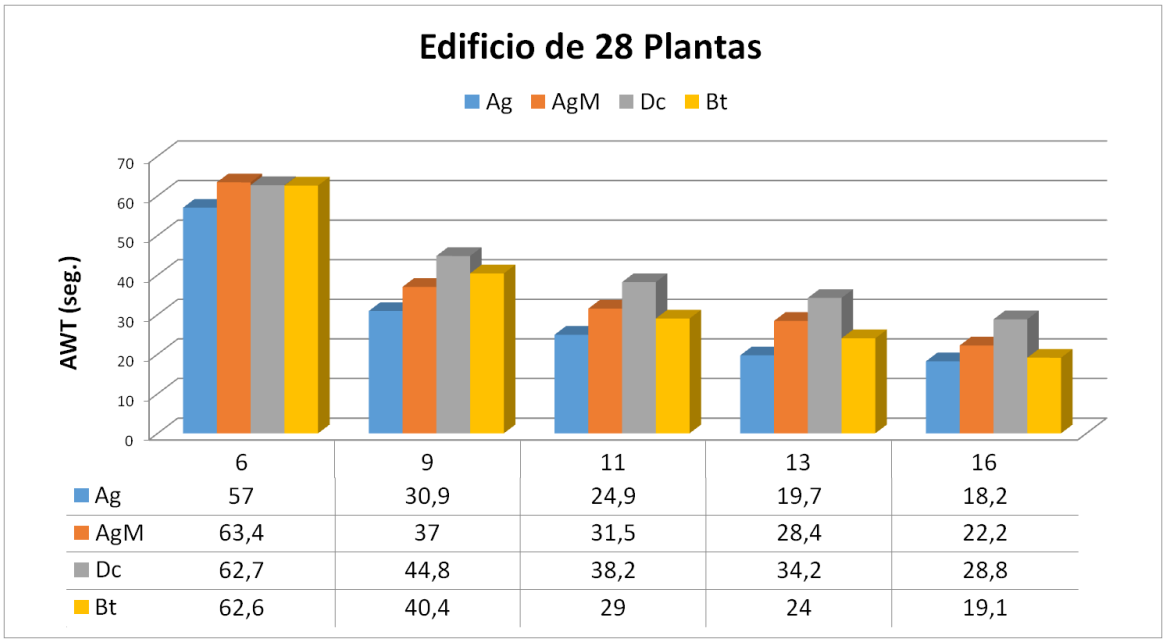


	5	7	8	10	13
Ag	51,3	32,2	28,8	23,5	17,1
AgM	51,6	38,4	34,6	29,1	21,7
Dc	55,5	43,7	42,1	34,8	26,8
Bt	60,7	39,8	30,9	25,8	18,8

Figura 49: AWT para edificio de 24 plantas

Se observa en la figura 41 que para un edificio con 24 plantas el algoritmo genético obtiene los mejores tiempos medios de espera para todo número de ascensores. A pesar de ello, para los edificios con 8, 10 y 13 ascensores el algoritmo de Búsqueda Tabú ofrece unos tiempos muy parejos al mejor, dados por el algoritmo genético.

28 Plantas

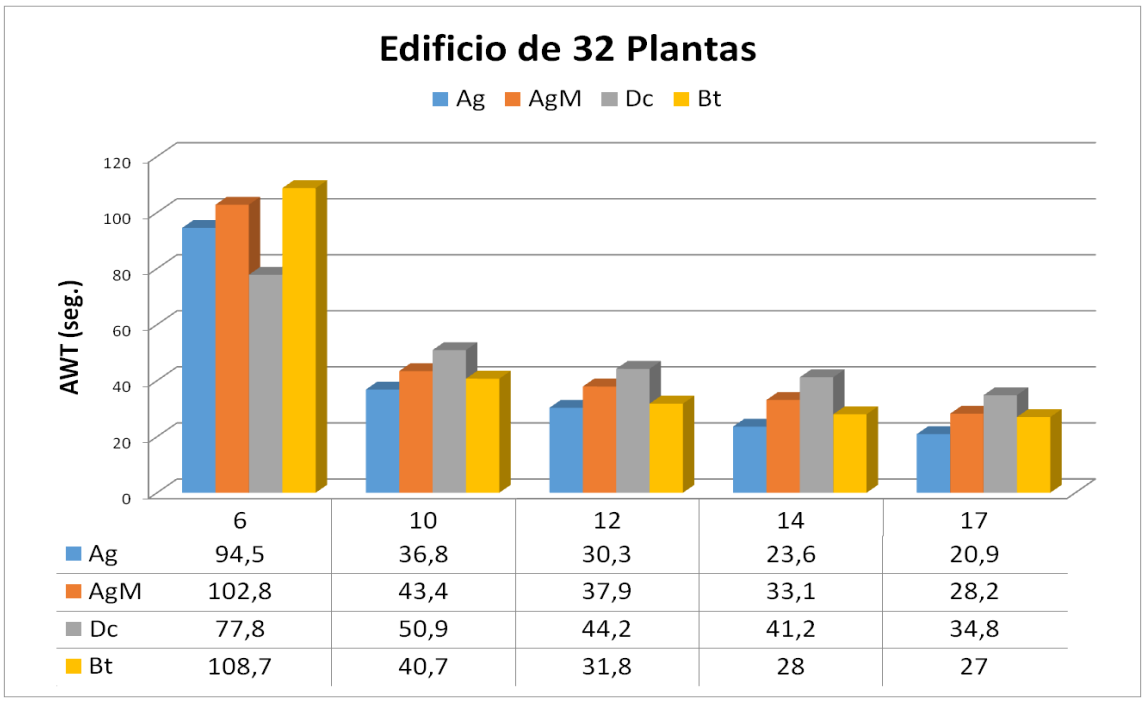


	6	9	11	13	16
Ag	57	30,9	24,9	19,7	18,2
AgM	63,4	37	31,5	28,4	22,2
Dc	62,7	44,8	38,2	34,2	28,8
Bt	62,6	40,4	29	24	19,1

Figura 50: AWT para edificio de 28 plantas

Se observa en la figura 42 que para un edificio con 28 plantas el mejor tiempo medio de espera lo da el algoritmo genético para todo número de ascensores aunque para un edificio con 16 ascensores el algoritmo de Búsqueda Tabú ofrece un resultado muy parecido.

32 Plantas

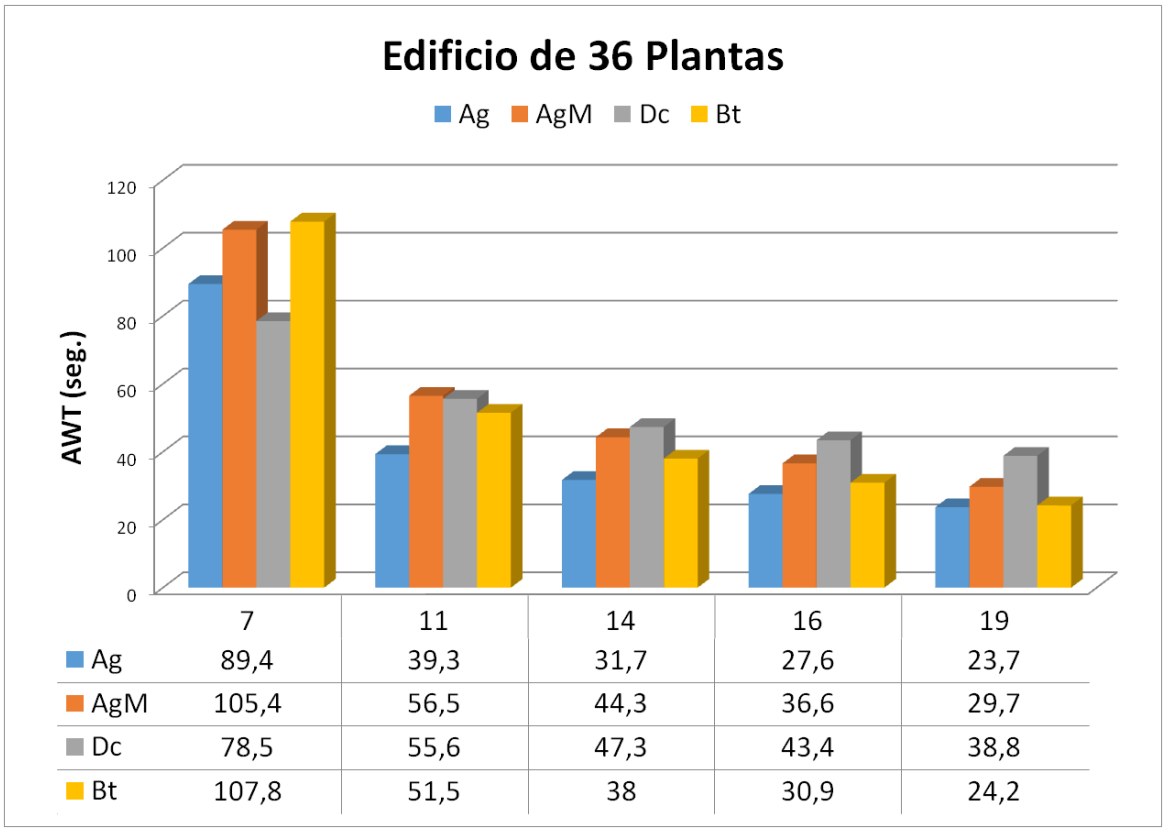


	6	10	12	14	17
Ag	94,5	36,8	30,3	23,6	20,9
AgM	102,8	43,4	37,9	33,1	28,2
Dc	77,8	50,9	44,2	41,2	34,8
Bt	108,7	40,7	31,8	28	27

Figura 51: AWT para edificio de 32 plantas

Para un edificio con 32 plantas se observa en la figura 43 que los mejores tiempos medios los ofrece el algoritmo genético aunque para un edificio con 12 plantas, el algoritmo de Búsqueda Tabú obtiene un buen resultado muy parecido al mejor para ese número de ascensores.

36 Plantas

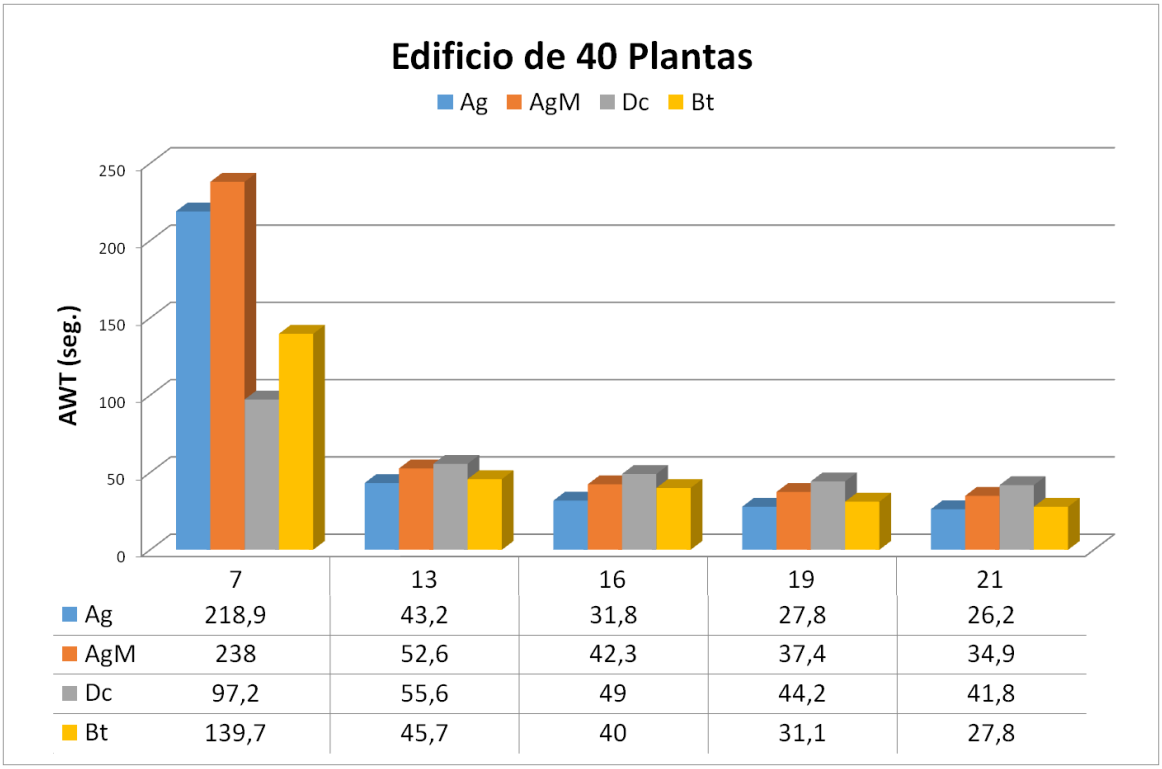


	7	11	14	16	19
Ag	89,4	39,3	31,7	27,6	23,7
AgM	105,4	56,5	44,3	36,6	29,7
Dc	78,5	55,6	47,3	43,4	38,8
Bt	107,8	51,5	38	30,9	24,2

Figura 52: AWT para edificio de 36 plantas

El algoritmo genético ofrece los mejores tiempos medios de espera para todo número de ascensores, excepto para el edificio de 7 plantas que los ofrece el algoritmo *Destination Control*, del edificio de 36 plantas que se muestra en la figura 44. Para edificios con 16 y 19 ascensores, el algoritmo de Búsqueda Tabú obtiene un resultado bastante parecido al mejor.

40 Plantas



	7	13	16	19	21
Ag	218,9	43,2	31,8	27,8	26,2
AgM	238	52,6	42,3	37,4	34,9
Dc	97,2	55,6	49	44,2	41,8
Bt	139,7	45,7	40	31,1	27,8

Figura 53: AWT para edificio de 40 plantas

En último lugar se experimenta con un edificio con 40 plantas. Se observa a través de la figura 45 que ante un elevado número de plantas del edificio, con 7 ascensores el algoritmo que mejor responde es el algoritmo *Destination Control* seguido del algoritmo de Búsqueda Tabú obteniendo un valor para el tiempo medio de espera muy inferior al resto. Para edificios con 13 y 16 ascensores, el algoritmo genético presenta los mejores valores. El algoritmo de Búsqueda Tabú y el algoritmo genético obtienen valores de tiempo parecidos para edificios con 19 y 21 ascensores.

Tras el análisis de cada edificio se puede comprobar lo comentado en el punto anterior. El algoritmo de Búsqueda Tabú obtiene unos tiempos de espera de pasajeros mejores que los que ofrece el algoritmo genético modificado o el implementado en *Elevate*. Sin embargo, se observa que cuando el edificio en cuestión tiene una cantidad pequeña de ascensores, el algoritmo no reacciona bien generándose para todos los edificios el mayor AWT excepto en el edificio de 40 plantas. Esto es debido a que el algoritmo debe en cada iteración recorrer la Lista Tabú en busca de zonas del espacio de soluciones que se hayan visitado para buscar zonas nuevas, esto implica que se ha de emplear un tiempo computacional importante y que si el número de ascensores es pequeño haga que la asignación sea más difícil, es decir, cada vez que el algoritmo asigna un ascensor debe antes haber recorrido la Lista Tabú pero si hay pocos ascensores puede darse el caso de que una vez recorrida aun no haya ascensores disponibles para la asignación.

Tanto el algoritmo de Búsqueda Tabú como el algoritmo genético ofrecen muy buenos resultados de asignación de ascensores a llamadas que hacen que el tiempo medio de espera sea bastante pequeño en comparación con los otros dos algoritmos. A pesar de que el algoritmo genético suele obtener tiempos más pequeños que la Búsqueda Tabú, se puede considerar que ambos responden muy bien ante el problema de asignación de llamadas,

cuando el número de ascensores no es pequeño, ya que la mayor diferencia de tiempo entre ambos métodos es de 12,2 segundos que se da en un edificio con 36 plantas y 11 ascensores.

6.6. Análisis de los tiempos medios de tránsito

Una vez se ha estudiado el objetivo principal, se pasa a analizar el objetivo secundario. A pesar de que minimizar el tiempo medio de espera de pasajeros es la prioridad, no se puede olvidar el tiempo que el pasajero tarda en llegar a su destino una vez ha entrado en el ascensor. De nada serviría obtener un tiempo muy pequeño de espera del pasajero frente al ascensor si luego el ascensor tarda mucho tiempo en llevar al pasajero a su destino. Se muestra a continuación en la tabla 8 el tiempo medio de tránsito de edificios según el número de ascensores y algoritmo de asignación empleado.

Tabla 8: Tiempos medios de tránsito de la experimentación

Nº Plantas	Ascensores	Ag	AgM	Dc	Bt
12	3	43,1	42,7	35,1	35,7
	5	38,5	38,9	28	29,4
20	5	65,3	64,8	46,1	60,1
	8	59,3	58	37,6	45,5
24	7	72	69,9	45,3	65,9
	10	70,3	68	42,3	54,1
28	9	79,8	77	48,4	70,9
	13	77,3	72,8	44,2	58,1
32	10	92,7	89,2	54,7	77,4
	14	81,9	85,9	50,6	64,7
36	11	103,2	100,3	61	89,2
	16	97,7	93	54,6	70,8
40	13	110,8	105,5	63,4	93,8
	19	102,1	96,8	57,1	76,5

La tabla 9 recoge además de la diferencia de tiempos entre métodos y el porcentaje de mejora/empeoramiento del tiempo medio de tránsito del algoritmo de Búsqueda Tabú con respecto a los otros algoritmos tal y como sucede en la tabla 7. En rojo se encuentran aquellos porcentajes de empeoramiento que el algoritmo de Búsqueda Tabú ofrece contra el algoritmo con el que se compara. En cambio, aquellos porcentajes verdes indican que el algoritmo de Búsqueda Tabú ofrece un tiempo mejor que el algoritmo con el que se compara.

A modo de ejemplo, para un edificio con 20 plantas y 8 ascensores se encuentra que el algoritmo de Búsqueda Tabú obtiene un tiempo de 13,8 segundos mejor que el algoritmo genético, este hecho supone una mejora del 23,27% del algoritmo de Búsqueda Tabú frente al algoritmo genético para ese número de plantas y ascensores.

Tabla 9: Comparativa porcentual de de los tiempos medios de tránsito

Número Plantas	Número Ascensores	Ag-Bt (s)	Ag-Bt %	AgM-Bt (s)	AgM-Bt %	Dc-Bt (s)	Dc-Bt %
12	3	7,4	17,17	7,00	16,39	-0,60	-1,71
	5	9,1	23,64	9,50	24,42	-1,40	-5,00
20	5	5,2	7,96	4,70	7,25	-14,00	-30,37
	8	13,8	23,27	12,50	21,55	-7,90	-21,01
24	7	6,1	8,47	4,00	5,72	-20,60	-45,47
	10	16,2	23,04	13,90	20,44	-11,80	-27,90
28	9	8,9	11,15	6,10	7,92	-22,50	-46,49
	13	19,2	24,84	14,70	20,19	-13,90	-31,45
32	10	15,3	16,50	11,80	13,23	-22,70	-41,50
	14	17,2	21,00	21,20	24,68	-14,10	-27,87
36	11	14	13,57	11,10	11,07	-28,20	-46,23
	16	26,9	27,53	22,20	23,87	-16,20	-29,67
40	13	17	15,34	11,70	11,09	-30,40	-47,95
	19	25,6	25,07	20,30	20,97	-19,40	-33,98

Se observa en la figura 48 la evolución del tiempo medio de tránsito que ofrece cada algoritmo con respecto a la evolución del número de ascensores del edificio.

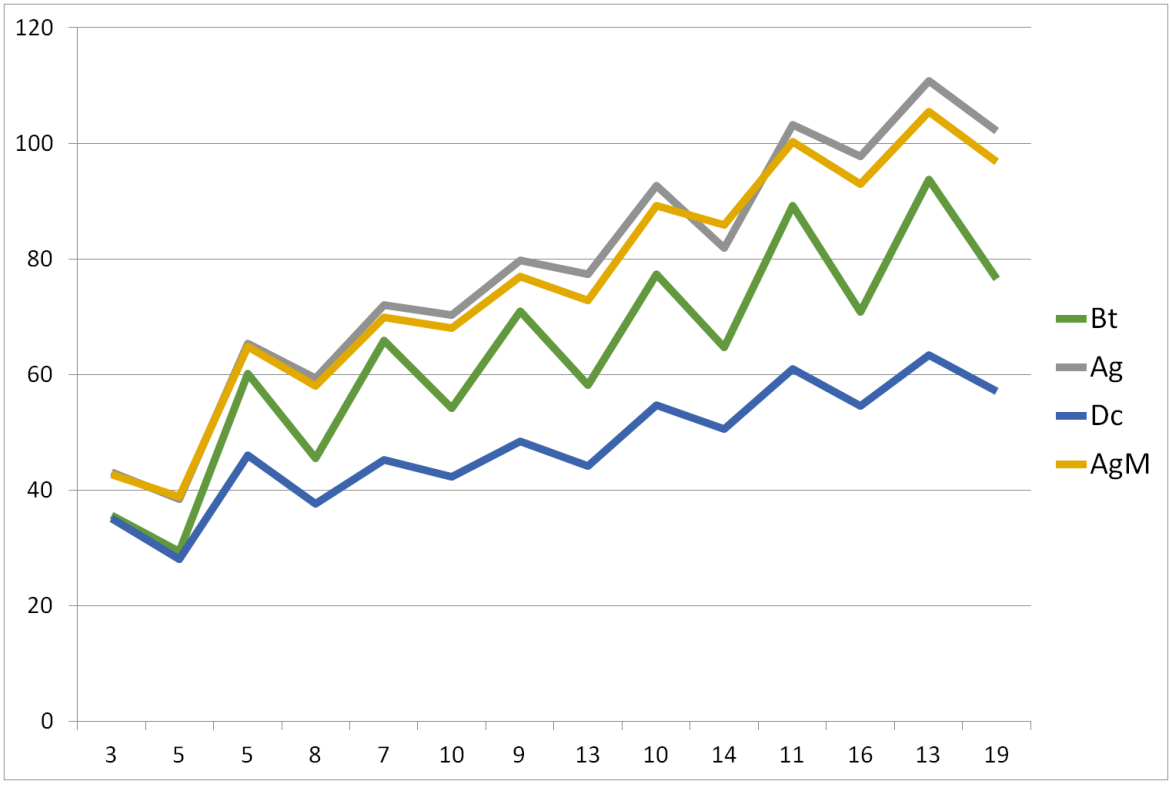


Figura 54: Gráfica 2D para ATT

Se puede observar que los mejores tiempos medios de tránsito los ofrece el algoritmo *Destination Control* en todas las ocasiones. Esto es debido a que este algoritmo se implementa con el objetivo de no minimizar solo el tiempo medio de espera de los pasajeros sino también

el tiempo medio de tránsito. Si analizamos los mejores algoritmos, en cuanto al tiempo medio de espera se refiere, nos encontramos con que el algoritmo de Búsqueda Tabú supera al algoritmo genético en todas las ocasiones analizadas.

De igual modo que en el punto 6.4, se ha realizado una gráfica 3D mediante el software *Matlab* en la que en el eje vertical se representa el tiempo medio de tránsito y en los ejes horizontales el número de plantas y el número de ascensores. Se puede observar mediante colores (Amarillo: *Destination Control*, Verde: Búsqueda Tabú, Azul: Genético Modificado y Rojo: Genético) la evolución del tiempo medio de tránsito en la figura 55.

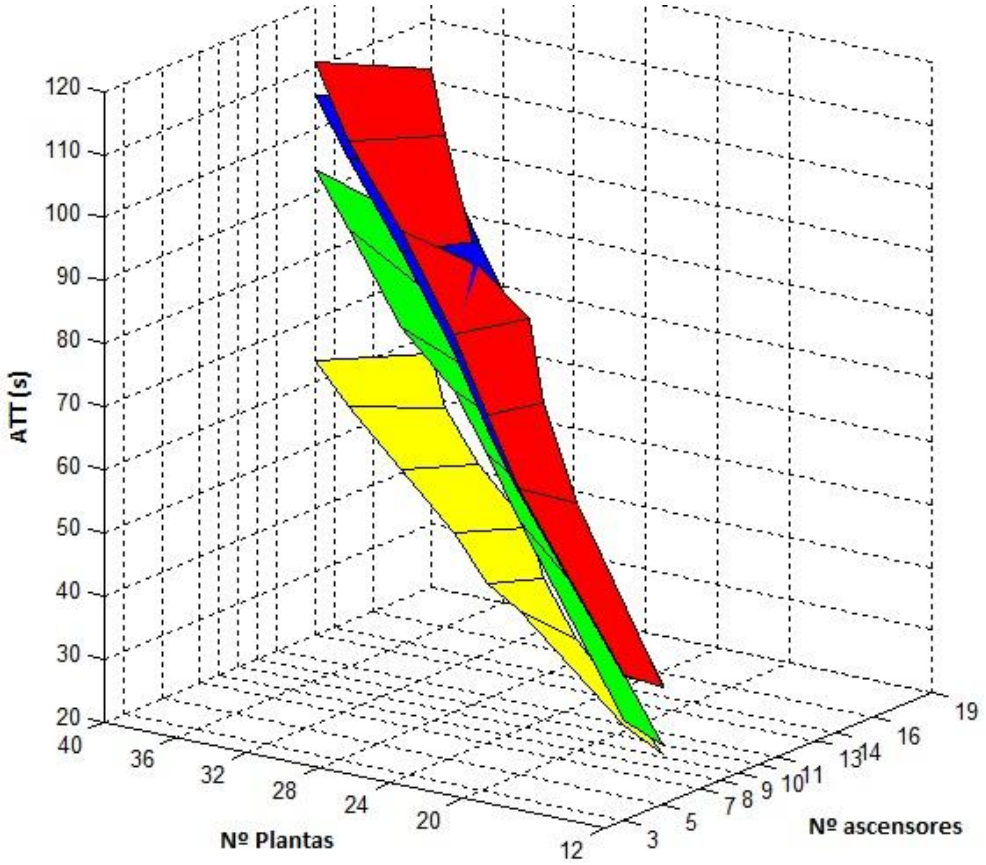


Figura 55: Gráfica 3D del ATT

Se observa que tal y como se ha comentado con anterioridad, el algoritmo que mejor tiempo medio de tránsito ofrece es el algoritmo implementado por *Elevate* siendo el algoritmo de Búsqueda Tabú el segundo mejor método para esta variable de estudio.

