Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Resolución del problema de enrutamiento del autobús escolar

Trabajo Fin de Grado
Grado en Ingeniería Informática

Autor: Domenech Sena, Eric
Tutor: Alonso Jordá, Pedro
Curso 2018-2019

“A mis padres y a mi mujer.”

Resumen

La finalidad de este trabajo es obtener una solución exacta y eficiente del problema de enrutamiento de un autobús escolar. Lo que se pretende es realizar una aplicación con la que poder planificar las rutas óptimas para transportar estudiantes desde el colegio a sus casas y viceversa.

Se trata de un problema derivado del problema de enrutamiento de vehículos y se diferencia de estos en añadir variables extra que proporcionan mayor dificultad en la obtención de la solución. Para obtener la solución exacta desarrollaremos un algoritmo de back-tracking que sea capaz de expandir de forma recursiva todos los nodos de las posibles soluciones.

Debido a que estamos ante un problema de tipo NP-Completo, obtener una solución exacta tendrá un coste temporal y computacional elevado. Para reducir el tiempo de computo lo máximo posible utilizaremos técnicas de programación paralela mediante OpenMP que nos permitirán aprovechar toda la capacidad de computo de la máquina.

Palabras clave: VRP, OpenMP, SBRP, TSP

Abstract

The purpose of the work is to obtain an exact and efficient solution to the problem of routing a school bus. What is intended is to make an application with the planned power of the optimal routes to transport students from school to their homes and vice versa.

This is a problem derived from the problem of routing vehicles and differs from these in adding extra variables that require more difficulty in obtaining the solution. To obtain the exact solution, we will develop a recoil algorithm that is capable of recursively expanding all the nodes of the possible solutions.

Because we are facing an NP-Complete problem, obtain an exact solution that will have a high temporal and computational cost. To reduce the computing time as much as possible, we use parallel programming techniques using OpenMP that allows us to reduce the entire computing capacity of the machine.

Keywords : VRP, OpenMP, SBRP, TSP

Índice

Tabla de Ilustraciones

Índice de tablas

1. Introducción

Mucho antes del nacimiento de la informática, el ser humano ya mostraba interés en la resolución de problemas siguiendo patrones sistemáticos. El termino de algoritmo proviene del matemático persa del siglo IX Abdullah Muhammad ibn Musa Al-Khwarizmi escritor del primer libro de algebra “Compendio de cálculo por reintegración y comparación” [1].

Con el nacimiento de la informática hemos visto la oportunidad de aprovechar su potencial en la resolución de algoritmos. Desde sus inicios en la década de 1940 hasta la actualidad, la capacidad de cálculo de los ordenadores ha ido evolucionando de forma exponencial. Por este motivo hemos visto como en aproximadamente 80 años grandes maquinas como el ENIAC, con una potencia de cálculo de 5000 operaciones por segundo y un peso de 30 toneladas, han sido sustituidas por pequeños dispositivos capaces de realizar millones de operaciones por segundo.

Nuestro problema tiene su origen en la década de 1959 cuando George Dantzig y John Ramser [2] intentaron obtener la solución óptima para la entrega de combustible en su red de gasolineras, es lo que se conoce como el problema de ruteo de vehículos, con siglas en inglés *VRP* (*vehicle route problem*). Para su solución Dantzig y Ramser realizan una aproximación algorítmica del problema del viajante o *TSP* (*Travel sales problem*).

Posteriormente este problema se ha ramificado en problemas más específicos como el problema de enrutamiento de un autobús escolar, en el cual basaremos nuestro proyecto.

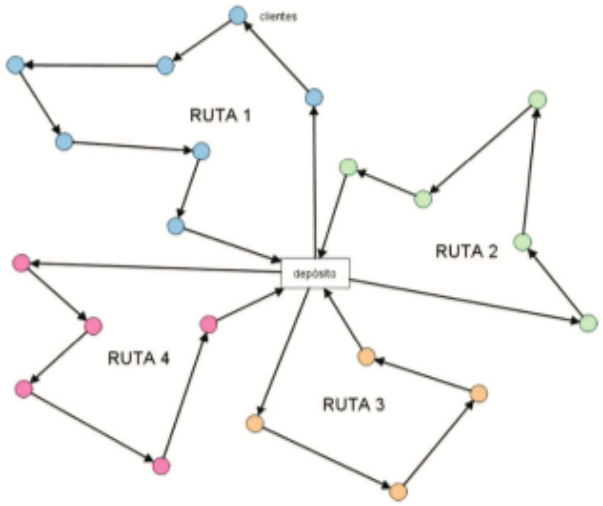


Ilustración 1 - Imagen rutas VRP (Wikipedia)

Todos estos problemas de enrutamiento de vehículos se caracterizan por ser problemas de optimización combinatoria y programación de enteros. Se trata de problemas NP-completos con un coste computacional elevado que buscan obtener la respuesta de "¿Cuál es el conjunto óptimo de rutas para una flota de vehículos que debe satisfacer las demandas de un conjunto dado de clientes?".

El problema de enrutamiento del autobús escolar es un problema específico del tipo de problema de ruteo de vehículos. En numerosas ocasiones las personas encargadas de

elaborar una ruta de transporte, debido al desconocimiento de algoritmos de optimización o a la utilización de herramientas informáticas, establecen unas rutas que a priori nos pueden resultar validas, pero tras un primer análisis podemos observar claramente que no siempre se obtienen los resultados óptimos. No obtener el resultado óptimo siempre implica dos cosas: perder tiempo y dinero.

En este proyecto vamos a programar un algoritmo para resolver el problema de enrutamiento de un autobús escolar y lo analizaremos desde una perspectiva de optimización computacional. Para ello crearemos un algoritmo que únicamente hará uso de un núcleo del procesador y posteriormente lo modificaremos utilizando técnicas de programación paralela para hacer uso de todos los núcleos disponibles en el computador (pueden ser varios núcleos y varios procesadores). Con esto veremos la importancia que tiene la adaptación de código a las capacidades tecnológicas de cada época.

1.1 - Motivación

He escogido el problema de enrutamiento de un autobús escolar, en adelante *SBRP*, (*School bus route* problem) debido a que me gusta la resolución de problemas y la optimización de soluciones. De todas las opciones que había finalmente me decanté por esta debido a que vi la oportunidad de poner en practica conocimientos sobre creación de programas con interfaz gráfica en JAVA, desarrollo de algoritmos en C, optimización usando OpenMP♦, etc.

El problema del SBRP es un tipo de problema con cierta relevancia en el ámbito científico debido a su elevada complejidad computacional. Debido a ello, la obtención de las rutas de transporte para los colegios es un trabajo costoso que implica tiempo y esfuerzo. Con este proyecto creo una base funcional de un programa multiplataforma que puede ayudar a que los colegios creen sus rutas de transporte cada año.

1.2 - Objetivos

Queremos crear un algoritmo que resuelva el problema de enrutamiento de un autobús escolar de forma paralela, utilizando todo el potencial de los procesadores multinúcleo. De esta forma mostraremos la reducción de tiempo que se obtiene tras aplicar técnicas de paralización de tareas usando OpenMP.

Para que el programa sea lo más amigable posible con personas ajenas a la utilización de la “Terminal” del sistema operativo el segundo objetivo es crear una aplicación multiplataforma que permita a los colegios obtener las rutas de transporte más eficientes para su año lectivo, permitiéndoles ahorrar tiempo y dinero en esta tarea.

1.3 - Impacto esperado

Como hemos comentado anteriormente esperamos reducir el tiempo de respuesta del algoritmo.

Creando una interfaz gráfica con un lenguaje de programación multiplataforma como JAVA y haciendo de forma transparente para el usuario todas las interconexiones con los diferentes lenguajes y sistemas operativos esperamos ampliar al máximo el abanico de posibles colegios que puedan usar la aplicación.

♦ OpenMP es una API para la programación paralela. Se puede encontrar una explicación más detallada en puntos posteriores.

1.4 - Metodología

Para poder lograr cada uno de los objetivos descritos anteriormente y lograr el impacto esperado vamos a estructurar el proyecto y utilizar los siguientes lenguajes de programación.

- Algoritmo de búsqueda atrás (*back-tracking)* escrito en lenguaje C con el fin de lograr la mejor optimización posible en el tratamiento y acceso a memoria.
- Modificación del algoritmo de búsqueda atrás utilizando metodologías de programación paralela usando OpenMP.
- Creación de la interfaz en JAVA para dotar al programa de una funcionalidad multiplataforma.
- Conexión a la API de GoogleMaps para obtener las coordenadas.
- Conexión entre los diferentes lenguajes de programación haciendo uso de scripts que permitan la compilación y ejecución de código de forma transparente para el usuario.

1.5 - Estructura de la memoria

El proyecto se divide en siete capítulos. A continuación, se muestra un pequeño resumen de los temas tratados en cada capítulo:

Capítulo 1. Introducción: pequeña introducción del tema desarrollado en el proyecto junto con aquellos aspectos que han motivado su elección

Capítulo 2. Estado del arte: se hace una introducción más teórica en el problema de enrutamiento de un autobús escolar, las diferentes opciones que hay para su solución y la explicación de cuales han sido los motivos que nos han llevado a escoger la nuestra.

Capítulo 3. Análisis del problema: expondremos las diferentes opciones que teníamos para abordar el problema. Cuales han sido los motivos que nos han llevado a escoger las opciones que hemos seleccionado. Realización del esbozo de la interfaz gráfica de la aplicación y estructura de nuestro algoritmo.

Capítulo 4. Diseño de la solución: aquí puede encontrarse una explicación de las diferentes tecnologías empleadas en el desarrollo del trabajo, se explica el problema de una forma mucho más profunda y como se ha implementado.

Capítulo 5. Implantación: información referente a requisitos mínimos del hardware de la máquina, sistemas operativos, dependencias de compiladores de C, dependencias de versiones de JAVA y obtención de la contraseña de GoogleMaps para poder usar la aplicación.

Capítulo 6. Pruebas: punto que contiene toda la información referente a las pruebas obtenidas en la comparación de algoritmo *back-tracking* para la solución del problema de ruteo de un autobús escolar haciendo uso de programación paralela y sin ella.

Capítulo 7. Conclusiones: recogerá las conclusiones obtenidas tras la finalización del proyecto.

Capítulo 8. Futuras mejoras: posibles mejoras o implementaciones futuras que añadirían funcionalidades extra.

En la parte final del proyecto pueden encontrarse los anexos que hacen referencia a toda la nomenclatura científica, abreviaturas empleadas y fuentes utilizadas para la realización del proyecto.

1.6 - Convenciones

Durante la memoria aparecen pequeños fragmentos de código para explicar las distintas fases del algoritmo, estos fragmentos de código irán en letra cursiva claramente diferenciados y con la explicación correspondiente de que realiza cada fragmento de código.

Las palabras extranjeras aparecerán debidamente distinguidas usando letra cursiva.
Las citas textuales, expresiones y preguntas retoricas al lector irán remarcadas entre comillas. Los nombres extranjeros no se han remarcado en cursiva.

Por cuestiones de evitar repeticiones que aumenten de forma innecesaria el texto, se utilizarán abreviaturas para referirse al problema y a cada una de las variantes de este, Dichas abreviaturas pueden encontrarse junto a su significa en el glosario al final del proyecto. Las notas al pie de página irán referenciadas con el símbolo del rombo.

Las referencias utilizadas irán debidamente numeradas y podrán encontrarse al final del documento en su correspondiente apartado.

2. Estado del arte

El concepto que tenemos actualmente de la informática no tiene más de cinco o seis décadas. Con la finalidad de situarnos en el estado actual que se encuentra la informática vamos a ver cuáles fueron sus inicios y que hitos hicieron posible que la informática tomase un papel tan importante en nuestras vidas como el que tiene actualmente.

Podemos diferenciar cinco generaciones desde el inicio de la informática hasta la actualidad.

Primera generación: podemos enmarcarla entre la década de 1940 y 1960. Se caracterizaba por la utilización de válvulas de vacío que daban como resultado maquinas con un gran tamaño, una capacidad de cálculo limitada y un consumo energético elevado. En esta época deberíamos destacar maquinas como el Mark I de la empresa IBM y el ENIAC creada por John Von Neumann en Estados Unidos.

Segunda generación: se desarrolla durante los años 1960 y 1965. Viene caracterizada por la sustitución de las válvulas de vacío por transistores y el aumento de la memoria. Los transistores permitieron reducir el tamaño de los dispositivos, aumentar su fiabilidad y reducir consumo.

El transistor es un dispositivo electrónico generado con silicio. Su funcionamiento está basado en tener dos posibilidades: la transmisión o no transmisión de energía.

Durante esta generación aparecieron las impresoras, los discos magnéticos y los primeros lenguajes de programación de alto nivel. Podemos considerar el primer ordenador de transistores al ATLAS 1962.

Tercera generación: se produce ente 1965 y 1975. En ella aparecen los primeros circuitos integrados, estos abren el camino de la miniaturización. Aumenta considerablemente la velocidad de cálculo y aparecen numerosos programas y lenguajes de programación. Es en esta época cuando se desarrollan los sistemas operativos que controlan el ordenador. El primer dispositivo caracterizado por el uso de circuitos integrados es el IBM serie 360.

Cuarta generación: abarca la década entre 1975 y 1985. Surgen los primeros microprocesadores, con ellos se consigue aumentar más todavía la potencia de cálculo, reducir aún más el tamaño y disminuir los costes de producción. Aparecen los primeros ordenadores personales como el Apple I y el primer ordenador personal de IBM con sistema operativo MS-DOS.

Quinta generación: desde los años 1985 hasta la actualidad. Cada año los componentes han ido reduciendo su tamaño, aumentando su capacidad de cómputo y reduciendo su consumo energético. Esto ha dado la posibilidad de crear una gran cantidad de programas, lenguajes de programación, etc.

En 1990 se produce una gran revolución con el nacimiento de internet, protocolo que permite el intercambio de información de forma instantánea entre ordenadores situados en diferentes partes del mundo.

El avance tecnológico siempre ha ido ligado de la necesidad de solucionar los problemas que el ser humano ha ido tenido. El origen del problema sobre el cual basamos nuestro proyecto tiene su definición en 1800 pero no es hasta la época de 1930 cuando comienza su estudio. Este problema original, se denomina “El problema del vendedor viajero” y tiene como objetivo la obtención de la ruta más corta que necesita un comercial para visitar a sus clientes y regresar a su ciudad.

Se trata de uno de los problemas de optimización más estudiados debido a su complejidad computacional, posee una gran cantidad de heurísticas y métodos exactos de resolución conocidos. Es un problema que ha sido portado a otros ámbitos como los puntos de soldadura en la fabricación de circuitos o la lectura de ADN. Se trata de un problema NP-Completo, lo que conlleva que el caso peor tenga un aumento exponencial del tiempo de cálculo respecto al número de ciudades.

Como ha sucedió con la informática, los problemas han ido evolucionando y adaptándose a las necesidades de cada época. Es así como nace “el problema de enrutamiento de vehículos” o *VR*P (*vehicle route problem*). George Dantzig y John Ramser en 1959 plantean una aproximación algorítmica que les permita obtener las mejores rutas para abastecer su red de gasolineras.

El problema de enrutamiento de un autobús escolar forma parte de esta familia de problemas *VRP* (Kinable, Spieksma, & Vanden, 2014, [3]). Se caracterizan como hemos comentado anteriormente en su complejidad computacional, son de tipo NP-Completo (Park & Kim, 2010, [4]) (Fügenschuh, 2007, [5]) y en la división en subproblemas: asignación de paradas a cada autobús, asignación de estudiantes a los autobuses y selección de la ruta.

Tanto el “problema del vendedor viajero” como “los problemas de ruteo de vehículos” poseen variables y características comunes como: rutas y número de vehículos, pero es en las restricciones de cada tipo de problema donde radica su principal diferencia y clasificación.

Por poner un ejemplo, los problemas *VRP* cuentan con aproximadamente 36 subtipos de problemas, todos ellos clasificados según el tiempo de resolución y la metodología empleada para su resolución como problemas NP-Completos. (Rocha, González & Orejuela, 2011, [6]).

El problema de enrutamiento de un autobús escolar (*SBRP*), lo que pretende es encontrar la mejor ruta para recoger estudiantes (o dejarlos) en sus respectivas paradas. Tiene una serie de variaciones que cabe remarcar con respecto a los problemas anteriores. Este problema añade limitaciones extra como: capacidad del autobús, tiempo máximo que puede permanecer cada estudiante en él y distancia máxima que pueden permanecer los estudiantes dentro del autobús (Yigit & Unsal, 2016, [7]).

Es un problema de movilidad y transporte que afecta tanto a grandes ciudades como a zonas rurales donde deben agrupar esfuerzos en la obtención de rutas conjuntas que satisfacen sus necesidades, es por ello que muchos estudios están de acuerdo en tratarlos como problemas importantes del transporte público (Kelly & Fu, 2014, [8]).

Hay diferentes versiones del *SBRP* y diversos textos científicos que intentan dar solución a cada una de estas versiones. Una de las principales diferencias radica en cuál es la estructura del transporte escolar en la zona. Cada zona tiene unas particularidades de transporte e introducen variables importantes que deben considerarse en la obtención del resultado.

Así pues, elementos como: el número de escuelas que comparten autobús (Angel et al,1972, [9]), tipo de zona (Thangiah & Nygard, 1992, [10]) (Bowerman et al, 1995, [11]) (Desrosiers et al. 1981, 1986ª, [12]), posibilidad de subir y bajar gente en la misma parada (Hargroves & Demetsky, 1981, [13]), distancia entre paradas, la escasez de estudiantes en cada parada y las distancias hasta el colegio, etc. generan verdaderos problemas logísticos que han sido estudiados en numerosos textos científicos.

