UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Tour Planner BackEnd

Presentado por Ignacio Aparicio Blanco
en Universidad de Burgos — 7 de febrero de 2020
Tutor: Bruno Baruque Zanon

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

A la hora de realizar un proyecto es imprescindible dedicar tiempo a la primera fase del mismo, la planificación. Es una fase importante ya que de ella depende el alcanzar los objetivos propuestos.

En este anexo se detallarán las tareas realizadas así como el tiempo dedicado a cada una de estas con el fin de acortar el tiempo de producción del proyecto.

Por otro lado se tendrá en cuenta un punto de vista más económico para valorar la posibilidad de llevar el producto final a un ámbito comercial.

Nos centraremos en los siguientes puntos:

- *Planificación temporal* se estudiara la gestión del tiempo que se ha llevado a lo largo del proyecto y las reuniones en las que se realizó la división de tareas.
- *Estudio de viabilidad* se tratará si podría ser beneficioso realizar el proyecto.
    - *Viabilidad económica* se realizará un estudio sobre la posibilidad de llevar el proyecto a un ámbito comercial y las consecuencias que tendrían ciertos cambios.
    - *Viabilidad legal* se detallarán los detalles legales que tengan que ver con el proyecto.

A.2. Planificación temporal

Como se mencionó en la Memoria, se ha utilizado la metodología ágil Scrum [**?**]. Los pasos que se han seguido con esta metodología son:

- Se define un tiempo, en nuestro caso entre dos semanas y un mes, llamado sprint. Al comienzo de cada intervalo se decidieron las tareas a realizar y el tiempo invertido en cada tarea.
- Al final de cada sprint se tuvo otra reunión con el tutor (en un principio de forma presencial en la Universidad de Burgos y más adelante a través de Skype debido a mi traslado a Málaga) para poner en común el trabajo realizado, los problemas que surgieron y los siguientes pasos a seguir

El proyecto comenzó el 05/Abril/2019 y se dio por finalizado el 13/Febrero/2020. Cabe destacar que este tiempo no se ha dedicado íntegramente al proyecto, sino que se aprovechó también para acabar asignaturas pendientes y cursar un máster en Málaga.

La realización del proyecto no ha sido regular, sino que hay meses muy activos y otros menos. Esto se debe a complicaciones para realizar reuniones entre profesores y alumnos ya que cada alumno estaba en una ciudad y con horarios de trabajo diferentes. Por suerte fuimos capaces de encontrar fechas comunes a todos y pudimos realizar reuniones a través de Skype.

Sprint 1 (05/04/19 - 19/4/19)

Primera reunión del proyecto. El tutor nos indico cuales iban a ser los objetivos del proyecto a realizar. Ese mismo día mi compañero Jesus Manuel y yo decidimos sobre que parte del proyecto íbamos a trabajar cada uno. Coincidimos en que yo me encargaría del *backend* mientras el trabajaría sobre el *frontend* Se indicaron las siguientes tareas:

- Documentarse sobre el servidor Glassfish
- Leer la información sobre le proyecto del que partíamos para hacernos una idea de sobre que íbamos a trabajar.
- Comenzar la memoria y realizar los objetivos iniciales para poder empezar cuanto antes a trabajar.

Sprint 2 (19/4/19 - 03/05/19)

Se comentó que conocimiento se tenían sobre las partes en las que se iban a trabajar, en mi caso, bases de datos y servicios REST en servidores. Aunque si que había trabajado con bases de datos anteriormente en la carrera nunca había trabajado sobre un servidor, por lo que se decidió centrarse en ese punto.

Las tareas propuestas para el sprint fueron las siguientes:

- Realizar tutoriales sobre Glassfish y hacer pequeñas pruebas.
- Documentarse sobre las bases de datos geo-espaciales OSM y Google Maps.
- Instalar en una partición el sistema operativo Ubuntu 18.04.

Sprint 3 03/05/19 - 17/05/19

Tras poner en común lo aprendido en el sprint anterior, se debatieron las ventajas e inconvenientes de cada una de las bases de datos geo-espaciales de la que se iban a obtener los datos. Se decidió utilizar OSM debido a las restricciones de Google Maps.

En esta reunión se trato también el tema de los algoritmos. El tutor explicó lo que era el problema base sobre el que íbamos a tratar (problema de la orientación - OP) y el sub-problema sobre el que se iba a trabajar (problema de la orientación con problemas de tiempo - OPTW).

Se acordaron los siguientes objetivos para el sprint:

- Instalar las máquinas virtuales del proyecto del que partíamos y probar que todo funcionase correctamente.
- Estudiar el formato de los datos referentes a las ventanas de tiempo de los diferentes nodos de OSM.
- Descargar datos sobre diferentes ciudades en OSM y cargar estos en una base de datos local para poder obtener los datos de tiempo y analizarlos.
- Estudiar diferentes algoritmos y elegir dos a implementar.
- Continuar trabajando en la memoria.

Sprint 4 (17/05/19 - 07/06/19)

Se comentaron las diferentes opciones de algoritmos que se encontraron y se decidió trabajar sobre dos completamente diferentes:

- Algoritmo iterativo
- Algoritmo de colonia de hormigas

Debido a no haber trabajado nunca con el último tipo de algoritmo, se propuso comenzar a implementar el algoritmo iterativo para entender mejor el problema sobre el que se iba a trabajar, y de ese modo hacer un poco más sencilla la implementación del segundo algoritmo.

