UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Planificación y técnicas de mejora de planes para smart cities

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Azocar Marques, Luis Ignacio

Tutor/a: Garrido Tejero, Antonio

CURSO ACADÉMICO: 2022/2023

Resumen

Las smart cities o ciudades inteligentes son aquellas capaces de utilizar la tecnología de la información y la comunicación (TIC), con el fin de promover un eficiente desarrollo sostenible y mejorar la calidad de vida de sus ciudadanos. Hoy en día el crecimiento poblacional de las distintas ciudades del mundo es mucho mayor y cada día la sociedad demanda un mejor aprovechamiento de los recursos, desde personal empleado del sector público para la correcta y eficiente gestión administrativa, así como de personal empleado en distintos sectores de servicios, como salud, bomberos, policías, hasta recursos físicos como ambulancias, patrullas, camiones, entre otros.

Con el presente trabajo se diseña y se implementa un modelo de optimización con el fin de realizar una planificación y/o asignación eficiente de recursos a las distintas demandas que puedan existir en una o muchas ubicaciones, de forma que se cubra la mayor demanda posible minimizando en su medida el gasto monetario que puede generar dicha asignación y empleabilidad de estos recursos.

Palabras clave: optimización, planificación, asignación, gasto, recursos.

Abstract

Smart cities are those capable of using information and communication technology (ICT) to promote efficient sustainable development and improve the quality of life of their citizens. Today the population growth in different cities around the world is much greater and every day society demands a better use of resources, from personnel employed in the public sector for the correct and efficient administrative management, as well as personnel employed in different sectors such as healthcare, firefighters, police, even physical resources such as ambulances, patrol cars, trucks, among others.

With the present work, an optimization model is designed and implemented in order to carry out and efficient planning and/or allocation of resources to the different demands that may exist in one or many locations, so that the greatest possible demand is covered minimizing to the extent the monetary expense of the allocation and employability of these resources can generate.

Keywords: optimization, plan, allocation, expenditure, resources.

Agradecimientos

Primero quiero agradecerle a mi tutor de trabajo Antonio Garrido por permitirme la realización de este trabajo de fin de grado, así como toda su colaboración, aportación de ideas y correcciones a lo largo de la realización de este.

Agradecer a todos mis amigos desde los más antiguos hasta los conocidos, gracias a esta carrera y a una nueva vida, por todo el apoyo y ánimo recibido para alcanzar el fin de un objetivo por el que llegué aquí en primer lugar.

Finalmente, agradecerle a mi mamá Claudia Cristina Marqués De Antunes por estar siempre para mí, por darme todo el apoyo y cariño a lo largo de mi vida y por demostrarme cada día que los límites los pone uno mismo para lograr las metas que se propone. Ninguno de mis logros serían posibles sin ella.

Tabla de contenidos

Tabla de figuras

1. Introducción

Para darle inicio a este trabajo, se comenzará definiendo y explicando que es una smart city, sus características, ventajas y desventajas; así como técnicas de planificación y la propuesta de desarrollo que cubrirá este TFG. Seguidamente se detallarán los objetivos planteados y la estructura de la memoria.

1.1 Smart city

Una smart city, o ciudad inteligente, es un concepto que surge a finales del siglo XX por Enrique Ruz, quién presentó la primera Ciudad Digital de la mano de empresas como ZTE, Telefónica, Gas Natural, INDRA, RACE y otras; pero que años después IBM modifica su nombre por Smart City. El concepto de smart city hoy en día se refiere a una ciudad que utiliza tecnología y datos para mejorar la calidad de vida de sus habitantes, así como la eficiencia de los servicios urbanos y la sostenibilidad medioambiental. [1]

Una smart city utiliza la infraestructura digital y las tecnologías de información y comunicación (TIC) para recopilar y analizar datos en tiempo real, con el fin de tomar decisiones informadas y proporcionar servicios de manera más eficiente. Tienen diferentes elementos interconectados que la componen como lo pueden ser las TIC, datos abiertos, sistemas de gestión inteligente, servicios públicos eficientes, etc; sino que además está conformada por todo lo que se encuentra dentro de ella, haciendo referencia a comercios, empresas, escuelas, hospitales, habitantes y muchos otros. La representación visual de una smart city se puede apreciar en la figura 1.