Referencia	Escuelas	Tipo de zona	Recogidas múltiples	Área
Kinable, Spieksma, & Vanden (2014)	Múltiples	Ambas	X	X
Park & Kim (2010)	Una	Ambas	X	X
Fügenschuh (2007)	Múltiples	Rural	No	Alemania
Rocha, González & Orejuela (2011)	Múltiples	Rural	No	X
Yigit & Unsal (2016)	Una	Rural	No	X
Kelly & Fu (2014)	Múltiple	Ambas	No	Dublín, Irlanda
Angel et al (1972)	Múltiples	Ambas	No	Tippercanoe, Indiana
Thangiah & Nygard (1992)	Una	Ambas	No	X
Bowerman et al (1995)	Una	Urbana	No	Ontario, Canada
Desrosiers et al. 1981 (1986a)	Multiples	Ambas	No	Drummondville, Canada
Hargroves & Demetsky (1981)	Multiple	Ambas	Si	Albemarle, Virginia
La Voz de Galicia (2019/04/20)	X	X	X	Galicia, España
Chen et al (1990)	Multiple	Rural	Si	Choctaw, Alabama

Tabla 1 - Referencias (elaboración propia)

Por ejemplo, hay zonas como el norte de España, donde la dificultad que supone a los colegios la elaboración de rutas acordes a las limitaciones presupuestarias ha llevado a la elaboración de rutas en las que escolares y ciudadanos comparten autobús, esto podemos leerlo en noticias como la aparecida en el diario “La Voz de Galicia” en 2019 [14]. Pero, no solo en España el transporte escolar tiene peculiaridades, en zonas de Estados Unidos con baja población demográfica la solución al problema del transporte escolar ha llevado a la adopción de modelos en los que diferentes escuelas compartan autobuses para reducir sus costes (Chen et al ,1990, [15]).

Nuestro modelo de SBRP es distinto, nosotros basaremos el estudio en una única escuela, que tiene la necesidad de obtener la ruta optima de recogida o regreso de los estudiantes. Este modelo es mucho más común en zonas como Valencia, Madrid o Barcelona donde los núcleos urbanos tienen más habitantes y por tanto la necesidad de compartir autobús entre escuelas no es un requisito obligatorio para el cumplimiento de unos presupuestos de transporte, aunque no por ello dejaremos de tenerlos en cuenta en la elaboración de la ruta.

2.2 - Programas para resolver VRP

Actualmente hay varios programas libres para obtener soluciones del problema VRP.

JSprit: basado en JAVA, se trata de un conjunto de herramientas de código abierto para solucionar varios tipos de problemas del ruteo de vehículos.

Abierto-VRP: se trata de un *framework* de Lisp. Al igual que JSprit permite implementar el mismo tipo de problemas mediante el lenguaje Lisp.

```
(solve-prob test-vrp (make-instance 'tabu-search :iterations 10 :animatep T))
(solve-prob solomon100 (make-instance 'tabu-search :iterations 100))
(solve-prob christofides-2 (make-instance 'tabu-search :iterations 50))
```

Ilustración 2 - Abierto-VRP (web del programa)

Esta desarrollado en JAVA y para la resolución de los problemas utiliza algoritmos de optimización basados en inteligencia artificial.

VRP Spreadsheet Solver: Se trata de una solución de código abierto propuesta por Microsoft. Esta herramienta permite representar, solucionar y visualizar los resultados de problemas VRP. Para ello utiliza Excel, GIS y metaheurísticas.

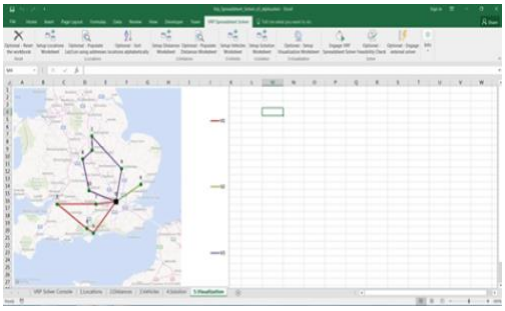


Ilustración 3 - VRP Spreadsheet Solver (fuente web del programa)

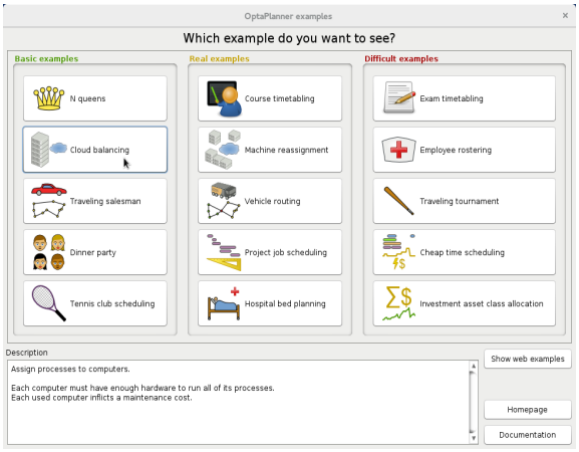


Ilustración 4 – Pantalla OptaPlaner (web del programa)

OptaPlaner: es un programa con interfaz gráfica y licencia apache. Se trata de un software mucho más potente que las dos opciones anteriores.

Permite la solución de problemas de ruteo de vehículos y otros problemas como la asignación de jornadas de trabajo, optimización de sistemas en la nube, asignación de tareas, programación de conferencias, embalaje de contenedores, etc.

Programa RUTAS: herramienta desarrollada por Alejandro Rodríguez Villalobos, Ingeniero de Organización Industrial y Profesor en el Departamento de Organización de Empresas de la Universidad Politécnica de Valencia. Permite la resolución de problemas VRP.

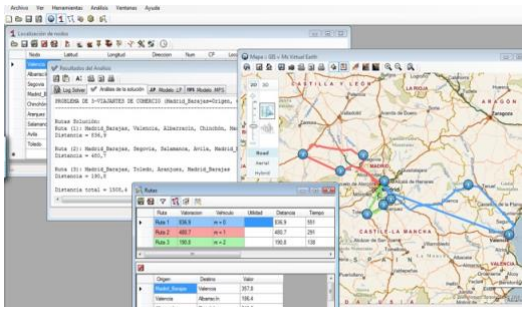


Ilustración 5 - Rutas (web del programa)

2.2 - Crítica al estado del arte

Cómo se ha descrito anteriormente, este tipo de problemas han sido enfocados desde diferentes perspectivas del ámbito científico dando como resultado diferentes formas de obtención de las soluciones. Si lo que se busca es priorizar el tiempo de búsqueda frente a la solución óptima se optara por algoritmos heurísticos o metaheurísticos y si lo que se pretende es encontrar la solución óptima se optara por algoritmos exactos.

Actualmente en el mercado hay diversas herramientas para la representación de los problemas y la obtención de las soluciones, cada una basa su resultado en uno o varios de los algoritmos descritos anteriormente.

Uno de los principales hándicaps que hemos detectado es que aplicaciones como JSprit y abierto-VRP necesitan de unas nociones previas de programación para su uso. Otras en cambio como OptaPlaner y VRP Spreadsheet Solver necesitan de una formación previa para su manejo y no permiten la representación exacta del problema de ruteo de un autobús escolar.

2.3 - Propuesta

Estudios y propuestas actuales, utilizan algoritmos exactos, heurísticos o metaheurísticos para la obtención de la solución óptima o aproximada. Como nosotros lo que pretendemos es encontrar la solución óptima, los algoritmos heurísticos y metaheurísticos quedan descartados.

Basaremos nuestro proyecto en un algoritmo exacto y nos diferenciaremos del resto de propuestas en la introducción de mejoras mediante la programación paralela con OpenMP que permitan reducir considerablemente el tiempo de obtención de resultados.

Como no queríamos que nuestro proyecto fuera únicamente accesible para personas con conocimientos informáticos avanzados, desarrollaremos un programa con interfaz gráfica que permita a cualquier persona con unos mínimos conocimientos sobre informática la introducción de los datos para la obtención de las rutas optimas de transporte.

3. Análisis del problema

En este apartado hablaremos en un primer lugar sobre las diferentes opciones que tenemos para abordar el problema y posteriormente explicaremos cual hemos seleccionado, el motivo de su elección y cómo vamos a desarrollarla.

Para ello haremos uso de técnicas de modelado conceptual como: diagrama UML de los diferentes estados del algoritmo, diagrama UML de casos de uso de la aplicación y un *mockup* (boceto de la interfaz gráfica) con las diferentes vistas de la aplicación.

3.1 - Identificación y análisis de posibles soluciones

Existen varias formas de resolver problemas VRP, dependiendo de cuáles son nuestras restricciones o que es aquello que buscamos priorizar a la hora de resolver el problema unas técnicas se adaptaran mejor que otras. Hay varios métodos para solucionar este tipo de problemas, según (Toth & Vigo, 2002, [16]) podemos agrupar estos métodos en tres grupos: exactos, heurística y metaheurística.

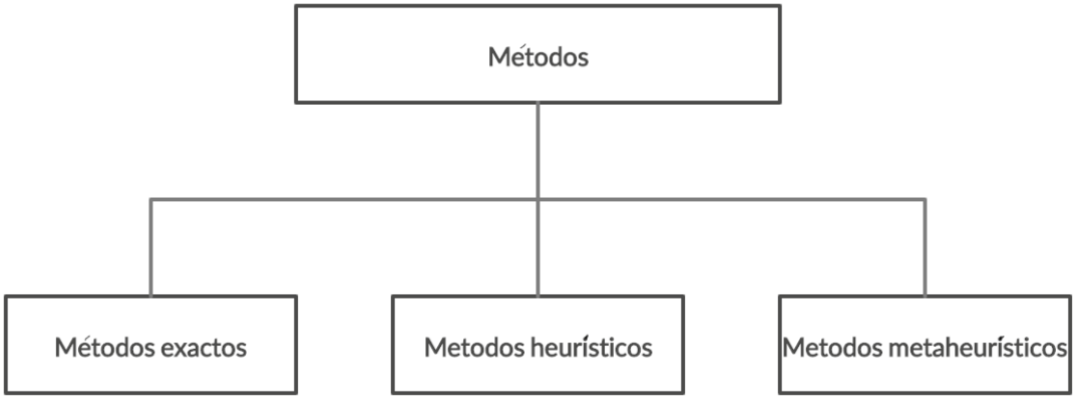


Ilustración 6 - Tipos de métodos (elaboración propia)

Métodos exactos: al tratarse de problemas de complejidad elevada, obtener la solución óptima haciendo una búsqueda exhaustiva no es un cálculo eficiente. Entre los métodos exactos podemos encontrar:

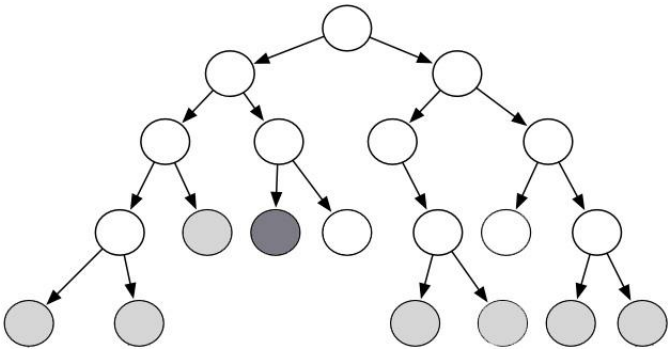
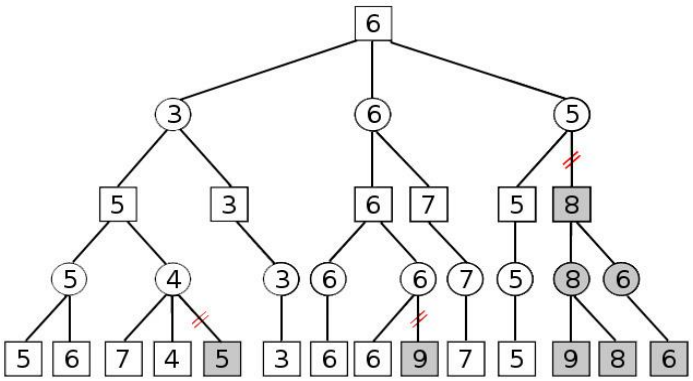


Ilustración 7 - Ramificación y acotamiento (elaboración propia)

Ramificación y acotamiento: esta técnica divide el problema principal en subproblemas más pequeños que posteriormente resuelve de forma individual, por último, agrupa todas estas soluciones parciales en una solución final.



Ramificación y poda: proviene del método anterior de ramificación y acotamiento. Se trata básicamente de un algoritmo más potente ya que detecta todas las ramificaciones no optimas y las poda, para no gastar potencial en recorrerlas y de esta forma aprovechar los recursos en la búsqueda de la solución óptima.

Ilustración 8 - Ramificación y poda (elaboración propia)

Métodos heurísticos: este tipo de algoritmos se utiliza para obtener una aproximación de las soluciones optimas en un tiempo de computación razonable. Este tipo de algoritmos priorizan el tiempo de obtención de una solución aproximada a la solución óptima en lugar de obtener la solución óptima. Podemos destacar:

Clark & Wright: se parte de dos rutas que cuando se combinan generan una nueva ruta, el resultado es ahorrar en el coste al generar una sola ruta y no dos. (Lysgaard & Sørensen, 1997, [17]).

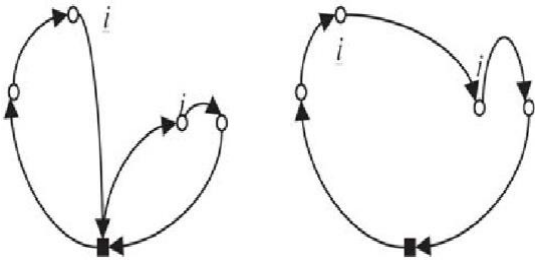


Ilustración 9 - Clark & Wright (Wikipedia)

Vecino más cercano: el algoritmo genera las rutas de forma secuencial, marcando los puntos con menor distancia para marcarlos a la ruta.
Este algoritmo establece que la mejor solución siempre es tomar el camino más corto, no profundiza en ver si ese camino nos introducirá en un camino que realmente está más alejado de la solución final. (Contreras P 2010, [18]).

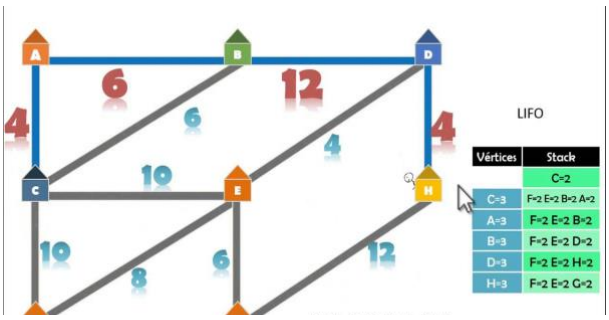


Ilustración 10 - Vecino más cercano (google imágenes)

Primero *clúster*, luego rutas: se trata de un algoritmo dividido en dos fases. En la primera fase es donde se agrupan los puntos en función de las restricciones de capacidad de cada vehículo, en la segunda fase es cuando se crean las rutas. (Prins, Lacomme, & Prodhon, 2014, [19]).

Primero rutas, luego *clúster*: es un algoritmo de dos fases. En la primera fase se hacen las rutas de visita a los clientes y en la segunda se crea el *clúster* según las restricciones de capacidad. (Beasley, 1983, [20]) (Lüer, Benavente, Bustos, & Venegas, 2009, [21]).

Métodos metaheurísticos: son algoritmos más avanzados que los heurísticos. Pretenden obtener soluciones más cercanas a las óptimas, mediante la mejora de los algoritmos heurísticos generales. Estos algoritmos no visitan iterativamente cada elemento del espacio de búsqueda, lo que hacen es guiar la búsqueda a regiones que pueden dar soluciones de alta calidad (Dib , Manier , Moalic , & Caminada , 2015, [22]). Algunas de las más importantes son:

Algoritmo de Colonia de Hormigas: al igual que las hormigas hacen en la búsqueda de alimento, este algoritmo se fija en el camino de salida. La salida en busca de la solución se hace de forma aleatoria, una vez se encuentra una solución regresa al punto de partida dejando una marca en el recorrido seguido, para reforzar y mejorar las soluciones. (C.A Peñuela, J.F Franco, E.M Toro, 2010, [23]).

Algoritmo Genético: ha evolucionado del concepto alemán de programación evolutiva, concebido en la época de 1975 por Colin R. Reeves y Jhonathan E. Rowe, 2003, [24]. Muchos autores como Golberg (1989, [25]) afirman teórica y empíricamente que este tipo de algoritmos son capaces de obtener resultados robustos en espacios complejos.

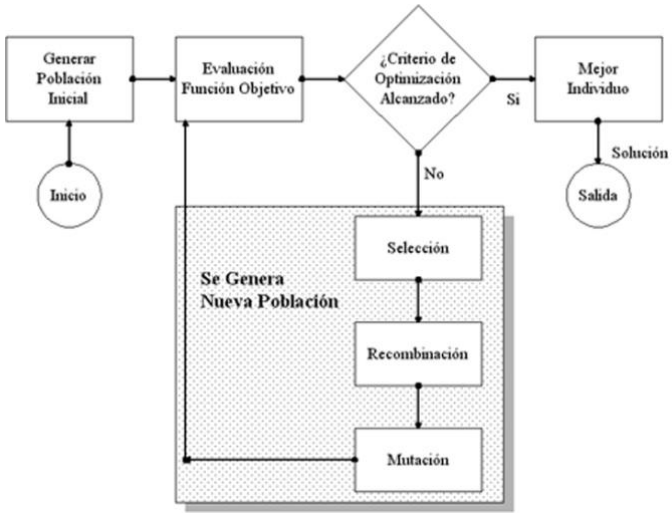


Ilustración 11 - Algoritmo genético (Google imágenes)

Este tipo de algoritmo se diferencia del resto en cuatro aspectos básicos en sus procedimientos de búsqueda y optimización:

1. Trabajan con una codificación de parámetros (o genotipo) y no con los parámetros en sí mismos (fenotipo), por esta razón en los AG cada solución (individuo de la población) está representada por un vector denominado cromosoma, en el que cada uno de sus componentes (gen) representa un parámetro de la solución.