Las tareas para el sprint fueron:

- Cargar los datos de la ciudad de burgos en la base de datos.
- Implementar el algoritmo iterativo.
- Encontrar sets de datos de prueba para poder probar las diferentes partes del algoritmo según se va implementando.

Sprint 5 (07/06/19 - 05/07/19)

Debido a no haber implementado nunca un algoritmo parecido, no se terminó de implementar el algoritmo iterativo, por lo que esa tarea se mantuvo para el sprint 5. Por otro lado, se determinó que era posible acabar de implementar el algoritmo iterativo y , al menos, comenzar a implementar el segundo algoritmo.

Las tareas para el sprint fueron:

- Acabar de implementar el algoritmo iterativo.
- Implementar el algoritmo de colonia de hormigas.

Sprint 6 (05/07/19 - 03/09/19)

Última reunión anterior al verano, meses durante los que iba a ser complicado tener más reuniones. En el sprint anterior se consiguió acabar de implementar ambos algoritmos. Aunque si se realizaron pruebas que

determinaron que los dos algoritmos resolvían el problema de forma correcta, no hubo tiempo para realizar pruebas de eficiencia.

A su vez, antes del sprint se trató de conectar el cliente (proyecto de Jesús Manuel Calvo Ruiz de Temiño) y el servidor, pero sin éxito.

Las tareas para el sprint fueron las siguiente:

- Realizar pruebas de eficiencia de los algoritmos.
- Encontrar el problema en la conexión de cliente y servidor.
- Trabajar en la memoria del proyecto.

Sprint 7 (03/09/19 - 04/11/19)

Primera reunión tras los meses de verano en la que se comentaron los avances realizador. Tratando el tema de conexión, se arregló un problema con el servidor que, a pesar de su baja complicación, llevó mucho mas tiempo del esperado, ya que los errores del log del servidor en Glassfish NO indicaban correctamente el error, por lo que la depuración fue complicada.

Se decidió documentar los errores y cambios en la memoria así como hacer una máquina virtual con el servidor y la base de datos ya que aunque no era posible conectarlo con el nuevo cliente si se podía hacer la conexión con el antiguo, aunque este no pudiese acceder a los nuevos algoritmos implementados.

Se plantea también buscar nuevas formas de seguir trabajando en el proyecto.

Las tareas fueron las siguientes:

- Continuar trabajando en la memoria.
- Hacer una máquina virtual con Ubuntu 18.04 y todo lo necesario para un correcto funcionamiento del servidor.
- Buscar nuevas formas de mejorar el proyecto.

Sprint 8 (04/11/19 - 18/11/19)

La reunión tuvo lugar por Skype, al igual que todas las que se tendrían a partir de este momento, debido a que , por continuar con mis estudios, tuve

que trasladarme a Málaga de forma permanente, por lo que era imposible tener reuniones presenciales.

Por otro lado apareció un problema con los horarios ya que Jesús Manuel trabajaba por las mañanas mientras que yo tenía que cursar clases del máster por las tardes. Por suerte fuimos capaces de encontrar una fecha para el siguiente sprint.

Por último, debido a los meses que restaban para la entrega del proyecto se decidió implementar un nuevo algoritmo.

Las tareas para el sprint fueron las siguientes:

- Estudiar diferentes algoritmos para implementar.
- Comenzar a implementar el nuevo algoritmo.

Sprint 9 (18/11/19 - 02/12/19)

Se pusieron en común diferentes posibles algoritmos a implementar.

Se decidió implementar un algoritmo genético, ya que no queríamos trabajar en uno que fuese demasiado parecido a alguno de los que ya había implementado anteriormente y que, como nunca había trabajado con algoritmos genéticos, podía tomármelo como un nuevo reto y aprender más.

La tarea del sprint fue la siguiente:

- Implementar el algoritmo genético OPTW.

Sprint 10 (02/12/19 - 16/12/19)

Debido a entregas en el máster no pude trabajar demasiado en este sprint, por lo que el algoritmo aún no estaba implementado del todo correctamente. Se mantuvo la tarea del sprint anterior:

- Continuar trabajando en el algoritmo genético.

Sprint 11 (16/12/19 - 13/01/20)

Reunión anterior a navidad, por lo que teníamos que establecer que se iba a hacer en esas fechas.

Seguía habiendo un error de conexión del que nadie estaba del todo seguro si la causa era el cliente o el servidor (o ambos).

Por otro lado, el algoritmo genético no era del todo eficiente, por lo que se propuso intentar nuevos métodos de *selección* y *crossover* y comparar los resultados obtenidos con los del resto de algoritmos.

Las tares por lo tanto fueron las siguientes:

- Analizar de donde provenía el error de conexión.
- Mejorar el algoritmo genético.
- Realizar test de eficiencia de los diferentes algoritmos OPTW.

Sprint 12 (13/01/20 - 07/02/20)

Penúltima reunión antes de la entrega del proyecto. Después de poner en común las diferentes pruebas realizadas se estableció que el problema de conexión derivaba del cliente y no del servidor.