6.7. Análisis del tiempo total de viaje

Se va a analizar en este apartado el tiempo total de viaje de los pasajeros del edificio con el objeto de comprobar qué algoritmo de los estudiados actúa mejor frente a esta variable.

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, esto es, el tiempo que transcurre desde que un pasajero llama a un ascensor hasta que llega a su destino.

Se muestra en la tabla 10 el tiempo total de viaje para los casos anteriormente estudiados, según el número de plantas del edificio y número de ascensores.

Tabla 10: Tiempos totales de viaje

Nº Plantas	Ascensores	Ag	AgM	Dc	Bt
12	3	73,1	78,2	67,8	66,8
	5	55	57,8	47,7	45,2
20	5	102,1	107,9	92,1	106,2
	8	81,1	84,3	68,5	67,3
24	7	104,2	108,3	89	105,7
	10	93,8	97,1	77,1	79,9
28	9	110,7	114	93,2	111,3
	13	97	101,2	78,4	82,1
32	10	129,5	132,6	105,6	118,1
	14	105,5	119	91,8	92,7
36	11	142,5	156,8	116,6	140,7
	16	125,3	129,6	98	101,7
40	13	154	158,1	119	139,5
	19	129,9	134,2	101,3	107,6

Se muestra en la figura 49 la evolución del tiempo total de viaje según se incrementa el número de plantas del edificio y por tanto el número de ascensores.

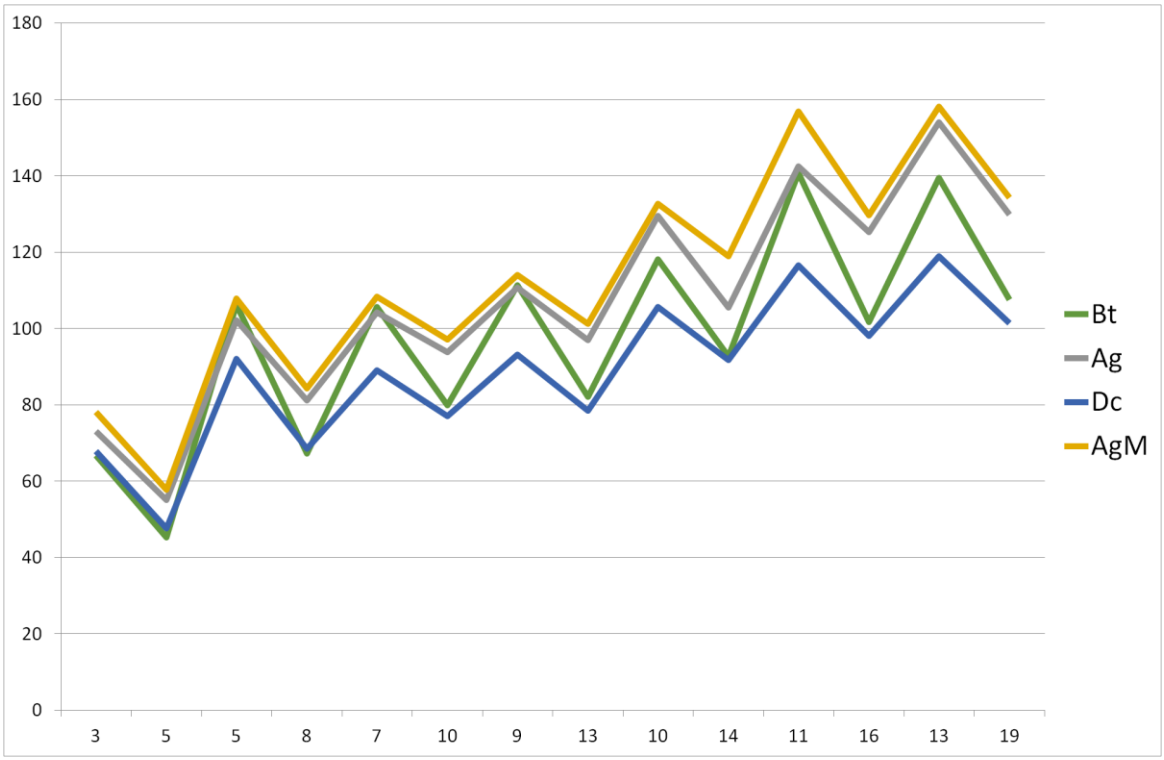


Figura 56: Gráfica 2D para el tiempo total de viaje

Se extrae de la gráfica que el algoritmo genético modificado y el algoritmo genético se encuentran en todo momento por encima del algoritmo de Búsqueda Tabú y el algoritmo *Destination Control* implementado por *Elevate*. Los mejores tiempos totales de viaje se consiguen mediante el algoritmo *Destination Control*, aunque el algoritmo de Búsqueda Tabú obtiene tiempos muy parejos a este.

7. CONCLUSIONES

Debido al incremento del precio del suelo, el tráfico vertical adquiere una gran relevancia en la actualidad ya que se pretende aprovechar todo el espacio disponible lo máximo posible. Esto hace que el número de plantas de los edificios que se construyen sea mucho mayor que antiguamente. Por ello, la importancia del modo de transportar tanto personas como mercancías en el edificio adquiere un peso muy importante. Se usan ascensores para realizar dichas labores, pero debido al incremento del número de plantas de los edificios, la tecnología que se emplea para dicho transporte debe avanzar permitiendo un desplazamiento más rápido y seguro.

En este aspecto, el estado del arte respecto al transporte vertical ha hecho que la evolución del ascensor a lo largo de la historia haya mejorado tanto en la forma de transporte, más silenciosa y rápida, como en la estética ya que se ha integrado como un recurso arquitectónico de decoración más. Además, también el estudio de los patrones de tráfico existentes ha supuesto una mejora en dicho estado del arte, de modo que los algoritmos para la asignación de llamadas a ascensores puedan también mejorar. En la actualidad, el tráfico conocido como *Interfloor* tiene un gran peso debido a la flexibilidad de horarios y la prohibición de fumar ya que los trabajadores entran y salen del edificio en más ocasiones con respecto al tráfico que se tenía antiguamente en el que una vez el trabajador comenzaba su horario de trabajo (*Uppeak*) ya no salía del edificio hasta terminar su jornada laboral (*Downpeak*) o en la hora del almuerzo (*Lunch Peak*). Es por ello por lo que en este trabajo las simulaciones se realizan teniéndose en cuenta que el 30% de los trabajadores a lo largo del día entran, un 30% salen del edificio y un 40% se desplaza entre plantas del edificio. Se consigue de este modo una simulación fehaciente a la realidad del día a día de un edificio de oficinas.

Ante el incremento del número de plantas de los edificios, nace la necesidad de buscar métodos que hagan que el transporte vertical sea eficiente ya que se debe lograr que el incremento de altura del edificio no sea un impedimento debido a un transporte lento. Es por ello por lo que se inventa el ascensor con arquitectura *Double* Deck. El uso de ascensores con arquitectura *Double Deck* permite la movilidad de una mayor población por cada viaje que el ascensor realiza, ya que emplea 2 cabinas acopladas y no 1 solo como los ascensores tradicionales. Sin embargo, se requiere de un algoritmo de asignación de llamadas lo suficientemente eficiente como para que se sirvan todas las llamadas del edificio de modo 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. El algoritmo de asignación objeto de estudio es la Búsqueda Tabú con la que se pretende abarcar un amplio abanico de soluciones en busca de una buena solución para el problema.

El algoritmo de Búsqueda Tabú emplea un listado, denominado Lista Tabú, en el que se guardan posibles soluciones de asignación de llamadas a ascensores. Para cada una de las soluciones se calcula la función objetivo que no es más que la suma de las distancias que hay entre los ascensores que se van a emplear y las llamadas a ascensores de las plantas del edificio. Una vez ha finalizado la búsqueda de soluciones se escoge aquella asignación de llamadas a ascensores que ha obtenido la menor función objetivo y por tanto es la solución

que va a generar que el tiempo medio de espera de los pasajeros frente a la puerta del ascensor sea menor. Al emplearse ascensores con arquitectura *Double Deck*, una vez se han asignado los ascensores a las plantas del edificio hay que escoger si la llamada es acogida por la cabina superior o inferior del ascensor. Dicha elección se realiza en función de si la llamada proviene de la planta principal del edificio o de una planta impar (cabina inferior) o si proviene de la última planta del edificio o de una planta par (cabina superior). Además, en caso de que se exceda la carga máxima permitida por parte de alguna de las cabinas la llamada se desasigna de ese ascensor.

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 simulaciones del edificio. Para la conexión entre ambos programas se requiere la previa instalación de la interfaz de desarrollador para la cual se recomienda encarecidamente el empleo de la versión 2008 del software *Microsoft Visual Studio* pues con otras versiones se ha probado que no es compatible.

Para las simulaciones que se han realizado se han empleado 7 edificios con diferente número de plantas (12, 20, 24, 28, 32, 36 y 40 plantas) en los que se ha variado el número de ascensores. Se han elaborado simulaciones de 2 horas de duración para un tipo de tráfico especial para ascensores con arquitectura *Double Deck* ya que los pasajeros van de plantas pares a pares y de plantas impares a impares como se requiere en este tipo de ascensor.

Se han estudiado varios tipos de algoritmos de asignación de llamadas a ascensores resultando que el algoritmo de Búsqueda Tabú, ante un número de ascensores por edificio lo suficientemente grande, esto es, sin contar la primera casuística de ascensores estudiada para cada edificio del apartado 6, ofrece unos tiempos medios de espera que se encuentran un 85,7 % de las ocasiones por debajo de 45 segundos y un 96,4 % de las ocasiones por debajo de 50 segundos.

Para edificios con pocos ascensores se demuestra que el algoritmo genético ofrece mejores tiempos medios de espera de pasajeros. Pero según se incrementa el número de ascensores del edificio, el algoritmo de Búsqueda Tabú va mejorando llegando a obtener una calificación del grado de servicio de Bueno/Excelente según el apartado 3.2 de evaluación de la calidad del servicio.

Se demuestra en el apartado 6.6 que el algoritmo de Búsqueda Tabú ofrece muy buenos resultados en cuanto a tiempo medio de tránsito se refiere, siendo los mejores resultados tras los obtenidos por el algoritmo *Destination Control* implementado por *Elevate.*

Se aprecia en el apartado 6.7 el tiempo total de viaje, es decir, la suma del tiempo de espera de los pasajeros frente al ascensor más el tiempo que el pasajero tarda en llegar a su destino una vez ha ingresado en el ascensor. Ante esta variable se observa que tanto el algoritmo genético como el algoritmo genético modificado obtienen los peores tiempos. En cambio, el algoritmo *Destination Control* y el algoritmo de Búsqueda Tabú ofrecen mejores tiempos totales de viaje. Los mejores tiempos de este apartado los obtiene el algoritmo *Destination Control* implementado por *Elevate* aunque la Búsqueda Tabú se encuentra ante valores muy parecidos.

A pesar de que el algoritmo de Búsqueda Tabú se encuentra programado de forma que se optimice el tiempo medio de espera de los pasajeros, se observa que obtiene muy buenos tiempos medios de tránsito. Gracias a esta doble optimización que se produce, a pesar de minimizar de forma principal tan solo el AWT, se consigue que el tiempo total de viaje de los pasajeros se encuentre muy cercano al tiempo total de viaje que ofrece el algoritmo *Destination Control*. Cabe recordar que este último algoritmo se encuentra optimizado por *Elevate* para minimizar tanto el tiempo medio de espera como el tiempo medio de tránsito.

Aunque el tiempo medio de espera del algoritmo implementado por *Elevate* y el algoritmo de Búsqueda Tabú es muy parecido en las simulaciones realizadas, hay que reseñar que el primero de los algoritmos consigue ese bajo tiempo a base de reducir mucho el tiempo de tránsito, dejando crecer el tiempo medio de espera. En cambio, el algoritmo de Búsqueda Tabú consigue tiempos bajos tanto para el tiempo medio de espera como para el tiempo medio de tránsito, como se demuestra en los apartados 6.4 y 6.6. Se demuestra que un pasajero se encuentra más cómodo una vez ha entrado al ascensor, prestándole menos importancia a la duración del trayecto hacia el destino. Es por ello por lo que en este aspecto es mejor el algoritmo de Búsqueda Tabú ya que minimiza en gran parte el tiempo medio de espera del pasajero frente al ascensor.

Se puede concluir con que tras las simulaciones realizadas a través de *Elevate* mediante la implementación en *Microsoft Visual Studio* para ascensores con arquitectura *Double Deck*, el algoritmo de Búsqueda Tabú bajo el que se centra el estudio cumple las especificaciones del problema y obtiene buenos resultados tanto para el objetivo principal de minimizar el tiempo medio de espera como para el objetivo secundario de minimizar el tiempo medio de tránsito de pasajeros.

8. REFERENCIAS

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

[2] 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.

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

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

[5] Mesa J. (2015). Sistemas de asignación de llamadas para ascensores tipo Double Deck basados en algoritmos genéticos, Trabajo Fin de Grado.

[6] Grupo ascensores Enor. (2016). http://www.enor.es/catalogo_otea_ES.pdf Último acceso: 26/06/2016

[7] Un poco de historia. (2016). http://www.silcon.com.ar/un_poco_de_historia.htm Último acceso: 26/06/2016

[8] Tipos de ascensores para las edificaciones. (2013). http://www.t3mascensores.com/tipos-de-ascensores-para-las-edificaciones/ Último acceso: 26/06/2016

[9] Ingesor. (2016). http://www.ingesor.com/inicio/ascensores/ascensores-electromec%C3%A1nicos/ Último acceso: 26/06/2016

[9] La nueva era de los ascensores revoluciona la construcción de mediana y gran altura. (2016). http://www.urban-hub.com/es/ideas/la-nueva-era-de-los-ascensores-revoluciona-la-construccion-de-mediana-y-gran-altura/ Último acceso: 26/06/2016

[10] Torre Picasso. (2016). http://www.torre-picasso.com/es/torre-picasso/el-edificio Último acceso: 26/06/2016

[11] The Shard. (2016). https://www.londres.es/the-shard Último acceso: 26/06/2016

[12] Otis en la Torre Burj Khalifa. (2016). http://www.otis.com/site/es-esl/pages/TorreBurjKhalifa.aspx Último acceso: 26/06/2016

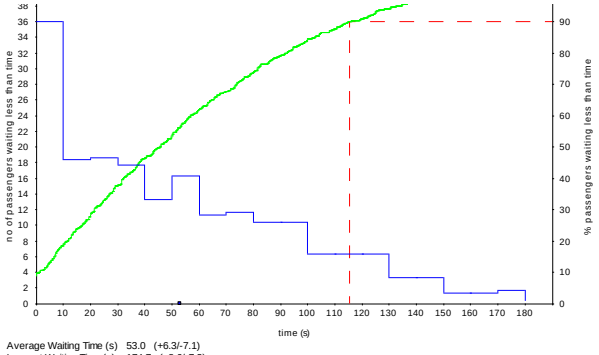
ANEXO A: GRÁFICAS EXPERIMENTALES

Este Anexo A contiene todas las gráficas del tiempo medio de espera de los pasajeros frente al ascensor y el tiempo medio de tránsito que *Elevate 8* ofrece para cada una de las simulaciones realizadas. Se muestran las gráficas agrupadas según el algoritmo empleado para la asignación de llamadas a ascensores. Cada gráfica se encuentra acompañada por el número de plantas del edificio, la cantidad de ascensores y el valor de la semilla aleatoria.

Tiempo medio de espera (AWT)

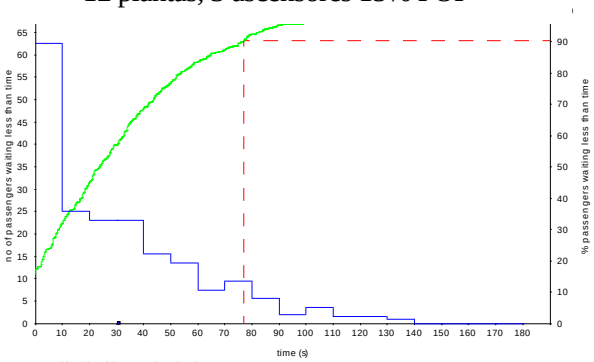
Búsqueda Tabú

12 plantas, 2 ascensores 12,5% POP



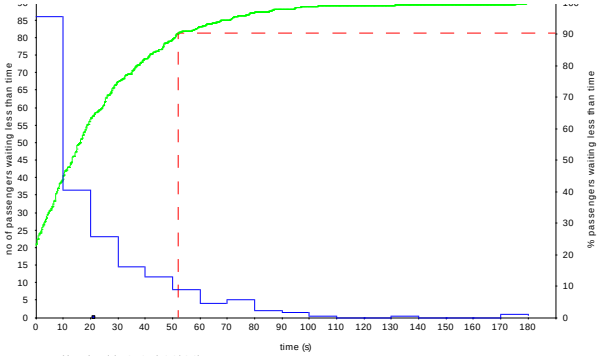
Average Waiting Time (s) 53.0 (+6.3/-7.1)
Longest Waiting Time (s) 174.7 (+8.6/-7.3)

12 plantas, 3 ascensores 15% POP



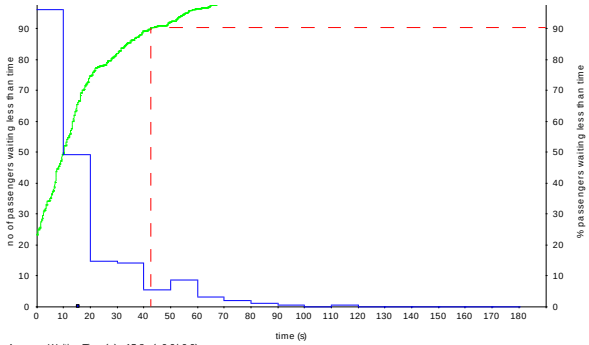
Average Waiting Time (s) 31.1 (+1.3/-1.3)
Longest Waiting Time (s) 131.3 (+0.6/-0.6)

12 plantas, 4 ascensores 17,5% POP



Average Waiting Time (s) 21.3 (+2.8/-2.8)
Longest Waiting Time (s) 139.5 (+44.7/-44.7)

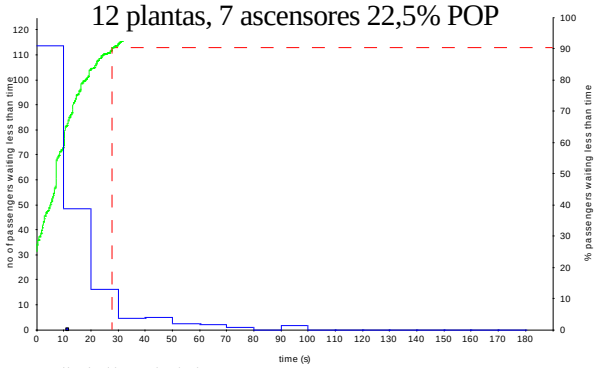
12 plantas, 5 ascensores 20 % POP



Average Waiting Time (s) 15.8 (+0.0/-0.0)
Longest Waiting Time (s) 105.3 (+12.7/-12.7)

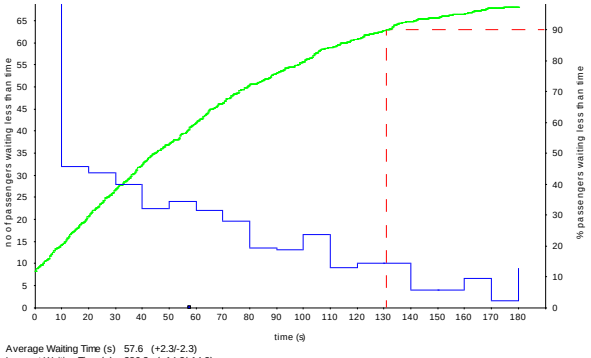
7 No. 1000 kg elevators @ 2.50 m/s
Average o

12 plantas, 7 ascensores 22,5% POP



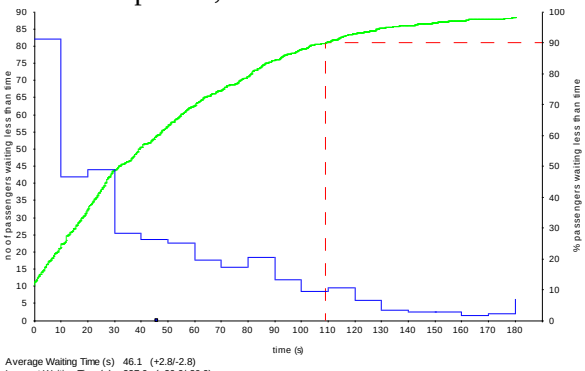
Average Waiting Time (s) 11.5 (+0.2/-0.2)
Longest Waiting Time (s) 86.3 (+12.3/-12.3)

20 plantas, 4 ascensores 12,5% POP



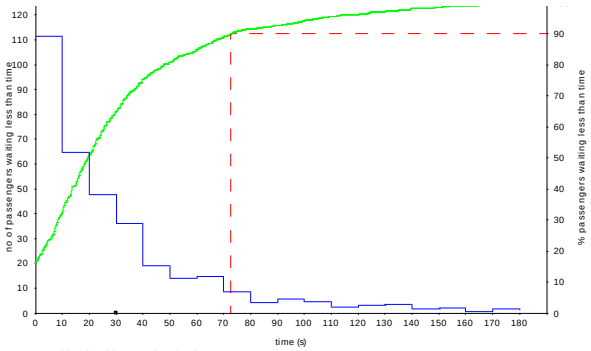
Average Waiting Time (s) 57.6 (+2.3/-2.3)
Longest Waiting Time (s) 286.3 (+14.2/-14.2)

20 plantas, 5 ascensores 15% POP



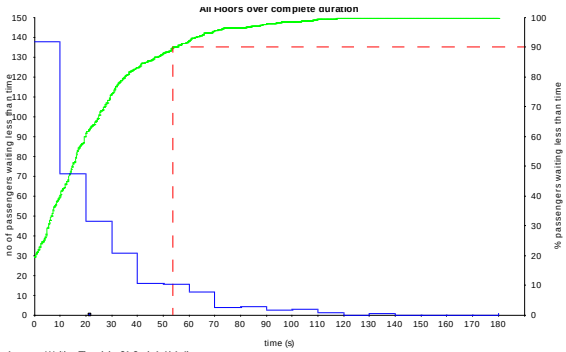
Average Waiting Time (s) 46.1 (+2.8/-2.8)
Longest Waiting Time (s) 227.0 (+30.9/-30.9)

20 plantas, 7 ascensores 17,5% POP



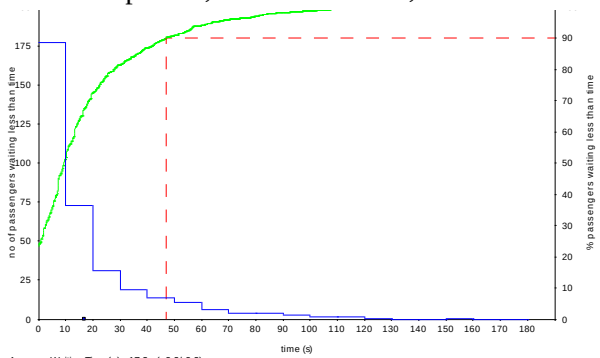
Average Waiting Time (s) 30.1 (+0.9/-0.9)
Longest Waiting Time (s) 190.4 (+9.4/-9.4)

20 plantas, 8 ascensores 20% POP



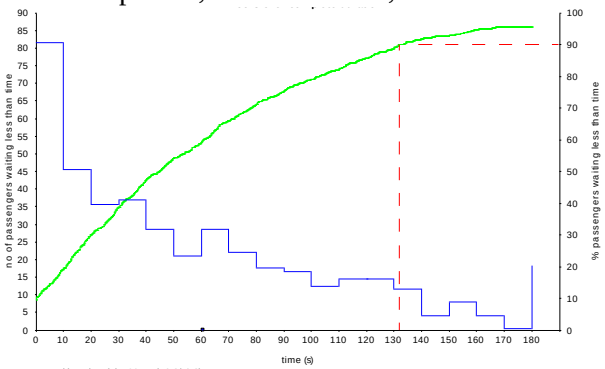
Average Waiting Time (s) 21.8 (+1.4/-1.4)
Longest Waiting Time (s) 163.4 (+51.5/-51.5)

20 plantas, 11 ascensores 22,5% POP



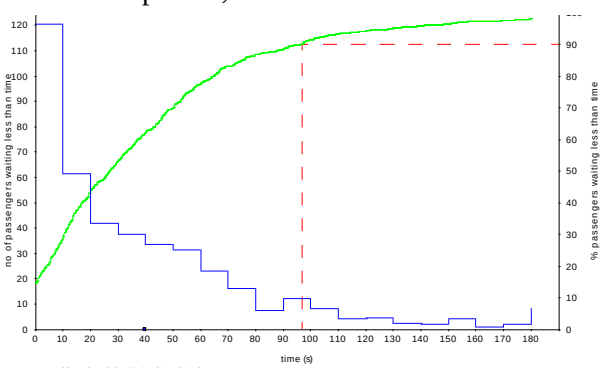
Average Waiting Time (s) 17.2 (+0.9/-0.9)
Longest Waiting Time (s) 130.0 (+22.6/-22.6)