2. Los algoritmos genéticos realizan la búsqueda a partir de una población de soluciones y no desde una sola solución, lo cual, según LeBlanc et al. (1999, [26]), asegura la exploración de una mayor porción del espacio de soluciones y evita la caída en óptimos locales.

3. Utilizan la información de la evaluación de la función objetivo (fitness) para guiar la búsqueda, no conocimiento auxiliar.

4. Tienen reglas de transición probabilística y no determinística.

Algoritmo de Búsqueda Tabú: es un tipo de algoritmo que usa memoria adaptativa, es decir, se ayuda de memoria a corto plazo para evitar la aparición de ciclos de búsqueda. Para ello esta memoria se implementa como una lista tabú donde se guardan registros con las soluciones ya obtenidas. Este tipo de algoritmo tiene el inconveniente de filtrar soluciones permitidas y por tanto excluir buenas soluciones, para evitar esto se puede usar una variable de criterio de aspiración que otorga la posibilidad de incluir soluciones que en principio hubieran estado prohibidas.

3.2 - Problema de enrutamiento del autobús escolar

Como el principal objetivo en la resolución del problema es utilizar las técnicas de paralización para ver cómo afecta la optimización de código en la reducción del tiempo de computo en obtener la solución tanto técnicas heurísticas como metaheurísticas quedan descartadas. Ya que, si bien estas técnicas son correctas para obtener soluciones aproximadas con buenos tiempos de respuesta, nuestro objetivo aquí no es este.

Para la resolución del problema nosotros utilizaremos un algoritmo que nos proporcione un valor exacto de la solución, independientemente del tiempo de computo que el ordenador necesite para encontrar la solución.

El algoritmo utilizado será un algoritmo de "*back-tracking* con poda" adaptado a este tipo de problema, estará programado de forma recursiva, estableciendo claramente cuál es su caso base y expandiendo todas las opciones posibles que nos conduzcan a la solución exacta.

El diagrama UML del algoritmo es:

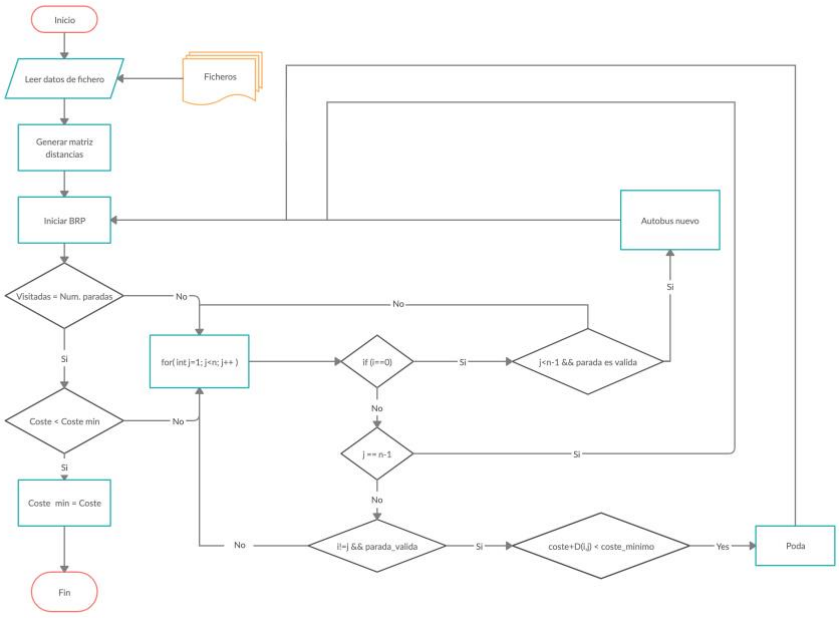


Ilustración 12 - Diagrama UML del algoritmo BRP (elaboración propia)

Podemos observar el caso base, que sería cuando todas las paradas han sido visitadas y el coste obtenido es menor que el coste mínimo que teníamos establecido en el inicio. Si esto no sucediera el problema no tendría solución.

El resto de casos se producen cuando todas las paradas no han sido visitadas, aquí podemos encontrar tres opciones:

1. Cuando no hay ninguna parada visitada y por tanto debemos generar un autobús y comenzar de nuevo el algoritmo.

2. Cuando se han visitado todas las paradas de esa rama y debemos seguir viendo otras ramas.

3. Cuando la rama que estamos visitando tiene mayor coste que una rama anterior, entonces realizamos una poda y volvemos al inicio para seguir verificando otras ramas.

Toda esta serie de decisiones se toman siguiendo más restricciones como las de distancia máxima que puede recorrer un estudiante, capacidad y coste de fletar un autobús.

La estructura de este algoritmo de *back-tracking* puede ser paralelizada utilizando OpenMP. Para ello hay que tener en cuenta una serie de factores y puntos clave para su paralelización:

1. Hay un único hilo que ejecutará la llamada principal y será el encargado de generar las tareas para el resto de hilos.

2. Hay una sección critica que solo puede ser accesible por un hilo a la vez. Esta sección critica se encuentra cuando se va a proceder a comparar para copiar a memoria el coste actual de la ruta con el coste mínimo que existe hasta ese momento.

3. Hay tres secciones del código que generaran tareas independientes para que sean paralelizadas por los hilos.

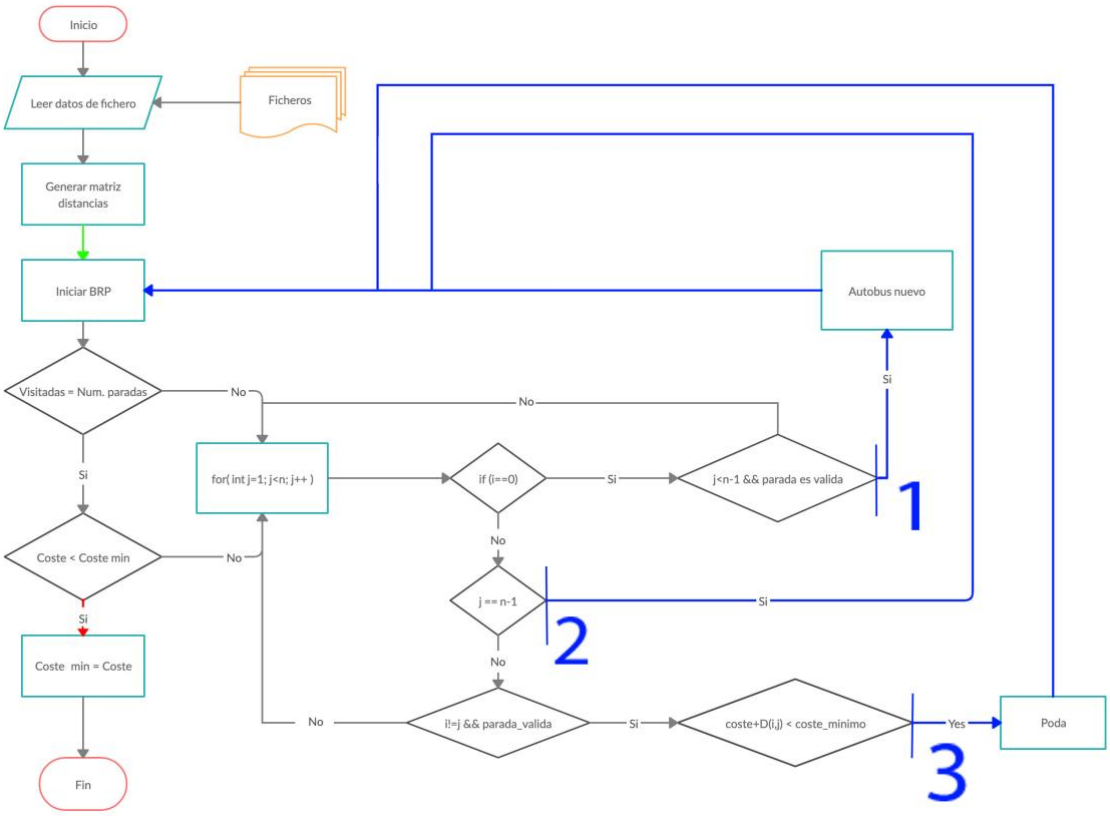


Ilustración 13 - UML algoritmo back-tracking paralelizado (elaboración propia)

La flecha verde sería la sección que únicamente accedería un hilo de los creados, la flecha roja indica la sección critica en la que solo pueden entrar los hilos de uno en uno y las flechas azules son las tres secciones que se deberán paralelizar mediante tareas.

3.2 - Análisis de creación de la interfaz gráfica

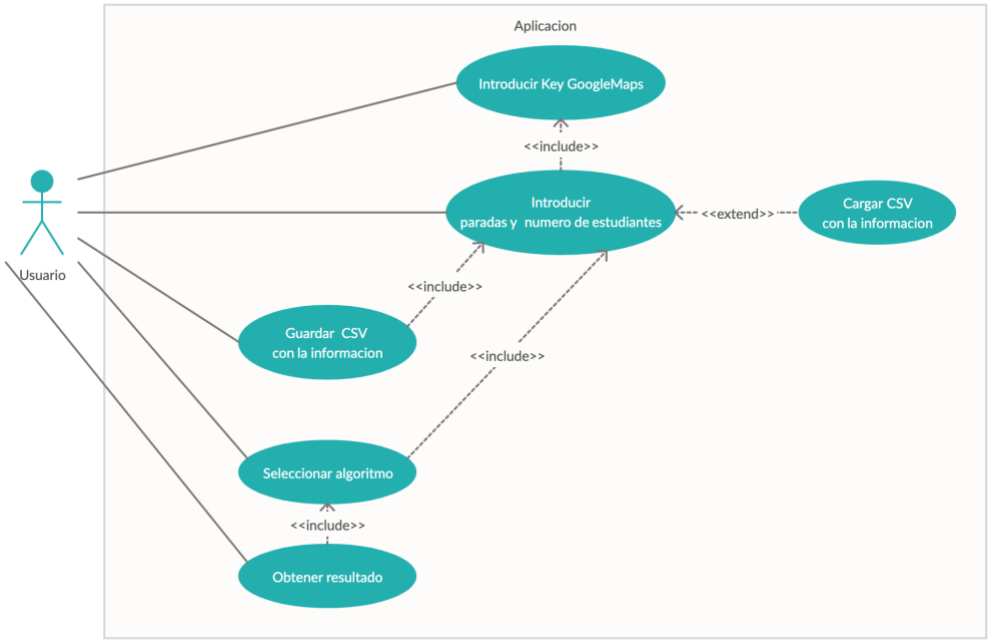


Ilustración 14 - Diagrama UML de casos de uso (elaboración propia)

Como podemos ver en el diagrama UML anterior, nuestra aplicación solo interactuara con un actor, el usuario.

Las acciones que necesiten un requerimiento de una acción previa son las que están modeladas con una flecha discontinua con la palabra <<*include*>>. Acciones que pueden derivar de otras están modeladas con la flecha discontinua y la palabra <<*exclude*>>. Así pues, entre las acciones principales que podrá realizar el usuario están:

Introducir clave de GoogleMaps: El usuario debe introducir su clave de GoogleMaps para poder hacer uso de la aplicación. Es necesario para poder conectarse a la API de Google y obtener las coordenadas geográficas de las paradas. Si la aplicación detecta que el fichero de la clave está vacío muestra un mensaje de error.

Introducir paradas y número de estudiantes: el usuario debe poder introducir la información de paradas y pasajeros desde el menú. Si se equivoca en la introducción de los datos, poniendo caracteres incorrectos u omitiendo información, aparece un mensaje de alerta indicándolo.

Cargar un fichero CSV: para mayor comodidad el usuario puede introducir la información de las paradas usando un CSV. Esto es útil debido a que si únicamente quiere añadir una parada a una consulta anterior puede hacer uso de esta función. Si el fichero CSV no tiene el formato correcto se muestra un mensaje de error.

Guardar un fichero CSV: cuando el usuario ha introducido paradas tiene la opción de guardar dichas paradas en un fichero CSV. Este fichero incluirá las coordenadas geográficas de las paradas, así de esta forma se evitará tener que hacer una consulta a la API de GoogleMaps la siguiente vez que el usuario cargué el fichero CSV.

Seleccionar algoritmo: El usuario tendrá la opción de escoger tres algoritmos: ruteo de un autobús escolar (algoritmo *BRP*), Prim o Dijkstra. Para poder seleccionar un algoritmo hay que introducir paradas, si no se introducen paradas la aplicación mostrara un mensaje de alerta.

Si lo que el usuario desea es obtener la ruta optima de transporte para una flota de autobuses escolares deberá escoger BRP, si por el contrario desea conocer el camino mínimo que recorre todas las paradas seleccionará Prim y si desea conocer el camino mínimo entre dos paradas usará Dijkstra. Estas dos opciones son añadidos extras que hemos introducido en nuestro TFG y que al igual que la aplicación no eran necesarias.

Obtener el resultado: Una vez se selecciona la opción que desea el usuario el programa mostrara el resultado en la aplicación. Si no se selecciona algoritmo la aplicación no deja continuar y muestra un mensaje de alerta.

3.3 - Solución propuesta

Hemos diseñado el siguiente *mockup* de interfaz para guiarnos en el diseño de la misma durante su implementación.

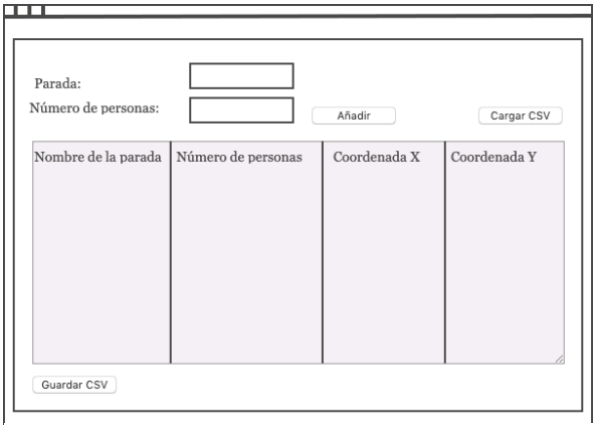


Ilustración 15 - Pantalla inicial (elaboración propia)

En esta pantalla el usuario podrá introducir el nombre de la parada, el número de personas de dicha parada, podrá si lo desea cargar un fichero CSV con dicha información o guardar un registro de paradas ya generado. La información introducida se mostrará en una tabla central.

Una vez finalizada la introducción de datos el usuario tendrá un botón de siguiente donde pasar a la siguiente interfaz.

En esta segunda pantalla es donde el usuario escogerá el algoritmo deseado de las tres opciones. Siendo la primera opción la que dará respuesta al problema de ruteo de un autobús escolar y las otras dos únicamente mostraran información sobre el camino mínimo de recubrimiento y mínimo entre dos puntos.

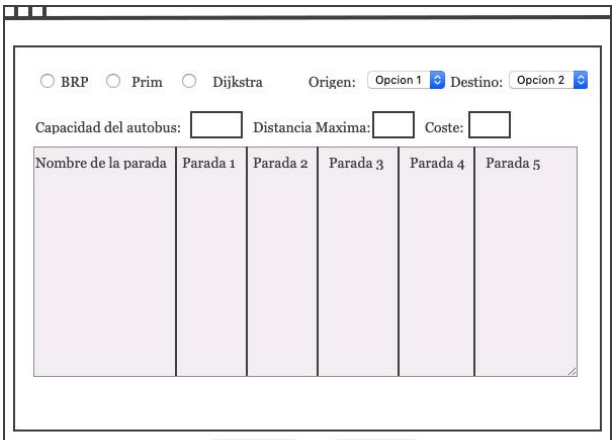


Ilustración 16 - Pantalla selección (elaboración propia)

En el algoritmo BRP el usuario deberá introducir: la capacidad, la distancia máxima a recorrer por el alumno y el coste de fletar el autobús.

Aparecerá una tabla con la matriz de distancias entre paradas, cuando el usuario de al botón siguiente procederá a calcular la solución el programa.

La tercera pantalla será donde el usuario pueda ver la solución al problema y escoger entre volver atrás, bien para modificar datos o escoger otro algoritmo, o ver la ruta de cada uno de los autobuses desde el botón de la parte superior derecha.

Ilustración 17 - Pantalla de solución (elaboración propia)

4. Diseño de la solución

Con la finalidad de obtener la mayor optimización posible para la resolución de nuestro algoritmo de *back-tracking* hemos escogido el lenguaje de programación C, ya que se trata de un lenguaje de carácter general pero que a su vez ofrece la oportunidad de acceso a memoria a bajo nivel. Este lenguaje de programación lo complementaremos con OpenMP, una API que nos permitirá dotar de concurrencia a nuestro algoritmo y de esta forma reducir el tiempo de ejecución.

Para llegar al mayor número de usuarios posibles y no cerrar su uso únicamente a un sistema operativo la aplicación será desarrollada en JAVA.

La unión entre ambos lenguajes se realizará mediante el uso de scripts programados específicamente para cada sistema operativo.

4.1 - Arquitectura del sistema

En la siguiente imagen puede observarse las diferentes partes en las que se ha divido el problema:

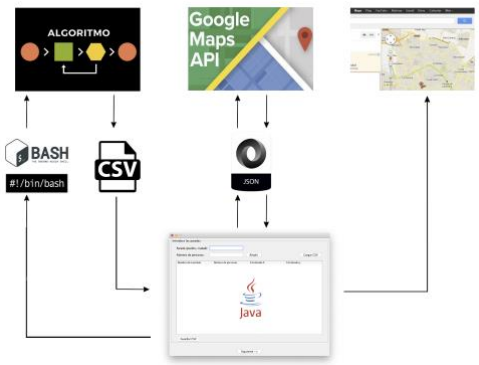


Ilustración 18 - Estructura de la aplicación (elaboración propia)

La aplicación de escritorio desarrollada en JAVA pide los datos para hacer el cálculo de la ruta al usuario. Este tiene la opción de insertarlos uno a uno o cargar un CSV con la información.