El mayor de los trabajos restantes era acabar la memoria para que los tutores pudiesen revisarla a tiempo para hacer mejoras en caso necesario.

- Finalizar la memoria.
- Buscar *code smells* para depurar código.

Sprint 13 (07/02/20 - Fin de Proyecto)

Ultima reunión del proyecto. Se ponen en común los resultados y las opiniones sobre la memoria, de la que se propusieron algunos cambios.

- Últimos cambios en la depuración del código.
- Mejoras en la memoria.
- Realizar la pancarta para la presentación del proyecto.

A.3. Estudio de viabilidad

En este apartado se estudiará la viabilidad del proyecto, desde un punto de vista tanto económico como legal, teniendo en cuenta las diferentes acciones que se pueden tomar durante el desarrollo del proyecto así como otros factores de importancia tales como el dinero disponible, el conocimiento previo al inicio del proyecto, el tiempo disponible, el tiempo necesario etc.

Se trata de un punto crítico para un correcto desarrollo de cualquier proyecto ya que de los resultados que se obtengan se determinará si el proyecto es viable, y por tanto merece la pena seguir adelante, o no.

Viabilidad económica

En esta sección se estudiarán y detallaran los diferentes costes desglosados que conllevaría la realización del proyecto.

Análisis de costes

Se tendrán en cuenta cuatro tipos de costes diferentes:

- Costes de personal
- Costes de hardware
- Costes de software
- Otros costes

Aunque sería posible no incluir el último apartado, *otros gastos*, en el que se incluiría renta de un lugar de trabajo, servicios de Internet, electricidad, material de oficina etc, debido a la posibilidad de obtener el apoyo de una *incubadora de empresas*, que suelen quedarse al acabar el desarrollo con un 5 % de los beneficios del proyecto.

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

B.2. Objetivos generales

B.3. Catalogo de requisitos

B.4. Especificación de requisitos

Apéndice C

Especificación de diseño

C.1. Introducción

Los apartados que se estudiarán en esta sección son los siguientes:

- **Diseño de datos**
- **Diseño procedimental**
- **Diseño arquitectónico**

C.2. Diseño de datos

Diseño de base de datos geo-espacial

Como se ha comentado en puntos anteriores, se ha utilizado una base de datos geo-espacial, más específicamente *OpenStreetMaps*. Debido a que los nuevos algoritmos no solo necesitan datos referentes a los puntos y su posición, sino que requieren de las ventanas de tiempo de los establecimientos, se ha añadido una columna adicional que contenga esta información.

Respecto al resto de la estructura de la base de datos, se ha utilizado la implementada en versiones anteriores de este proyecto.

Modelo de datos

AC: Decir que se ha cambiado X para implementar las ventanas de tiempo

Tablas

AC: Parecido a lo anterior

Estructura de paquetes (Servidor)

Como en versiones anteriores del proyecto, los paquetes del servidor se encuentran divididos en dos secciones claras:

- Main
- Test

Cada uno de estas secciones contienen a su vez sub-paquetes, de los que se destacarán a continuación aquellos en los que se han realizado cambios para implementar la funcionalidad correspondiente a el calculo de rutas teniendo en cuenta las ventanas de tiempo.

Paquete *main*

Se trata del paquete principal de la aplicación, que a su vez contiene 7 **AC: Cambiar numero y paquetes** paquetes más. Estos paquetes son: *algorithms, crypt, database, model, performance, resource, resource.request, resource.response, router, util*

Paquete algorithmsTW

Paquete modelTW

Paquete *test*

Este paquete contiene los test unitarios realizados para llevar un buen control del funcionamiento de los algoritmos.