Dentro de sus ventajas se encuentran: efectividad en la toma de decisiones basada en datos, creación de comunidades más seguras, mejora del transporte urbano y del medio ambiente, optimización en tiempos de espera de hospitales y servicios públicos, optimizan la asignación de recursos y ayudar a reducir gastos innecesarios y elevan el grado de satisfacción de los habitantes. Como toda idea ambiciosa que percibe mejoras extraordinarias, también presenta desventajas para su implementación como lo son la alta inversión de capital en tecnología, dependencia de las compañías de servicios tecnológicos, encarecimiento en la construcción de inmuebles y mayores brechas tecnológicas entre las smart cities y otras ciudades. [2]

Figura 1. Representación visual de una smart city
Recuperado de: https://canteradeempresas.com/que-son-las-smart-cities-y-que-importancia-tienen-en-nuestro-proyecto/

La inteligencia, el avance tecnológico, la innovación y la conectividad motorizan al desarrollo urbano, apuntando a que las ciudades inteligentes pueden ser solución al indetenible crecimiento demográfico que están experimentando las urbes, y ofrecer mayor calidad de vida. Las ciudades inteligentes deben dirigirse hacia un modelo horizontal, eficaz eficiente, económico y escalable, que unifique infraestructuras y reaproveche capacidades, apalancado a una cultura responsable con los recursos naturales y el medio ambiente, innovando en el modelo de gobernanza y de gestión de servicios públicos e involucrando y educando a los ciudadanos.

Una smart city es la orquestación de gobierno inteligente, economía inteligente, ciudadanos inteligentes, vida inteligente, ambiente inteligente y movilidad inteligente. Al plantear un proyecto de smart city es primordial adecuarlo a la realidad particular de cada ciudad, entendiendo los aspectos que la pueden hacer diferenciadora y así potenciarlos. Las iniciativas se deben adaptar a su tamaño físico, a su tamaño poblacional, a sus posibilidades y a sus principales intereses. La tecnología habilita el desarrollo de múltiples iniciativas, pero el plan estratégico es el que permite a la Administración definir los objetivos a corto, mediano y largo plazo, e impulsar las iniciativas de forma cohesionada, fundamentado en 4 fases:

1. Estudio de necesidades.
2. Planteamiento estratégico.
3. Programa de desarrollo y actuación.
4. Seguimiento y medición de logros.

Desde instalación de granjas solares para generar energía limpia, construcción y/o reforma de centros de servicios públicos, desarrollo de vías de circulación ecológicamente responsables, instalación de antenas de comunicaciones para interconectar todos los dispositivos inteligentes, hasta el ambicioso proyecto de tratamiento y purificación de agua, no hay duda que el factor base común para ejecutar cualquier proyecto es la asignación eficiente de recursos

físicos y humanos adecuada a las necesidades, con el fin de aprovechar ventajas de las posibles interrelaciones en costo o calidad de servicio permitiendo así alcanzar los objetivos planteados.

Enfocado en uno de los seis componentes de una smart city representado en la figura 2, el presente proyecto tiene como objetivo el desarrollo de un modelo de optimización de asignación de recursos para satisfacer diversas demandas, informando al usuario que lo utiliza la propuesta idónea de asignación de recursos que tiene a su disposición, así como el número de recursos adicionales que requiere para satisfacer el 100% de sus demandas. El modelo incrementa la efectividad en la asignación de recursos que se traduce en reducción de gastos, con la idea de que ese ahorro monetario pueda ser dirigido a nuevos proyectos que contribuyan con el bienestar social y calidad de vida de los ciudadanos.