Con los datos introducidos, la aplicación se conecta utilizando JSON a la API de GoogleMaps con el fin de obtener las coordenadas de cada parada para calcular la distancia entre ellas.

Con los datos obtenidos de GoogleMaps y haciendo uso de scripts la aplicación compila y ejecuta el algoritmo en C, este algoritmo genera un fichero de texto con la solución que posteriormente la aplicación mostrara. Finalmente, el usuario puede decidir ver la ruta de cada autobús en GoogleMaps desde la aplicación.

Se puede ver las diferentes capas que forman la aplicación:

Capa de usuario: es la que verá el usuario, en ella es donde introduce los datos y donde verá los resultados. Hemos tenido en cuenta para su desarrollo los principios básicos que debe tener una interfaz gráfica: ser amigable, fácil de usar y tener filtrados todos los posibles casos de uso errores que pueda cometer el usuario en la introducción de la información.

Capa de negocio: en esta capa es donde incluimos los métodos y clases que obtienen la información de la interfaz, *scritps*, algoritmos y las peticiones de información a la API de GoogleMaps. Esta capa se encarga de recibir las peticiones del usuario y obtener los resultados.

Capa de datos: en este caso y debido a las características de la aplicación y a la facilidad de uso por parte de los usuarios hemos usado el formato de archivos CSV y TXT para almacenar y cargar la información.

4.2 - Tecnología utilizada

Las tecnologías escogidas y los motivos de su elección son los siguientes:

Lenguaje de programación C: Fue desarrollado entre los años 1969 y 1972 por Dennies Ritchie en los laboratorios Bell de AT&T. Es una evolución del lenguaje ensamblador y es utilizado en la mayor parte del *kernell* de UNIX.

Entre las características más destacadas que nos han llevado a la elección de este lenguaje hay que destacar:

- Es fácil de portar a otros lenguajes.
- Permite la programación en diferentes estilos.
- Se puede acceder a memoria de bajo nivel usando punteros, esta función es realmente útil para el desarrollo de nuestro algoritmo con técnicas de programación paralela.
- Al compilarse en cada máquina hace un uso más optimizado del *Hardware.*

En definitiva, el lenguaje C es muy flexible y potente y nos permitirá junto con OpenMP la creación de tareas para la programación paralela del algoritmo.

Lenguaje de programación JAVA: Fue creado en Sun Microsystems por James Gosling en 1982 y adquirido en 2010 por Oracle. Su sintaxis proviene de C y C++ pero no tiene tanto acceso a bajo nivel como ellos.

Se trata de un lenguaje de propósito general, orientado a objetos y concurrente. Fue diseñado específicamente para poder ser ejecutado en cualquier máquina y sistema operativo, de esta forma se evita el tener que reescribir código. Una vez compilada la aplicación a *bytecode* la máquina virtual de JAVA es la encargada de ejecutarla.

Hemos empleado este lenguaje para la creación de la interfaz debido al conocimiento de uso de la clase SWING para la creación de interfaces y al potencial que otorga JAVA a la hora de escribir un código que puede ser ejecutado en cualquier máquina independientemente de la arquitectura o sistema operativo.

API OpenMP: Se trata de una API de programación multiproceso de memoria compartida. Se utiliza para dar concurrencia a lenguajes escritos en C, C++ y Fortran, existen librerías que portan esta API a entornos JAVA como JOpenMP, pero nosotros en el proyecto hemos decido usar la versión nativa para C.

Es una API disponible para varios sistemas operativos como Unix y Windows. Básicamente se trata de un conjunto de rutinas de biblioteca, directivas de compilación y variables de entorno que modifican el comportamiento en tiempo de ejecución mediante la creación de hilos y tareas.

Se basa en el modelo *Fork-Join* paradigma que desciende de UNIX, básicamente se basa en dividir una tarea muy pesada en K-hilos con menor peso, recolectando al final de cada hilo los valores para finalmente unirlos mostrando un único resultado.

El uso de directivas OpenMP tiene como consecuencia una sincronización obligatoria de todo el bloque de código, es decir, este bloque se marcará como paralelo y se crearan hilos según las características de dicha directiva, cuando finaliza el bloque se realizará la sincronización de los diferentes hilos.

OpenMP también permite un paralelismo asíncrono. Desde la versión lanzada en 2013 permite la creación de tareas que irán procesándose a medida que estén disponibles hilos de ejecución en el procesador. Este paralelismo asíncrono es el que utilizaremos nosotros.

El uso de OpenMP lleva un sobre coste temporal en la ejecución del programa, este sobrecoste se ira amortizando a medida que aumente el potencial de paralelización de la tarea que se requiera paralelizar.

API GoogleMaps: En 2005 Google abre al mercado un servidor de aplicaciones de mapas para la creación de aplicaciones. En su inicio fue una API gratuita, actualmente es una API de pago, pero con un coste aplicable solo si se sobrepasan un número de consultas máximo mensual, por lo que para el desarrollo de nuestro proyecto y su uso futuro por los usuarios estimamos que no va a tener coste por su uso.
Para su uso es necesario una *Key*, clave de usuario que se obtiene de forma gratuita en la página web de la API. En el punto correspondiente explicaremos como el usuario puede obtener su API.
La API de GoogleMaps básicamente es un conjunto de funciones relacionadas con posicionamiento, distancia, coordenadas, tiempo de ruta, trafico real, etc.
Hemos escogido esta API porque:

- Nos permite obtener la distancia entre puntos para rellenar nuestra matriz de distancias.
- Nos permite estimar la duración de cada trayecto.
- Podemos mostrar el itinerario que debe seguir cada autobús con el tráfico que tiene cada zona.

JSON: Es el acrónimo de *Java Script Open Notation*, básicamente se trata de un formato de texto para el intercambio de información entre lenguajes, está empezando a reemplazar a XML. Suele utilizarse con mucha frecuencia en aplicaciones cliente-servidor.

Lo utilizamos para intercambiar información entre la aplicación JAVA y la API de GoogleMaps.

Scripts: Se trata de un lenguaje de programación interpretado, es decir, el código fuente no se compila para su ejecución. Para su ejecución hace uso de un intérprete de comandos. Nosotros hemos utilizado Bash para los sistemas UNIX y BATCH para Windows.

Utilizamos los scripts debido a que el algoritmo programado en C y OpenMP necesita compilarse para poder ser ejecutado, el fichero resultante de esta compilación solo puede ejecutarse en la máquina donde se compila. Con el Script lo que hemos hecho es crear un “programa” que compile el algoritmo en C en función del sistema operativo donde ejecutemos la interfaz, esto se hace de forma totalmente transparente para el usuario y evitamos que personas que desconozcan el uso de terminal o de compiladores de C puedan usar la aplicación.

También hemos creado unos scripts para que la ejecución del algoritmo en C sea totalmente transparente para el usuario.

Netbeans: se trata de un entorno de desarrollo que creo Sun MicroSystems en el 2000, actualmente pertenece a Oracle. Hemos escogido este entrono debido a que es un proyecto de código abierto y gratuito. Es un entorno de desarrollo modular que soporta todos los tipos de aplicaciones JAVA (J2SE, web, EJB y aplicaciones móviles). Adicionalmente, soporta el desarrollo de Aplicaciones empresariales con Java EE 5, incluyendo herramientas de desarrollo visuales de SOA, herramientas de esquemas XML, orientación a web servicies y modelado UML. Mediante la instalación de un *plugin* soporta proyectos de C/C++.

4.3 - Diseño detallado

En este punto explicaremos de forma detallada las diferentes partes de la aplicación y cómo funciona el algoritmo

4.3.1 - Diseño de la aplicación

En un primer lugar explicaremos la estructura de directorios de la aplicación, la estructura interna de la aplicación junto con sus métodos y clases, los componentes gráficos que aparecerán en la aplicación, la comunicación entre lenguajes de programación, la conectividad con la API de Google.

4.3.1.1 - Estructura de directorios

Hemos estructurado las diferentes partes que forman la aplicación en carpetas, en cada una de ellas se encuentra:

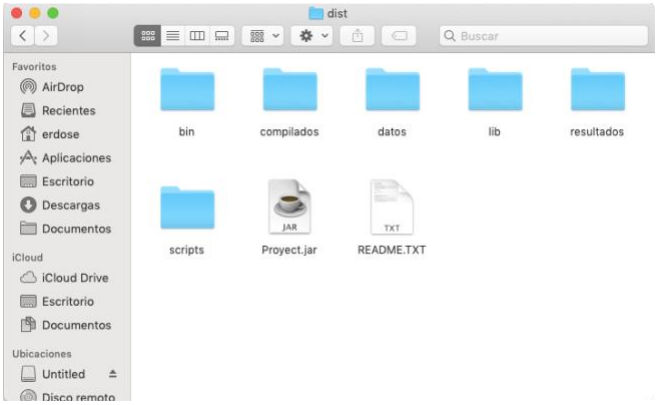


Ilustración 19 - Estructura de directorios (elaboración propia)

Bin: Aquí se encuentran los algoritmos en C sin compilar que utilizaremos para la resolución del problema de ruteo. En su interior podemos observar dos ficheros: BRP.c y BRP_openmp.c. El primero es el algoritmo sin paralelizar y el segundo el algoritmo paralelizado.

Compilados: En esta carpeta se encuentran los algoritmos compilados para nuestra máquina. En un inicio, esta carpeta estará vacía y será cuando ejecutemos por primera vez la aplicación cuando los scripts compilaran el algoritmo y lo depositaran en esta carpeta para que la aplicación pueda utilizarlo.

Datos: en esta carpeta hay depositados unos ficheros de rutas de ejemplo, para que el usuario pueda ver la estructura del fichero y lo modifique a sus necesidades

A medida que vamos ejecutando la aplicación en esta carpeta se guardan automáticamente los datos que el usuario va introduciendo con las características de su problema.

Este directorio también contiene un fichero muy importante y sin el cual nuestra aplicación no funciona, se trata del fichero *"key.txt"*. Este fichero contiene la clave de usuario que permite el uso de la API de GoogleMaps, si cuando arranca la aplicación este fichero está vacío la aplicación mostrara una señal de alerta indicándolo.

Lib: esta carpeta tiene algunas librerías que utilizamos para obtener y representar información en nuestra aplicación. Entre ellas están librerías como *OpenCSV* que se encarga de facilitarnos el acceso, tanto de lectura como de escritura, a ficheros CSV.

Resultados: como su nombre indica esta carpeta contiene el conjunto de resultados que genera el algoritmo. Serán dos ficheros, uno con el tiempo de ejecución y otro con los valores de la solución óptima. El usuario no necesita abrir estos ficheros para conocer la solución, la aplicación se encarga de una vez han sido generados mostrarlos en la interfaz gráfica de forma totalmente transparente para el usuario.

Scripts: esta carpeta contiene los diferentes scripts para la compilación y ejecución del algoritmo en C desde la interfaz en Java. Cada sistema operativo hace uso de su script de compilación y de su script de ejecución por lo que la carpeta contendrá un total de 6 ficheros.

Fichero "Proyecto.jar": es la aplicación multisistema JAVA compilada para su ejecución.

Fichero "leeme.txt": contiene información técnica sobre la configuración del fichero "key.txt" y modificación de Scripts para adaptarlos a la ruta donde cada usuario tenga instalado el compilador de C.

4.3.1.3 - Estructura interna de la aplicación

En este punto vamos a explicar en qué paquetes hemos estructurado la aplicación.

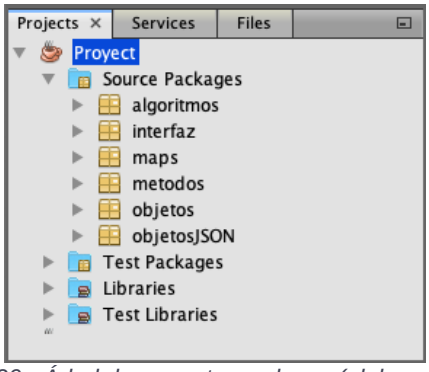


Ilustración 20 - Árbol de paquetes y clases (elaboración propia)

Paquete algoritmos: contiene el conjunto de los diferentes algoritmos que utilizamos en la aplicación. Contiene las clases:

- BRP.c: es el algoritmo de *back-tracking* escrito en lenguaje de programación C y sin optimizar mediante el uso OpenMP. Este algoritmo no lo ejecuta la aplicación, lo utilizamos nosotros para hacer la comparativa entre el algoritmo optimizado y no optimizado.

- BRP_openmp.c: contiene el algoritmo paralelizado utilizando OpenMP.

- Dijkstra.java: esta clase contiene el algoritmo que empleamos para calcular la distancia mínima entre dos puntos.

- Prim.java: clase que contiene el algoritmo que nos muestra el árbol de recubrimiento mínimo.

Paquete interfaz: aquí se encuentran las clases que contienen las diferentes pantallas de la aplicación. Estas pantallas las hemos estructurado en:

- Inicio: contenedor FPanel que hace de base sobre la que mostraremos los diferentes JPanel. En ella se encuentran los métodos encargados de las llamadas a la clase "control.java" y al intercambio de información con la clase "datos.java".

- JPanel0, JPanel1 y JPanel2: son los diferentes paneles donde el usuario introduce, selecciona y obtiene la información del problema.

- control.java: contiene las variables y métodos que manejan el desplazamiento entre las diferentes pantallas.

Paquete *maps:* aquí se encuentran todas las clases que hemos creado para la comunicación con la API de GoogleMaps, están explicadas con más detalle en putos posteriores.

Paquete métodos: contiene algunos métodos utilizados en la aplicación para: interpretar, leer y crear ficheros CSV.

Paquete objetos: contiene clases que nos sirven como estructura para nuestro proyecto. Hay clases con más métodos de los utilizados por si en un futuro se desea ampliar la aplicación con más opciones, como por ejemplo la de añadir los nombres de los alumnos de cada autobús y parada para llevar un control de ellos.
Paquete objetosJSON: este paquete contiene la clase que utilizamos para formatear la información recibida por la API de GoogleMaps, nos permite la creación de objetos con información para nuestra aplicación.

4.3.1.4 - Componentes gráficos

El paquete Swing de JAVA nos facilita la construcción de interfaces gráficas de usuario. En este punto vamos a explicar los componentes que hemos usado en la creación de la interfaz.

Los componentes gráficos son elementos que permiten interactuar al usuario con el sistema. En JAVA cada componente tiene su clase, para poder usarlo es necesario instanciar su clase, esto es, para usar un componente de texto deberemos crear un objeto de la clase *JTextArea*.

```
1  JTextArea jtx = new JTextArea();
2  JTextArea jtx = new JTextArea(10,50);
3  JTextArea jtx = new JTextArea( "Texto del JTextArea" );
```

Ilustración 21 – Código invocación JTextArea (elaboración propia)

La primera línea crea el objeto JTextArea, la segunda línea le indica el número de elementos y columnas, la tercera introduce el texto.

Existen diferentes categorías que dependen de un pequeño árbol de herencias, a continuación, vamos a clasificar algunos componentes gráficos según su funcionalidad.

Contenedores: son los encargados de establecer la base sobre la que diseñaremos la interfaz gráfica. Existen contenedores principales como el *JFrame* y *JDialog*, pero también existen otros tipos con funcionalidades diferentes dentro de la interfaz.

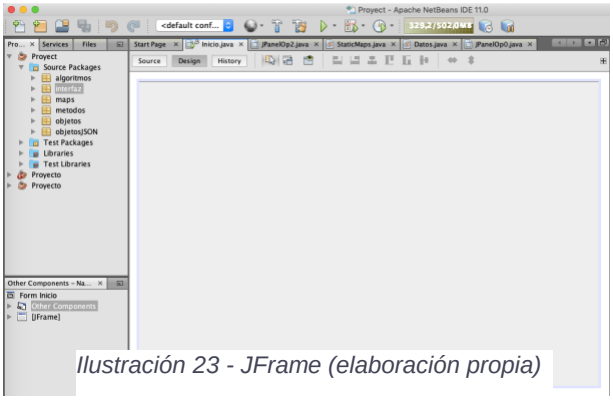


Ilustración 23 - JFrame (elaboración propia)

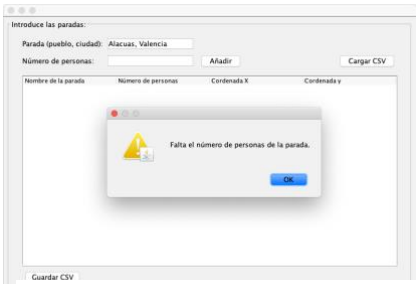


Ilustración 22 - JDialog (propia)

- *JFrame*: contenedor principal de la aplicación.
- JDialog: se trata de una ventana de tipo diálogo, en ocasiones puede ser el contenedor de la aplicación principal.
- *JPanel*: se utiliza para la creación de paneles independientes en los que insertar otros componentes. Nosotros hemos diseñado 3 paneles distintos, cada uno para una sección de la aplicación. El control y avance entre paneles esta explicado de forma detallada más adelante.