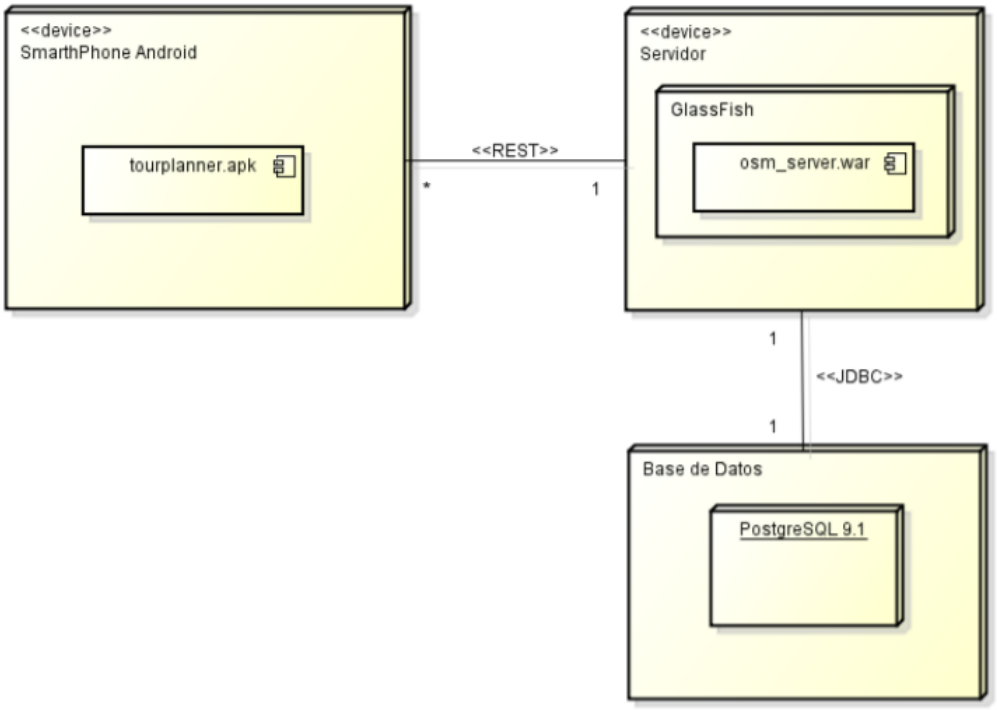
Diagrama de despliegue

En este apartado se muestra el diagrama de despliegue de la aplicación. Cabe destacar que debido a que el funcionamiento de la aplicación es el mismo que en los proyectos anteriores, también lo es el diagrama de despliegue.

Podemos dividir dicho funcionamiento en tres bloques:

- **Lanzamiento de petición (cliente)**: el cliente hará una petición dese la aplicación a través de una petición REST.

- **Atención de la petición:** la petición será atendida por un *servlet* ejecutándose en un contenedor de aplicaciones *Glassfish*.
- **Consulta**: Para atender la petición, el servidor realizará una consulta a una base de datos *PostgreSQL*, a través de un conector *JDBC*.



C.3. Diseño procedimental

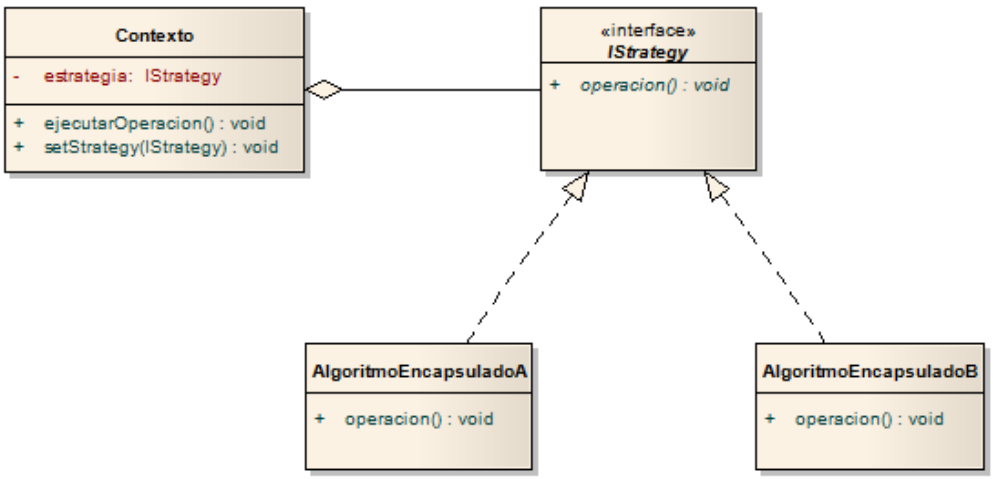
AC: Preguntar a bruno que pongo yo aqui

C.4. Diseño arquitectónico

Patrón Strategy

El patrón Strategy (Estrategia en español) es un patrón de diseño que pertenece a la categoría de *behavioral patterns* (patrones de comportamiento), cuya función es identificar patrones de comunicaciones entre diferentes objetos y realizar correctamente el intercambio de mensajes entre ellos.

El patrón Strategy permite encapsular los algoritmos en clases y elegir cual queremos utilizar en tiempo de ejecución.



Este patrón se ha utilizado para seleccionar que algoritmo queremos que se ejecute : Chao et al, GRASP, Iterativo, Colonia de hormigas o Genético.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En este anexo se describirá la documentación técnica de programación, en la que se incluyen la estructura de directorios, una explicación de las bibliotecas y herramientas utilizadas, la instalación de las mismas y los algoritmos implementados.

En algunos apartados es posible que se haga referencias al cliente, desarrollado por mi compañero Jesús Manuel Calvo Ruiz de Temiño. Se puede encontrar más información del cliente en su memoria del proyecto.

La mayor parte del proyecto se basa en la implementación de algoritmos que resuelvan un nuevo problema de cálculo de rutas: el problema de la orientación con ventanas de tiempo (OPTW).

Es posible que se hagan referencias a apartados del proyecto que no han sido modificados en esta versión pero que son necesarios para una completa comprensión del mismo.

D.2. Estructura de directorios

En esta sección se detallará la estructura de los directorios que se pueden encontrar en la memoria USB entregada junto con la memoria del proyecto, así como en el repositorio de *github* del mismo.