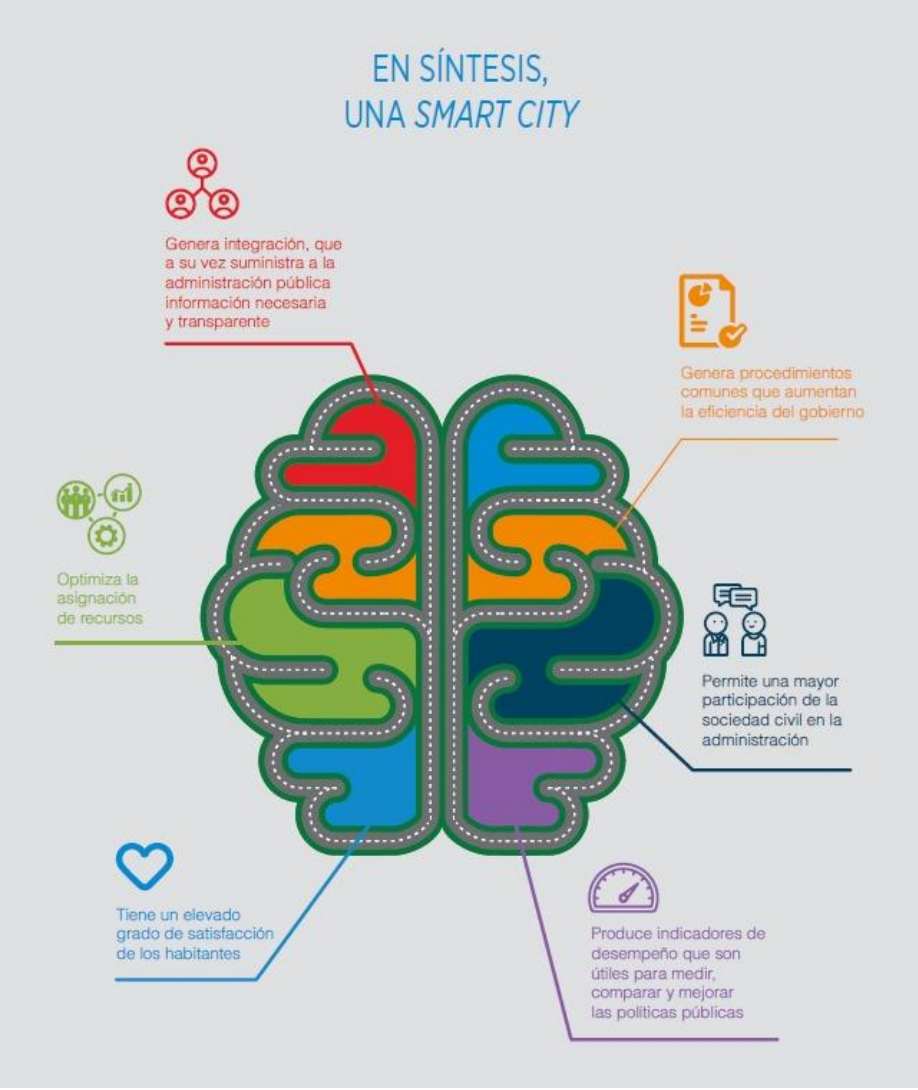

Figura 2. Componentes de una smart city
Recuperado de: https://blogs.iadb.org/ciudades-sostenibles/es/ciudades-inteligentes-2/

1.2 Motivación

La prioridad de todo gobierno o alcaldía es poder proveer el mejor bienestar social para sus ciudadanos, dicho bienestar viene dado por distintos factores como puede ser el transporte público, limpieza de calles, mantenimiento de vías públicas, seguridad, salud, educación, entre otros. Estos factores poseen una demanda según la población que se trate, bien sea si se trata de una ciudad en concreto y sus distintos barrios o incluso de un país y sus distintas provincias; esta demanda ha de ser cubierta por los distintos recursos que se poseen y además se han de asignar y utilizar de la forma más eficiente posible. Si se realiza una correcta asignación de recursos minimizando el gasto que puede ocupar, cualquier ente encargado de cubrir dicho gasto pudiera, en el caso de tener un sobrante del dinero presupuestado, utilizar los fondos sobrantes en el desarrollo de nuevos bienes para sus ciudadanos, desde parques infantiles, espacios públicos para el esparcimiento, centros educativos, hospitales, estaciones de bomberos y policías, polideportivos, centros de acogida para personas sin recursos, programas públicos de ayuda a personas con discapacidades, programas de reinserción social e inclusive llegar a la posibilidad de disminuir los impuestos alcanzando tasas más reducidas que conllevarían a una mejora en el ingreso neto de los ciudadanos.

Este trabajo propone un modelo de optimización que garantiza el cubrimiento de la mayor cantidad de demanda posible con los recursos ofertados, teniendo en cuenta la prioridad que pueda tener dicha demanda en una ubicación, proporcionando el gasto total que conllevaría, así como la propuesta de solución de asignaciones de dicho caso. Es importante mencionar que este proyecto aplica para cualquier área, función, recurso y demanda.