24 plantas, 5 ascensores 12,5% POP



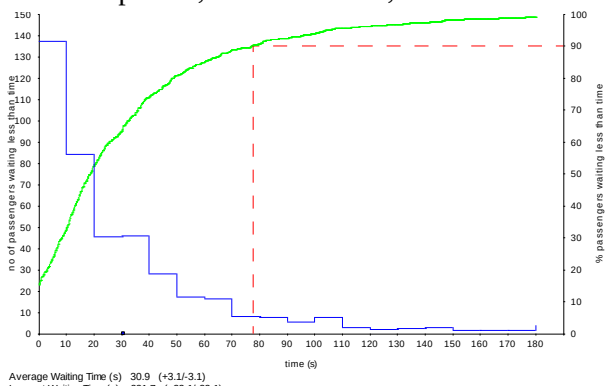
Average Waiting Time (s) 60.7 (+2.3/-2.3)
Longest Waiting Time (s) 352.6 (+77.7/-77.7)

24 plantas, 7 ascensores 15% POP



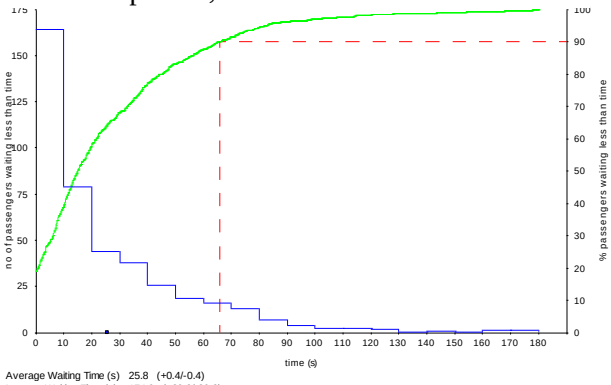
Average Waiting Time (s) 39.8 (+0.4/-0.4)
Longest Waiting Time (s) 282.8 (+19.1/-19.1)

24 plantas, 8 ascensores 17,5% POP



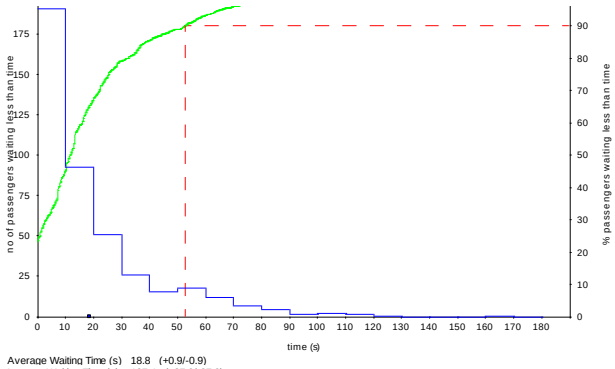
Average Waiting Time (s) 30.9 (+3.1/-3.1)
Longest Waiting Time (s) 221.7 (+22.1/-22.1)

24 plantas, 10 ascensores 20% POP



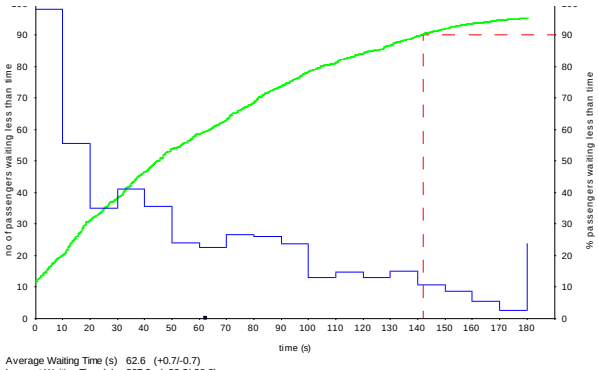
Average Waiting Time (s) 25.8 (+0.4/-0.4)
Longest Waiting Time (s) 174.0 (+29.2/-29.2)

24 plantas, 13 ascensores 22,5% POP



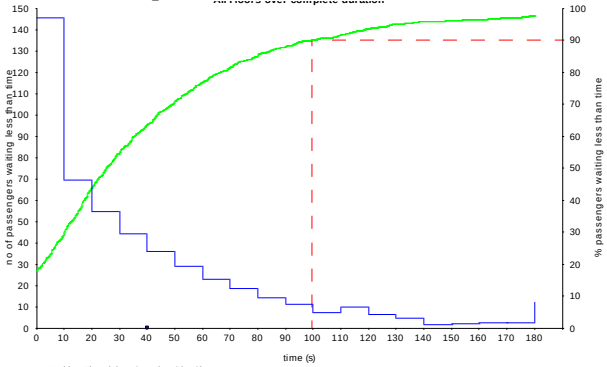
Average Waiting Time (s) 18.8 (+0.9/-0.9)
Longest Waiting Time (s) 137.4 (+27.0/-27.0)

28 plantas, 6 ascensores 12,5% POP



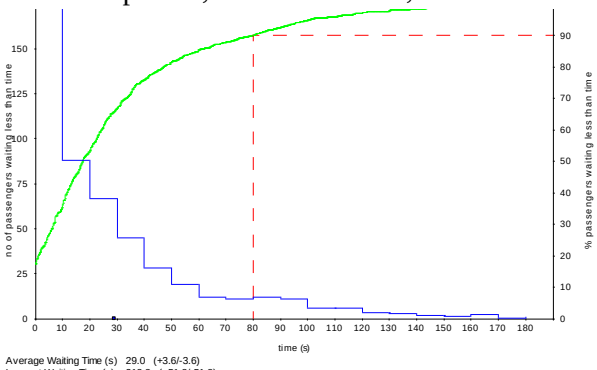
Average Waiting Time (s) 62.6 (+0.7/-0.7)
Longest Waiting Time (s) 387.3 (+30.2/-30.2)

28 plantas, 9 ascensores 15% POP



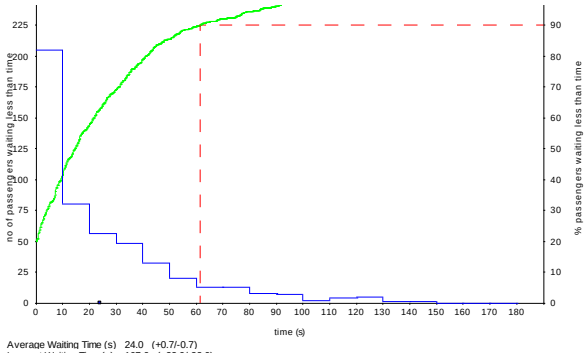
Average Waiting Time (s) 40.4 (+4.9/-4.9)
Longest Waiting Time (s) 291.1 (+31.5/-31.5)

28 plantas, 11 ascensores 17,5% POP



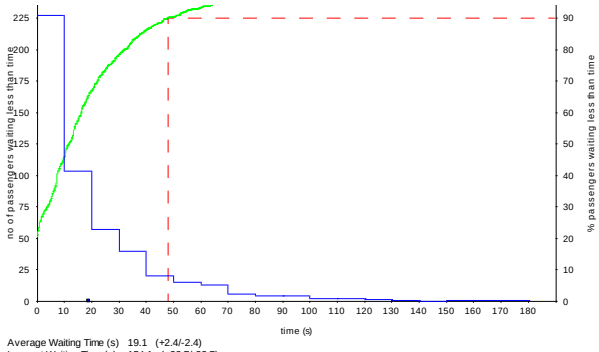
Average Waiting Time (s) 29.0 (+3.6/-3.6)
Longest Waiting Time (s) 218.8 (+51.8/-51.8)

28 plantas, 13 ascensores 20% POP



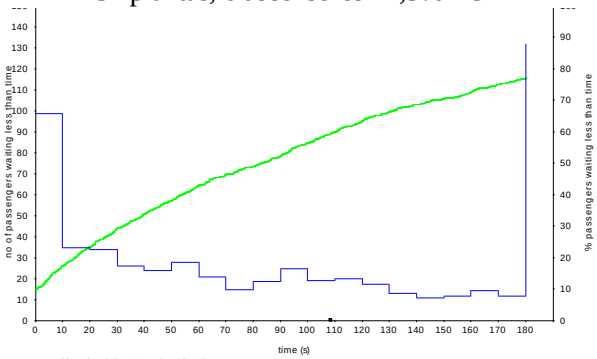
Average Waiting Time (s) 24.0 (+0.7/-0.7)
Longest Waiting Time (s) 167.0 (+22.9/-22.9)

28 plantas, 16 ascensores 22,5% POP



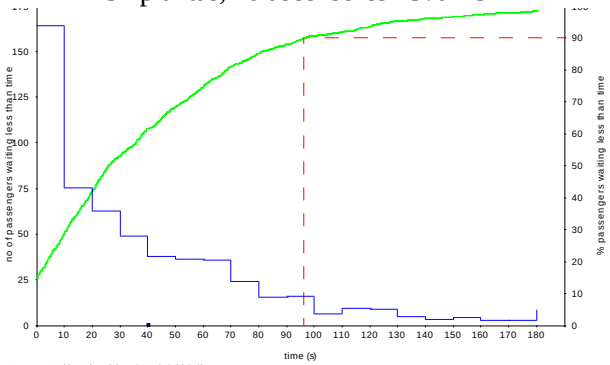
Average Waiting Time (s) 19.1 (+2.4/-2.4)
Longest Waiting Time (s) 154.1 (+32.7/-32.7)

32 plantas, 6 ascensores 12,5% POP



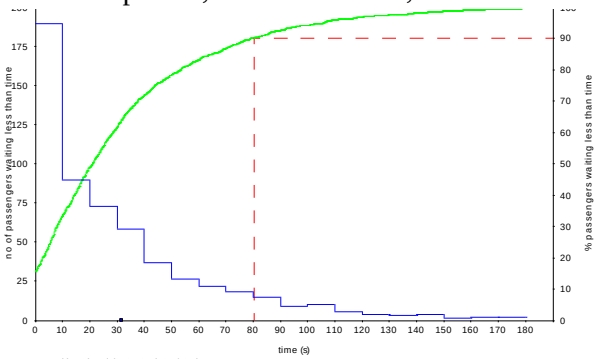
Average Waiting Time (s) 108.7 (+7.9/-7.9)
Longest Waiting Time (s) 641.8 (+100.0/-100.0)

32 plantas, 10 ascensores 15% POP



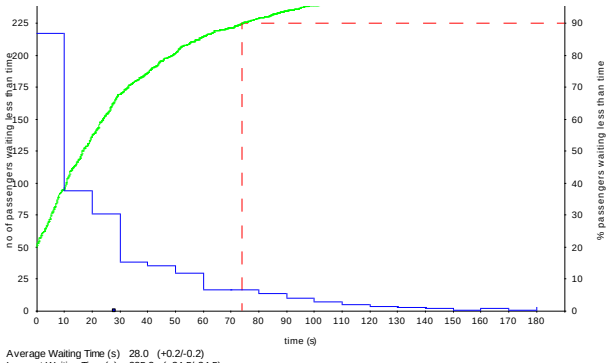
Average Waiting Time (s) 40.7 (+0.8/-0.8)
Longest Waiting Time (s) 241.0 (+17.3/-17.3)

32 plantas, 12 ascensores 17,5% POP



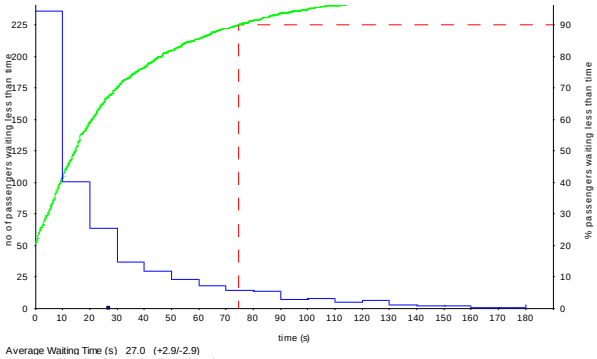
Average Waiting Time (s) 31.8 (+2.7/-2.7)
Longest Waiting Time (s) 202.5 (+20.1/-20.1)

32 plantas, 14 ascensores 20% POP



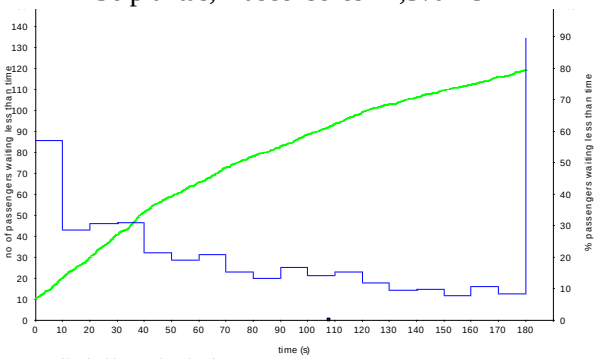
Average Waiting Time (s) 28.0 (+0.2/-0.2)
Longest Waiting Time (s) 225.2 (+34.5/-34.5)

32 plantas, 17 ascensores 22,5% POP



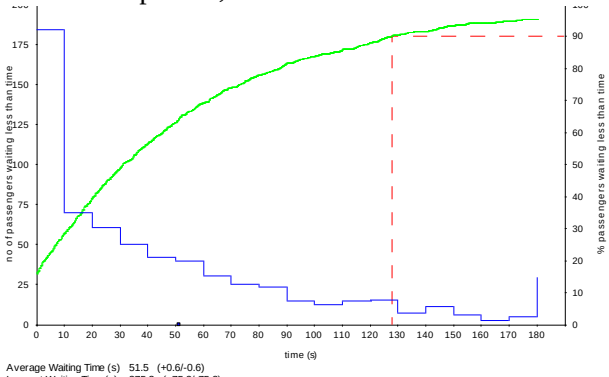
Average Waiting Time (s) 27.0 (+2.9/-2.9)
Longest Waiting Time (s) 238.9 (+18.7/-18.7)

36 plantas, 7 ascensores 12,5% POP



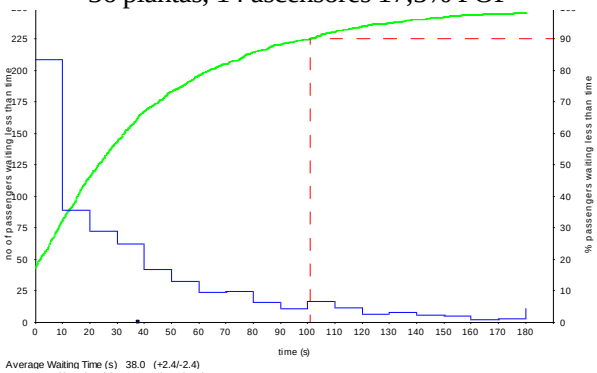
Average Waiting Time (s) 107.8 (+10.3/-10.3)
Longest Waiting Time (s) 640.1 (+58.7/-58.7)

36 plantas, 11 ascensores 15% POP



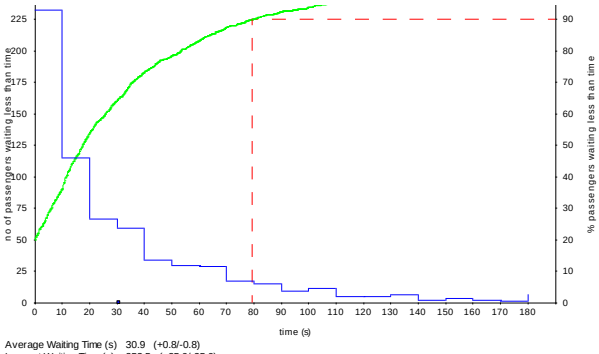
Average Waiting Time (s) 51.5 (+0.6/-0.6)
Longest Waiting Time (s) 375.3 (+75.2/-75.2)

36 plantas, 14 ascensores 17,5% POP



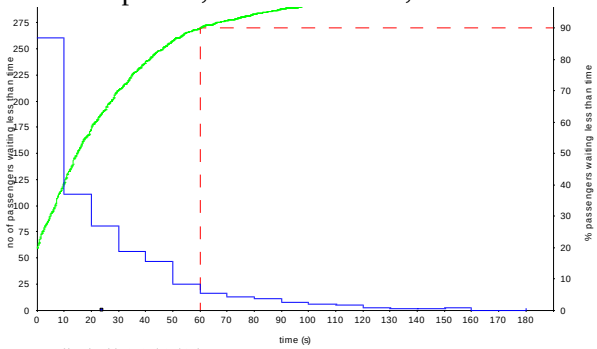
Average Waiting Time (s) 38.0 (+2.4/-2.4)
Longest Waiting Time (s) 256.7 (+3.8/-3.8)

36 plantas, 16 ascensores 20% POP



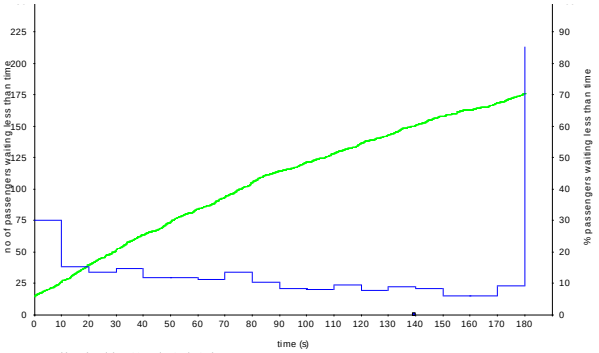
Average Waiting Time (s) 30.9 (+0.8/-0.8)
Longest Waiting Time (s) 253.5 (+25.0/-25.0)

36 plantas, 19 ascensores 22,5% POP



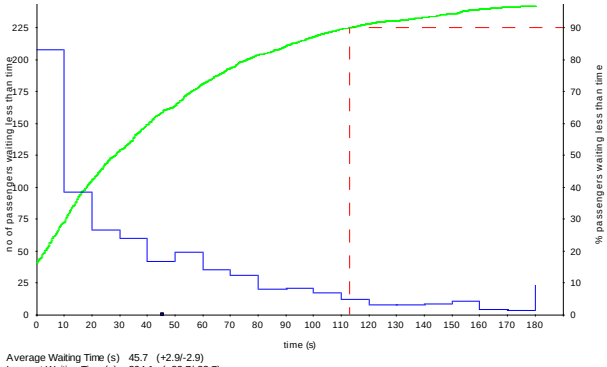
Average Waiting Time (s) 24.2 (+0.4/-0.4)
Longest Waiting Time (s) 191.9 (+1.8/-1.8)

40 plantas, 7 ascensores 12,5% POP



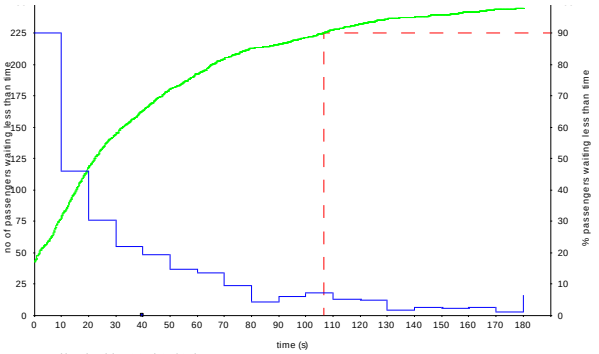
Average Waiting Time (s) 139.7 (+13.4/-13.4)
Longest Waiting Time (s) 718.8 (+12.1/-12.1)

40 plantas, 13 ascensores 15% POP



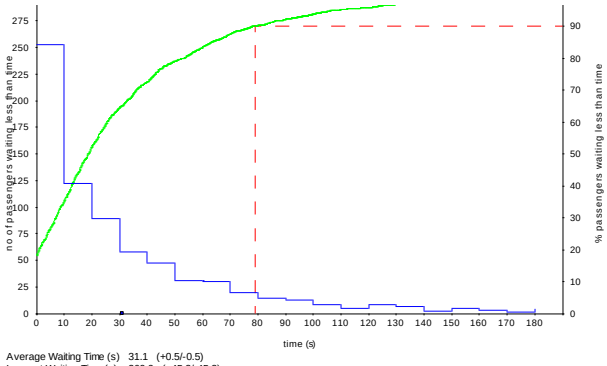
Average Waiting Time (s) 45.7 (+2.9/-2.9)
Longest Waiting Time (s) 304.1 (+33.7/-33.7)

40 plantas, 16 ascensores 17,5% POP



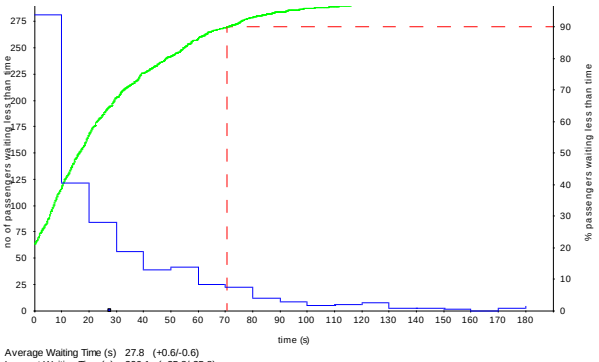
Average Waiting Time (s) 40.0 (+0.7/-0.7)
Longest Waiting Time (s) 333.3 (+66.4/-66.4)

40 plantas, 19 ascensores 20% POP



Average Waiting Time (s) 31.1 (+0.5/-0.5)
Longest Waiting Time (s) 269.9 (+45.2/-45.2)

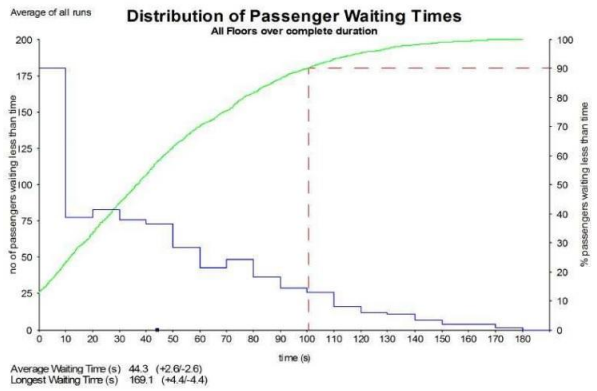
40 plantas, 21 ascensores 22,5% POP



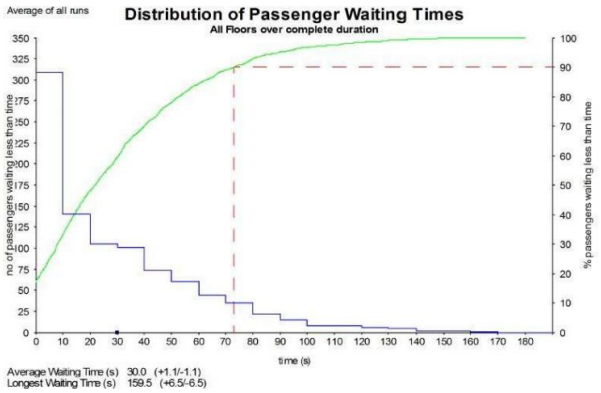
Average Waiting Time (s) 27.8 (+0.6/-0.6)
Longest Waiting Time (s) 296.1 (+35.8/-35.8)