- **Cliente**
  - Aplicación: carpeta en la que se encuentra el archivo *TourPlanner.apk*, que contiene la aplicación cliente lista para ser instalada en el dispositivo.
  - Código fuente del proyecto: carpeta en la que se encuentra el código de la aplicación cliente.
  - Javadoc: carpeta en la que se encuentra la documentación sobre el código del cliente.
  - Datos: carpeta que contiene un backup de la base de datos utilizada en el proyecto.
  - Máquina virtual: carpeta que contiene una máquina virtual con sistema operativo Windows donde se encuentra listo para ser ejecutada la aplicación cliente.
  - Documentación:
    - PDF: contiene la memoria y anexos en formato .pdf
    - LaTex contiene la memoria y anexos en formato latex.
- **Servidor**
  - Aplicación: carpeta en la que se encuentra el archivo osm_server.war que deberá ser cargado en el servidor.
  - Código fuente del proyecto: carpeta que contiene el código java del servidor.
  - Javadoc: carpeta en la que se encuentra la documentación sobre el código del servidor.
  - Datos: carpeta en la que se encuentra un backup de la base de datos utilizada en el proyecto.
  - Software: carpeta que contiene distintos programas/bibliotecas que se han utilizado en el proyecto.
  - Máquina virtual: carpeta que contiene una máquina virtual con el sistema operativo *Ubuntu 18.04* y el servidor listo para ser ejecutado.
  - Documentación
    - PDF: contiene la memoria y anexos en formato .pdf
    - LaTex contiene la memoria y anexos en formato latex.

D.3. Manual del programador

Algoritmos para el cálculo de rutas

En esta sección se detallará todo lo necesario para una completa comprensión de los algoritmos de cálculo de rutas implementados en el proyecto.

Descripción del problema

Una de las mayores preocupaciones del problema de la orientación (a partir de este momento OP) es que dado que *Golden et al* probó que, hablando de complejidad algorítmica, OP es de complejidad NP es muy improbable que el problema de la orientación con ventanas de tiempo (a partir de ahora OPTW) se pueda resolver en tiempo polinómico.

Al tratarse de un problema que requiere ser resuelto en segundos, ya que está destinado a ser utilizado en dispositivos móviles en tiempo real se trata de un gran inconveniente.

Algoritmo iterativo

A pesar del inconveniente descrito en la sección anterior, se ha seguido el modelo de *Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe y Dirk Van Oudheusden* para implementar un algoritmo basado en una heurística de búsqueda iterativa.

Heurística

La heurística utilizada se basa en dos pasos: *inserción* y *shake*. En la *inserción* se construye una ruta comenzando por los puntos inicial y final definidos previamente intentando, mediante formulas matemáticas explicadas en el siguiente apartado, reducir el coste computacional de insertar nuevos puntos. En el *shake*, se pretende escapar del óptimo local reduciendo la ruta calculada e insertando nuevos puntos que no estaban presentes anteriormente.

Inserción

Primer paso a la hora de calcular una nueva ruta. Se basa en insertar nuevos puntos uno a uno.

Para reducir el coste computacional del algoritmo, **antes** de insertar un nuevo punto es necesario comprobar que el resto de visitas que tendrían lugar después del nuevo punto siguen cumpliendo la restricción de sus ventanas

de tiempo, de esta forma no probamos rutas que sabemos de antemano que no van a ser válidas.

Es necesario reducir lo máximo posible el cálculo de si una inserción es posible o no. Debido a que comprobar la validez de un nuevo punto en cada paso de la inserción es imposible que sea eficiente, se han utilizado unas variables auxiliares en cada punto de la ruta que almacenan dos datos imprescindibles para el buen funcionamiento del algoritmo: *Wait* y *MaxShift*.

Wait contiene el tiempo que se debe esperar en caso de que se llegue al punto **antes** de que se abra su ventana de tiempo.

$$Wait_i = max[0, O_i - a_i]$$

Siendo O_i la hora de apertura (opening) del punto i y a_i la hora de llegada (arrival) al punto i.

MaxShift contiene el máximo tiempo que la visita a un punto se puede retrasar sin hacer que el resto de visitas posteriores a ese punto se retrasen y no cumplan sus ventanas de tiempo. Gracias a esta variable se consigue que el algoritmo sea eficiente.

La fórmula utilizada para calcular el *MaxShift* de un punto es:

$$MaxShift_i = min[C_i - s_i, Wait_(i+1) + MaxShift_(i+1)]$$

Lo que esta formula quiere decir es que *MaxShift* de un punto será la suma de *Wait* y *MaxShift* del punto siguiente menos en el caso de que el propio punto esté limitado por su propia ventana de tiempo. Un ejemplo de esto puede ser el último punto de una ruta. Como no existe ningún punto posterior al último de la ruta, este estará limitado por la hora a la que comienza el servicio (no confundir con el *arrival*, ya que esta no tiene en cuenta el tiempo de espera *wait*, mientras que la hora a la que comienza el servicio SI lo tiene en cuenta) y la hora de cierre del establecimiento, dada por su ventana de tiempo.

Una vez que tenemos *MaxShift*, nos falta una forma eficiente de comprobar si el punto que se va a insertar cumple esa restricción.

La solución es otra variable que contendrá cada punto de la ruta: *Shift*.

Shift almacena el tiempo que consume insertar un nuevo punto entre un punto i y otro punto j. La fórmula que lo define es la siguiente:

$$Shift_j = c_i j + Wait_j + T_j + c_j k - c_i k$$

Siendo $c_y z$ el tiempo en recorrer el camino entre los puntos y z y T_j el tiempo de servicio de el punto j.