1.3 Objetivos

El objetivo principal es desarrollar un algoritmo de optimización que asigne los recursos necesarios según una lista de oferta para cubrir una demanda existente. Este ha de ser los más general posible, de forma que pueda ser utilizado por cualquier institución pública o privada que le convenga y que además pueda en un futuro ser escalable para cubrir otros ámbitos y adaptable a casos más específicos según el ramo de la institución o las necesidades particulares. Para ello es necesario conocer los aspectos, parámetros y condiciones a restringir al momento de asignar recursos, por ejemplo: máximo de ubicaciones a las cuales se puede asistir por día, máximo de horas a asignar, jerarquía de tipos, entre otros.

Como objetivo secundario, se requiere que este modelo obtenga la mayor cobertura posible de demanda asignando los recursos preestablecidos por él usuario, minimizando el coste que conlleva la asignación de éstos, logrando la mejor relación precio-valor.

También se desea realizar un estudio del funcionamiento del modelo desarrollado midiendo el número de restricciones generadas para cada caso, cantidad de demanda no satisfecha completa o parcialmente y tiempo de ejecución en una escala de diversos niveles de complejidad, así como determinar la escalabilidad según las herramientas utilizadas.

1.3 Estructura de la memoria

Para una mejor compresión de la memoria se explica a continuación su estructura:

- **Capítulo 1:** En este capítulo se habla de smart cities, su estructura, componentes, ventajas y desventajas y la correlación con el propósito de este trabajo, que conlleva a la motivación para su realización y los objetivos planteados.
- **Capítulo 2:** En el desarrollo de este capítulo se comentarán las diferentes herramientas existentes que pueden ser empleadas para la realización de la solución con su definición, ventajas y desventajas.
- **Capítulo 3:** Se dará a conocer la problemática que se desea resolver y la solución propuesta para ello.
- **Capítulo 4:** En este capítulo se detallará la implementación de la solución con las distintas librerías empleadas y funciones desarrolladas.
- **Capítulo 5:** Se mostrará la comprobación de funcionamiento de la solución y se realizarán diversas pruebas para el estudio de los resultados obtenidos.
- **Capítulo 6:** El capítulo final presentará las conclusiones obtenidas este trabajo, la relación que posee con el grado cursado y las mejoras a futuro que se pueden realizar.

2. Estudio estratégico

Hoy en día con el desarrollo constante de la tecnología se han creado diversas formas para resolver problemas de todo tipo, desde la simple resolución de una operación matemática, creación de algoritmos más complejos para elegir las mejores decisiones de un negocio e incluso, creación de algoritmos que sugieren a un usuario una selección de películas, series, artículos, entre otros, según sus preferencias o de acuerdo con su historial de búsqueda.

Para resolver distintos tipos de problemáticas como lo pueden ser la asignación de diferentes recursos en distintas demandas existen diversas herramientas para afrontarlo. Éstas son herramientas de búsqueda, heurísticas, algoritmos de programación entera y mixta, entre muchos otros.

2.1 Herramientas de búsqueda

Esta herramienta se basa en la elección de estados. Un estado se puede considerar como la representación de elementos que describen el problema en un momento dado, es decir, la situación en que se encuentra o se podría encontrar el problema en cada instante de tiempo. [3]

La elección de estados no solo determina qué información se almacenará de las diversas situaciones por las que pasa el problema, sino que en muchos casos es determinante también para decidir cuáles son las reglas u operaciones básicas que se permiten para realizar transformaciones entre estados. [3]

Un ejemplo de esto puede ser si se plantea un hospital que requiere cubrir una demanda puesto de seis médicos, los cuales son dos cirujanos, un traumatólogo y un pediatra, también requiere de cinco enfermeros, como se observa en la figura 3 (los valores son acumulativos). Para cubrir esta demanda se cuentan con diferentes recursos de médicos y enfermeros que pueden ser o no de las especialidades solicitadas, en total 10 médicos los cuales hay tres cirujanos, dos traumatólogos, dos pediatras y tres ginecólogos y 8 enfermeros generales. Los médicos pueden ocupar una demanda que no es de su especialidad siempre y cuando sigan siendo de la misma rama, o sea que un cirujano puede ser asignado para la demanda de cirujanos o de médicos generales.