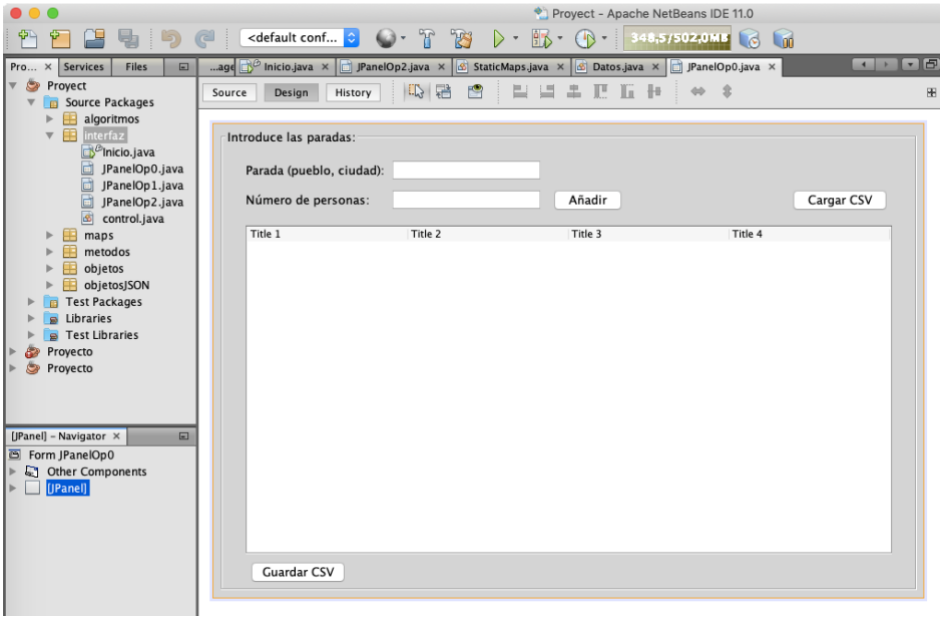


Ilustración 24 – Jpanel (elaboración propia)

- *JScrollPane:* se utiliza para enmarcar un área de texto y darle opción de desplazamiento vertical, horizontal o ambos.
- *JSplitPane:* sirve para la división del panel en dos áreas.
- *JTabbedPane:* Para la creación de pestañas, donde cada pestaña es un contenedor independiente, suele usarse en menús de configuración.
- *JDesktopPane:* para la creación de ventanas dentro de la ventana principal.

- *JToolBar:* es la barra de herramientas de la aplicación.

Componentes atómicos: Se trata de elementos que no permiten almacenar otro tipo de objetos o elementos gráficos. En nuestra interfaz hemos utilizado numerosos elementos de este tipo.

- *JLabel:* utilizados para introducir imágenes o texto fijo.
- *JButton:* nos permiten colocar botones de acción en la aplicación.
- *JCheckBox:* se trata de casillas de verificación, nos permite la selección múltiple de objetos.
- *JRadioButton:* permite la selección de una única opción entre varias disponibles. Básicamente se trata de un *JCheckBox* que, en lugar de permitir la selección múltiple, permite la selección de una única opción. Si se desea puede configurarse como un *JCheckBox* aunque el diseño visual de este elemento al usuario le transmitirá la sensación de que únicamente debe escoger una.

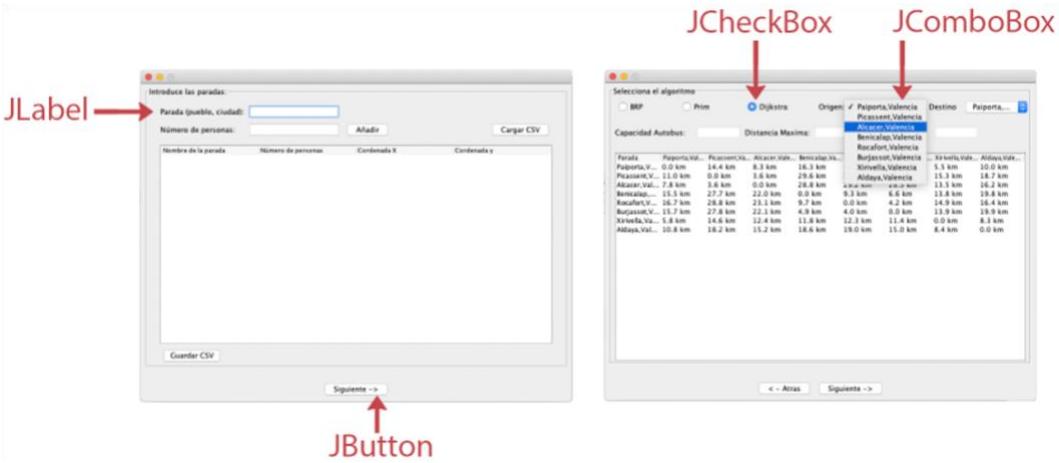


Ilustración 25 - Componentes atómicos (elaboración propia)

- *JToggleButton:* este botón se queda presionado hasta que ocurra otra acción que lo deshabilite.
- *JComboBox:* muestra una lista de elementos de selección dentro de un botón desplegable.
- *JScrollBar:* inserta una barra de desplazamiento horizontal, vertical o ambas en un área de texto.
- *JSeparator:* usado como su nombre indica para separar opciones.
- *JSlider:* añade un deslizador a nuestra ventana de aplicación.
- *JSpinner:* es una caja de texto con botones integrados que permiten seleccionar algún valor.
- *JProgressBar:* barra de progreso.

Componentes de texto: este tipo de elementos nos permiten trabajar con cadenas de texto, tanto para entrada como para salida de información.

- *JTextField:* para la introducción de un campo de texto simple.

- *JFormattedTextField:* permite la introducción de un campo de texto simple preestableciendo el formato de dicho campo.
- *JPasswordField:* sirve para ocultar el texto que el usuario introduce en este campo. Usado para insertar contraseñas.
- *JTextArea:* introduce áreas de texto de mayor tamaño que un *JTextField*, también permite mostrar texto procedente de otro método, clase u objeto de la aplicación.
- *JEditorPane:* otorga propiedades de formato al área de texto.
- *JTextPane:* similar al anterior, pero con muchas más opciones.

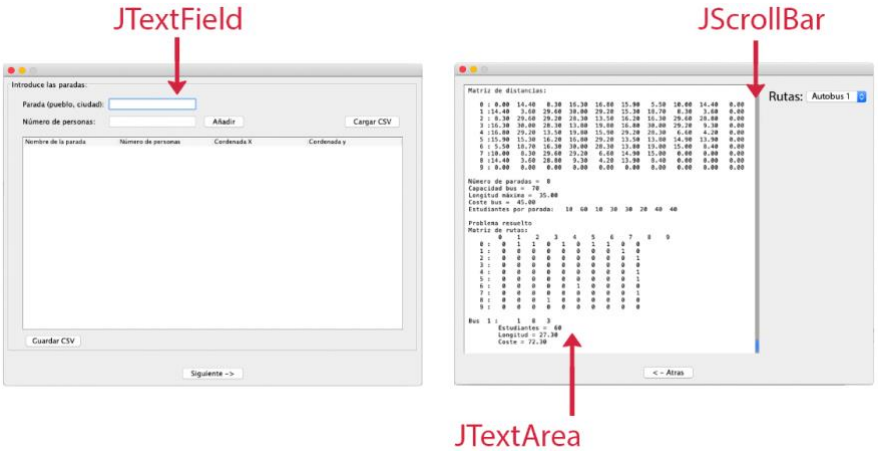


Ilustración 26 - Componentes de texto

Componentes de menús: añaden opciones de menú a las aplicaciones.

- *JMenuBar:* permite vincular una barra de menús.
- *JMenu:* permite vincular botones o enlaces que al ser pulsados despliegan un menú principal.
- *JMenuItem:* botón u opción que se encuentra en un menú.
- *JCheckBoxMenuItem:* es un elemento del menú con opciones de *CheckBox*.
- *JRadioButtonMenuItem:* es un elemento del menú como botón de selección.
- *JPopupMenu:* ventanas de menú emergentes.

Componentes complejos: se trata de elementos más complejos que cumplen con funciones enfocadas a procesos más específicos avanzados. Se utilizan para obtener información de base de datos, etc.

- *JTable:* muestra una tabla de datos con sus filas y columnas.
- *JTree:* se muestra un árbol que representa una jerarquía de visual como si fuese un tipo de directorio.
- *JList:* carga una lista de elementos, dependiendo de las propiedades puede tenerse una lista de selección múltiple.
- *JFileChooser:* componente que permite la búsqueda y selección de ficheros entre otras opciones.

- *JColorChooser:* componente que permite cargar un panel selector de color.
- *JOptionPane:* no es algo complejo sino más un componente independiente que permite mostrar un cuadro de diálogo personalizable.

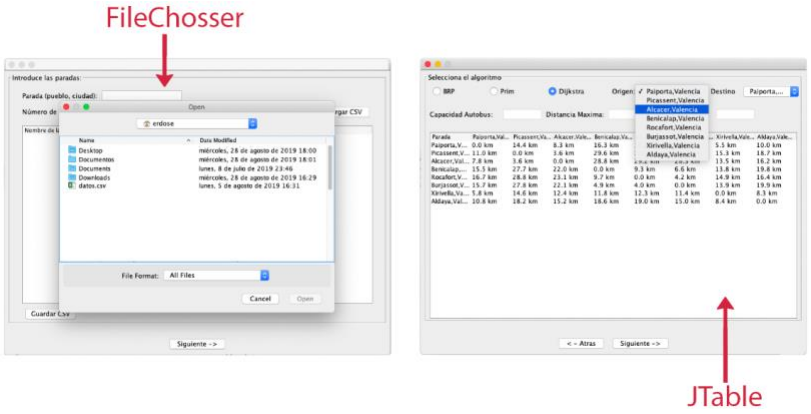


Ilustración 27 - Elementos complejos (elaboración propia)

A continuación, se encuentran numeradas todos los paneles por los que pasa la aplicación hasta obtener el resultado, a pesar de estar generados y programados con el fin de evitar ampliar con demasiadas imágenes el proyecto hemos obviado la inclusión de mensajes de alerta por introducción de texto incorrecto, ausencia de *clave*, etc.

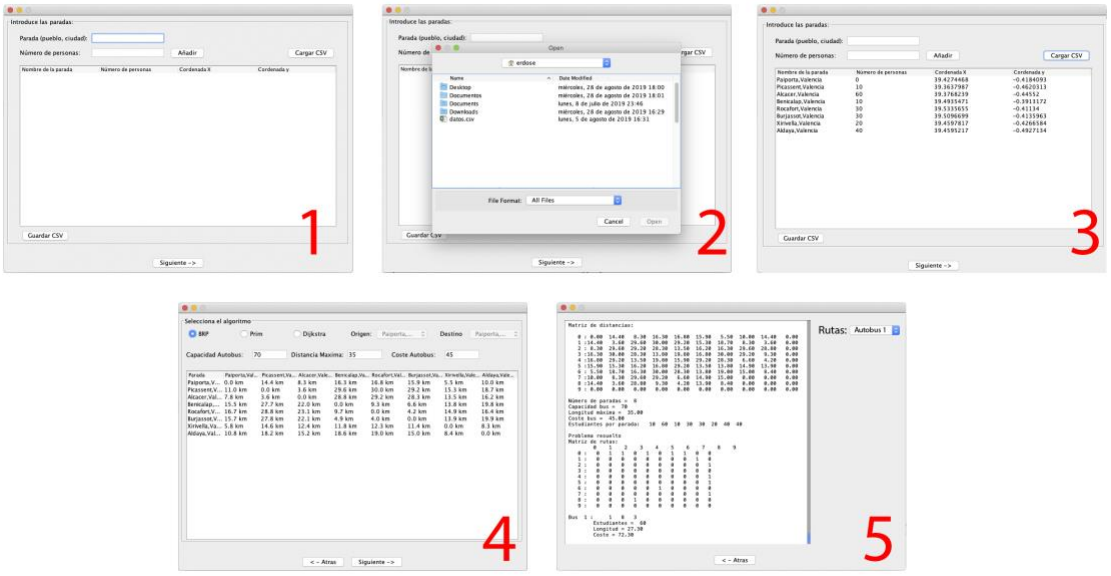


Ilustración 28 - Paneles de la aplicación (elaboración propia)

Si el usuario desea puede ver la ruta en la web pulsando el botón superior la imagen 5.

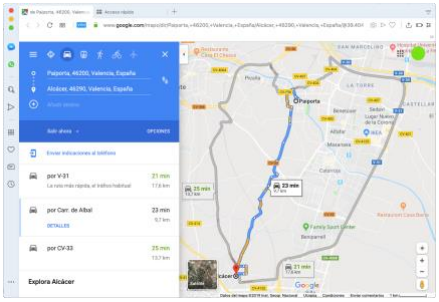


Ilustración 29 - Ruta web (elaboración propia)

4.3.1.5 - Comunicación entre lenguajes

Para poder hacer funcionar nuestro algoritmo en C desde la aplicación JAVA hemos hecho uso de la clase "java.lang.Runtime.exec()".

Esta clase lo que permite es tener acceso a la terminal del sistema operativo, utilizaremos esto para poder ejecutar los Scripts de compilación y ejecución.

```
//COMPILAR EL ALGORITMO
Runtime r = Runtime.getRuntime();
try {
    Process p = r.exec (rutaCom);
    try (BufferedReader input = new BufferedReader(new InputStreamReader(p.getInputStream()))) {
        while ((line = input.readLine()) != null) { System.out.println(line);}
    }
}
catch (IOException err) {}
```

Ilustración 30 - Fragmento de código comunicación JAVA y C (elaboración propia)

"rutaCom" es una variable de tipo *String* en la que almacenamos la ruta donde se encuentra el *script* de compilación, para ejecutar el *script* de ejecución el código es el mismo cambiando rutaCom por rutaEje.

Destacar aquí que para la creación de esta ruta y todas las rutas, que son de acceso tanto de lectura como de escritura de ficheros, hay que tener en cuenta que los sistemas operativos utilizan diferentes símbolos para la separación de directorios. En Unix utilizan el símbolo "/" en cambio el sistema Windows utiliza "\". Para hacer frente a esta problemática hemos creado el siguiente código:

```
//VER EL SO
String line;
String sistema = System.getProperty("os.name").toLowerCase();
String ruta = Datos.getDirectorio();
String rutaCom = "";
String rutaEje = "";
if (sistema.indexOf("win")>=0){
    rutaCom = ruta+File.separator+"scripts"+File.separator+"com_win.bat";
    rutaEje = ruta+File.separator+"scripts"+File.separator+"eje_win.bat";
}
else if (sistema.indexOf("lin")>=0){
    rutaCom = ruta+File.separator+"scripts"+File.separator+"com_lin.sh";
    rutaEje = ruta+File.separator+"scripts"+File.separator+"eje_lin.sh";
}
else if (sistema.indexOf("mac")>=0){
    rutaCom = ruta+File.separator+"scripts"+File.separator+"com_mac.sh";
    rutaEje = ruta+File.separator+"scripts"+File.separator+"eje_mac.sh";
}
else {
    JOptionPane.showMessageDialog(null, "Sistema operativo invalido.", nul
    System.exit(1);
```

Ilustración 31 - Fragmento de código para obtener el sistema operativo (elaboración propia)

En el hacemos uso de la clase *"System.getProperty("os.name")"* para obtener el nombre del sistema operativo, posteriormente mediante el uso de estructuras *IF* y *ELSE* filtramos el sistema operativo en el cual nos encontramos y generamos las rutas de ambos *scritps*. La clase "*File.separator*" es la encargada de en función del sistema operativo colocar el símbolo "\" o "/".

4.3.1.6 - Conectividad con la API de Google

Una API es una herramienta que nos permite utilizar un sistema sin necesidad de conocer como este está desarrollado. En este caso la API de GoogleMaps fue lanzada en 2005 y nos permite utilizar los servicios de posicionamiento y geolocalización sin tener que conocer como están estos desarrollados (conexión con sus satélites, mapeo de imágenes, etc.).

Por tato, lo que nos está ofreciendo es el acceso a diversos servicios entre los que se encuentran: gestión de rutas, imágenes satélites, trafico en tiempo real y muchos otros.

Nosotros lo utilizaremos para conocer las coordenadas geográficas de las paradas del problema y la distancia entra ellas, de esta forma evitamos que el usuario tenga que introducir la distancia entre paradas en una tabla, información que probablemente desconozca.

La comunicación entre la aplicación y la API de Google se realiza mediante URLs con un formato especifico que incluyen diversos tipos de parámetros que esperamos en la respuesta como: nombre del lugar, formato en el que se desea recibir la distancia, formato en que se desea recibir el tiempo, tipo de imagen del mapa, formato de la imagen mostrada, etc.

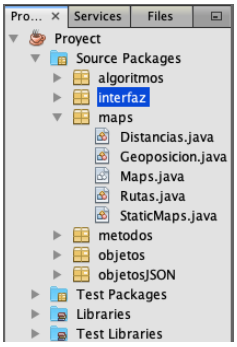


Ilustración 32 - Árbol de clases para la comunicación con API GoogleMaps (elaboración propia)

Geposicion.java: en este fichero se encuentra el codigo que nos permite obtener las coordenadas geograficas de una parada. Un fragmento de codigo importante dentro de este fichero es el siguinete:

```
private String urlGoogleGeo = "https://maps.googleapis.com/maps/api/geocode/json?";

public String getUrlGoogleGeo() {return this.urlGoogleGeo+"address=";}

//METODOS DE GEOPOSICION
public String codificarURL(String dir) throws UnsupportedEncodingException{
    String dirCod = URLEncoder.encode(dir, "UTF-8");
    String url = getUrlGoogleGeo()+dirCod+"&key="+Datos.getGoogleKey();

    return url;
}
```

Ilustración 33 - Fragmento código Geoposición.java (elaboración propia)

Como podemos ver, la variable "*urlGoogleGeo*" es la encargada de almacenar la base de la dirección URL que solicita google para la obtención de la geoposición.

El método "*codificarURL*" lo que nos permite es mediante el paso como parámetro del nombre de la parada como elemento de tipo *String* codificarlo como UTF-8, añadir nuestra clave para usar la API y codificar dicho texto para que cumpla con las directrices de un elemento URL. Esta clase de JAVA es la que utilizamos en la primera vista de la aplicación para completar la tabla con las coordenadas geográficas.