Para determinar cual de todos los puntos disponibles vamos a insertar, se calculará un ratio teniendo en cuenta el coste de insertar el punto (*Shift*) y la puntuación que nos aporta el punto. De esta forma nos aseguramos que un punto no es insertado solamente por ser el que menos cueste, ya que podríamos encontrarnos con una ruta muy larga pero con muy poco valor, ni por ser el que más puntuación nos de, ya que podríamos tener rutas con mucho valor pero muy poca duración.

La fórmula utilizada ha sido la siguiente:

$$Ratio_i = (S_i)^2 / Shift_i$$

Algorithm 1 Inserción

for each visita no incluida **do**
  Determinar mejor punto para insertar la visita
  Calcular *Shift*
  Calcular *Ratio*
end for
Insertar la visita con mayor *Ratio*(j);
j: Calcular *Arrive*, *Start*, *Wait*;
for each visita después de j **do**
  Actualizar *Arrive, Start, Wait, MaxShift, Shift*;
end for
j: actualizar *MaxShift*;
for each visita antes de j **do**
  Actualizar *MaxShift*;
end for

Actualización post-inserción Una vez se ha insertado un nuevo punto, es importante actualizar los valores de todas las visitas posteriores al punto insertado.

En vez de recalcular los valores con las formulas propuestas anteriormente, siguiendo el modelo de algoritmo propuesto por los autores mencionados anteriormente, se ha decidido actualizar los valores de los puntos mediante formulas que tengan menor coste computacional. Para poder hacer esto es necesario haber almacenado el valor *Shift* de cada punto para poder acceder a el en este proceso.

Los valores que se necesitan actualizar son los siguientes: *Shift*, *Wait*, *arrival (a)* (tiempo de llegada al punto), *comienzo del servicio (s)* y *MaxShift*. En la notación de las formulas que se encuentran a continuación se interpretará j como el punto insertado entre i y k.

$$Shift_j = c_i j + Wait_j + T_j + c_j k - c_i k$$

$$Wait_k = max[0, Wait_k - Shift_j]$$

$$a_k = a_k + Shift_j$$

$$Shift_k = max[0, Shift_j - Wait_k]$$

$$s_k = s_k + Shift_k$$

$$MaxShift_k = MaxShift_k + Shift_k$$

Las fórmulas se encuentran en orden de uso, si se implementan utilizando otro orden los resultados no serán correctos.

Este paso se repetirá para cada punto posterior a k, siguiendo el mismo formato.

Para todos los puntos anteriores al nuevo punto insertado j, se requerirá **recalcular** su *MaxShift* utilizando la fórmula propuesta en el apartado de *Inserción*.

Shake

Como se ha mencionado anteriormente, el paso de *shake* pretende escapar el óptimo local. El proceso se basa en quitar uno o más puntos de la ruta. Para ello se utilizarán dos enteros : R_d y S_d.

R_d indica cuantos puntos se van a quitar de la ruta mientras que S_d indica el punto de la ruta en el que se iniciará el proceso de *shake*.

Debido al cambio en ambos números entre rutas, aseguramos que las soluciones al problema sean muy diferentes y por tanto abrimos el abanico de posibilidades.

Algorithm 2 Shake

```
for each tour do
  Borrar set de visitas (i=>j);
  Calcular Shift;
  for each visita despues de j do
    Mover la visita hacia el punto correspondiente en la ruta;
    Actualizar Arrive, Start, Wait, MaxShift, Shift;
  end for
  for each visita antes de i do
    Actualizar MaxShift;
  end for
end for
```

Ejecución del algoritmo

El algoritmo comienza con un set de tours vacíos. Es necesario inicializar los parámetros del paso *Shake* a 1.

El algoritmo se ejecutará en bucle. La condición de salida es que se supere un numero establecido de veces que no se mejore la solución actual.

El primer paso será realizar *Inserción* hasta que se alcance el óptimo local, es decir, hasta que no se pueda introducir ningún punto más en la ruta generada.

Una vez tengamos la ruta, se la comparará con la ruta guardada actualmente (comenzaremos almacenando una ruta que solo contenga el nodo inicial y final ya que es la única ruta válida conocida en la primera iteración del algoritmo), y en caso de que la nueva ruta sea mejor, la almacenaremos y descartaremos la anterior.

En caso de que el óptimo local SI que sea mejor que la ruta almacenada, es importante reiniciar el valor de R a 1. También se reiniciará el contador de veces sin mejorar a 0.

Por el contrario, si la nueva ruta NO es mejor que la ruta almacenada, bastará con incrementar en 1 el contador de veces sin mejorar.

El segundo paso consiste en aplicar *Shake* a la ruta. Después de cada *Shake*, se incrementarán los valores R y S, mencionados en el apartado anterior, para modificar el rango de acción de *Shake*.

S se verá incrementada en el valor actual de R, mientras que R simplemente se incrementará en 1. En caso de que S sea mayor o igual que el tamaño del tour más pequeño, restaremos ese tamaño a el valor de S. De esta forma nos aseguramos de que *Shake* nunca supere el tamaño de la ruta.

En caso de que R sea igual a $m/3 * m$, el valor de R se reiniciará a 1.