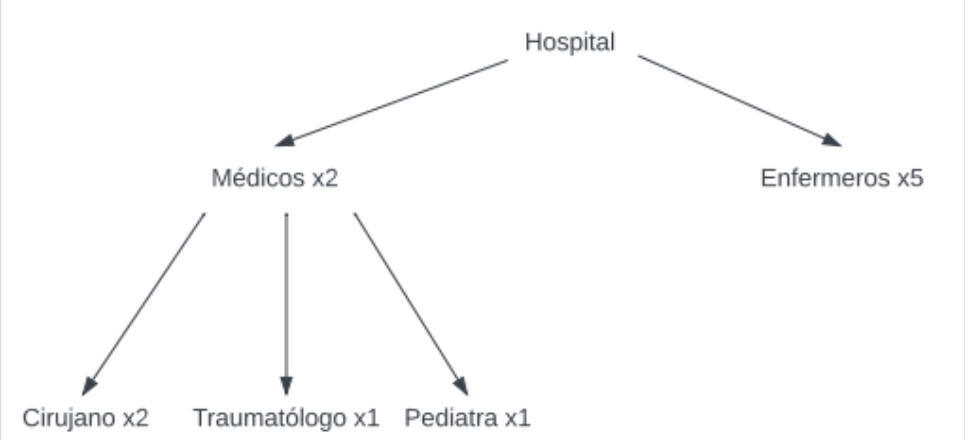

Figura 3. Diagrama representativo de la demanda

Con la oferta suministrada se pueden plantear diversos escenarios de resolución, estos son construidos paso a paso por distintas asignaciones como las presentadas en la figura 4 y 5 hasta llegar a una asignación más completa mostrada en la figura 6 y finalmente llegando a la solución en la figura 7, la cual satisface todas las demandas mencionadas anteriormente.