Algoritmo Genético

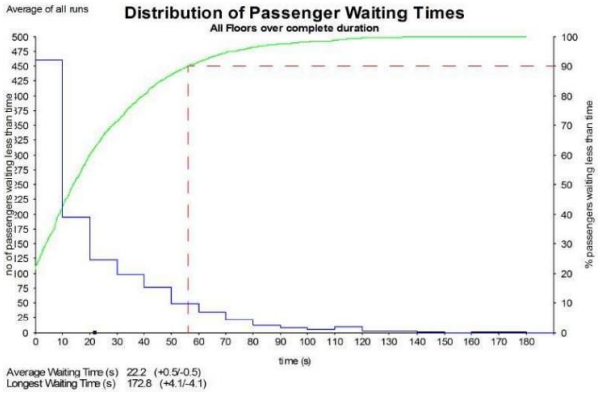
12 plantas, 2 ascensores 12,5% POP



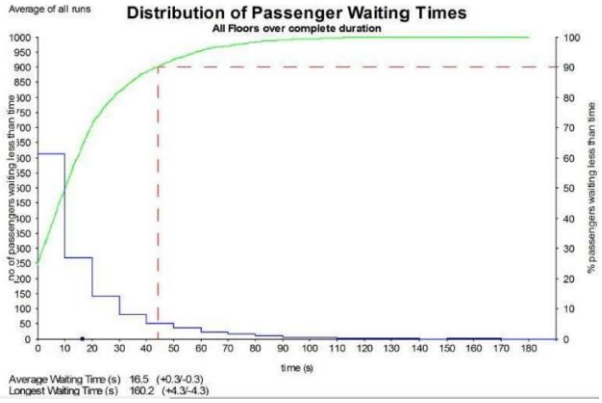
12 plantas, 3 ascensores 15% POP



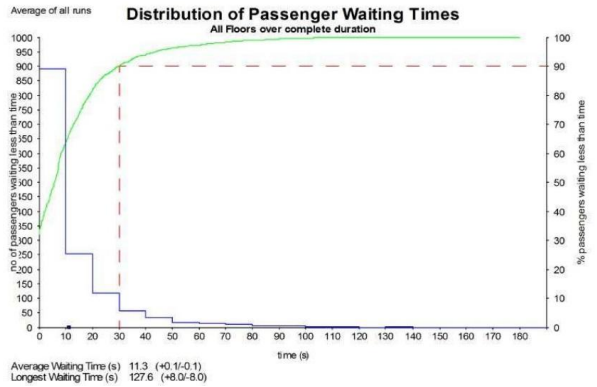
12 plantas, 4 ascensores 17,5% POP



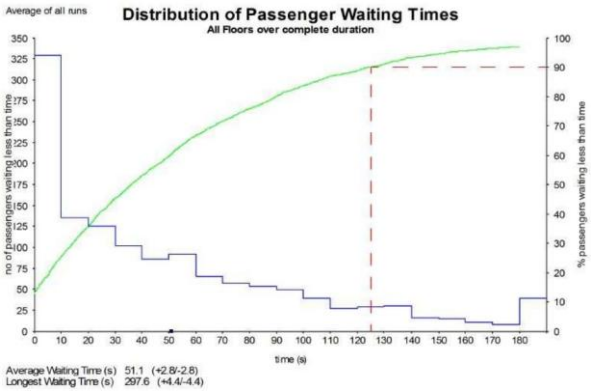
12 plantas, 5 ascensores 20% POP



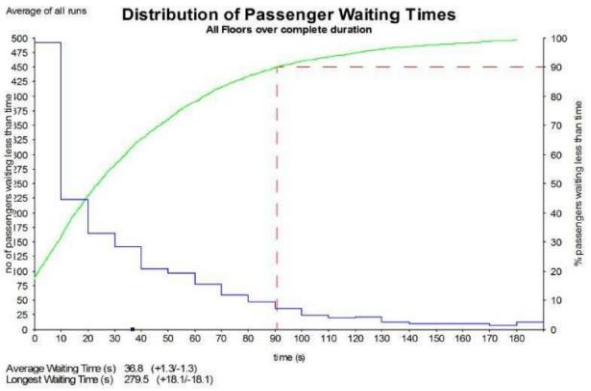
12 plantas, 7 ascensores 22,5% POP



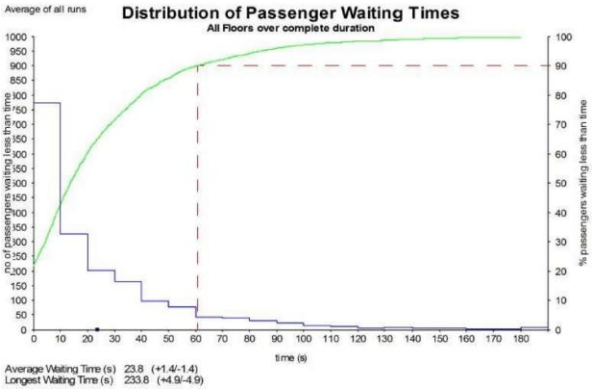
20 plantas, 4 ascensores 12,5% POP



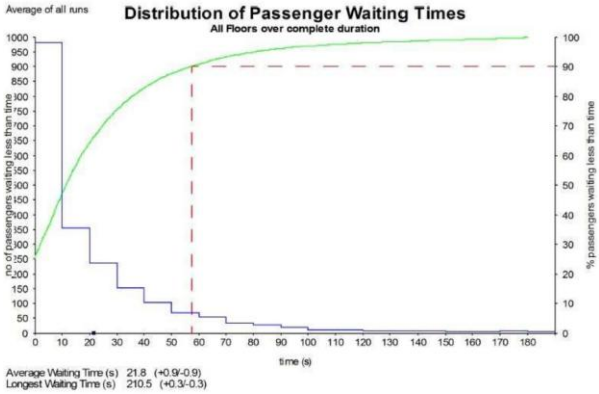
20 plantas, 5 ascensores 15% POP



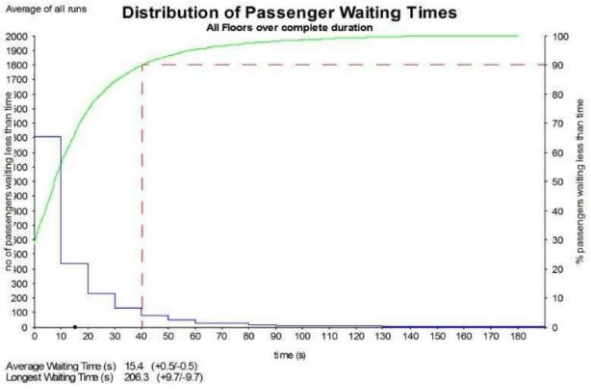
20 plantas, 7 ascensores 17,5% POP



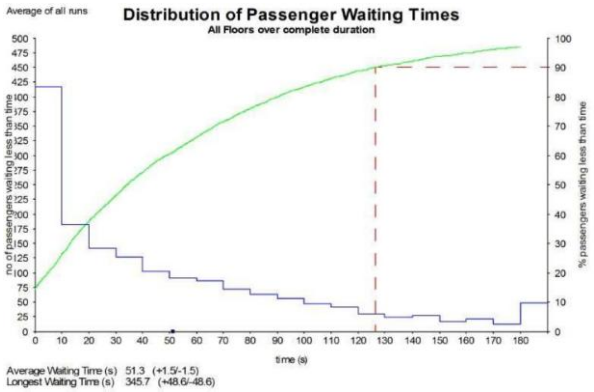
20 plantas, 8 ascensores 20% POP



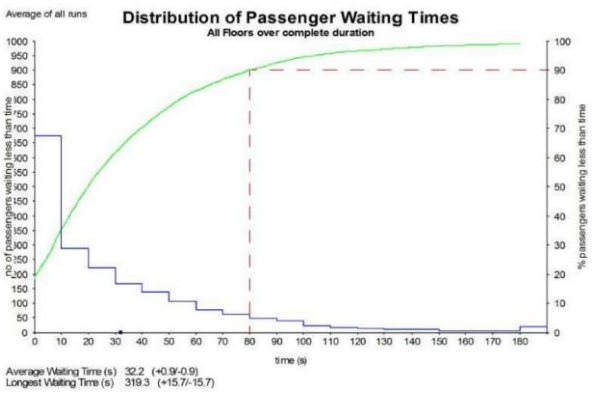
20 plantas, 11 ascensores 22,5% POP



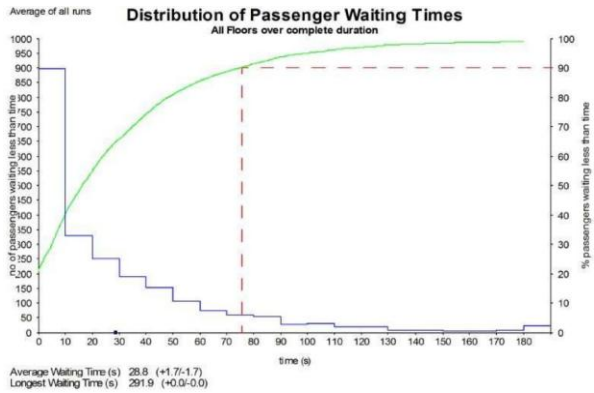
24 plantas, 5 ascensores 12,5% POP



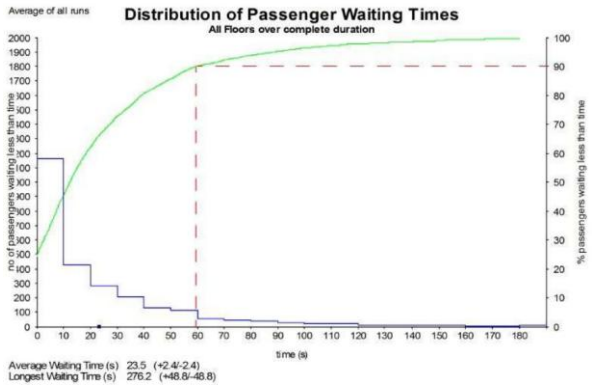
24 plantas, 7 ascensores 15% POP



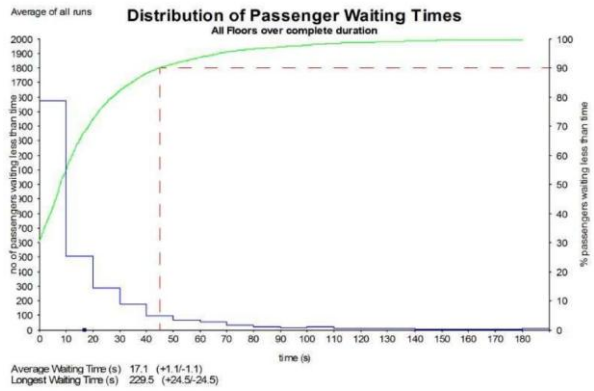
24 plantas, 8 ascensores 17,5% POP



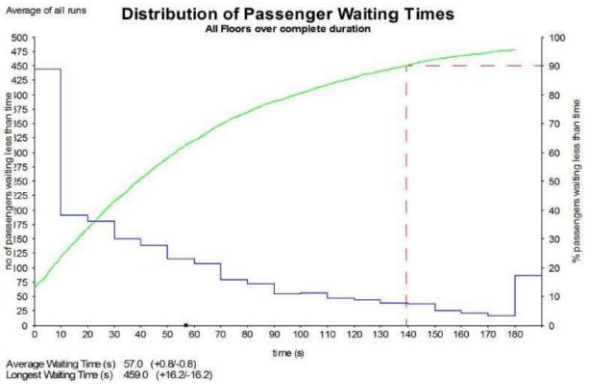
24 plantas, 10 ascensores 20% POP



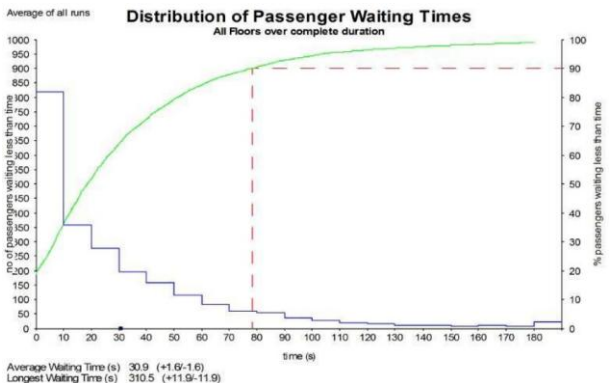
24 plantas, 13 ascensores 22,5% POP



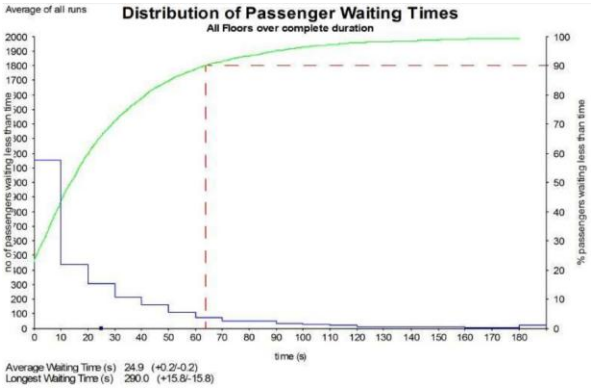
28 plantas, 6 ascensores 12,5% POP



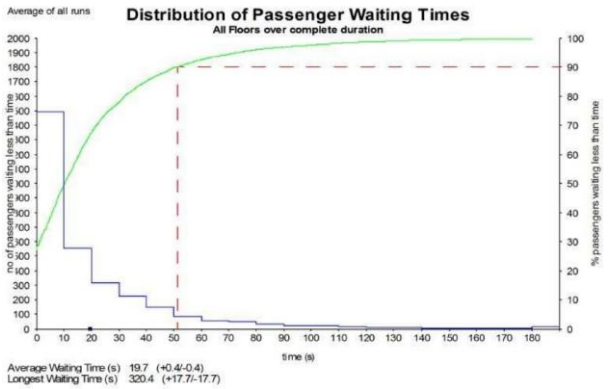
28 plantas, 9 ascensores 15% POP



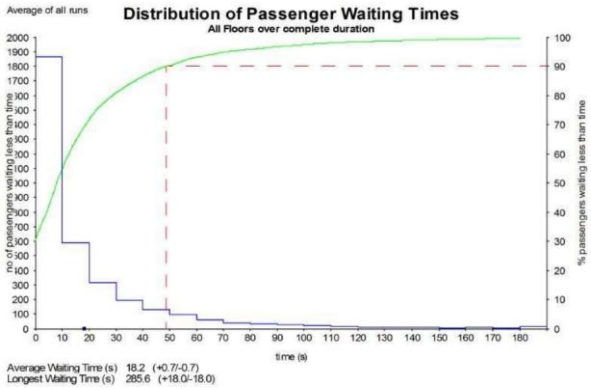
28 plantas, 11 ascensores 17,5% POP



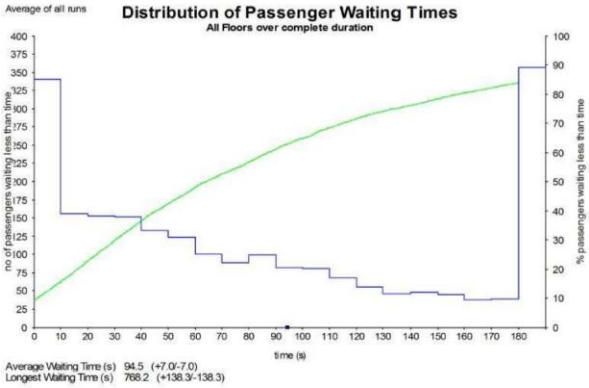
28 plantas, 13 ascensores 20% POP



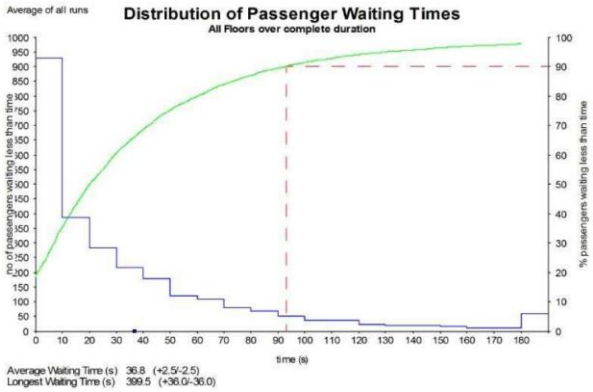
28 plantas, 16 ascensores 22,5% POP



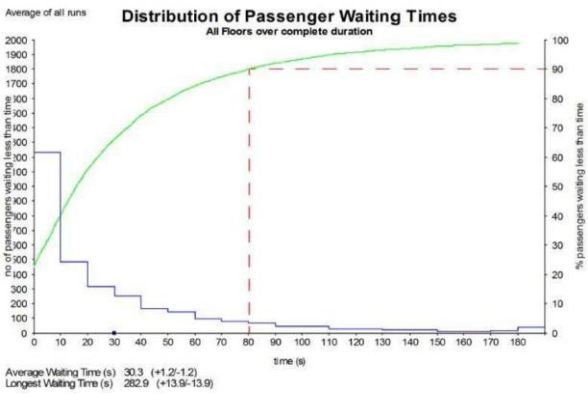
32 plantas, 6 ascensores 12,5% POP



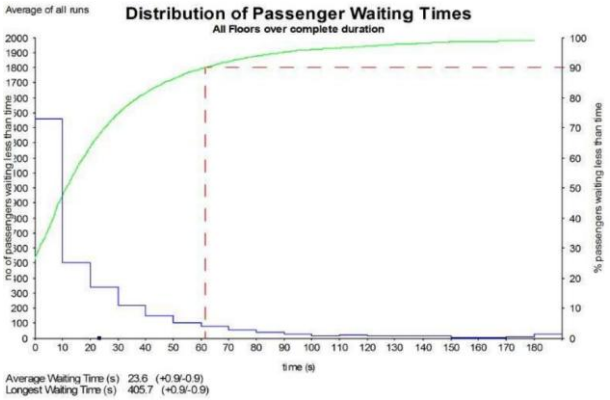
32 plantas, 10 ascensores 15% POP



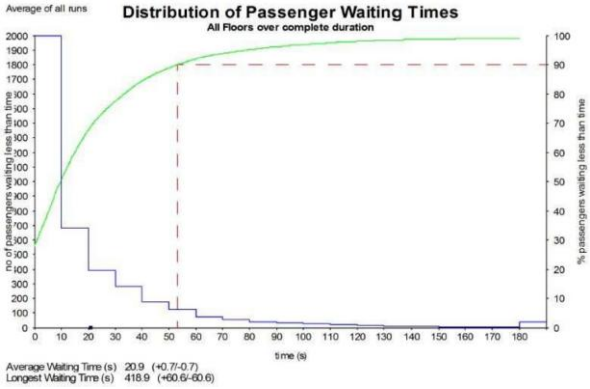
32 plantas, 12 ascensores 17,5% POP



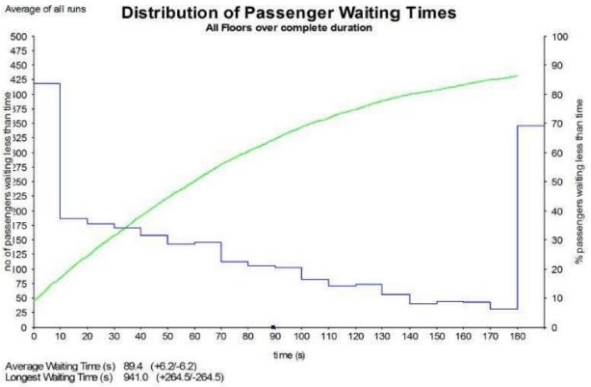
32 plantas, 14 ascensores 20% POP



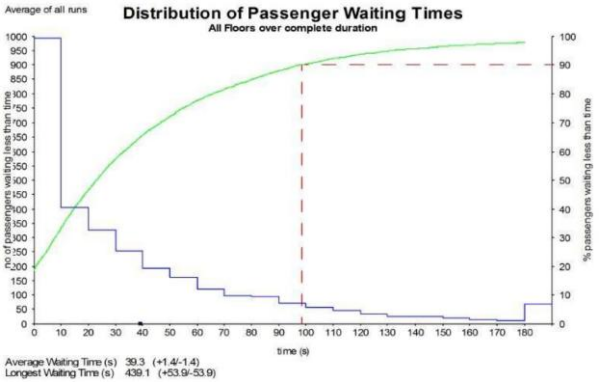
32 plantas, 17 ascensores 22,5% POP



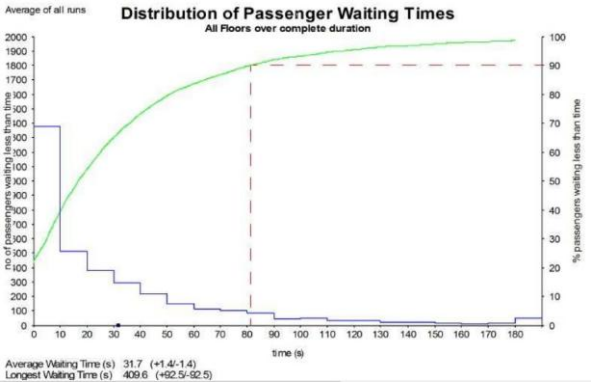
36 plantas, 7 ascensores 12,5% POP



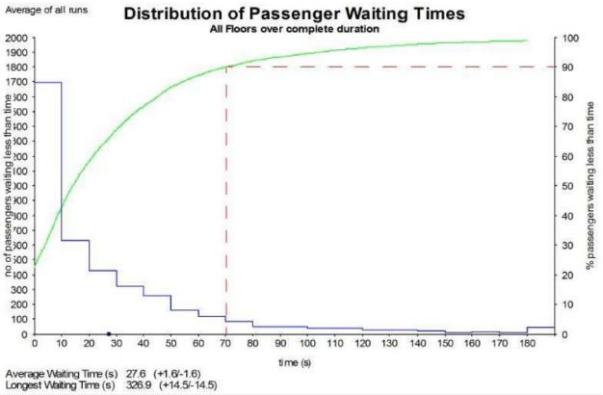
36 plantas, 11 ascensores 15% POP



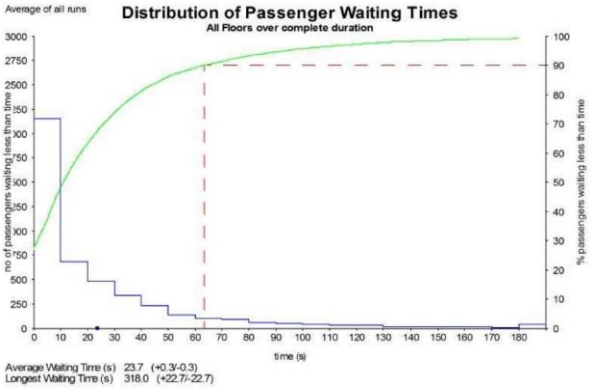
36 plantas, 14 ascensores 17,5% POP



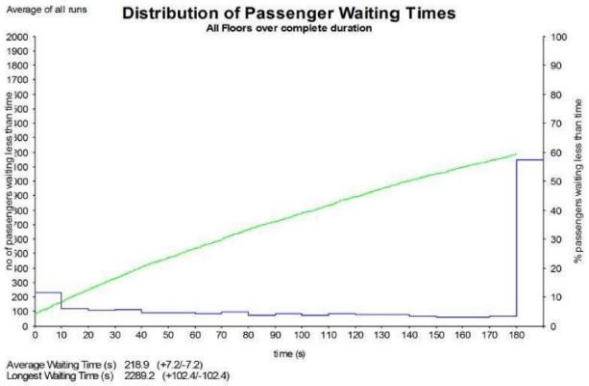
36 plantas, 16 ascensores 20% POP



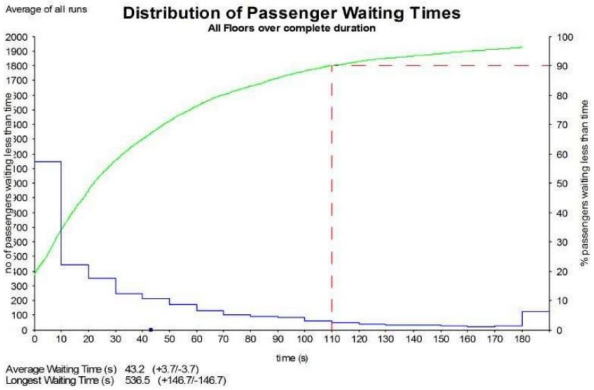
36 plantas, 19 ascensores 22,5% POP



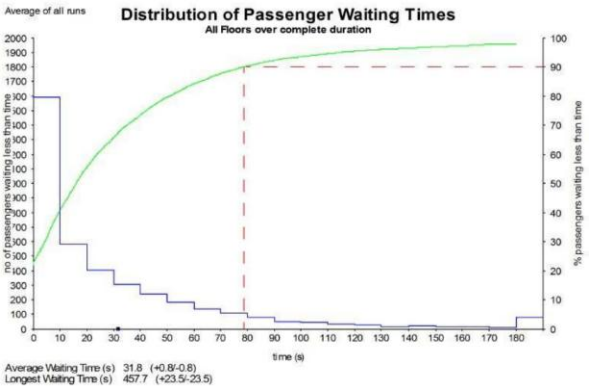
40 plantas, 7 ascensores 12,5% POP



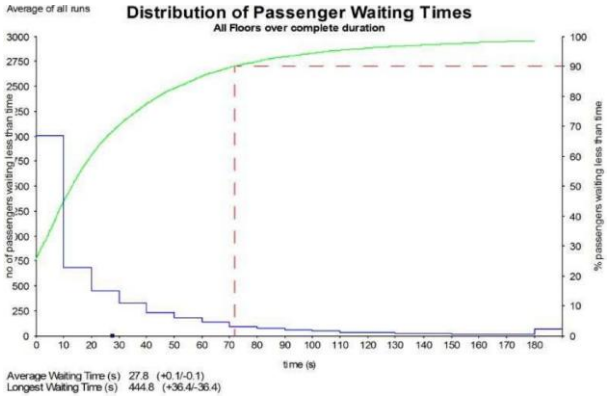
40 plantas, 13 ascensores 15% POP



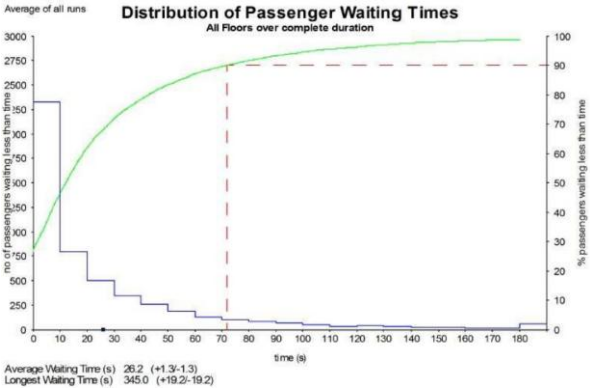
40 plantas, 16 ascensores 17,5% POP



40 plantas, 19 ascensores 20% POP

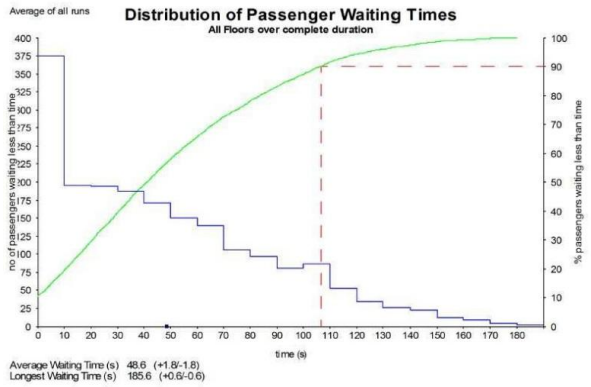


40 plantas, 21 ascensores 22,5% POP

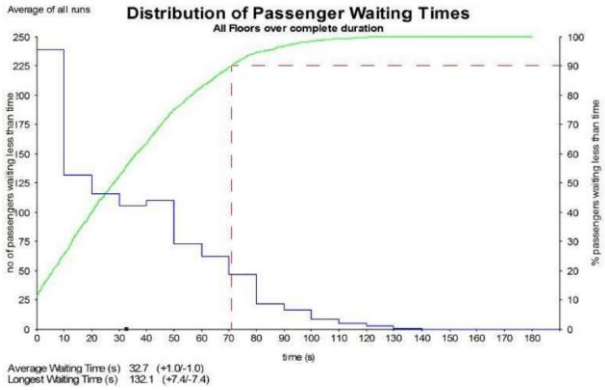


Algoritmo Genético Modificado

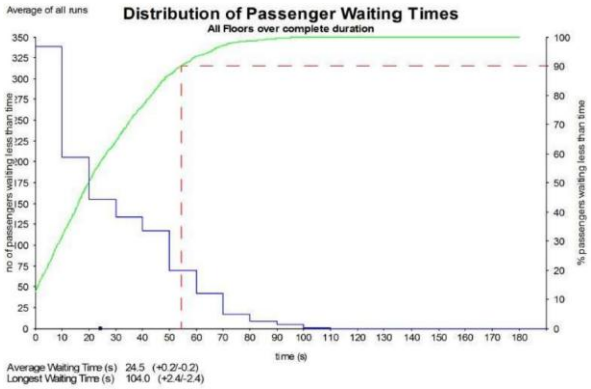