Algorithm 3 Ejecución completa

```
S ← 1;
R ← 1;
VecesSinMejorar ← 0;
while VecesSinMejorar <150 do
  while No sea óptimo local do
    Inserción
  end while
  if Solución es mejor que MejorSolución then
    MejorSolución ← Solución;
    R ← 1;
    VecesSinMejorar ← 0
  else
    VecesSinMejorar+ = 1;
  end if
  ShakeSolución(R,S);
  S ← R + S;
  R ← R + 1;
  if S >= Tamaño del tour más pequeño then
    S ← S − tamañodeltourmáspequeño;
  end if
  if  R == n/3*m then
    R ← 1;
  end if
end while
```

Algoritmo de colonia de hormigas

Algoritmo colonia

Algoritmo genético

Documentación de las bibliotecas

Osm2po

Junit

Cloning library

D.4. Compilación, instalación y ejecución del proyecto

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

En este apartado se detallarán las especificaciones mínimas tanto software como hardware que se requerirán para un correcto funcionamiento de la aplicación.

Requisitos hardware

Podemos diferenciar dos posibles formas de ejecutar la aplicación: a través de un emulador *Android* instalado en un PC y a través de un dispositivo móvil con sistema operativo *Android*.

- **Emulador en un PC:** será necesario que el PC tenga instalados tanto el compilador *Java* como un emulador con el SDK de *Android*. Se recomienda que el emulador utilizado sea *Android Emulator API 23*. Este emulador tiene sus propias restricciones hardware, que son:
  - Procesador de 64 bits. *Android Emulator* NO se encuentra disponible para procesadores de 32 bits desde Junio/2019
  - SDK Tools 26.1.1 (o posterior)
  - HAXM 6.2.1 o posterior (aunque se recomienda HAXM 7.2.0 o posterior)
  - Procesador *Intel* i3, i5 o i7. Se ha observado que aunque es posible utilizar procesadores *AMD*, se recomienda que estos sean actuales

ya que en un principio el emulador se probó con un procesador *AMD ryzen2700x* y no funcionó.

- **Dispositivo móvil:** se podrá utilizar cualquier dispositivo (móvil o tablet) que cuente con *Android 6.0* o superior. La aplicación se ha instalado y ejecutado en un dispositivo *Google Pixel 2* y un *BQ Aquaris E5* y en ambos ha dado buenos resultados.

Requisitos software

Los requisitos software son los siguientes:

- Compilador *Java*. Necesario para programar aplicaciones *Android* y ejecutar la máquina virtual de *Java* (JVM)
- *Android Studio*, desarrollado por *Google*, que nos permite desarrollar y ejecutar aplicaciones para la plataforma *Android*
- *Oracle Glassfish Server 3*, que contendrá el servidor encargado de las respuestas a las peticiones del cliente.
- *Oracle VM VirtualBox*, utilizado en caso de no querer instalar los programas mencionados anteriormente. Bastaría con cargar la máquina virtual con el cliente/servidor. Cabe destacar que en caso de que el ordenador no tenga demasiada potencia es posible que *VirtualBox* no sea capaz de funcionar correctamente.

E.3. Instalación

En esta sección se encuentra detallada la guía de instalación de la aplicación.

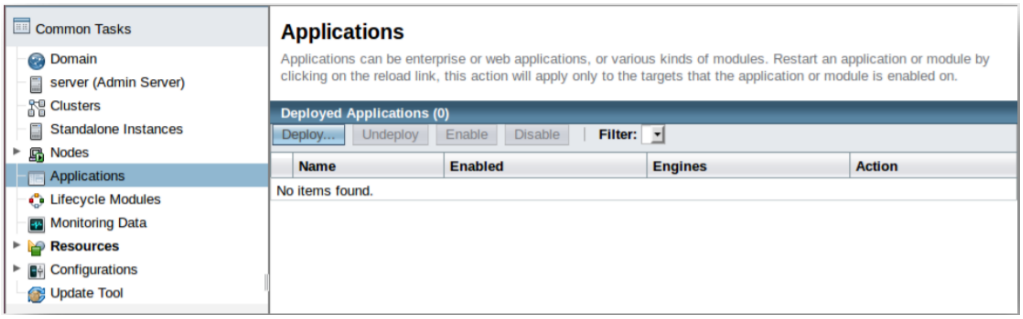
Cabe destacar que es necesario instalar tanto el cliente en un dispositivo *Android* (o emulador) como el servidor en *Glassfish*.