Distancias.java: esta clase es la encargada de obtener la matriz de distancias entre las paradas. Los métodos más destacables en ella son:

```
private String codificarURL(String url) throws UnsupportedEncodingException{
    return URLEncoder.encode(url, "UTF-8");
}

public  String codificarURLDistancias(String uni,String ori, ArrayList dest) throws UnsupportedEncodingException{
    String units="units=metric";
    String origenes = "";
    String destinos = "";

    if ("imperial".equals(uni)) units = "units=imperial";

    origenes = origenes + ori;
    for (int i = 0; i<dest.size();i++){
        destinos= destinos+dest.get(i).toString();
    }

    String url = getUrlGoogleDistancias()+units+"&origins="+codificarURL(origenes)+"&destinations="
            +codificarURL(destinos)+"&key="+Datos.getGoogleKey();


    return url;
}
```

Ilustración 34 - Fragmento código Distancias.java (elaboración propia)

El método "*codificarURL*" lo que nos permite es establecer el texto introducido en los JTextField en formato UTF-8 requisito obligatorio evitar errores en la formación de URL.

El método "*codificarURLDistancias*" es el utilizado para obtener la *String* (que posteriormente deberá codificarse como URL) con los parámetros que deseamos obtener como respuesta de la API de GoogleMaps. Observamos así que al método hay que pasarle: *String* con la unidad de medida de las distancias (hemos preestablecido las unidades métricas), un *String* con el origen y un *ArrayList* con objetos de tipo destino.

Con estos tres parámetros el metro construido es el encargado de formatear nuestra dirección para obtener la cadena de texto que utilizaremos para hacer la llamada a Google y recibir una respuesta.

```
public MatrizDistancias respuestaMatrizDistancias(String origenDestinos) throws MalformedURLException {

    URL url;
    HttpURLConnection conn;
    BufferedReader rd;
    String line;
    String resultado = "";
    try {

        //+URLEncoder.encode(destino, "UTF-8");
        url = new URL(origenDestinos);
        conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        while ((line = rd.readLine()) != null) {
            resultado += line;
        }
        rd.close();
    }
    catch (IOException e) { e.printStackTrace();}
    catch (Exception e)   { e.printStackTrace();}

    MatrizDistancias mDistancias = new Gson().fromJson(resultado, MatrizDistancias.class);

    return mDistancias;
}
```

Ilustración 35 - Fragmento código comunicación Distancias.java (elaboración propia)

El método "*respuestaMatrizDistancias*" es un método que devuelve un objeto JSON creado con la información recopilada de hacer la llamada con la URL generada en el método descrito anteriormente. La información que devuelve la API GoogleMaps y que nosotros formateamos tiene el siguiente aspecto:

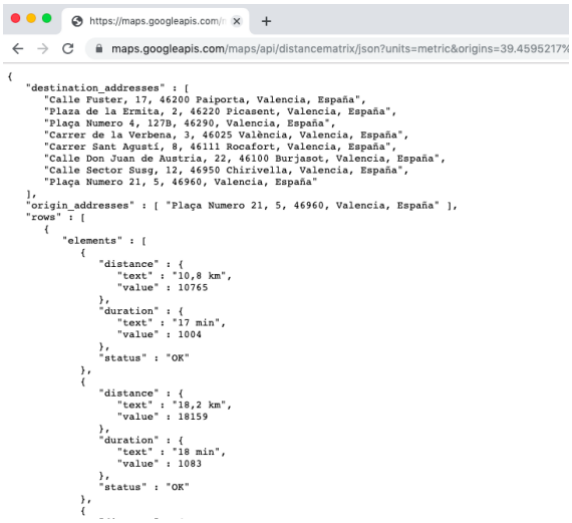


Ilustración 36 - Respuesta consulta API GoogleMaps (elaboración propia)

Es el mismo método el que se encarga de dar formato a esta información rellenando cada componente del objeto JSON generado con la información que le corresponde y necesitamos para rellenar entre otras cosas la matriz de distancias entre las rutas.

El resto de clases que incluye el paquete Maps (“Maps.java”, “Rutas.java”, “StaticMaps.java”) son clases similares que se encargan de obtener otro tipo de información como imágenes de mapas o información relevante para el resto de clases a la hora de hacer consultas y obtener respuesta.

4.3.2 - Diseño del algoritmo

Para el diseño del algoritmo vamos a utilizar el lenguaje de programación C y lo optimizaremos mediante el uso de OpenMP para utilizar toda la potencia computacional del equipo donde se ejecute.

Vamos a proceder a explicar cómo hemos estructurado nuestro algoritmo, este puede encontrarse íntegramente en el anexo.

1. El algoritmo tras la introducción (lectura de fichero) de los datos creara en memoria un vector con la matriz distancias añadiéndole una fila y una columna más que nos servirán para indicaran si el autobús a llegado al número máximo de alumnos, si sobrepasa la distancia máxima o si dicha ruta ya ha sido revisada por otro hilo.

Matriz de distancias:

0 :	0.00	14.40	12.20	16.30	16.80	15.90	5.50	10.00	0.00
1 :	14.40	0.00	3.60	29.60	30.00	29.20	15.30	18.70	0.00
2 :	12.20	3.60	0.00	24.30	24.70	23.80	13.50	16.20	0.00
3 :	16.30	29.60	24.30	0.00	9.30	6.60	13.80	19.80	0.00
4 :	16.80	30.00	24.70	9.30	0.00	4.20	14.90	16.40	0.00
5 :	15.90	29.20	23.80	6.60	4.20	0.00	13.90	15.40	0.00
6 :	5.50	15.30	13.50	13.80	14.90	13.90	0.00	8.30	0.00
7 :	10.00	18.70	16.20	19.80	16.40	15.40	8.30	0.00	0.00
8 :	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Ilustración 37 - Matriz distancias generada (elaboración propia)

2. En la siguiente parte del código mostraremos la información que usuario ha introducido.

```
Número de paradas =  7
Capacidad bus =  65
Longitud maxima =  70.00
Coste bus =  70.00
Estudiantes por parada:  10  60  10  30  30  20  40
```

Ilustración 38 - Entrada de datos (elaboración propia)

3. Luego ya se procede a la ejecución de las condiciones que recorrerán todas las posibles opciones. Finalmente mostraremos el resultado en caso de haberlo o indicaremos que no hay solución si no la hay. Como puede observarse en la siguiente imagen el hecho de que cada columna solo tenga un "1" indica que se cumple con la condición de que solo puede visitarse una parada por autobús.

```
Problema resuelto
Matriz de rutas:
     0  1  2  3  4  5  6  7  8
 0:  0  0  1  0  0  1  1  1  0
 1:  0  0  0  0  0  0  0  0  1
 2:  0  0  0  0  0  0  0  0  1
 3:  0  0  0  0  0  0  0  0  1
 4:  0  0  0  0  0  0  0  0  0
 5:  0  0  0  0  1  0  0  0  0
 6:  0  0  0  1  0  0  0  0  0
 7:  0  1  0  0  0  0  0  0  0
 8:  0  0  0  0  0  0  0  0  0

Bus 1:  2
        Estudiantes =  60
        Longitud = 12.20
        Coste = 82.20
Bus 2:  5  4
        Estudiantes =  60
        Longitud = 20.10
        Coste = 90.10
Bus 3:  6  3
        Estudiantes =  30
        Longitud = 19.30
        Coste = 89.30
Bus 4:  7  1
        Estudiantes =  50
        Longitud = 28.70
        Coste = 98.70

Coste total = 360.30
```

Ilustración 39 - Resultado (elaboración propia)

4. Tras la explicación del funcionamiento y estructura del algoritmo de back-tracking utilizado para resolver el problema de enrutamiento de un autobús escolar vamos a explicar las diferentes opciones que nos ofrecía OpenMP para su paralelización y cuales hemos escogido.

La API OpenMP permite mediante el modelo de programación *fork-join* la posibilidad de generar hilos de ejecución de subtareas. Inicialmente un único hilo ejecuta la aplicación hasta que llegamos al código OpenMP donde se encuentra el primer constructor paralelo, dentro de este constructor es donde posteriormente se generaran hilos esclavos y en el él donde se organizará la ejecución de cada uno de estos hilos.

El número de hilos que genera se obtiene por entrada de parámetros en la llamada de ejecución, si no se pasa como parámetro crea el máximo número de hilos disponible para nuestro procesador. Nosotros por defecto hemos establecido este parámetro para que genere el mayor número de hilos permitido.

La forma de hacer una llamada a OpenMP dentro del código de C es la siguiente:

```
#pragma omp parallel
#pragma omp single
BRP( 0, 0, X, 0, 0, 0 );
```

Ilustración 40 - Código OpenMP (elaboración propia)

Este fragmento de código sería el correspondiente a la llamada que indicaría el inicio de la sección paralela ("*#pragma omp parallel*") y el otro a la ejecución de la siguiente llamada a un método por un único hijo. A continuación de la llamada "*parallel*" pueden añadirse diversas formas de acceso a las variables de dentro del siguiente bloque de

código de la sección paralela, como: *private, firstprivate, deafult, shared, copyin y reduction.* Esta serie de variables de entrono se establecen sobre todo para establecer cómo deben acceder los hilos cuando se ejecutan bucles *for* o *while* en paralelo.

Sobre la información de acceso a bucles utilizando programación paralela hay información muy extensa donde pueden apreciarse todos los casos posibles. En nuestro caso como no hemos utilizado la paralelización de bucles (debido a que su coste computacional no era elevado) vamos a obviar todas las posibilidades que existen y nos centraremos en una opción que ofrece OpenMP como la creación de tareas o "*task*".

La posibilidad de crear tareas en OpenMP está presente desde la versión 3.0. La creación de tareas nos permite paralelizar fragmentos de código irregulares y que acceden a memoria. De esta forma podremos dividir el problema en subproblemas más simples que serán ejecutados por hilos independientes de nuestro procesador.

```
for( int j=1; j<n; j++ ) {
    if( i==0 ) {
        /* Autobus nuevo */
        if( j<n-1 && parada_valida( j, X, q[j-1], D(i,j) ) ) {
            float *X_aux = ( float* ) malloc( n*n*sizeof(float) );
            memcpy( X_aux, X, n*n*sizeof(float) );
            X_aux( i, j ) = 1;
            #pragma omp task
            {
            BRP( j, visitadas+1, X_aux, q[j-1], D(i,j), coste+coste_bus+D(i,j) );
            free(X_aux);
            }
        }
    }
```

Ilustración 41 - Uso de tareas (elaboración propia)

La imagen anterior hace referencia a la primera tarea generada. Si recordamos el diagrama UML visto anteriormente, haría referencia a la primera tarea creada. En él lo que se hace es:

1. El bucle *for* recorre un vector donde se encuentran las paradas.

2. Se comprueba si la parada ha sido visitada o no. Los "0" indica que la parada no ha sido visitada y los "1" indican que si han sido. Si la parada no ha sido visitada también debe comprobarse si la parada es válida.

3. Posteriormente reservamos memoria y copiamos el estado actual de la memoria. Hacemos esto para que la tarea pueda acceder a la memoria sin que los hilos que la ejecuten puedan variar el estado de sus variables.

4. Generamos la tarea que consistirá en llamar al algoritmo de back-tracking desde un hilo independiente. Es importante remarcar la liberación de memoria tras la ejecución de la tarea para no saturar el sistema.

El acceso a la memoria debe hacerse de forma privada, es decir, usando *"#pragma omp task private*" pero al ser el tipo de acceso por defecto privado nosotros no lo hemos puesto. No vamos a comentar el resto de tareas ya que se crean siguiendo el mismo patrón.

El acceso a la sección critica se define de la siguiente manera:

```
int n = n_paradas+2;
if( visitadas == n_paradas ) {
    if( coste < coste_minimo ) {
        #pragma omp critical
        if( coste < coste_minimo ) {
            coste_minimo = coste;
            memcpy( X_mejor, X, n*n*sizeof(float) );
        }
    }
}
```

Ilustración 42 - Sección critica (elaboración propia)

Establecemos como sección critica el momento en que el hilo intenta acceder a la comparación de si el coste de la ruta desarrollada es menor que el que tenía. Si no estableciésemos esta sección como critica el valor de la variable iría variando a medida que los hilos accedieran a ella y no podríamos compararlo con el valor que tenía cuando el hilo accedió a ella, por tanto, no obtendríamos el resultado esperado.

Hay que destacar la aparición de un doble bucle *if* con la misma condición, se realiza para hacer una función de filtro y descartar posibles hilos generados entre el transcurso de la lectura de código.

También hay que tener en cuenta que cuando finaliza una tarea y esta accede a la sección critica, pueden estar corriendo otras tareas con valores mayores (y por tanto no deseados), estos valores no influirán en el resultado ya que cuando intenten acceder a la sección critica serán descartados pero sí que influirán de alguna forma en la utilización de recursos del sistema, por esto (entre otras cosas) el tiempo de ejecución del algoritmo puede variar de una vez a otra utilizando los mismos parámetros.

5. Implantación

En este apartado trataremos temas relacionados con la instalación del programa, las dependencias en rutas de compiladores que existen, la modificación de scripts en caso que sea necesario y una breve descripción de cómo obtener la clave de GoogleMaps para poder usar la aplicación.

En primer lugar, hay que tener claro que, para el correcto funcionamiento de Java, C y OpenMP hay que tener instalado el software, compiladores y APIs correspondientes a cada uno de ellos, ya que los tres son herramientas que no vienen preinstaladas por defecto (en la mayoría de ocasiones) en ningún sistema operativo.

5.1 - Instalación de las herramientas

Para la instalación de JAVA podéis consultar la siguiente página web donde se explica detalladamente y con enlaces de descarga como hacerlo en los diferentes sistemas operativos www.java.com/es/download/help/download_options.xml.

Existen diversas opciones para la instalación de C en los diferentes sistemas operativos y en función del compilador seleccionado el código de compilación y ejecución será diferente. Este aspecto es un aspecto a tener muy encuentra para el correcto funcionamiento del programa, pues si no se usa el mismo que hemos utilizado nosotros, será necesario una pequeña variación en los scripts para que la aplicación funcione correctamente (esta variación será explicada en el punto siguiente).

Nosotros hemos utilizado los siguientes compiladores en cada sistema operativo:

- Windows: Cl podéis encontrar una guía de instalación aquí: https://docs.microsoft.com/es-es/cpp/build/walkthrough-compiling-a-native-cpp-program-on-the-command-line?view=vs-2019
- Linux: Gcc podéis encontrar una guía de instalación aquí: https://linuxize.com/post/how-to-install-gcc-compiler-on-ubuntu-18-04/
- Mac: Clang podéis encontrar una guía de instalación aquí: https://embeddedartistry.com/blog/2017/2/20/installing-clangllvm-on-osx

Para la instalación de OpenMP en windows, linux y mac se puede encontrar información en el siguiente enlace de su página web www.openmp.org/resources/openmp-compilers-tools

5.2 - Modificación de los scripts

Básicamente la función de un script es automatizar tareas que se ejecutan por consola de comandos. Como para la compilación y ejecución de código C es necesario hacer uso de comandos en terminal, la edición de los *scripts* en caso de no usar los mismos compiladores es necesaria.

Estructura del script de compilación en Linux:

```
#!/bin/bash
gcc brp/BRP.c -o compilados/BRP;
gcc brp/BRP_openmp.c -fopenmp -o compilados/BRP_openmp;
gcc brp/BRP_openmp.c -fopenmp -o compilados/BRP_openmp_ori;

exit 0
```

Ilustración 43 - Script Linux (elaboración propia)

en el inicio de la línea dos, tres y cuatro aparece el campo “gcc” hace referencia al compilador que estamos usando, si usamos otro compilador deberemos editar este campo por la llamada del compilador que usemos.

Estructura del script de compilación en Mac:

```
#!/bin/bash
/usr/local/opt/llvm/bin/clang -L/usr/local/opt/llvm/lib brp/BRP.c -o compilados/BRP
/usr/local/opt/llvm/bin/clang -fopenmp -L/usr/local/opt/llvm/lib brp/BRP_openmp.c -o compilados/BRP_openmp
/usr/local/opt/llvm/bin/clang -fopenmp -L/usr/local/opt/llvm/lib brp/BRP_openmp_ori.c -o compilados/BRP_openmp_ori

exit 0
```

Ilustración 44 - Script Mac (elaboración propia)

el campo “/usr/local/opt/llvm/bin/clang” hace referencia al compilador que hemos usado para mac, en este caso “clang”. Al igual que ocurria en Linux, si usamos otro deberemos modificarlo para que haga la llamada al compilador de forma correcta.

Para finalizar, en Windows la estructura del script es:

```
@echo off
cl src/BRP.c /out:BRP.c
cl src/BRP_openmp.c /out:BRP_openmp.c

MOVE /Y BRP bin
MOVE /Y BRP_openmp bin

exit
```

Ilustración 45 - Script Windows (elaboración propia)

si se utiliza otro compilador debera modificarse el comando “cl” por el correspondiente a la llamda del nuevo compilador.

5.3 - Obtener clave de GoogleMaps

El servicio que ofrece Google para obtener información satélite, coordenadas, etc. comenzó siendo un servicio gratuito, posteriormente ha pasado a ser un servicio de pago, pero para un uso “no abusivo” las tarifas son realmente reducidas y si tenemos en cuenta que nos otorgan con la creación de la cuenta 220€ anuales en la cuenta (en el momento de creación de este proyecto), el servicio será como gratuito. Con esto quiero remarcar que si la finalidad es hacer un uso comercial con elevadas consultas lo mejor es que se consulte su tabla de precios por número de peticiones disponible en su web.