12 plantas, 2 ascensores 12,5% POP



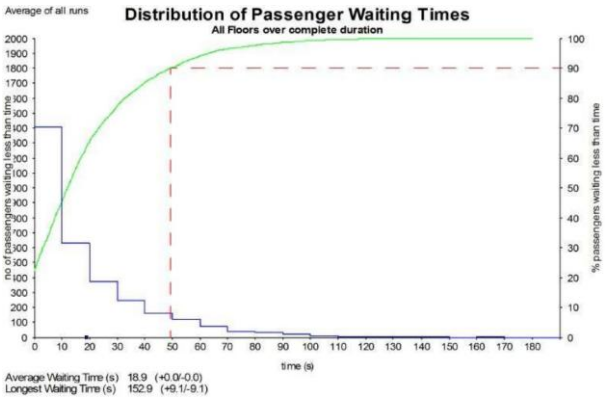
12 plantas, 3 ascensores 15% POP



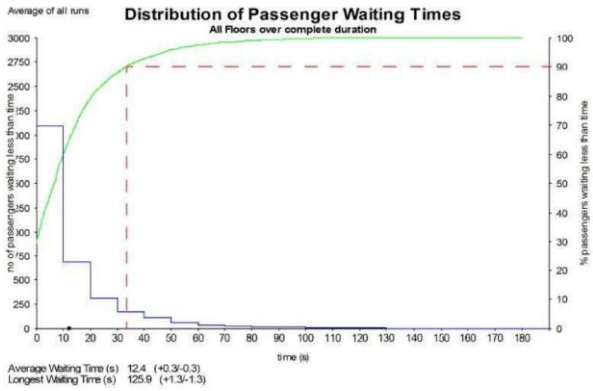
12 plantas, 4 ascensores 17,5% POP



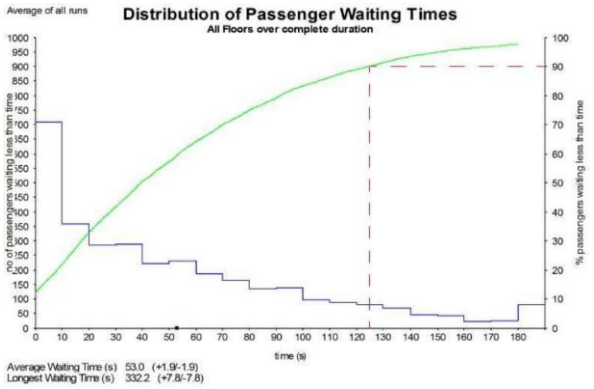
12 plantas, 5 ascensores 20% POP



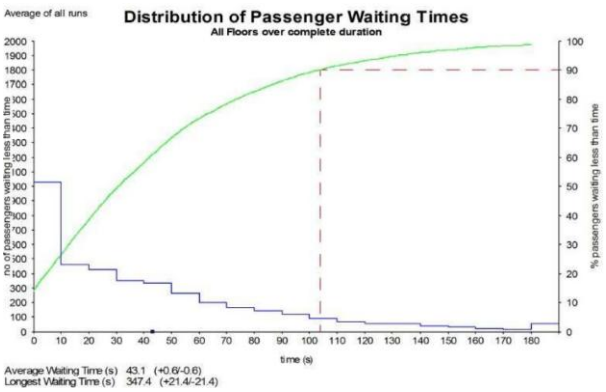
12 plantas, 7 ascensores 22,5% POP



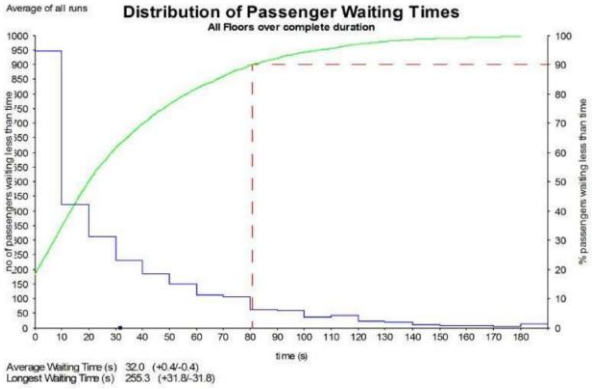
20 plantas, 4 ascensores 12,5% POP



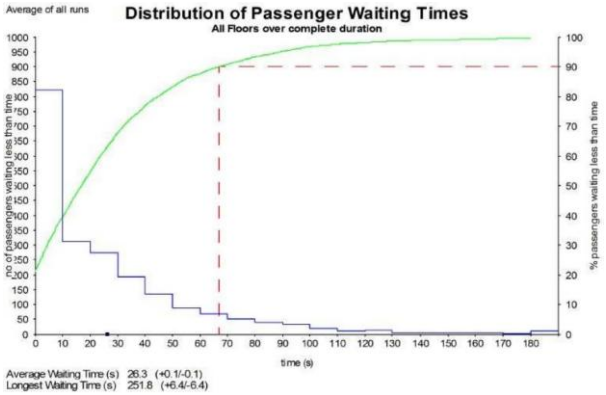
20 plantas, 5 ascensores 15% POP



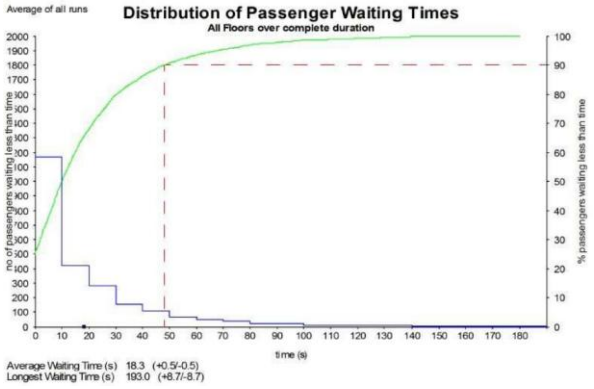
20 plantas, 7 ascensores 17,5% POP



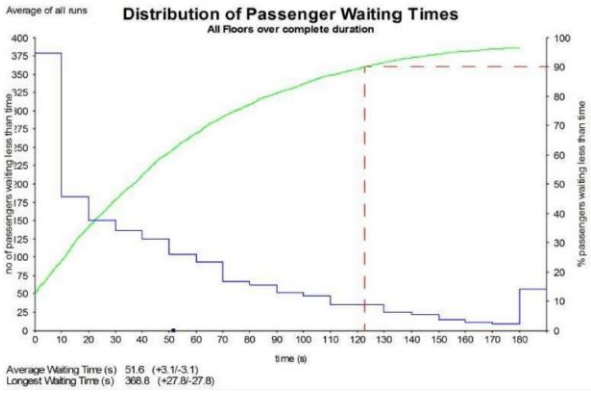
20 plantas, 8 ascensores 20% POP



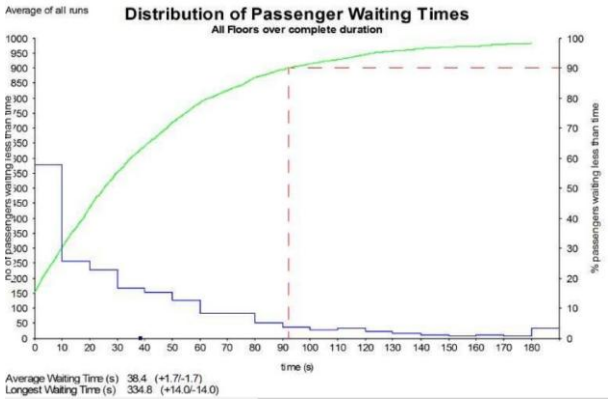
20 plantas, 11 ascensores 22,5% POP



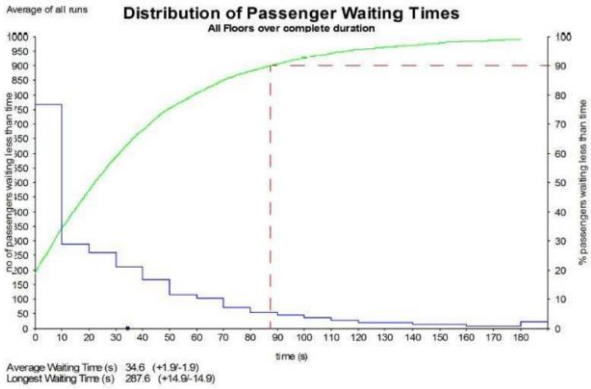
24 plantas, 5 ascensores 12,5% POP



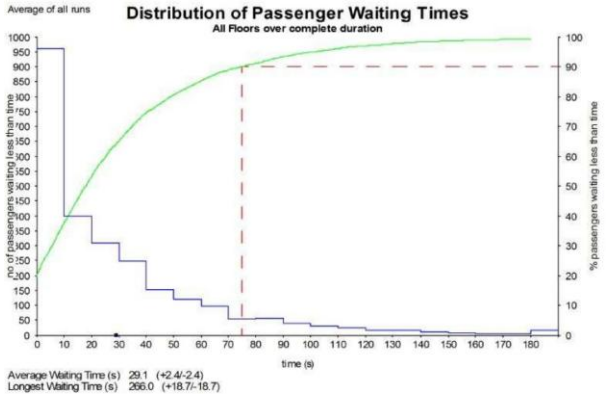
24 plantas, 7 ascensores 15% POP



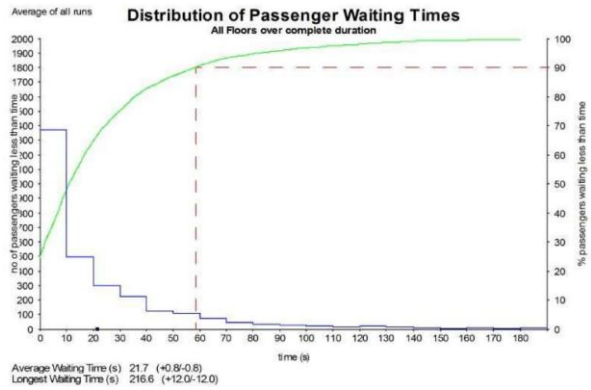
24 plantas, 8 ascensores 17,5% POP



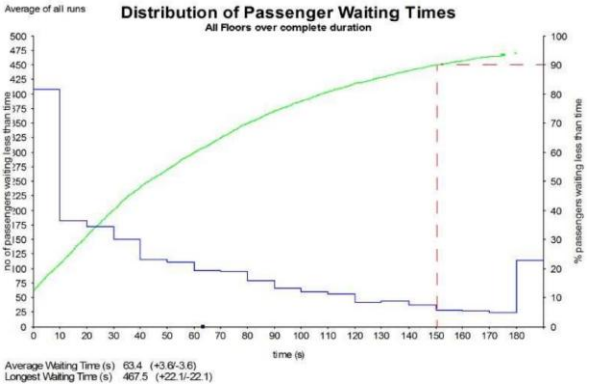
24 plantas, 10 ascensores 20% POP



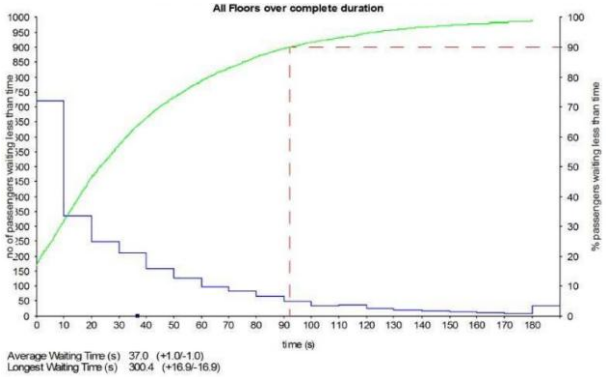
24 plantas, 13 ascensores 22,5% POP



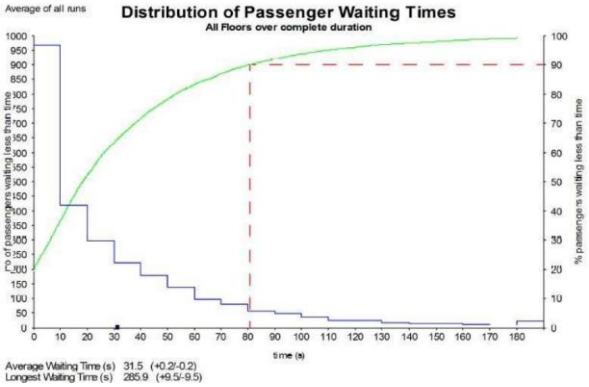
28 plantas, 6 ascensores 12,5% POP



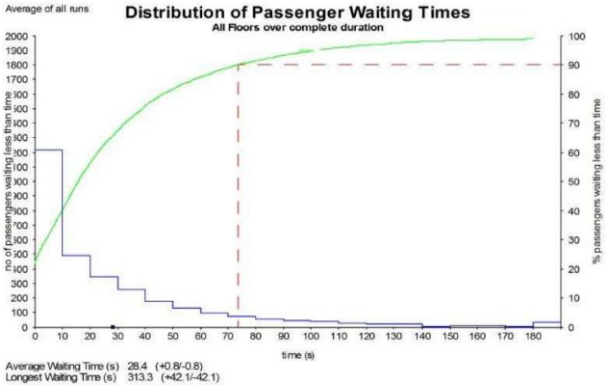
28 plantas, 9 ascensores 15% POP



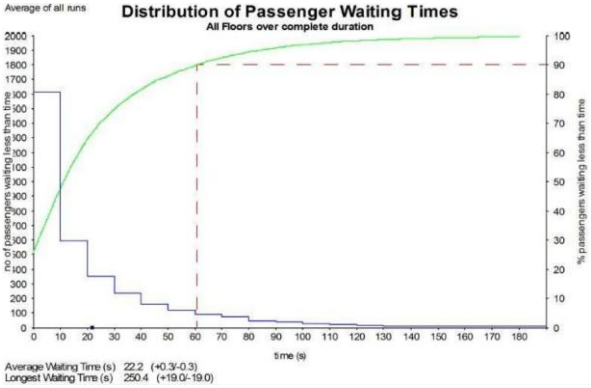
28 plantas, 11 ascensores 17,5% POP



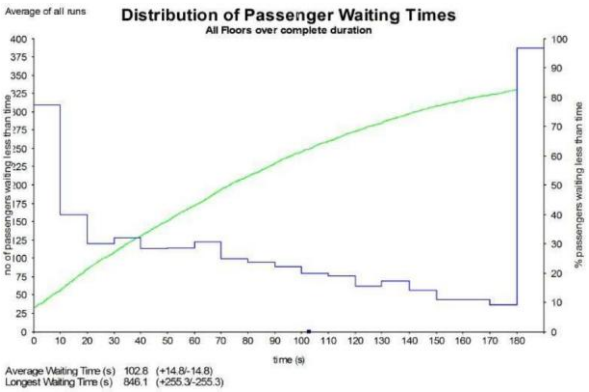
28 plantas, 13 ascensores 20% POP



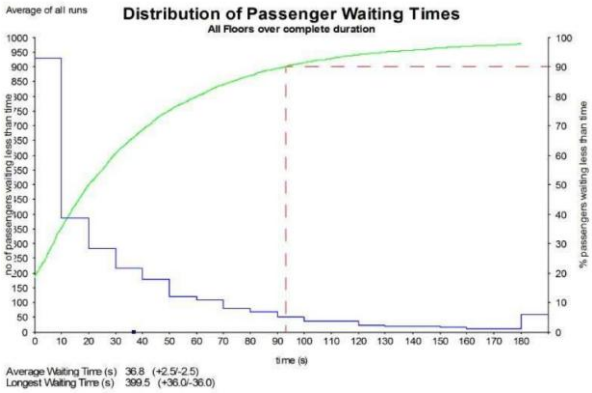
28 plantas, 16 ascensores 22,5% POP



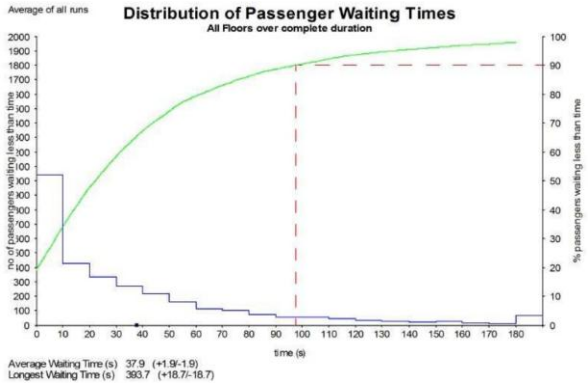
32 plantas, 6 ascensores 12,5% POP



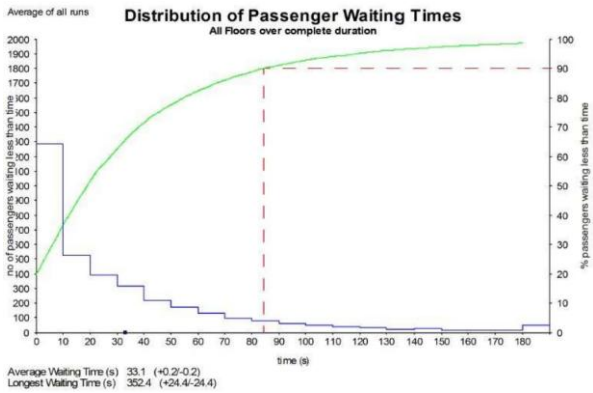
32 plantas, 10 ascensores 15% POP



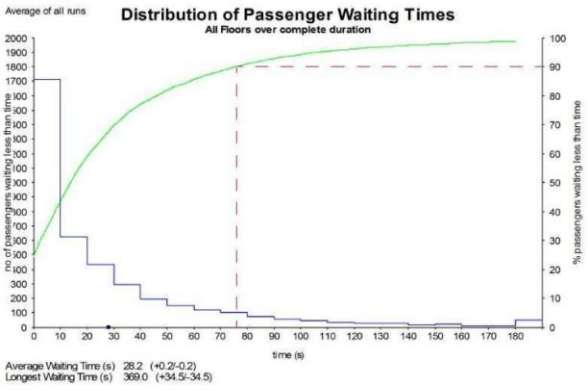
32 plantas, 12 ascensores 17,5% POP



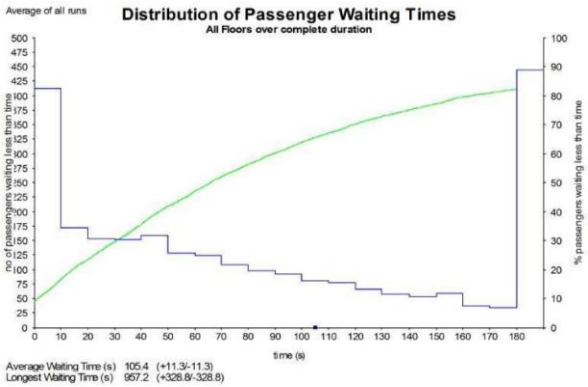
32 plantas, 14 ascensores 20% POP



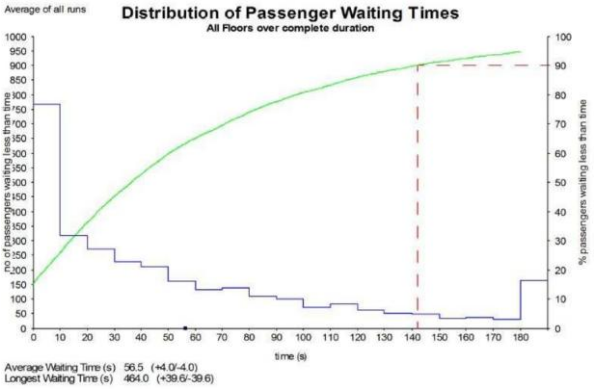
32 plantas, 17 ascensores 22,5% POP



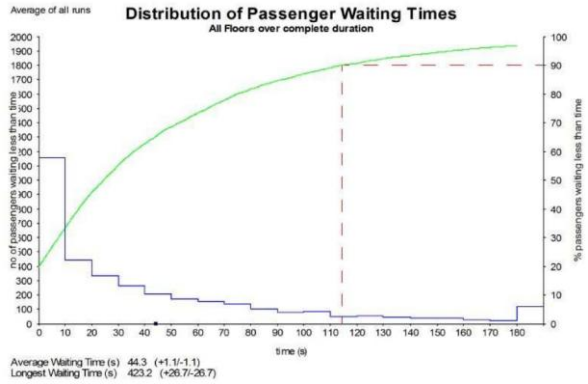
36 plantas, 7 ascensores 12,5% POP



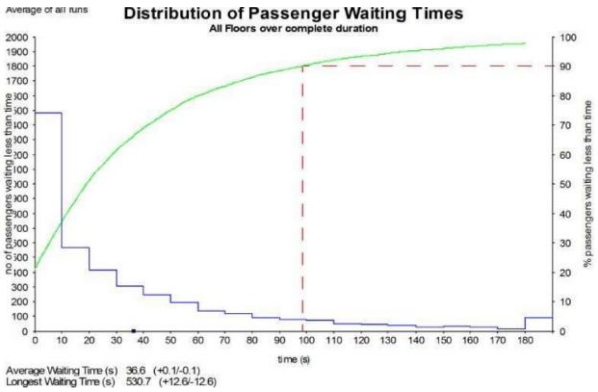
36 plantas, 11 ascensores 15% POP



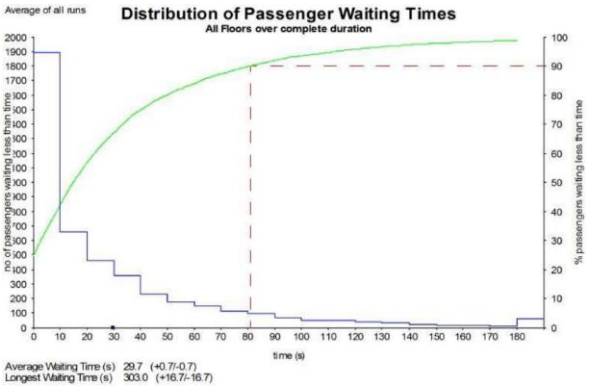
36 plantas, 14 ascensores 17,5% POP



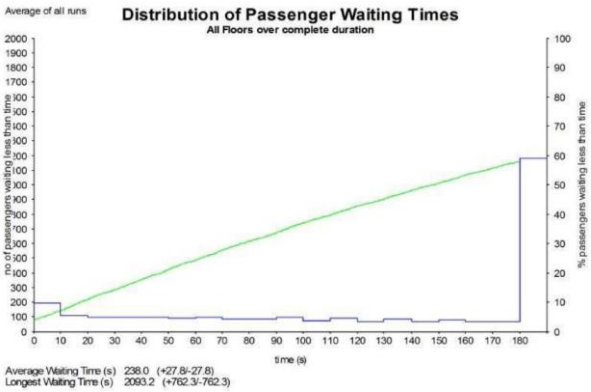
36 plantas, 16 ascensores 20% POP



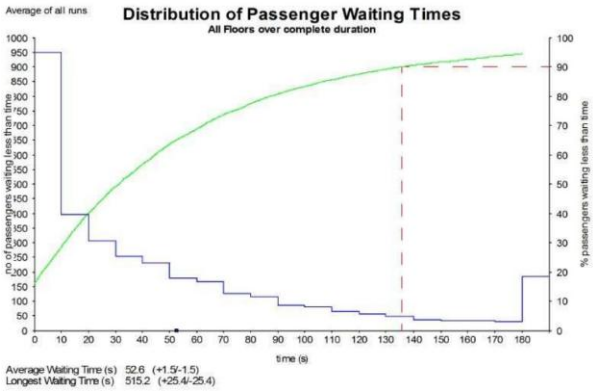
36 plantas, 19 ascensores 22,5% POP



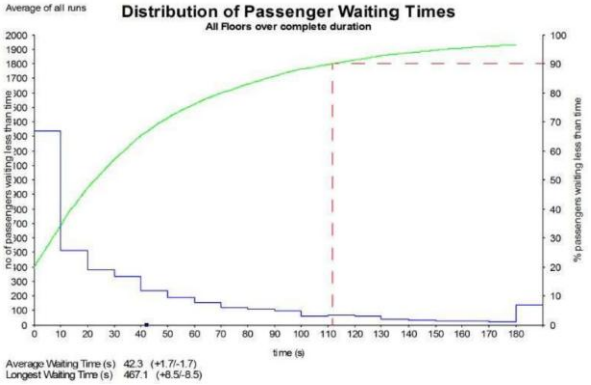
40 plantas, 7 ascensores 12,5% POP



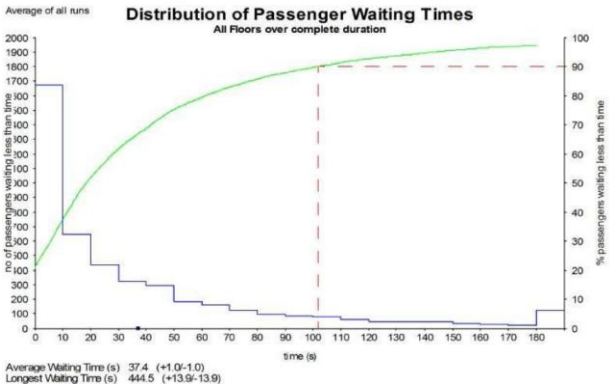
40 plantas, 13 ascensores 15% POP



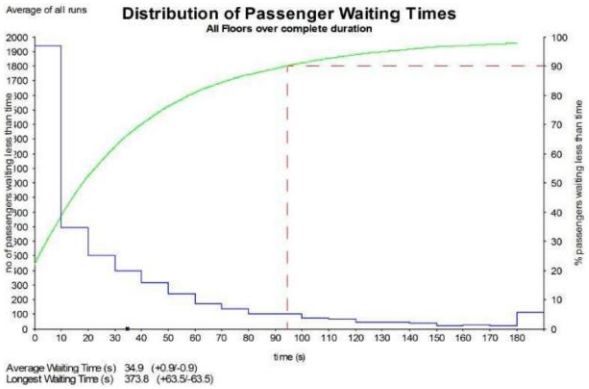
40 plantas, 16 ascensores 17,5% POP



40 plantas, 19 ascensores 20% POP

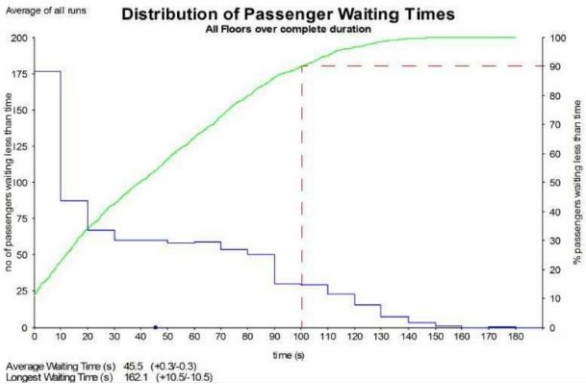


40 plantas, 21 ascensores 22,5% POP

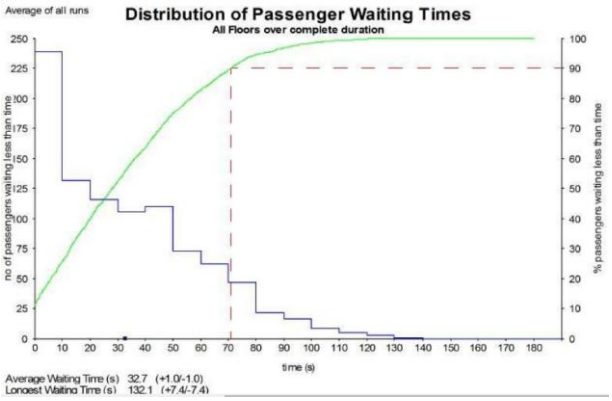


Double Deck Destination Control

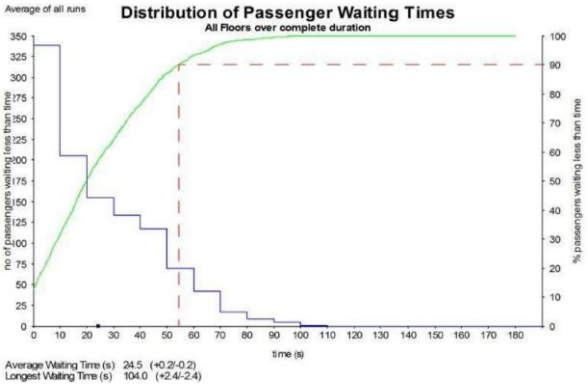
12 plantas, 2 ascensores 12,5% POP