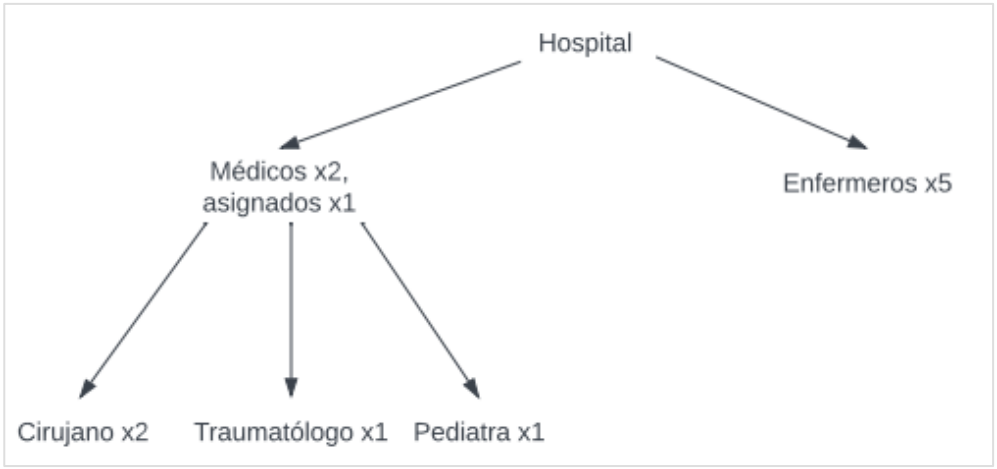

Figura 4. Asignación de médico cirujano

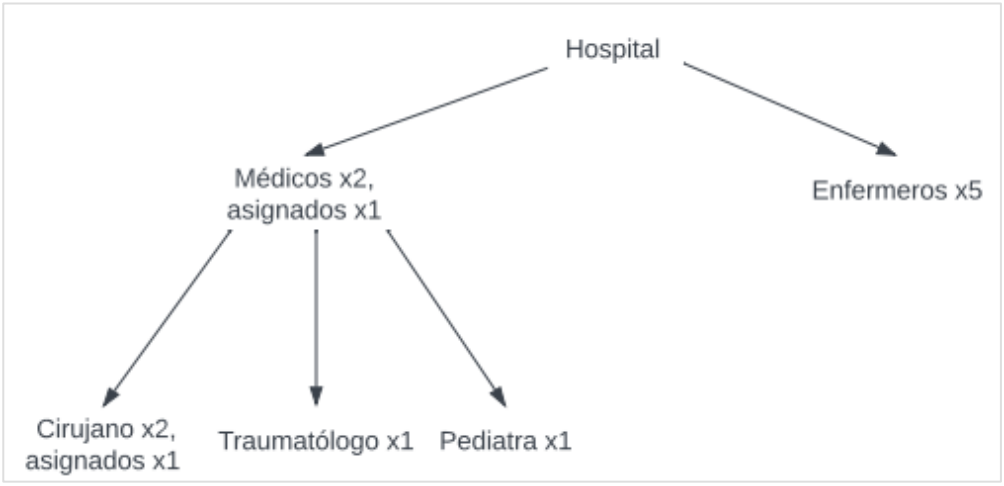

Figura 5. Asignación de médico traumatólogo y cirujano

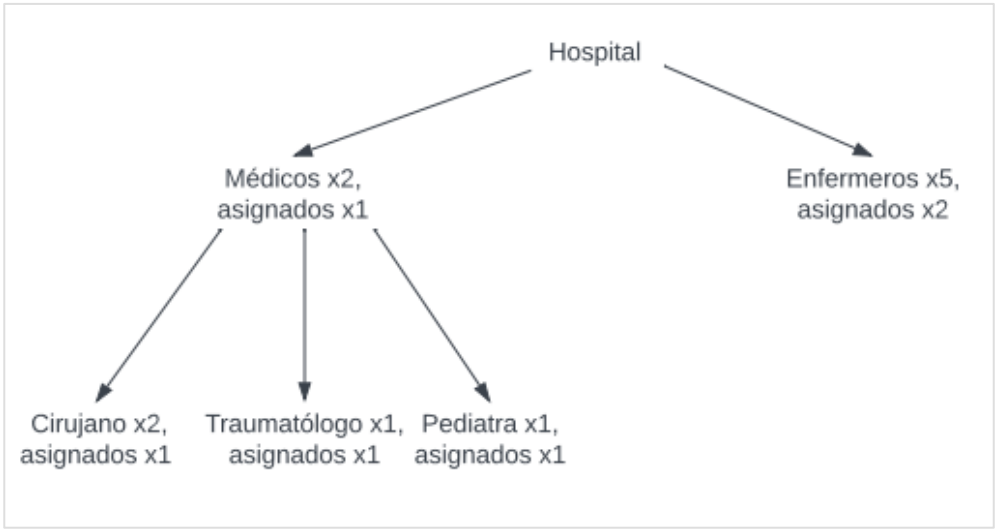

Figura 6. Asignación del segundo enfermero

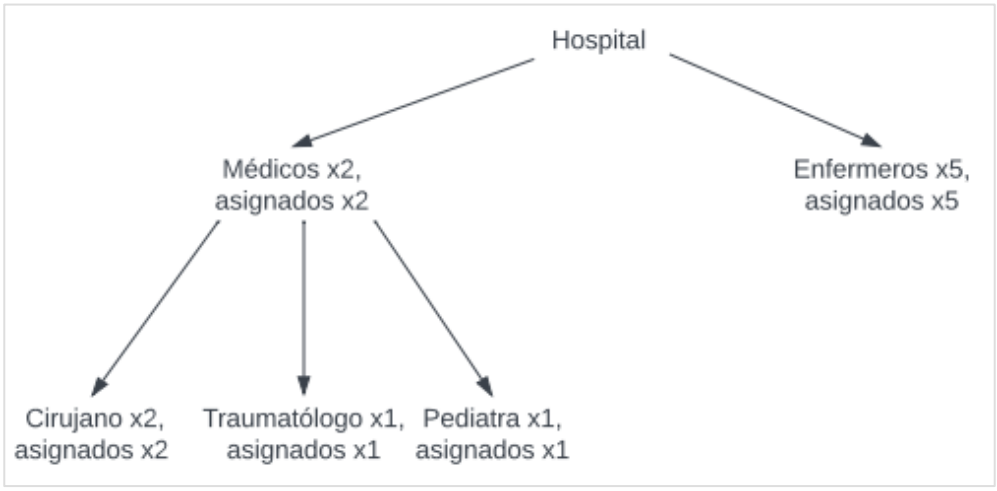

Figura 7. Asignación del quinto enfermero

A su vez puede ocurrir para la misma problemática planteada que los recursos disponibles no sean suficientes, suponiendo que se poseen menos enfermeros de los solicitados, causando que no se pueda llegar a una solución donde sí se satisfacen todos los estados pero que igual se pueda tomar como válida. Como resultado se obtendría lo visualizado en la figura 6, donde a pesar de no ser una solución completa, es una solución válida porque se utilizan todos los recursos disponibles para pueden cumplir con la demanda existente según las normas establecidas previamente.