Tras esto, lo primero que tendermos que hacer sera tener una cuenta de google y posteriormetne acceder a este enlace https://developers.google.com/maps/documentation/geocoding/get-api-key y seguir todos los pasos. Una vez tengamos la clave deberemos pegarla dentro del fichero “*key.txt*” situado en la carpeta “datos” de la aplicación.

Si arrancamos el programa sin introducir la clave nos mostrara una ventana de alerta indicando que no podra funcioanr correctametne.

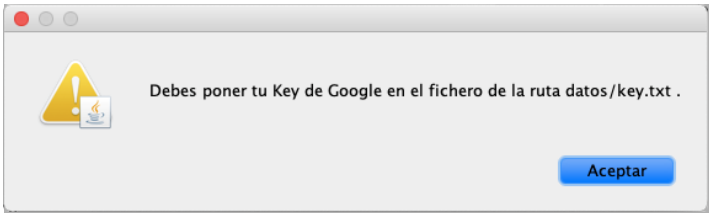


Ilustración 46 - Error clave (elaboración propia)

6. Pruebas

6.1 - Equipos

Hemos desarrollado las pruebas en tres sistemas operativos (Windows, Mac y Linux) y dos equipos distintos. Los equipos son:

- MacBook Air (principios 2014):

  Procesador Intel Core i5 (4260U) a 1,4Ghz
  8GB RAM DDR3 a 1600Mhz

- Asus ROG LG752:

  Procesador Intel Core i7 (6700HQ) a 2,6Ghz
  16GB RAM DDR4 a 2133Mhz

6.2 - Pruebas realizadas en equipos

Para la realización de las pruebas hemos creado una escuela ficticia con sede en Paiporta y hemos ido añadiendo paradas y alumnos hasta generar los siguientes casos de estudio:

Nombre de la parada	Número de personas	Cordenada X	Cordenada y
Paiporta,Valencia	0	39.4274468	-0.4184093
Picassent,Valencia	10	39.3637987	-0.4620313
Alcacer,Valencia	60	39.3768239	-0.44552
Benicalap,Valencia	10	39.4935471	-0.3913172
Rocafort,Valencia	30	39.5335655	-0.41134
Burjassot,Valencia	30	39.5096699	-0.4135963
Xirivella,Valencia	20	39.4597817	-0.4266584
Aldaya,Valencia	40	39.4595217	-0.4927134

Tabla 2 - Caso 1 (elaboración propia)

Nombre de la parada	Número de personas	Cordenada X	Cordenada y
Paiporta,Valencia	0	39.4274468	-0.4184093
Picassent,Valencia	10	39.3637987	-0.4620313
Alcacer,Valencia	60	39.3768239	-0.44552
Benicalap,Valencia	10	39.4935471	-0.3913172
Rocafort,Valencia	30	39.5335655	-0.41134
Burjassot,Valencia	30	39.5096699	-0.4135963
Xirivella,Valencia	20	39.4597817	-0.4266584
Aldaya,Valencia	40	39.4595217	-0.4927134
Benimaclet,Valencia	20	39.4871955	-0.3548312

Tabla 3 - Caso 2 (elaboración propia)

Nombre de la parada	Número de personas	Cordenada X	Cordenada y
Paiporta,Valencia	0	39.4274468	-0.4184093
Picassent,Valencia	10	39.3637987	-0.4620313
Alcacer,Valencia	60	39.3768239	-0.44552
Benicalap,Valencia	10	39.4935471	-0.3913172
Rocafort,Valencia	30	39.5335655	-0.41134
Burjassot,Valencia	30	39.5096699	-0.4135963
Xirivella,Valencia	20	39.4597817	-0.4266584
Aldaya,Valencia	40	39.4595217	-0.4927134
Benimaclet,Valencia	20	39.4871955	-0.3548312
Paterna,Valencia	30	39.5037093	-0.4431618

Tabla 4 - Caso 3 (elaboración propia)

Nombre de la parada	Número de personas	Cordenada X	Cordenada y
Paiporta,Valencia	0	39.4274468	-0.4184093
Picassent,Valencia	10	39.3637987	-0.4620313
Alcacer,Valencia	60	39.3768239	-0.44552
Benicalap,Valencia	10	39.4935471	-0.3913172
Rocafort,Valencia	30	39.5335655	-0.41134
Burjassot,Valencia	30	39.5096699	-0.4135963
Xirivella,Valencia	20	39.4597817	-0.4266584
Aldaya,Valencia	40	39.4595217	-0.4927134
Benimaclet,Valencia	20	39.4871955	-0.3548312
Paterna,Valencia	30	39.5037093	-0.4431618
Campanar,Valencia	20	39.4818921	-0.3948498

Tabla 5 - Caso 4 (elaboración propia)

Nombre de la parada	Número de personas	Cordenada X	Cordenada y
Paiporta,Valencia	0	39.4274468	-0.4184093
Picassent,Valencia	10	39.3637987	-0.4620313
Alcacer,Valencia	60	39.3768239	-0.44552
Benicalap,Valencia	10	39.4935471	-0.3913172
Rocafort,Valencia	30	39.5335655	-0.41134
Burjassot,Valencia	30	39.5096699	-0.4135963
Xirivella,Valencia	20	39.4597817	-0.4266584
Aldaya,Valencia	40	39.4595217	-0.4927134
Benimaclet,Valencia	20	39.4871955	-0.3548312
Paterna,Valencia	30	39.5037093	-0.4431618
Campanar,Valencia	20	39.4818921	-0.3948498
Alacuas,Valencia	10	39.4518434	-0.4714694

Tabla 6 - Caso 5 (elaboración propia)

Nombre de la parada	Número de personas	Cordenada X	Cordenada y
Paiporta,Valencia	0	39.4274468	-0.4184093
Picassent,Valencia	10	39.3637987	-0.4620313
Alcacer,Valencia	60	39.3768239	-0.44552
Benicalap,Valencia	10	39.4935471	-0.3913172
Rocafort,Valencia	30	39.5335655	-0.41134
Burjassot,Valencia	30	39.5096699	-0.4135963
Xirivella,Valencia	20	39.4597817	-0.4266584
Aldaya,Valencia	40	39.4595217	-0.4927134
Benimaclet,Valencia	20	39.4871955	-0.3548312
Paterna,Valencia	30	39.5037093	-0.4431618
Campanar,Valencia	20	39.4818921	-0.3948498
Alacuas,Valencia	10	39.4518434	-0.4714694
Mislata,Valencia	15	39.4779243	-0.4195876

Tabla 7 - Caso 6 (elaboración propia)

Nombre de la parada	Número de personas	Cordenada X	Cordenada y
Paiporta,Valencia	0	39.4274468	-0.4184093
Picassent,Valencia	10	39.3637987	-0.4620313
Alcacer,Valencia	60	39.3768239	-0.44552
Benicalap,Valencia	10	39.4935471	-0.3913172
Rocafort,Valencia	30	39.5335655	-0.41134
Burjassot,Valencia	30	39.5096699	-0.4135963
Xirivella,Valencia	20	39.4597817	-0.4266584
Aldaya,Valencia	40	39.4595217	-0.4927134
Benimaclet,Valencia	20	39.4871955	-0.3548312
Paterna,Valencia	30	39.5037093	-0.4431618
Campanar,Valencia	20	39.4818921	-0.3948498
Alacuas,Valencia	10	39.4518434	-0.4714694
Mislata,Valencia	15	39.4779243	-0.4195876
Alboraya,Valencia	10	39.4989552	-0.3510141

Tabla 8 - Caso 7 (elaboración propia)

Con estos casos de estudio, la prueba ha consistido en ejecutar en primer lugar el algoritmo sin paralelizar y posteriormente el paralelizado para ver la magnitud de la mejora de uno frente al otro. Hemos establecido las siguientes variables:

- Capacidad de cada autobús: 65 personas
- Distancia máxima que puede recorrer: 50 km
- Coste de cada autobús: 300€

El tiempo de ejecución de los diferentes casos en cada uno de los ordenadores ha sido el siguiente:

	MacBooK Air		
Caso	Paradas	Tiempo secuencial (seg)	Tiempo paralelo (seg)
1	7 (+1 colegio)	0,038	0,378
2	8 (+1 colegio)	0,383	0,591
3	9 (+1 colegio)	5,56	8,014
4	10 (+1 colegio)	88,142	93,125
5	11 (+1 colegio)	821,332	1760,885

Tabla 9 - Resultados ejecución i5 (elaboración propia)

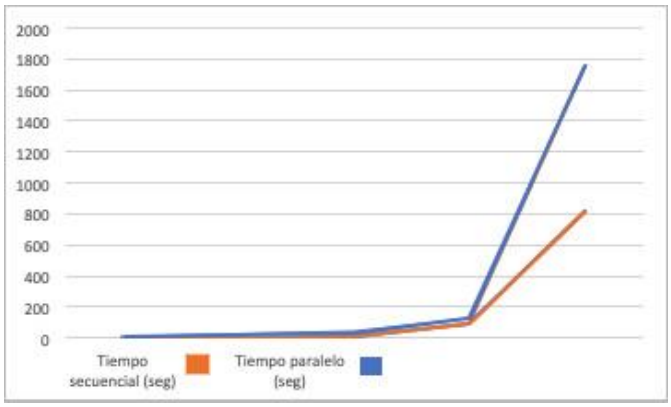


Tabla 10 - Gráfico ejecución i5 (elaboración propia)

	Intel Core i7 (6700HQ)		
Caso	Paradas	Tiempo secuencial (seg)	Tiempo paralelo (seg)
1	7 (+1 colegio)	0,038	0,021
2	8 (+1 colegio)	0,383	0,049
3	9 (+1 colegio)	5,56	0,66
4	10 (+1 colegio)	88,142	9,488
5	11 (+1 colegio)	486,902	163,631
6	12 (+1 colegio)	8630,452	2691,068

Tabla 11 - Resultados ejecución i7 (elaboración propia)

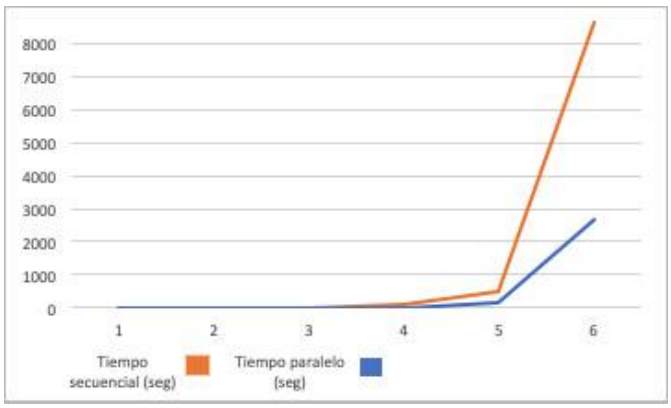


Tabla 12 - Gráfico ejecución i7 (elaboración propia)

Los datos anteriores nos sirven para cuantificar la mejoría que se produce cuando se optimiza código mediante paralelización. Es cierto que para pocas pardas la mejoría no existe o paralelizar código empeora los tiempos, pero si tenemos en cuenta que la diferencia de tiempo entre paralelizar o no paralelizar en casos en que hay pocas paradas es de minutos y en casos de excesivas paradas es de horas, la mejor opción siempre es paralelizar.

Hay que destacar que estamos ejecutando el código en dos equipos con configuraciones de hardware y sistemas operativos totalmente diferentes. El i5 tiene dos núcleos que son capaces de ejecutar dos hilos cada núcleo, en total cuatro hilos y el i7 tiene un total de 4 núcleos capaces de ejecutar dos hilos por núcleo en total 8 hilos. Hay que tener en cuenta que los procesadores están realizando otra serie de tareas mientras ejecutamos nuestro código (procesos del sistema operativo, aplicaciones que corren en segundo plano, etc.) es por ello que podemos apreciar una diferencia bastante elevada entre el rendimiento de un ordenador y otro, cuantos más núcleos tenga disponibles más rápido obtenemos resultados y menos notamos el hándicap de añadir código que nos permita paralelizar. Aun así, si tenemos en cuenta que un colegio siempre tendrá más de 12 paradas, podemos apreciar claramente como ejecutar el código de forma paralelizada reduce en 5939,384 segundos la obtención del resultado, o lo que es lo mismo en 99 minutos.

En este apartado también vamos a tratar la interpretación de los datos mostrados al obtener una solución. Para ellos vamos a utilizar como ejemplo el caso 2 con los siguientes valores de las variables comentados anteriormente. Podemos dividir el resultado en dos partes: la primera parte mostrara los datos de entrada y la segunda parte los resultados con la solución.

Parte 1:

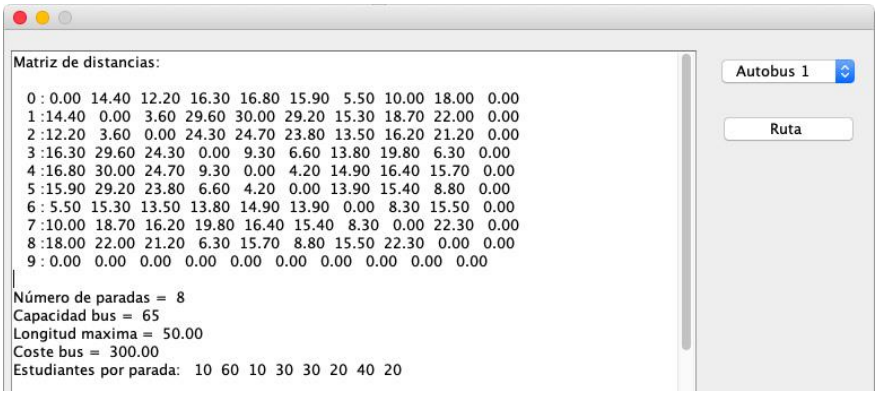


Ilustración 47 - Datos de entrada (elaboración propia)

En la ilustración 47 podemos apreciar la matriz de distancias del problema, hay que tener en cuenta que la última columna y la última fila son filas añadidas artificialmente como variables auxiliares que indican cuando una parada ha sido visitada o cuando está lleno el autobús. A continuación, podemos ver: el número de paradas introducidas, en este caso son ocho paradas (nueve si contamos el colegio), la capacidad de 65 personas, la distancia que puede recorrer un alumno es de 50 km, el coste de cada autobús que asciende a 300 € y un vector que indica la cantidad de alumnos por parada.

Parte 2:

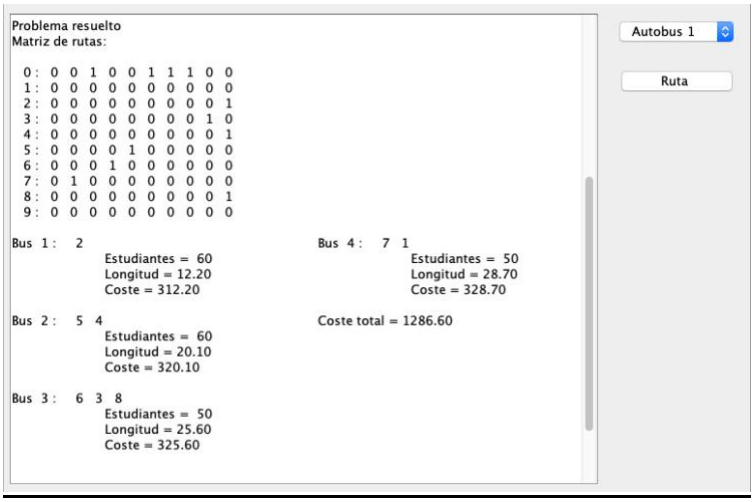


Ilustración 48 - Resultado del problema (elaboración propia)

En esta segunda parte podemos apreciar si se ha obtenido una solución o no, en este caso sí que se ha encontrado solución. Posteriormente podemos encontrar:

- La matriz de rutas que indica que se ha pasado por cada una de las paradas.
- El recorrido que debe tomar cada autobús junto:
  - Al número de estudiantes que transporta.
  - La distancia que recorre
  - El coste que supone cada autobús, hay que apreciar aquí que hemos tomado un valor de combustible de 1€ de ahí que muestre en cada uno de los autobuses la cifra de 300€ + (la longitud del recorrido*1€).
- El coste total del conjunto de autobuses, en este ejemplo 1286,60€ sería el coste de la ruta óptima.

6.3 - Pruebas realizadas en laboratorio Knights

Debido al potencial de cálculo necesario para elaborar pruebas superiores a 13 paradas en ordenadores domésticos (problemas de limitaciones temporales en tiempos de cálculo), vamos a realizar el estudio de estos casos en el laboratorio Knights que posee las siguientes características:

- Maquina con 24 Cores Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz

Varaibles de ejecución:

- Paradas 12, capacidad 65, distancia máxima 50Km, coste 300
- Paradas 13, capacidad 65, distancia máxima 50Km, coste 300

Knights	
Paradas	Tiempo paralelo (seg)
12	1691,068
13	4789,345

Tabla 13 - Resultados prueba Knights

En esta prueba sí que podemos apreciar claramente que es un problema NP-Completo, el tiempo de ejecución al añadir una parada aumenta de forma exponencial y el tiempo de computo entre la paralelización o no del algoritmo nos indica que es mejor optimizar.

7. Conclusiones

Desde un inicio suponíamos que estábamos frente a un tipo de problema complejo, que requeriría una potencia de cálculo elevada, algo común a todo tipo de problema NP-Completo, pero es ahora cuando realmente podemos apreciar que a medida que añadimos paradas el tiempo de procesamiento para encontrar una solución exacta crece de forma exponencial.

Cuando el problema de enrutamiento del autobús escolar se plantea con pocas paradas, la diferencia de tiempo en obtener la solución es prácticamente inexistente entre ejecutarlo de forma secuencial o en paralelo. Incluso en algunas ocasiones es mejor ejecutarlo de forma secuencial debido a que OpenMP añade instrucciones que ralentizan el proceso de ejecución unos milisegundos (tiempo prácticamente inapreciable).