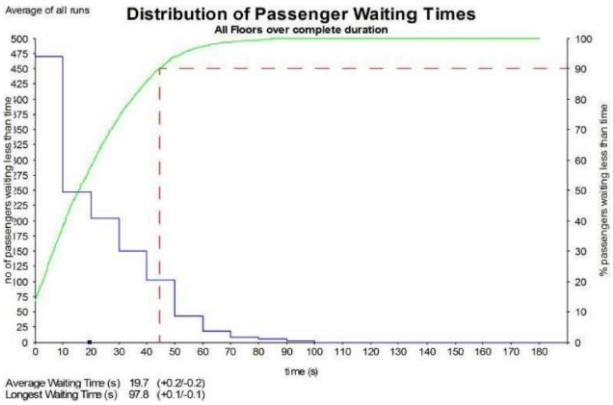
12 plantas, 3 ascensores 15% POP



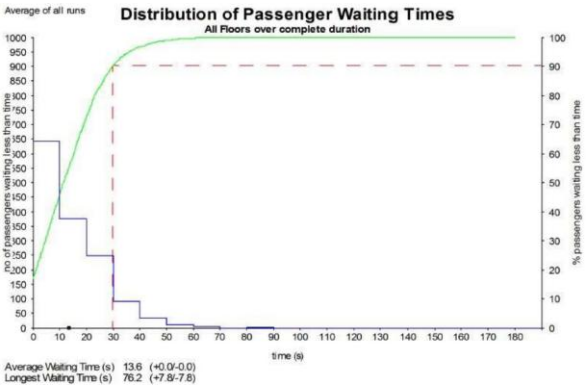
12 plantas, 4 ascensores 17,5% POP



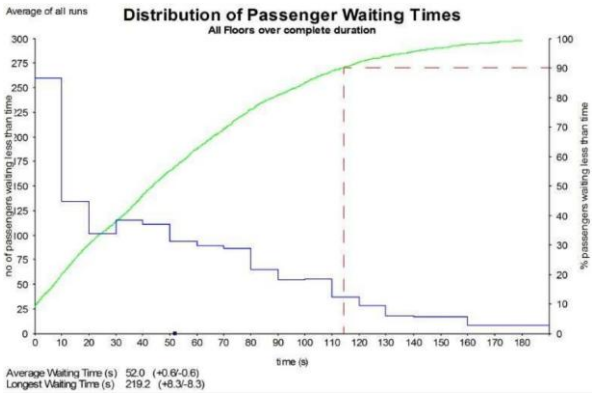
12 plantas, 5 ascensores 20% POP



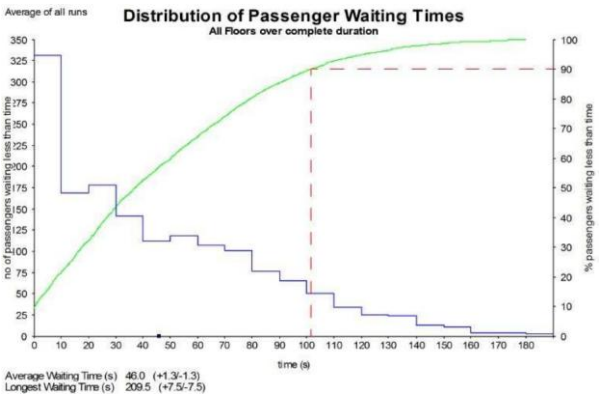
12 plantas, 7 ascensores 22,5% POP



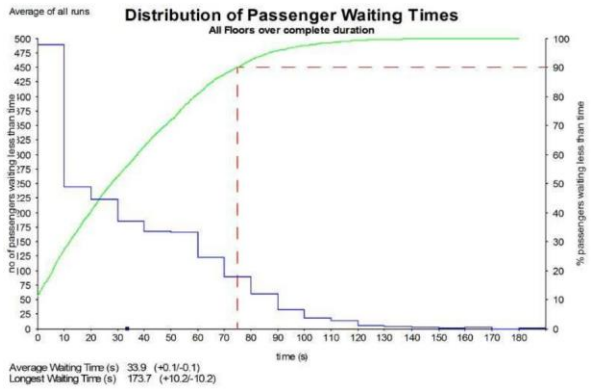
20 plantas, 4 ascensores 12,5% POP



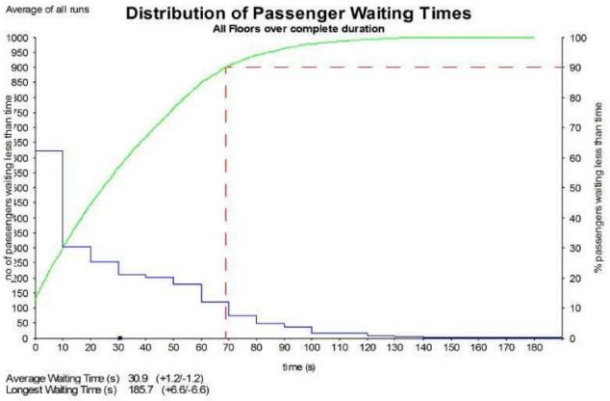
20 plantas, 5 ascensores 15% POP



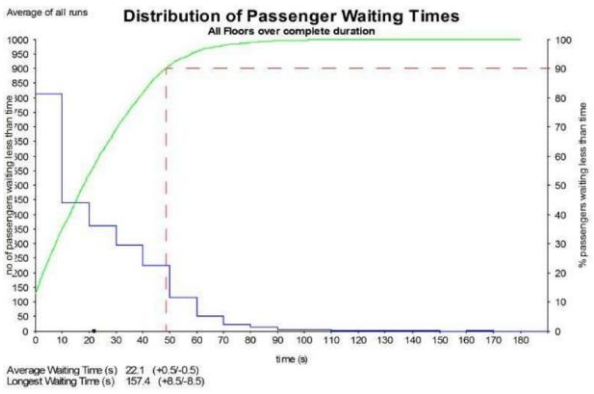
20 plantas, 7 ascensores 17,5% POP



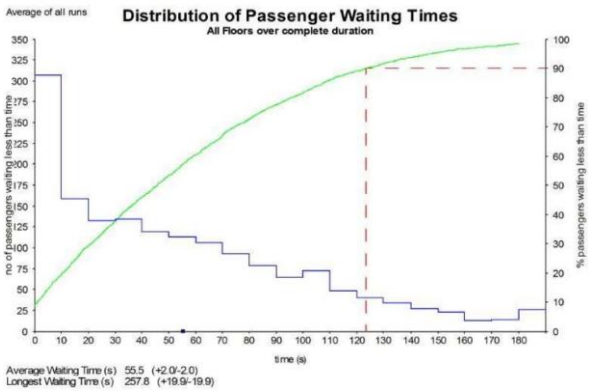
20 plantas, 8 ascensores 20% POP



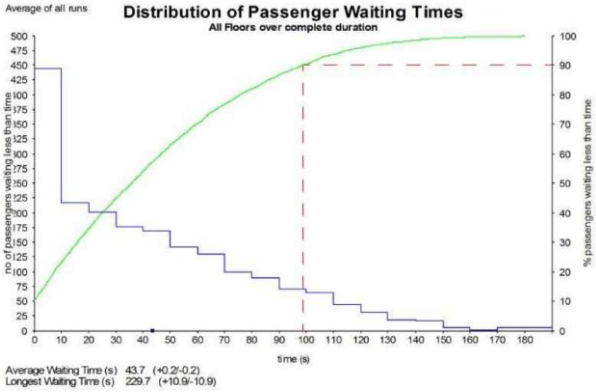
20 plantas, 11 ascensores 22,5% POP



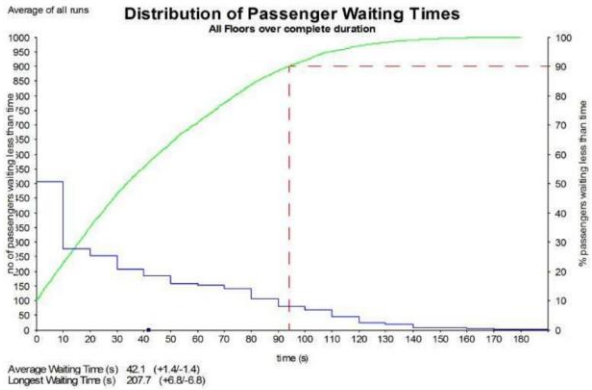
24 plantas, 5 ascensores 12,5% POP



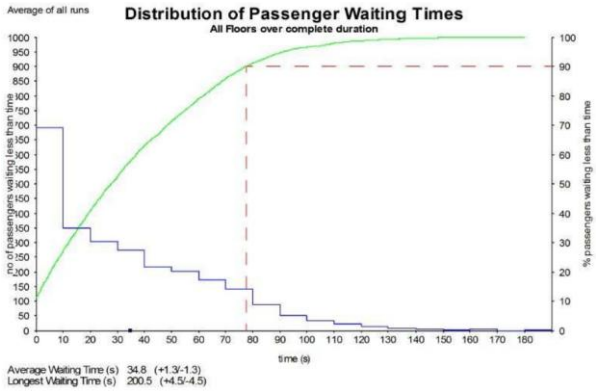
24 plantas, 7 ascensores 15% POP



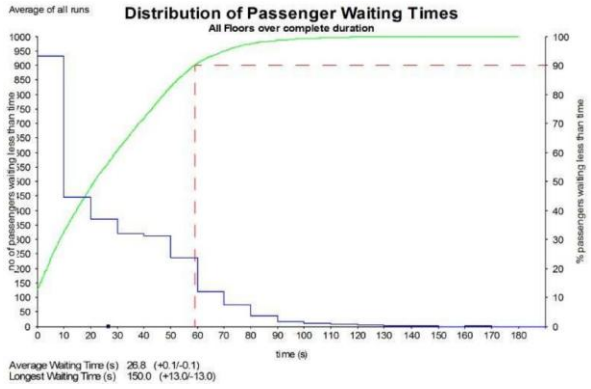
24 plantas, 8 ascensores 17,5% POP



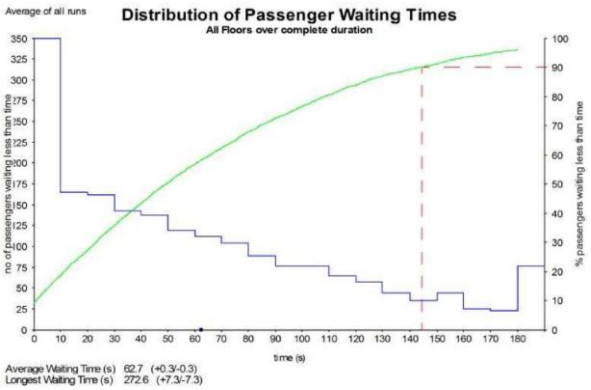
24 plantas, 10 ascensores 20% POP



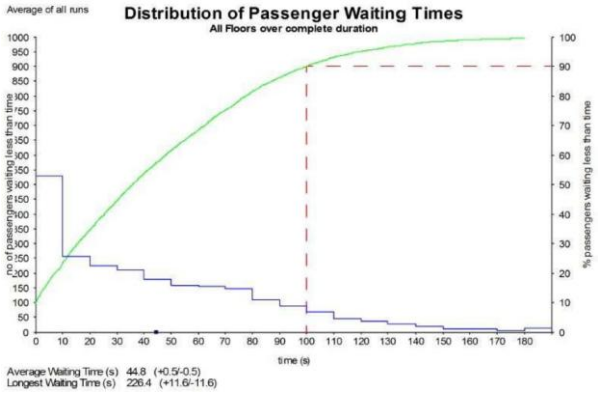
24 plantas, 13 ascensores 22,5% POP



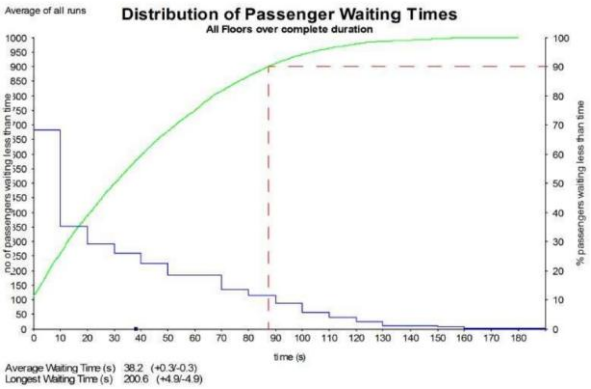
28 plantas, 6 ascensores 12,5% POP



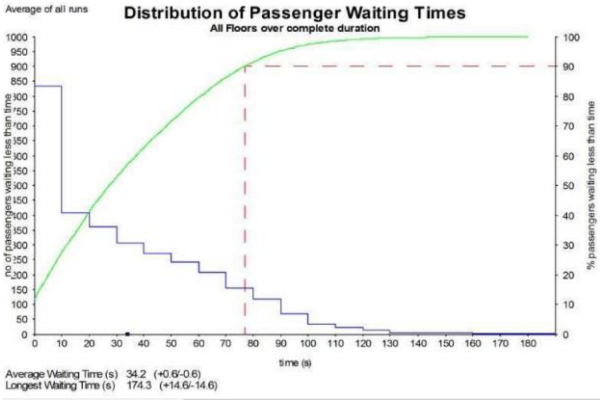
28 plantas, 9 ascensores 15% POP



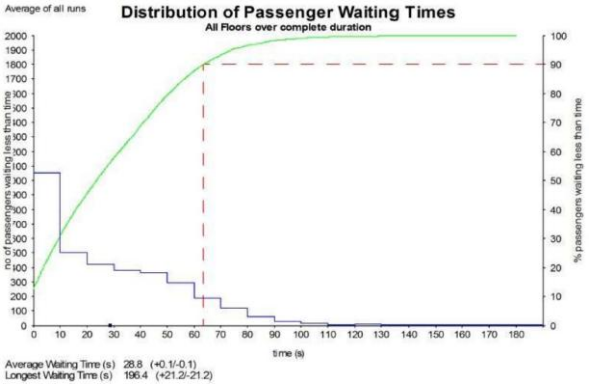
28 plantas, 11 ascensores 17,5% POP



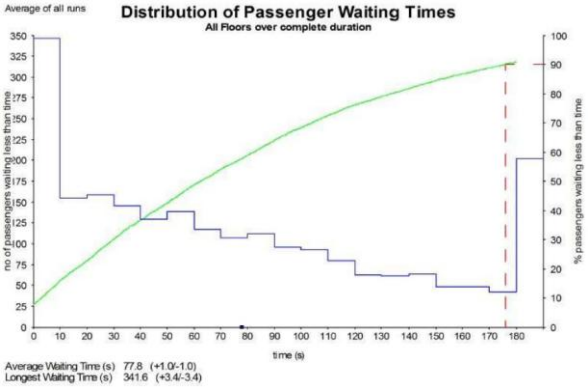
28 plantas, 13 ascensores 20% POP



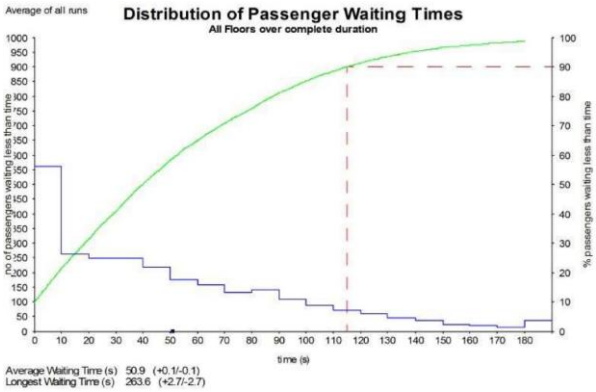
28 plantas, 16 ascensores 22,5% POP



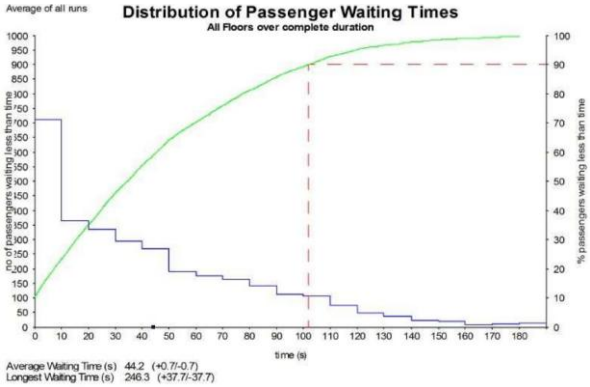
32 plantas, 6 ascensores 12,5% POP



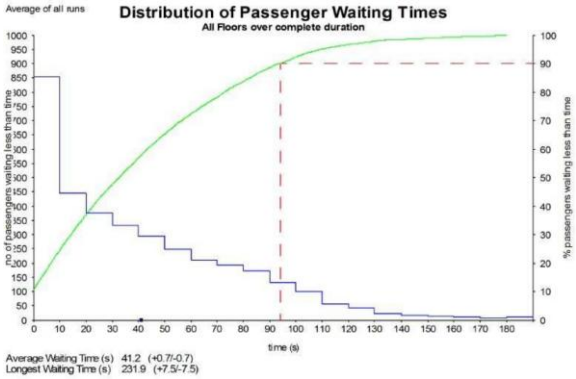
32 plantas, 10 ascensores 15% POP



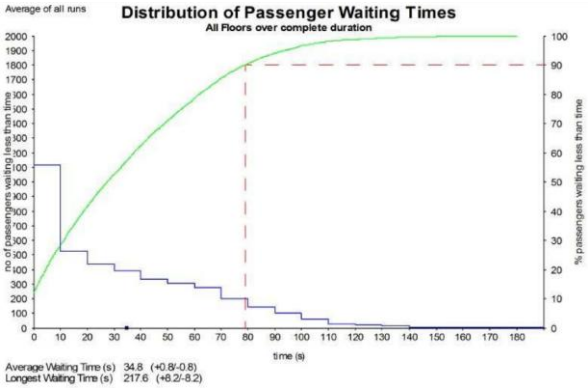
32 plantas, 12 ascensores 17,5% POP



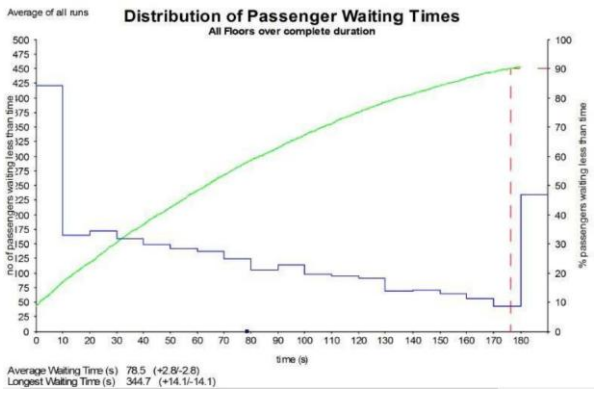
32 plantas, 14 ascensores 20% POP



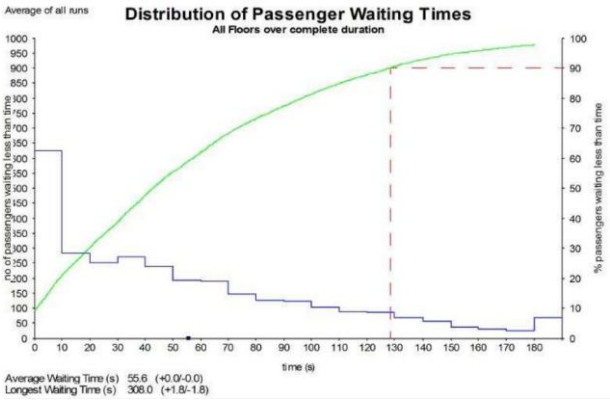
32 plantas, 17 ascensores 22,5% POP



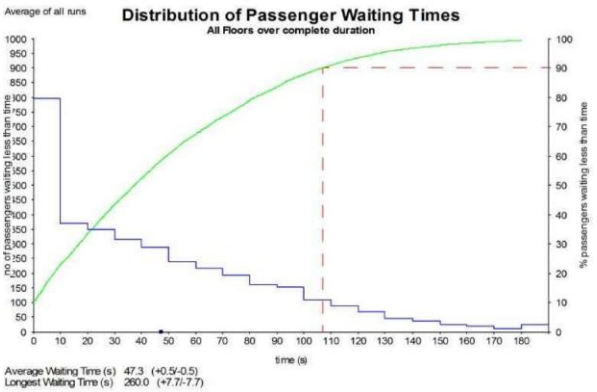
36 plantas, 7 ascensores 12,5% POP



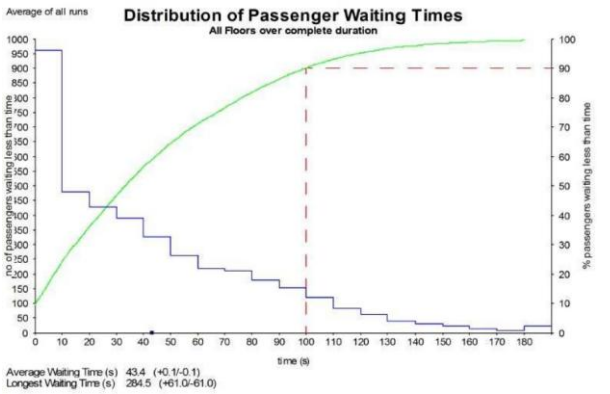
36 plantas, 11 ascensores 15% POP



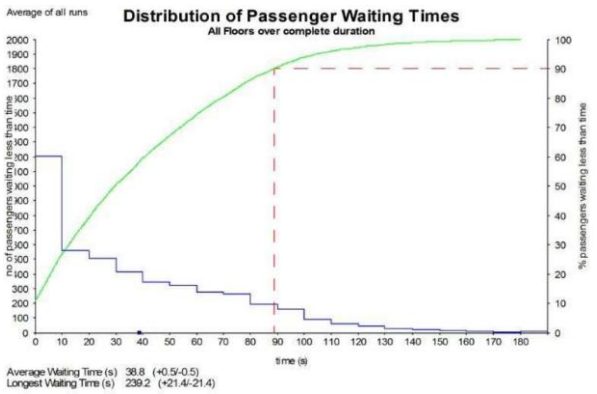
36 plantas, 14 ascensores 17,5% POP



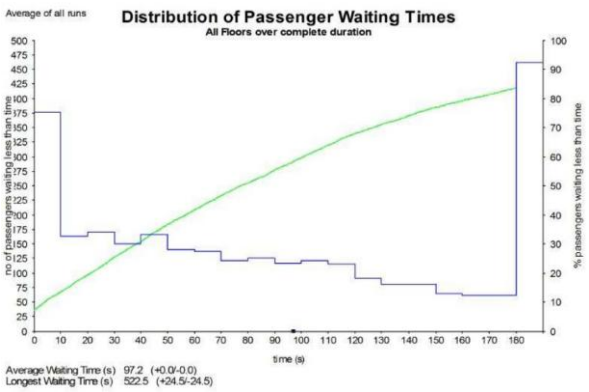
36 plantas, 16 ascensores 20% POP



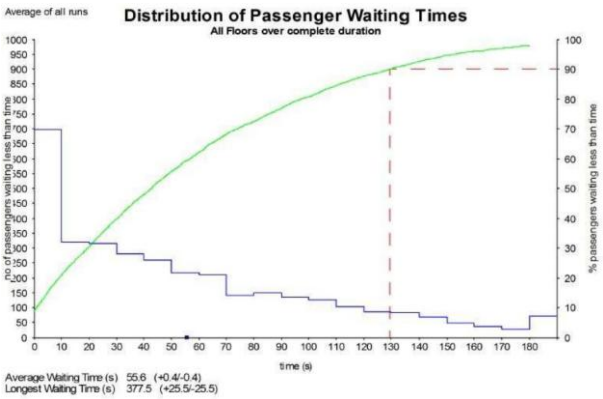
36 plantas, 19 ascensores 22,5% POP



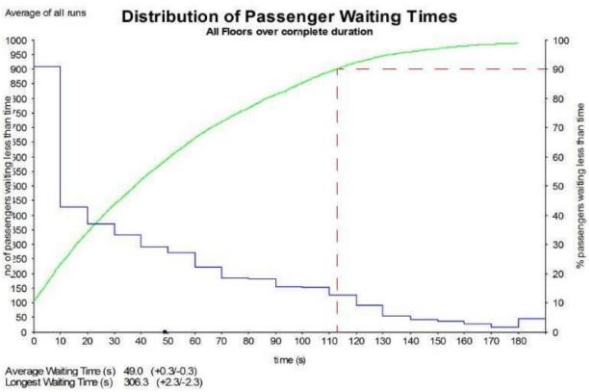
40 plantas, 7 ascensores 12,5% POP



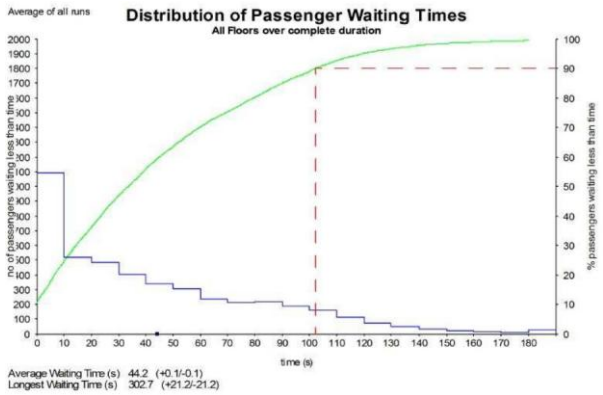
40 plantas, 13 ascensores 15% POP



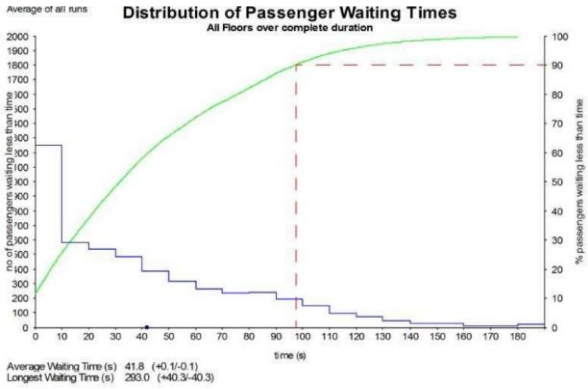
40 plantas, 16 ascensores 17,5% POP



40 plantas, 19 ascensores 20% POP



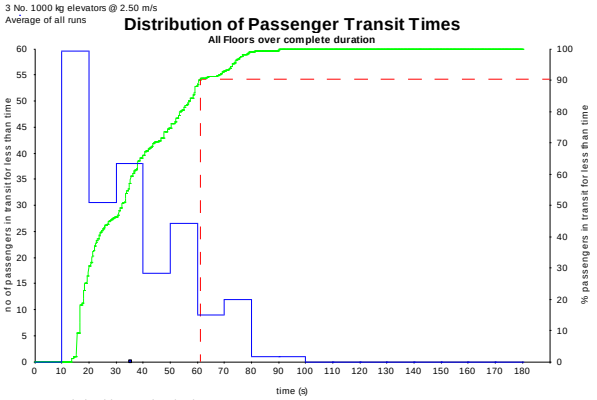
40 plantas, 21 ascensores 22,5% POP



Tiempo medio de tránsito (ATT)

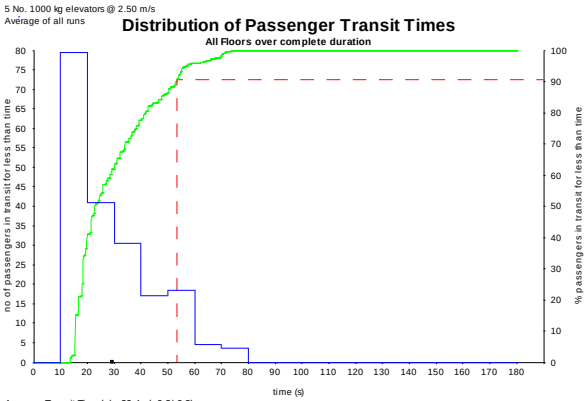
Búsqueda Tabú

12 plantas, 3 ascensores 15% POP



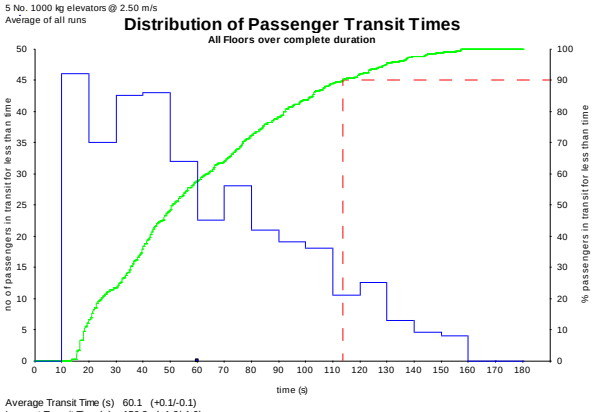
Average Transit Time (s) 35.7 (+0.0/-0.0)