Instalación de la aplicación en el servidor

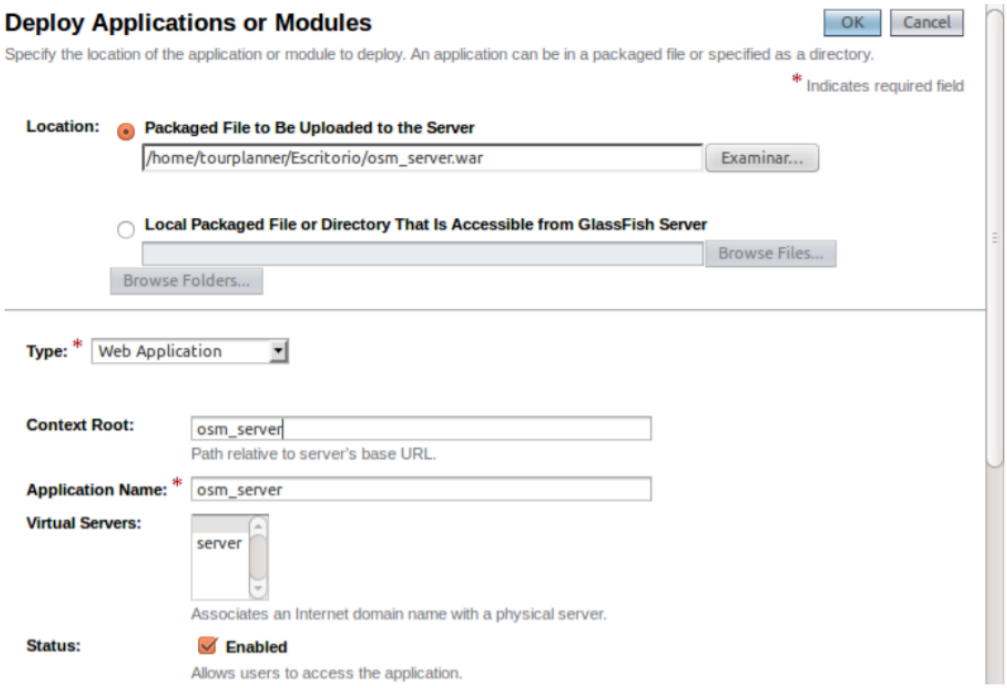
Debido a que la versión de *Glassfish* utilizada es la misma que en anteriores versiones del proyecto, se seguirán los mismos pasos para la instalación del servidor. Dichos pasos son los siguientes:

Para instalar la aplicación en el servidor, hay que abrir la consola de administración de *GlassFish* tecleando en un navegador web la dirección

del servidor con el puerto 4848 (http://localhost:4848 en este caso) e ir a la opción *Applications*.



A continuación hay que hacer *click* sobre el botón *Deploy* e indicar la ruta en la que se encuentra el archivo *.war* , después una vez que se haga *click* sobre el botón OK ya estará la aplicación publicada y funcionando.

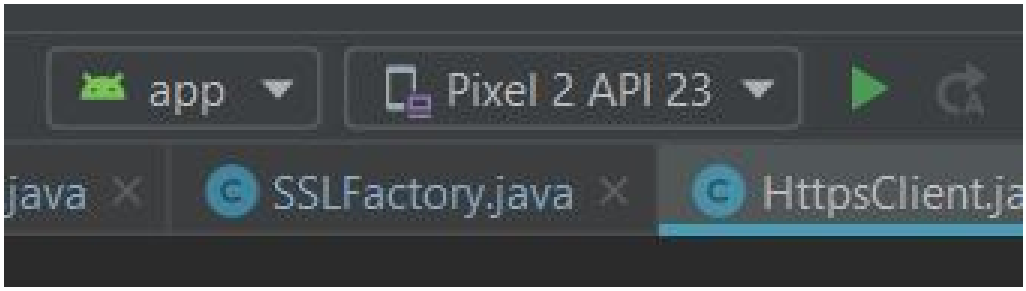


Instalación de la aplicación en el cliente

Para realizar la instalación de nuestra aplicación *Android* en un dispositivo físico o un emulador necesitaremos seguir los siguientes pasos:

- En caso de un dispositivo físico será necesario que éste se encuentre conectado al sistema en el que tengamos la aplicación.

- Una vez tengamos lista la aplicación pulsaremos el botón de *Play* en *Android Studio*.



- *Android Studio* será el encargado de instalar el archivo *.apk* dentro del dispositivo y lo ejecutará automáticamente.

AC: ACTUALIZAR SEGUN MANUAL DE SUSO Hay que tener en cuenta que para el correcto funcionamiento de la aplicación hay que seguir correctamente los pasos de compilación explicados en el apartado *Compilación del cliente*, prestando especial atención al punto en el que se establece la dirección IP y puerto del servidor.

Ejecución de la aplicación

Para lograr que la aplicación se ejecute correctamente sin la necesidad de contar con un servidor y un cliente independientes, se han incluido en el USB del proyecto las máquinas virtuales correspondientes a el cliente y el servidor. De esta forma podremos ejecutar la aplicación con un solo ordenador.

Cabe destacar que para lograr que ambas máquinas virtuales se comuniquen entre sí correctamente, se ha utilizado el programa *LogMeIn Hamachi* que nos permite crear redes virtuales entre ambas máquinas a través de Internet.

Ejecución de la aplicación servidor

Para ejecutar la aplicación servidor, se importa la máquina virtual *Servidor-Ubuntu 18.04 LTS* en *VirtualBox* (hay más software de creación de máquinas virtuales pero este es el que se ha utilizado).

AC: FOTO

Ejecución de la aplicación cliente

Para ejecutar la aplicación cliente, al igual que con la aplicación servidor, se importa la máquina virtual *Cliente-Windows 10* en *VirtualBox*.

Una vez se haya iniciado la máquina virtual, se hace doble *click* sobre el entorno de desarrollo *Android Studio*. A continuación, hacemos doble *click* sobre el icono *Play*.

AC: FOTO

E.4. Manual del usuario