También se aprecia que en estas situaciones hacer uso de un procesador con menor potencia tampoco es un factor determinante en el tiempo que tardamos en obtener la solución. Donde realmente notamos una importante mejora es cuando incrementamos las paradas en más de 11 unidades. Es aquí cuando empezamos a ver diferencias de potencia reales entre ambos equipos y donde realmente empezamos a apreciar el beneficio real que obtenemos al paralelizar el algoritmo.

Con las pruebas realizadas, la optimización de código para hacer uso de todo el potencial del equipo en que corremos una aplicación es imprescindible en problemas que exijan un poder de cálculo elevado.

7.1 - Aprendizaje y relación del proyecto con los estudios

Analizando todas las fases por las que hemos ido pasando hasta llegar a la finalización del proyecto podemos observar todo lo que hemos aprendido y todo aquello en lo que hemos ampliado nuestro conocimiento previo: programación en JAVA y C, conexión con APIs, uso de OpenMP, gestión de un proyecto, etc.

Este proyecto nos ha servido para mejorar y desarrollar nuestras capacidades de planificación, análisis de problemas y desarrollo de soluciones. A pesar de no haber estudiado directamente muchas de las tecnologías empleadas en el desarrollo de este problema, sí que nos hemos dado cuenta que durante estos cuatro cursos del grado en ingeniería informática se nos ha ido formando con las bases y herramientas que un buen informático debe tener para desarrollar su carrera profesional.

Este proyecto me ha servido para ver que la informática es un sector cambiante, en el que día tras día aparecen nuevas tecnologías, lenguajes, etc. Por tanto, es un sector que requiere estudio y motivación constante, tras este proyecto sé que seré capaz de afrontar el aprendizaje continuo que se requiere para desarrollar correctamente la profesión.

8. Futuras mejoras

Debido a la estructura que hemos tomado en el desarrollo de la aplicación, separando por un lado lo que sería el algoritmo y por otro la interfaz gráfica, una mejora futura podría ser la implementación de la aplicación utilizando una arquitectura cliente servidor. La parte del cliente seria donde se ejecutaría la interfaz y en la parte de servidor ejecutaríamos la parte del algoritmo. Esto otorgaría gran potencial de cálculo debido a que los procesadores de los servidores por norma general tienen más núcleos y pueden ejecutar más hilos.

Debido a que la estructura de clases generada tiene atributos creados pensando en una futura actualización en la que pueda llevarse un control más exhaustivo sobre que alumnos. Podría ampliarse la aplicación con esta opción y así poder realizar un seguimiento de los alumnos que suben o bajan en cada parada, etc. Incluir esta mejora llevaría unido la creación de una base de datos SQL y un menú de herramientas con la opción de añadir alumnos y sus respectivos nombres, etc.

También podríamos considerar las opciones de seguimiento de dispositivos que ofrece la API de Google para crear un menú en el que pudiéramos seguir la ruta de cada autobús en tiempo real.

9. Referencias

1. Abdullah Muhammad ibn Musa Al- Khwarizmi “Compendio de cálculo por reintegración y comparación :https://www.biografiasyvidas.com/biografia/k/khwarizmi.htm#targetText=Al%2DKhwarizmi,los%20principios%20fundamentales%20del%20%C3%A1lgebra.
2. Dantzig, G. and Ramser, J. (1959) The Truck Dispatching Problem. Management Science, 6, 80-91.
3. Kinable, J., Spieksma, F., & Vanden, G. (2014). School bus routing: A column generation approach. INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH, 453–478.
4. Park, J. and Kim, B.-I. (2010). "The school bus routing problem: A review". European Journal of Operational Research, vol. 202, No. 2 (April), pp. 311-319.
5. Fügenschuh, A., 2009. Solving a school bus scheduling problem with integer programming. European Journal of Operational Research 193 (3), 867–884
6. Rocha, L., González, E., & Orjuela, J. (2011). Una revisión al estado del arte del problema de ruteo de Vehículos: evolución histórica y métodos de solución. Ingeniería, 16(2), 35–55.
7. Yigit & Unsal 2016 Optimization of Dynamic School Bus RoutingProblem by Using Metaheuristic and ClusteringMethods
8. Kelly, J. A., & Fu, M. (2014). Sustainable school commuting - understanding choices and identifying opportunities. A case study in Dublin, Ireland. Journal of Transport Geography, 34, 221–230.
9. Angel, R.D., Caudle, W.L., Noonan, R., Whinston, A., 1972. Computer-assisted school bus scheduling. Management Science 18 (6), 279–288.
10. Thangiah & Nygard (1992) School bus routing using genetic algorithms
11. Bowerman, R., Hall, B., Calamai, P., 1995. A multi-objective optimization approach to urban school bus routing: formulation and solution method. Transportation research Part A 29 (2), 107–123.
12. Desrochers, M., Sauvé, M., 1986b. Methods for routing with time windows. European Journal of Operational Research 23 (2), 236–245.
13. Hargroves, B.T., Demetsky, M.J., 1981. A computer assisted school bus routing strategy: a case study. Socio-Economic Planning Sciences 15 (6), 341–345.
14. La voz de Galicia,(2019): https://www.lavozdegalicia.es/noticia/galicia/2019/04/20/bus-escolar-compartido-transforma-transporte-interior-galicia/00031555759880233479684.htm
15. Chen, D., Kallsen, H.A., Chen, H., Tseng, V., 1990. A bus routing system for rural school districts. Computers and Industrial Engineering 19, 322–325.

    Chen, D., Kallsn, H.A., Snider, R.C., 1988. School bus routing and scheduling: an expert system approach. Computers and Industrial Engineering 15, 179–183.
16. Toth, P. and Vigo, D. (2002) The Vehicle Routing Problem. Siam, Philadelphia.
17. Lysgaard & Sørensen, 1997 https://www.studocu.com/en/document/technische-universiteit-eindhoven/deterministic-operations-management/other/clarke-wrights-savings-algorithm/67425/view
18. Contreras P 2010. Algorithms for hierarchical clustering: an overview
19. Prins, Lacomme, & Prodhon, 2014. Technical Note: Split Algorithm in O(n) for the Capacitated Vehicle Routing Problem
20. Beasley, 1983. Order-first split-second methods for vehicle routing problems: A review
21. Lüer, A., Benavente, M., Bustos, J., & Venegas, B. (2009). El problema de rutas de vehículos: Extensiones y métodos de resolución estado del arte.
22. Dib , Manier , Moalic , & Caminada , 2015. Combining VNS with Genetic Algorithm to Solve the One-to-One Routing Issue in Road Network.
23. C.A Peñuela, J.F Franco, E.M Toro, 2010. Colonia de hormigas aplicada a la programación óptima de horarios de clase
24. Colin R. Reeves, Jhonathan E. Rowe, 2003. Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory
25. Golberg, 1989. Genetic Algorithms in Search, Optimization and Machine Learning.
26. Leblanc et al.: Genetic Algorithms and aperiodic order. 621

Agradecimientos

Me gustaría agradecer a Pedro Alonso Jordá, tutor de mi TFG, la ayuda prestada durante todo este tiempo.

Anexo: código fuente del algoritmo paralelizado

/Users/erdose/Documentos/TFG/Proyecto/src/algoritmos/BRP_openmp.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>

#define D(i,j) D[(i)*n+(j)]
#define X(i,j) X[(i)*n+(j)]
#define X_mejor(i,j) X_mejor[(i)*n+(j)]
#define X_aux(i,j) X_aux[(i)*n+(j)]

void leer_valores( );
void leer_matriz( );
void leer_estudiantes( );

void generar_distancias( int n, float *D );
void generar_paradas( int n_paradas, int *q );

void imprimirdistancias( int n, float *D );
void imprimirrutas( int n, float *X );
void imprimirparadas( int n_paradas, int *q );
void imprimirsolucion( );

int BRP( int i, int visitadas, float *X, int c, float l, float coste );

int n_paradas;
float coste_bus;
float *D;
int *q;
int C;
float L;
float matriz [1000][1000];

float coste_minimo = RAND_MAX;
float *X_mejor;

int main( int argc, char *argv[] ) {

    leer_valores();

    if( L<20 ) {
        fprintf(stderr,"%s: La longitud maxima debe ser mayor que 20\n",argv[0]);
        return 1;
    }

    leer_matriz();

    //Creación de vectores
    int n = n_paradas+2;
    D = (float*) malloc(n*n*sizeof(float));
    q = (int*) malloc(n_paradas*sizeof(int));
    float matriz[n][n];

    generar_distancias(n,D);
    imprimirdistancias(n,D);

    //Salida por pantalla
    FILE *fp;
    fp = fopen ( "resultados/resultado.txt", "a+" );
```

/Users/erdose/Documentos/TFG/Proyecto/src/algoritmos/BRP_openmp.c

```
    fprintf(fp, "%s %d","\nNúmero de paradas = ",n_paradas);
    fprintf(fp, "%s %d","\nCapacidad bus = ",C);
    fprintf(fp, "%s %.2f","\nLongitud maxima = ",L);
    fprintf(fp, "%s %.2f","\nCoste bus = ",coste_bus);
    fprintf(fp, "%s", "\n");

    fclose(fp);

    generar_paradas(n_paradas,q);
    imprimirparadas(n_paradas,q);

    float *X = (float*) malloc(n*n*sizeof(float));
    X_mejor = (float*) malloc(n*n*sizeof(float));

    for( int i=0; i<n; i++ )
        for( int j=0; j<n; j++ ) {
            X( i, j ) = 0;
            X_mejor( i, j ) = 0;
    }

        #pragma omp parallel
    #pragma omp single
    BRP( 0, 0, X, 0, 0, 0 );

    FILE *fp2;
    fp2 = fopen ( "resultados/resultado.txt", "a+" );
    if( coste_minimo<RAND_MAX ) { fprintf(fp2, "%s","Problema resuelto\n");}
    else { fprintf(fp2, "%s","Problema sin solucion\n");}
    fclose(fp2);

    imprimirrutas( n, X_mejor );
    imprimirsolucion( );

        FILE *fp3;
        fp3 = fopen ( "resultados/resultado.txt", "a+" );
    fprintf(fp3, "%s%.2f%s","\nCoste total = ",coste_minimo,"\n");
        fclose(fp3);

        free(X);
    free(q);
    free(D);
    return 0;
    }

    int parada_valida( int j, float *X, float c, float l ) {
        int n = n_paradas+2;
    int valido = 1;
        int i=0;

    while( i<n && valido ) {
            if( i!=j && X( i, j )==1 ) {
                valido = 0;
            }
            i++;
    }
            return valido && c<C && l<L;
    }
```

/Users/erdose/Documentos/TFG/Proyecto/src/algoritmos/BRP_openmp.c

```
/**
* i             Parada que se está visitando
* visitadas     Número de paradas visitadas hasta el momento
* X             Matriz de recorridos
* c             Número de alumnos en la ruta actual (con el autobús que se está trabajando
* l             Longitud de la ruta actual (con el autobús que se está trabajando)
* coste         Coste total acumulado hasta el momento
*/

int BRP( int i, int visitadas, float *X, int c, float l, float coste ) {
    int n = n_paradas+2;
    if( visitadas == n_paradas ) {
        if( coste < coste_minimo ) {
        #pragma omp critical
            if( coste < coste_minimo ) {
                coste_minimo = coste;
                memcpy( X_mejor, X, n*n*sizeof(float) );
            }
        }
    }
    else {
        for( int j=1; j<n; j++ ) {
            if( i==0 ) {
                /* Autobus nuevo */
                if( j<n-1 && parada_valida( j, X, q[j-1], D(i,j) ) ) {
                    float *X_aux = ( float* ) malloc( n*n*sizeof(float) );
                    memcpy( X_aux, X, n*n*sizeof(float) );
                    X_aux( i, j ) = 1;
                    #pragma omp task
                    {
                    BRP( j, visitadas+1, X_aux, q[j-1], D(i,j), coste+coste_bus+D(i,j) );
                    free(X_aux);
                    }
                }
            }
            else {
                if( j==n-1 ) {
                    float *X_aux = ( float* ) malloc( n*n*sizeof(float) );
                memcpy( X_aux, X, n*n*sizeof(float) );
                X_aux( i, j ) = 1;
                #pragma omp task
                {
                BRP( 0, visitadas, X_aux, c, l, coste );
                    free(X_aux);
                }
                }
                else if( i!=j && parada_valida( j, X, c+q[j-1], l+D(i,j) ) ) {
                    if( coste+D(i,j) < coste_minimo ) { /* Poda */
                        float *X_aux = ( float* ) malloc( n*n*sizeof(float) );
                        memcpy( X_aux, X, n*n*sizeof(float) );
                        X_aux( i, j ) = 1;
                        #pragma omp task
                        {
                        BRP( j, visitadas+1, X_aux, c+q[j-1], l+D(i,j), coste+D(i,j) );
                        free(X_aux);
                        }
                    }
```

/Users/erdose/Documentos/TFG/Proyecto/src/algoritmos/BRP_openmp.c

```
                    }
                }
            }
        }
    }

    void generar_distancias( int n, float *D ) {
        for( int i=0; i<n; i++ ) {
            for( int j=i; j<n; j++ ) {
                D( i, j ) = ( j==n-1 ) ? 0 : matriz[i][j];
            D( j, i ) = D( i, j );
            }
    }
    }

    void imprimirdistancias( int n, float *D ) {
    FILE *fp;
    fp = fopen ( "resultados/resultado.txt", "a+" );
    fprintf(fp,"%s", "Matriz de distancias: \n");
    fprintf(fp,"%s","      ");

    //for( int j=0; j<n; j++ ) { printf("%4d",j);}

    fprintf(fp,"%s","\n");
    for( int i=0; i<n; i++ ) {
            fprintf(fp,"%4d %s",i,":");
            for( int j=0; j<n; j++ ) {
            if (D(i,j)<10){ fprintf(fp," %.2f  ",D( i, j ));}
                else {fprintf(fp,"%.2f  ",D( i, j )); }
            }
            fprintf(fp,"%s","\n");
    }
    fclose(fp);
    }

    void generar_paradas( int n_paradas, int *q ) {
        FILE *f = fopen("datos/estudiantes.txt", "r");
        int aux;

    for( int i=0; i<n_paradas; i++ ) {
            fscanf(f,"%d",&aux);
            while( ( q[ i ] = aux ) < 3 );
    }
    fclose(f);
    }

    void imprimirparadas( int n_paradas, int *q ) {
    FILE *fp;
        fp = fopen ( "resultados/resultado.txt", "a+" );

    fprintf(fp,"%s","Estudiantes por parada: ");
    for( int i=0; i<n_paradas; i++ ) {
            fprintf(fp,"%4d",q[ i ]);
    }
    fprintf(fp, "%s","\n\n");
    fclose(fp);
    }
```

/Users/erdose/Documentos/TFG/Proyecto/src/algoritmos/BRP_openmp.c

```
    void imprimirrutas( int n, float *X ) {
    FILE *fp;
    fp = fopen ( "resultados/resultado.txt", "a+" );

    fprintf(fp,"%s","Matriz de rutas: \n");
    fprintf(fp,"%s","    ");

    for( int j=0; j<n; j++ ) { fprintf(fp," %4d",j);}

    fprintf(fp,"%s","\n");

    for( int i=0; i<n; i++ ) {
            fprintf(fp,"%4d %s",i,":");
            for( int j=0; j<n; j++ ) { fprintf(fp,"%4.0f",X( i, j ));}
            fprintf(fp,"%s","\n");
        }
    fclose(fp);
    }

    //Funciones para leer valores
    void leer_valores(){
    char parametro1 [10];
    char parametro2 [10];
    char parametro3 [10];
    char parametro4 [10];

    FILE *f = fopen("datos/datos.txt", "r");

    fgets (parametro1, 10, f);
    fgets (parametro2, 10, f);
    fgets (parametro3, 10, f);
    fgets (parametro4, 10, f);

    n_paradas = atoi(parametro1);
    C = atoi(parametro2);
    L = atoi(parametro3);
    coste_bus = atoi(parametro4);

    fclose(f);
    }

    void leer_matriz(int n){
    FILE *f = fopen("datos/matriz.txt", "r");

        int i, j;
        float temp;

        for(i=0;i<n_paradas+1;i++) {
            //printf(" \n");
            for(j=0;j<n_paradas+1;j++) {
                fscanf(f,"%f",&temp);
                matriz[i][j] = temp;//printf("%f ", matriz[i][j]);
            }
        }
        fclose(f);
    }

    void imprimirsolucion( ) {
```

/Users/erdose/Documentos/TFG/Proyecto/src/algoritmos/BRP_openmp.c

```
        int n = n_paradas+2;
    int bus = 1;
    int i = 0;

    FILE *fp;
    fp = fopen ( "resultados/resultado.txt", "a+" );

    for( int j=1; j<n; j++ ) {
            int estudiantes = 0;
            float longitud = 0;
            float coste = 0;

            if( X_mejor( i, j ) == 1 ) {
            fprintf(fp, "%s %d %s %4d", "\nBus ",bus++,": ",j);
            estudiantes = q[j-1];
            coste = coste_bus + D( i, j );
            longitud = D( i, j );
            int jj = j;
            int seguir = 1;

        while( seguir ) {
                    int k = 0;
                    while( X_mejor( jj, k ) == 0 ) k++;
                    if( k<n-1 ) {
                fprintf(fp,"%4d",k);
                estudiantes += q[k-1];
                longitud += D( jj, k );
                coste += D( jj, k );
                jj = k;
                    }
                    else {
                fprintf(fp,"%s %d%s%.2f%s%.2f%s","\n\tEstudiantes = ",estudiantes,
                                "\n\tLongitud = ",longitud,
                                                "\n\tCoste = ",coste,"\n");
                seguir = 0;
                    }
            }
            }
    }
    fclose(fp);
    }
```