Longest Transit Time (s) 84.5 (+5.7/-5.7)

12 plantas, 5 ascensores 20% POP



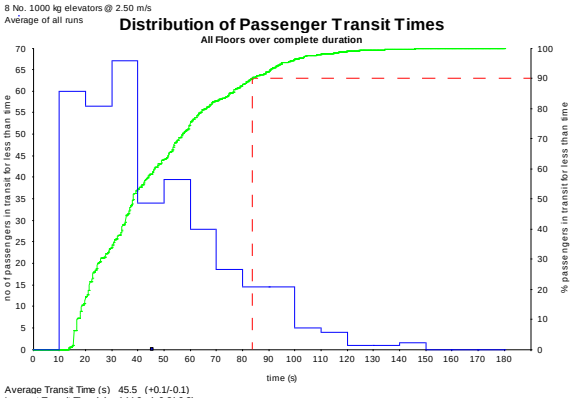
Average Transit Time (s) 29.4 (+0.3/-0.3)
Longest Transit Time (s) 73.0 (+1.4/-1.4)

20 plantas, 5 ascensores 15% POP



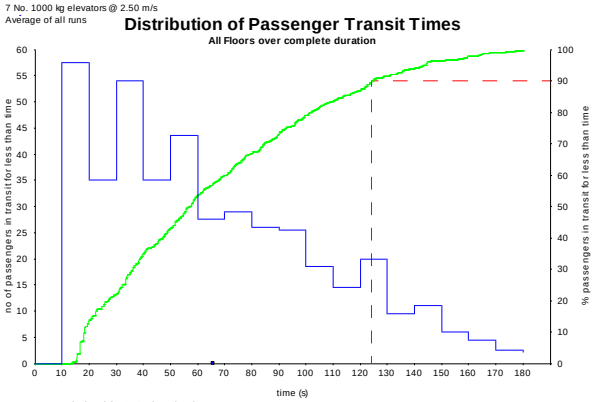
Average Transit Time (s) 60.1 (+0.1/-0.1)
Longest Transit Time (s) 156.3 (+1.0/-1.0)

20 plantas, 8 ascensores 20% POP



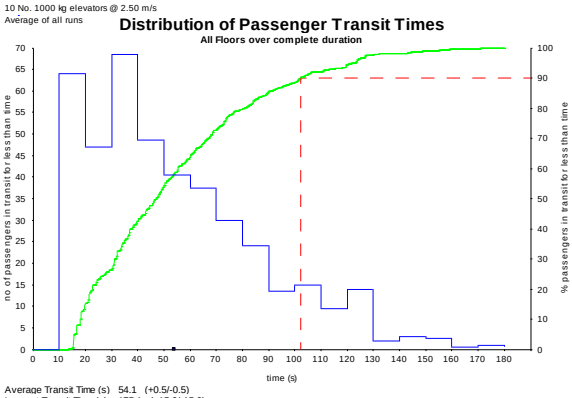
Average Transit Time (s) 45.5 (+0.1/-0.1)
Longest Transit Time (s) 144.9 (+0.8/-0.8)

24 plantas, 7 ascensores 15% POP



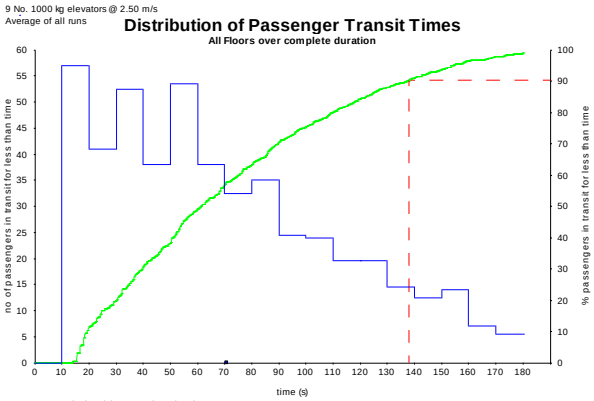
Average Transit Time (s) 65.9 (+1.7/-1.7)
Longest Transit Time (s) 220.7 (+40.5/-40.5)

24 plantas, 10 ascensores 20% POP



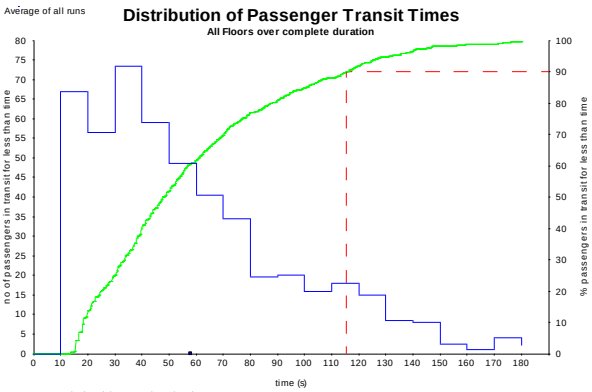
Average Transit Time (s) 54.1 (+0.5/-0.5)
Longest Transit Time (s) 175.1 (+15.0/-15.0)

28 plantas, 9 ascensores 15% POP



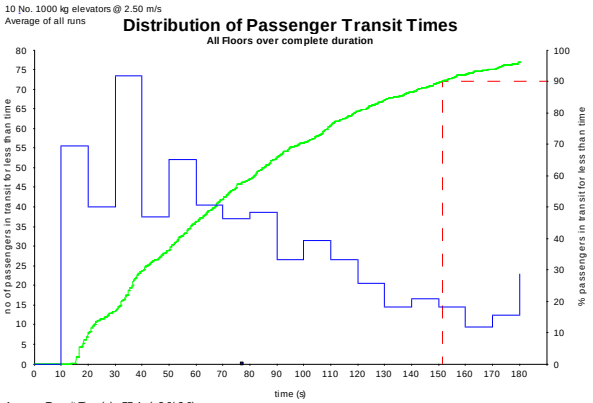
Average Transit Time (s) 70.9 (+3.3/-3.3)
Longest Transit Time (s) 210.6 (+3.3/-3.3)

28 plantas, 13 ascensores 20% POP



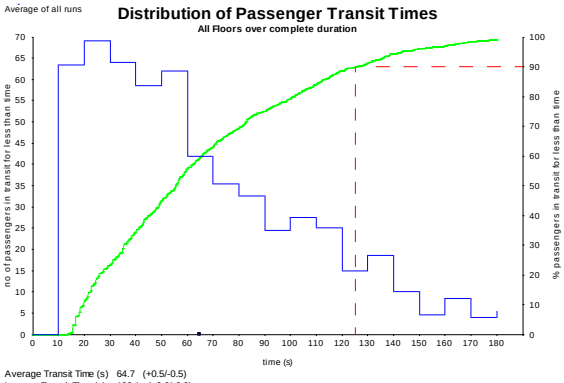
Average Transit Time (s) 58.1 (+6.3/-6.3)
Longest Transit Time (s) 167.5 (+19.6/-19.6)

32 plantas, 10 ascensores 15% POP



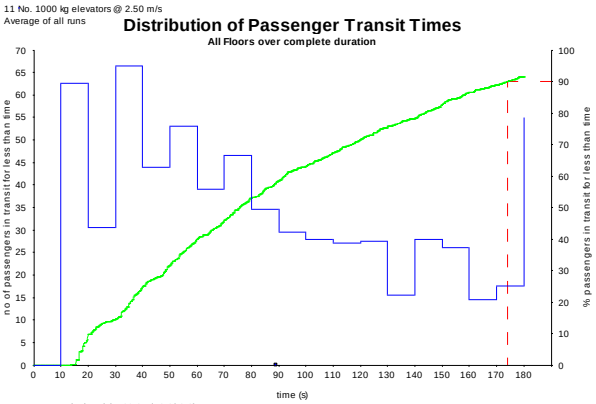
Average Transit Time (s) 77.4 (+2.6/-2.6)
Longest Transit Time (s) 240.1 (+2.4/-2.4)

32 plantas, 14 ascensores 20% POP



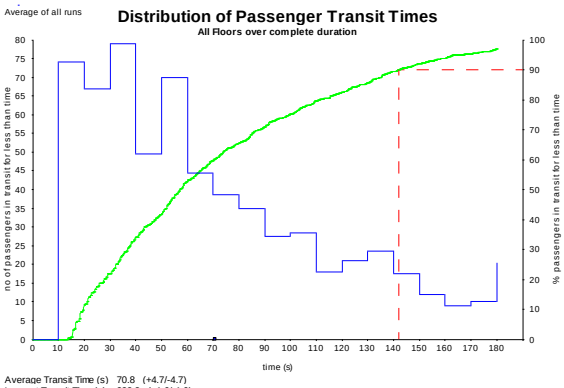
Average Transit Time (s) 64.7 (+0.5/-0.5)
Longest Transit Time (s) 198.1 (+2.0/-2.0)

36 plantas, 11 ascensores 15% POP



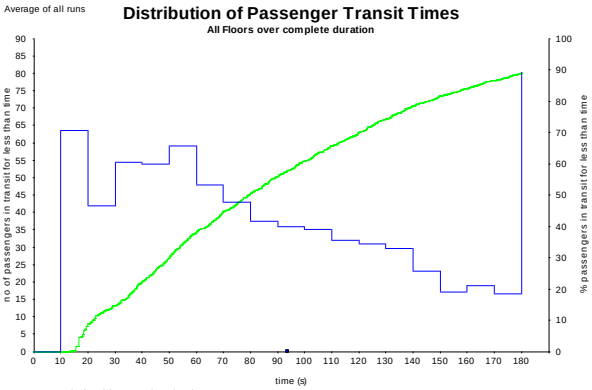
Average Transit Time (s) 89.2 (+2.3/-2.3)
Longest Transit Time (s) 271.6 (+5.6/-5.6)

36 plantas, 16 ascensores 20% POP



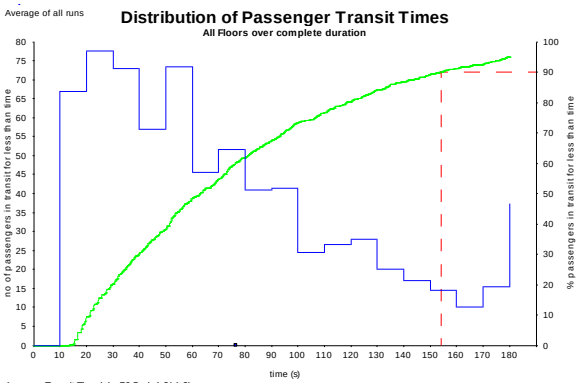
Average Transit Time (s) 70.8 (+4.7/-4.7)
Longest Transit Time (s) 228.8 (+1.9/-1.9)

40 plantas, 13 ascensores 15% POP



Average Transit Time (s) 93.8 (+0.7/-0.7)
Longest Transit Time (s) 312.8 (+14.5/-14.5)

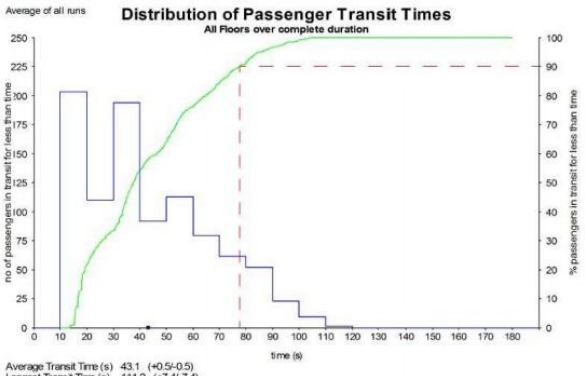
40 plantas, 19 ascensores 20% POP



Average Transit Time (s) 76.5 (+1.2/-1.2)
Longest Transit Time (s) 271.1 (+10.3/-10.3)

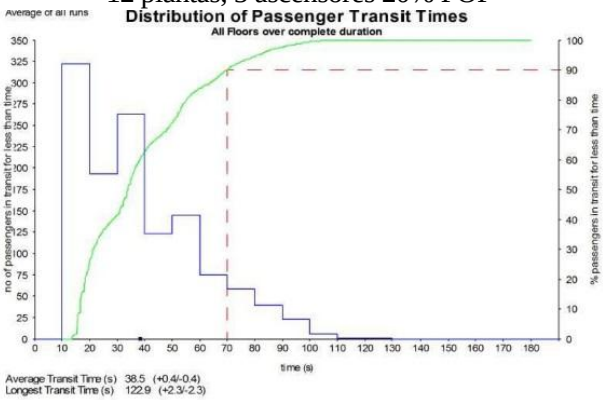
Algoritmo Genético

12 plantas, 3 ascensores 15% POP



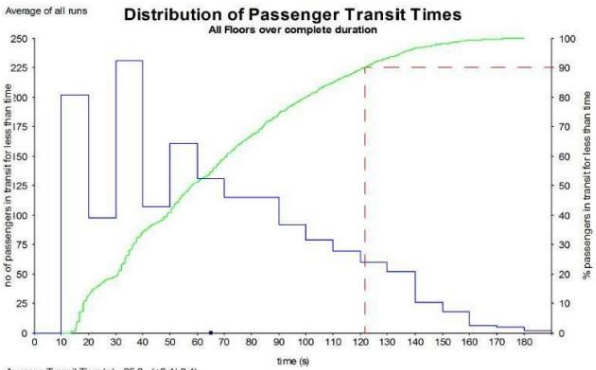
Average Transit Time (s) 43.1 (+0.5/-0.5)
Longest Transit Time (s) 111.2 (+7.4/-7.4)

12 plantas, 5 ascensores 20% POP



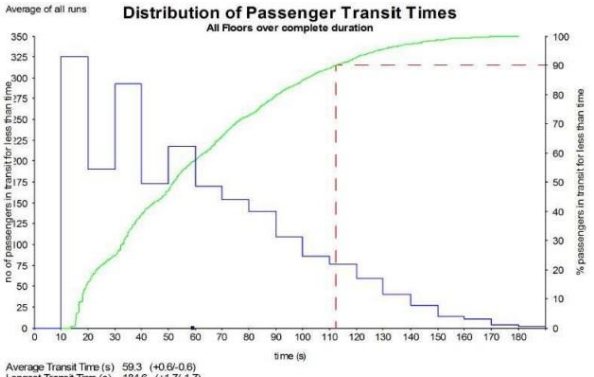
Average Transit Time (s) 38.5 (+0.4/-0.4)
Longest Transit Time (s) 122.9 (+2.3/-2.3)

20 plantas, 5 ascensores 15% POP



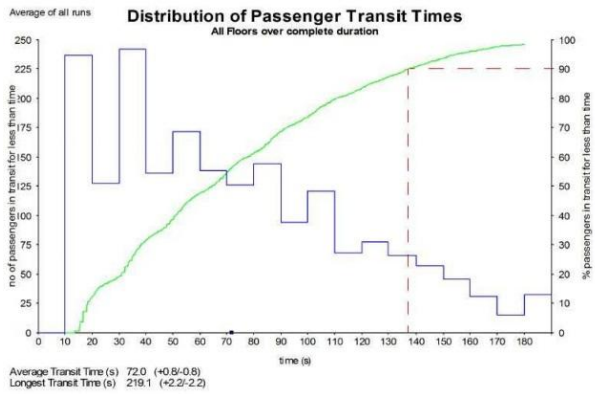
Average Transit Time (s) 65.3 (+0.4/-0.4)
Longest Transit Time (s) 200.6 (+0.0/-0.0)

20 plantas, 8 ascensores 20% POP

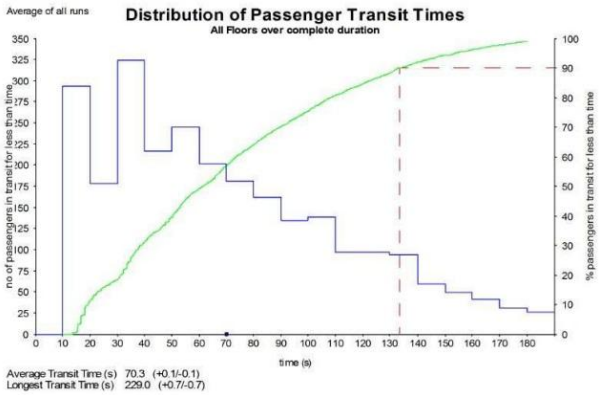


Average Transit Time (s) 59.3 (+0.6/-0.6)
Longest Transit Time (s) 184.6 (+1.7/-1.7)

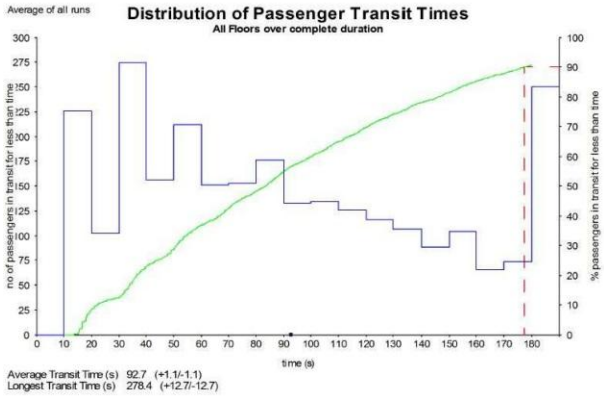
28 plantas, 9 ascensores 15% POP



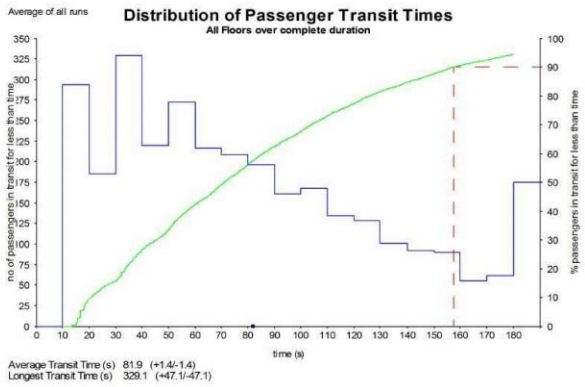
28 plantas, 13 ascensores 20% POP



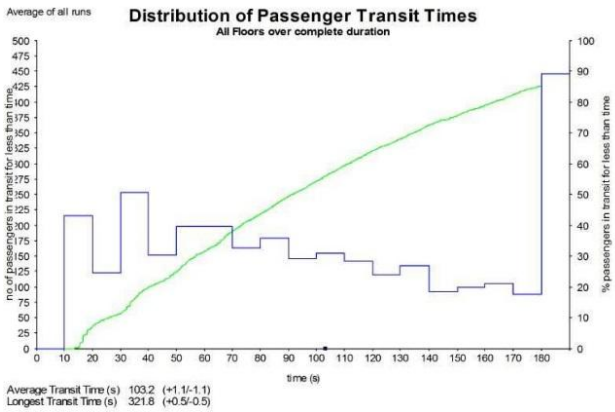
32 plantas, 10 ascensores 15% POP



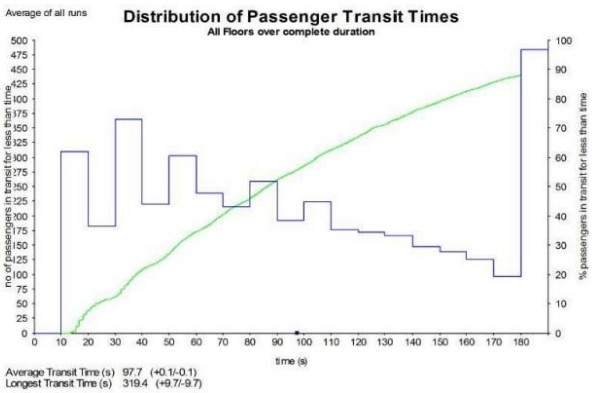
32 plantas, 14 ascensores 20% POP



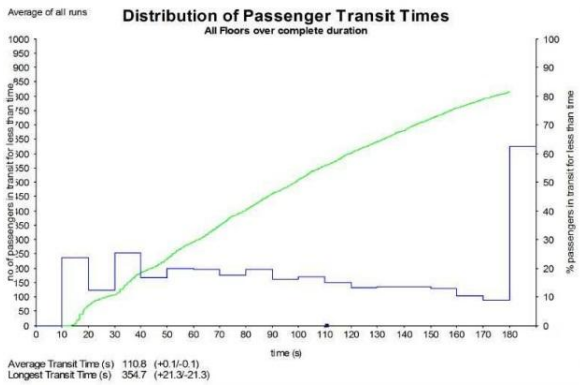
36 plantas, 11 ascensores 15% POP



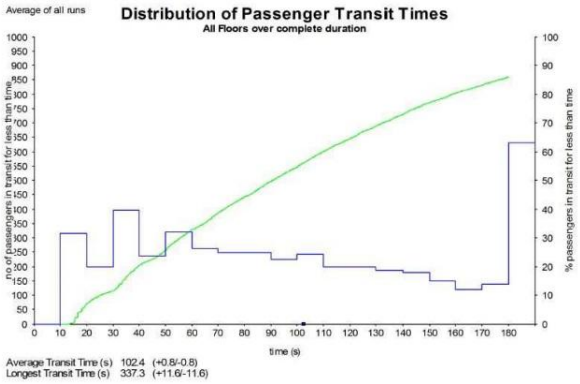
36 plantas, 16 ascensores 20% POP



40 plantas, 13 ascensores 15% POP

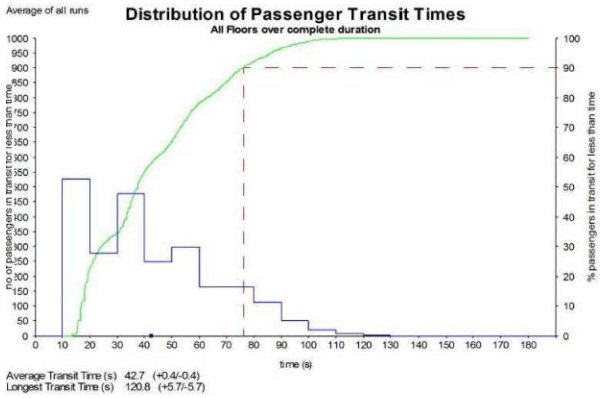


40 plantas, 19 ascensores 20% POP

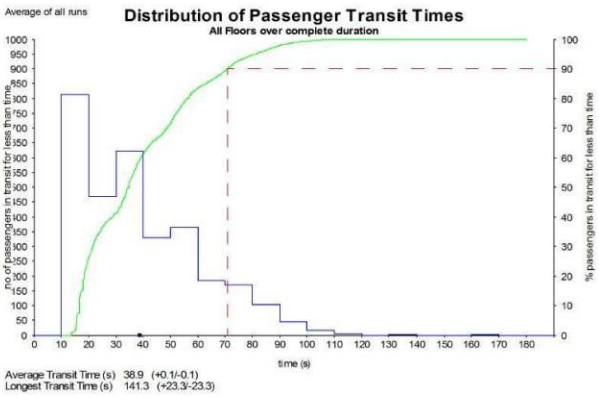


Algoritmo Genético Modificado

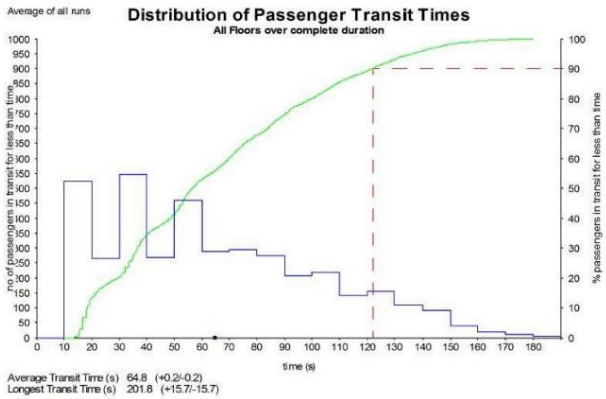
12 plantas, 3 ascensores 15% POP



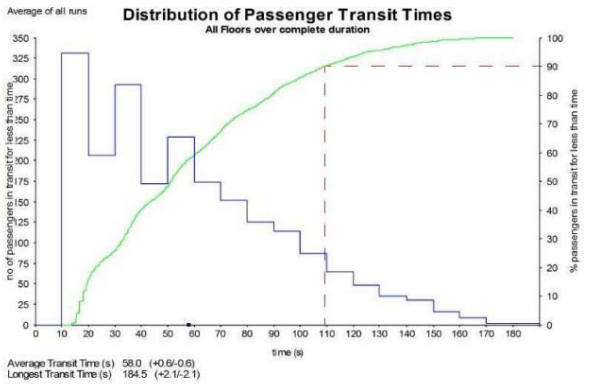
12 plantas, 5 ascensores 20% POP



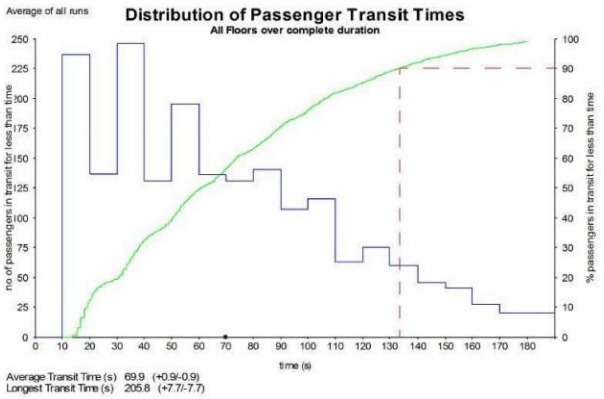
20 plantas, 5 ascensores 15% POP



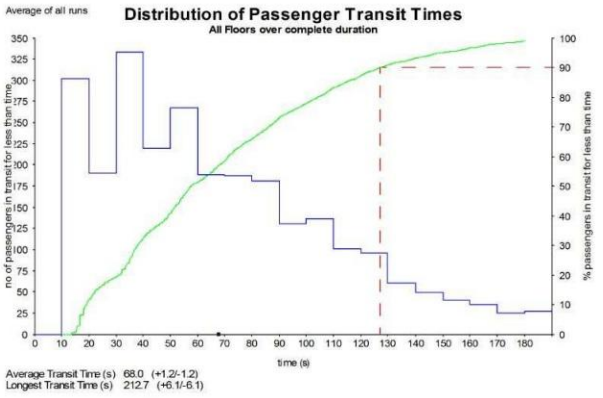
20 plantas, 8 ascensores 20% POP



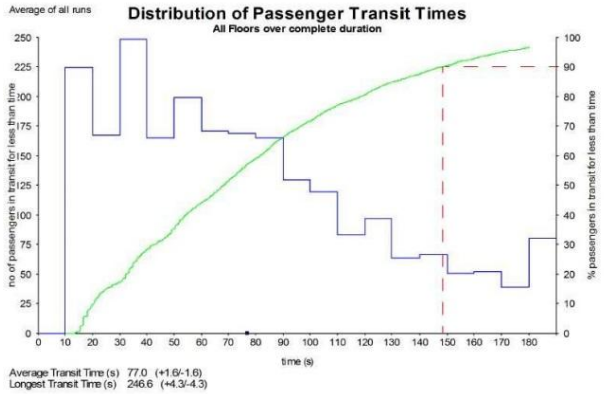
24 plantas, 7 ascensores 15% POP



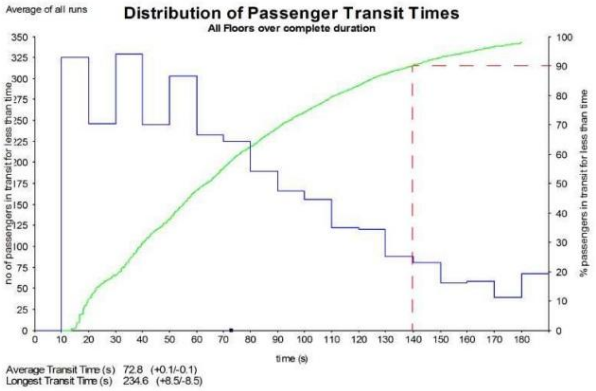
24 plantas, 10 ascensores 20% POP



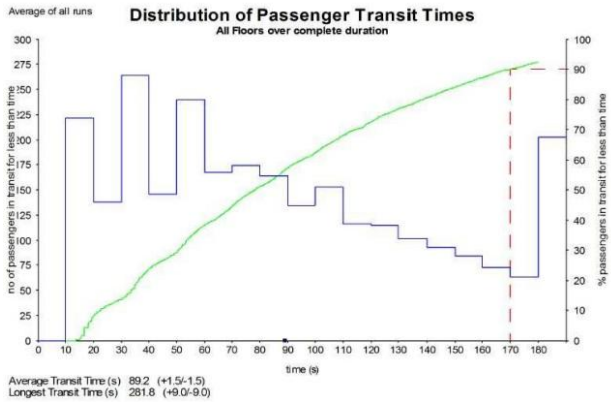
28 plantas, 9 ascensores 15% POP



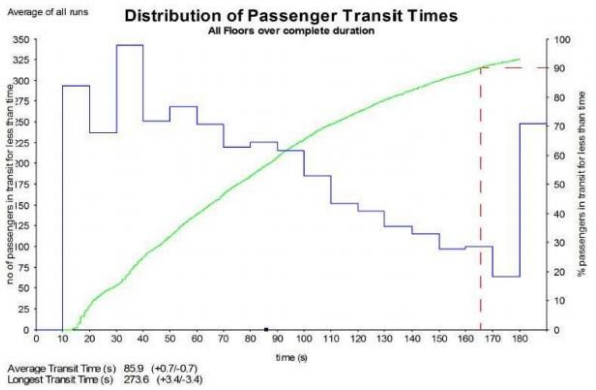
28 plantas, 13 ascensores 20% POP



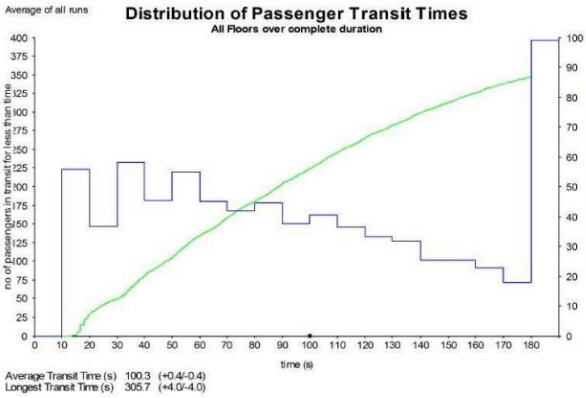
32 plantas, 10 ascensores 15% POP



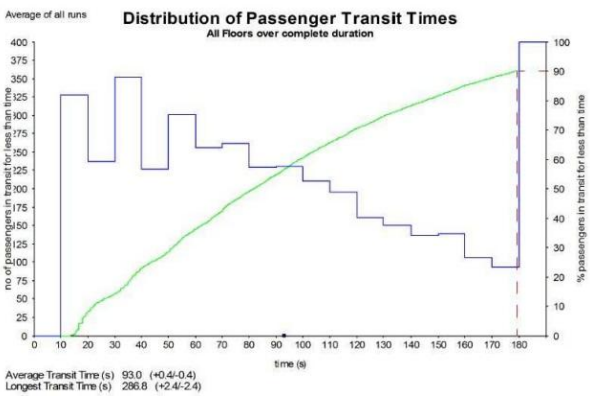
32 plantas, 14 ascensores 20% POP



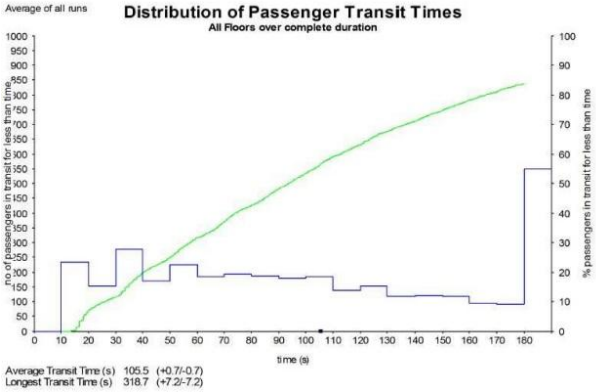
36 plantas, 11 ascensores 15% POP



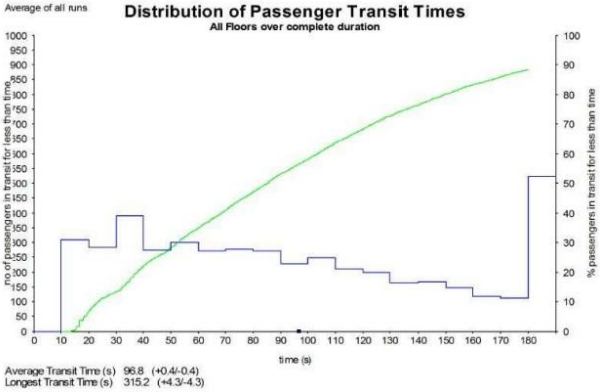
36 plantas, 16 ascensores 20% POP



40 plantas, 13 ascensores 15% POP

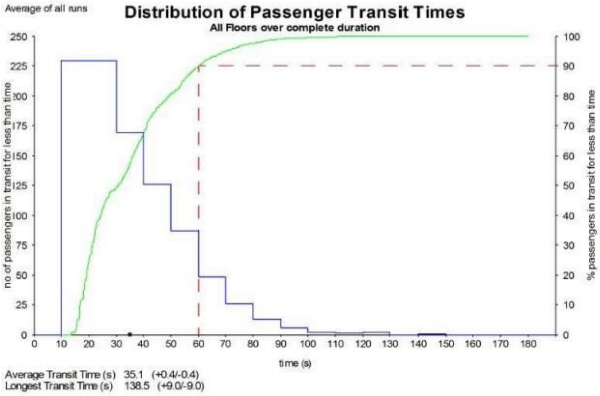


40 plantas, 19 ascensores 20% POP

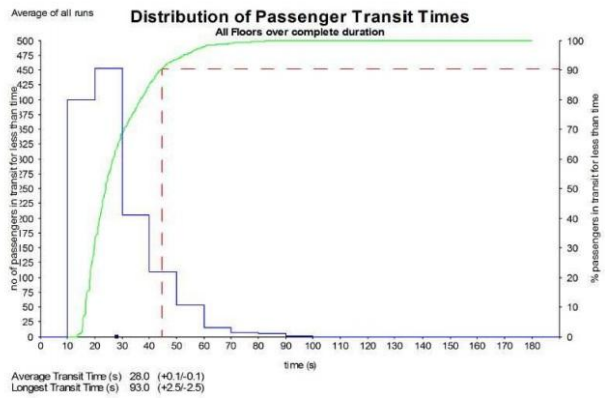


Double Deck Destination Control

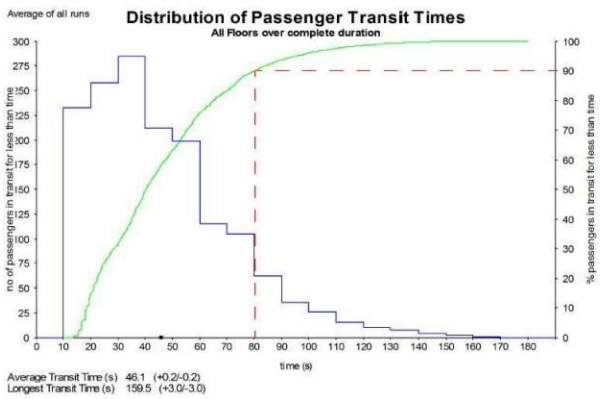
12 plantas, 3 ascensores 15% POP



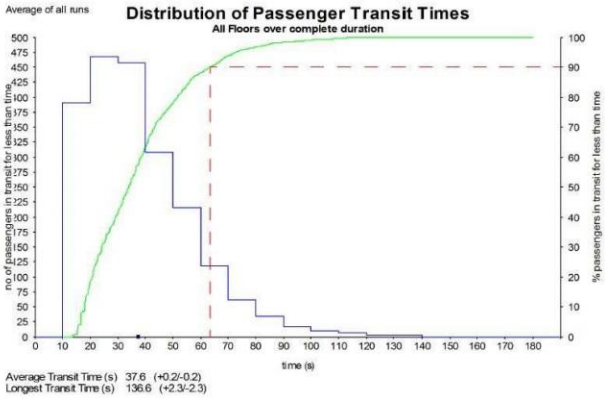
12 plantas, 5 ascensores 20% POP



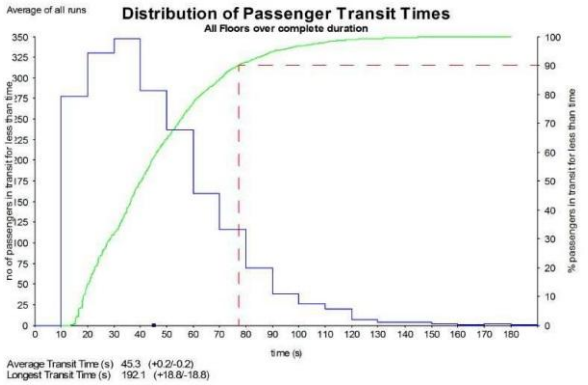
20 plantas, 5 ascensores 15% POP



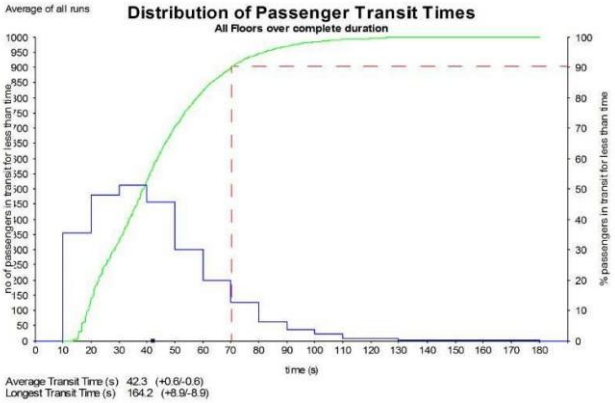
20 plantas, 8 ascensores 20% POP



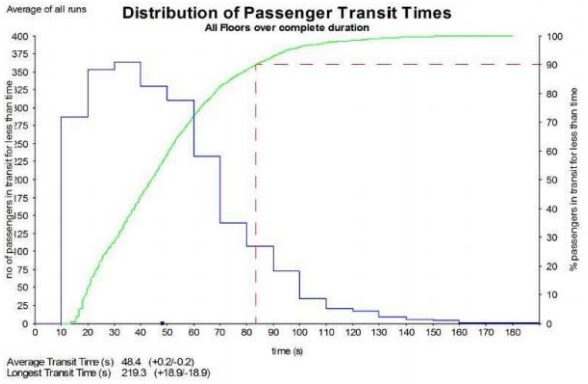
24 plantas, 7 ascensores 15% POP



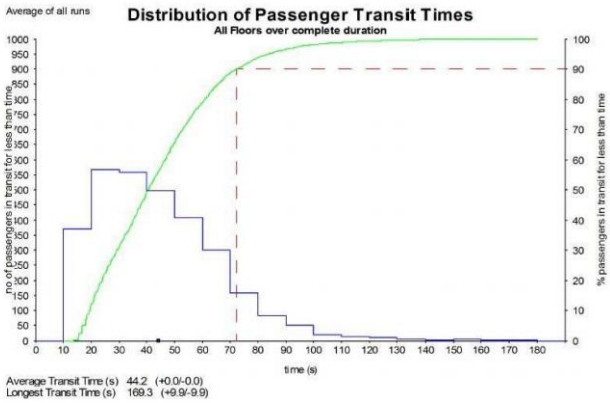
24 plantas, 10 ascensores 20% POP



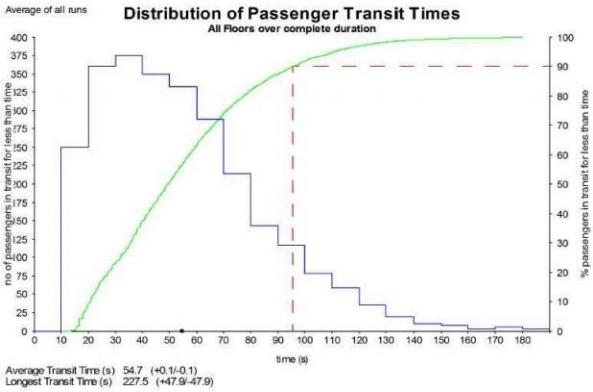
28 plantas, 9 ascensores 15% POP



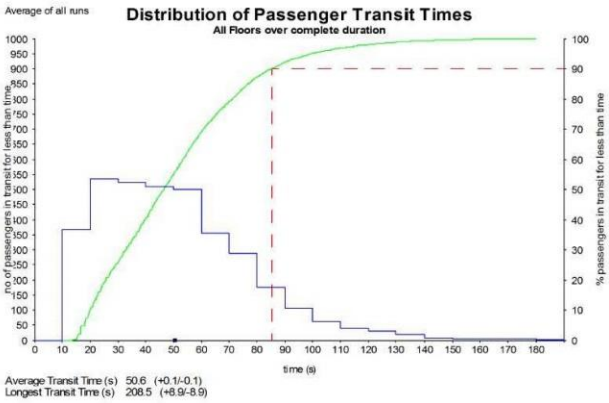
28 plantas, 13 ascensores 20% POP



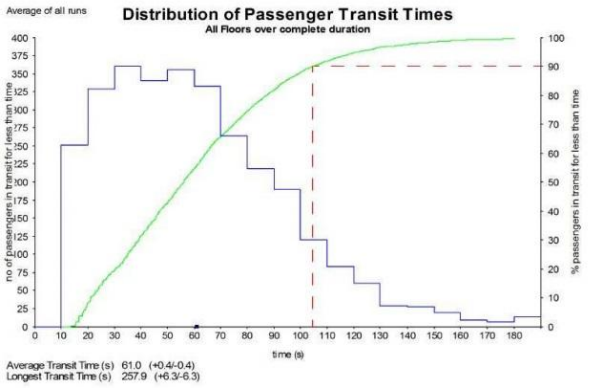
32 plantas, 10 ascensores 15% POP



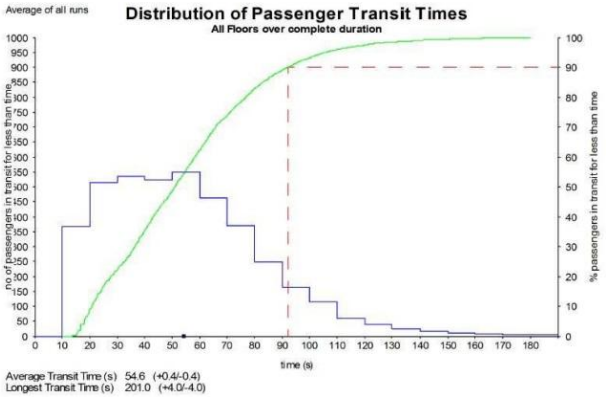
32 plantas, 14 ascensores 20% POP



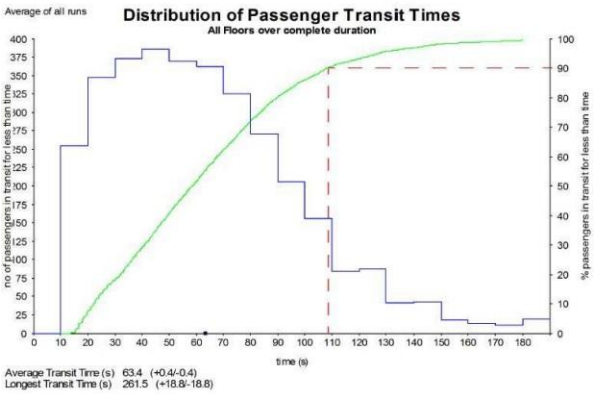
36 plantas, 11 ascensores 15% POP



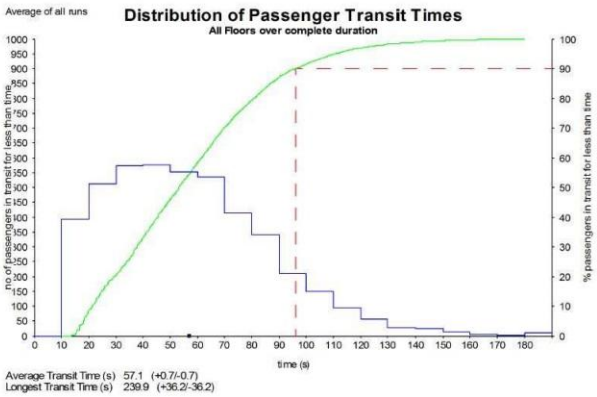
36 plantas, 16 ascensores 20% POP



40 plantas, 13 ascensores 15% POP



40 plantas, 19 ascensores 20% POP



ANEXO B: VARIABLES DE OBJETO

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

Se tiene un primer objeto cuyo nombre es "*lift.h*" en el cual se encuentran las variables y características asociadas al ascensor. Entre las variables más comunes que se encuentran en este objeto se tiene el estado de las puertas del ascensor, la capacidad total del ascensor, la capacidad actual, la velocidad y dirección del ascensor. Además, como se trabajan con ascensores de arquitectura *Double Deck* cada una de las variables anteriores dependen de si la cabina es superior o inferior.

El segundo objeto con el que se trabaja es el objeto "Dispatch.h" en el cual se encuentran las variables y características asociadas al edificio y a 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 "*Dispatch.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__INCLUDED_
)
#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__INCLUDED_
)
```