Ventajas y desventajas de las herramientas de búsqueda:

Ventajas	Desventajas
Eficiencia: permiten explorar de manera sistemática un espacio de soluciones, lo que puede llevar a una búsqueda más eficiente y evitar la necesidad de examinar todas las combinaciones posibles. Flexibilidad: adaptables a una amplia variedad de problemas de decisión y son aplicables en diferentes dominios, desde la planificación de rutas hasta la toma de decisiones en juegos. Enfoque sistemático: proporcionan una estructura clara y un enfoque sistemático para resolver problemas de decisión, lo que facilita la comprensión del problema y la implementación de la solución. Optimización: algunas herramientas de búsqueda, como los algoritmos heurísticos, pueden encontrar soluciones cercanas a la óptima en problemas complejos donde la búsqueda exhaustiva no es viable.	Complejidad computacional: En problemas grandes o con un espacio de búsqueda extenso, las herramientas de búsqueda pueden requerir una cantidad significativa de tiempo y recursos computacionales. Óptimo local: Algunas técnicas de búsqueda, como la búsqueda local, pueden quedar atrapadas en óptimos locales y no garantizar la solución óptima global. Limitaciones del modelo: dependen de la formulación adecuada del problema y la elección de las variables y restricciones. Una formulación inadecuada puede llevar a soluciones sub óptimas o incorrectas. Heurísticas inexactas: Las heurísticas utilizadas en las herramientas de búsqueda a menudo son aproximaciones y pueden no proporcionar soluciones precisas o garantizadas.

2.2 Heurísticas

Los métodos heurísticos son estrategias generales de resolución y reglas de decisión utilizadas por los que desean solucionar problemas, basadas en la experiencia previa con problemas similares. Estas estrategias indican las vías o posibles enfoques a seguir para alcanzar una solución. [4]

Los métodos heurísticos pueden varias en el grado de generalidad. Algunos son muy generales y se pueden aplicar a una gran variedad de dominios, otros pueden ser más específicos y se limitan a un área particular del conocimiento. [4]

Un ejemplo de heurística ampliamente utilizada es el algoritmo A* (A-Star) que se aplica en la búsqueda en grafos o árboles. A* combina la búsqueda en amplitud (BFS) con una función heurística para estimar la proximidad a la solución deseada. El algoritmo utiliza dos valores para cada nodo: el costo acumulado del nodo desde el inicio (g) y una estimación heurística del costo restante hasta la meta (h). La función de evaluación f(n) se define como f(n) = g(n) + h(n), y se selecciona el nodo con el valor f(n) más bajo en cada iteración.

Por ejemplo, supongamos que deseas encontrar la ruta más corta entre dos ciudades en un mapa con múltiples rutas posibles. El algoritmo A* puede utilizar una heurística como la distancia euclidiana entre cada ciudad y la ciudad objetivo como estimación h(n). Al explorar el mapa, A* considerará tanto el coste acumulado de llegar a cada ciudad (g(n)) como la distancia estimada hasta la ciudad objetivo (h(n)) para decidir qué ruta seguir en cada paso. El objetivo es encontrar la ruta con el valor f(n) más bajo, lo que representa la mejor estimación de la ruta más corta.

Es importante tener en cuenta que las heurísticas no siempre garantizan la solución óptima, ya que se basan en estimaciones y aproximaciones. Sin embargo, son útiles en situaciones en las que es necesario tomar decisiones rápidas y el espacio de búsqueda es demasiado grande para una búsqueda exhaustiva.

Ventajas y desventajas de las Heurísticas:

Ventajas	**Desventajas**
Eficiencia: permiten tomar decisiones rápidas al buscar soluciones aproximadas en problemas complejos. Pueden encontrar soluciones satisfactorias en un tiempo razonable, evitando la necesidad de una búsqueda exhaustiva. Adaptabilidad: son flexibles y pueden aplicarse a una amplia gama de problemas en diversos dominios. Pueden ser adaptadas y ajustadas según las características específicas del problema. Reducción del espacio de búsqueda: ayudan a reducir el espacio de búsqueda, ya que utilizan información y conocimiento previo para guiar la exploración hacia regiones más prometedoras del espacio de soluciones. Escalabilidad: útiles para problemas grandes y complejos, donde la búsqueda exhaustiva no es factible debido al tamaño del espacio de búsqueda. Pueden proporcionar soluciones satisfactorias incluso en problemas de gran escala.	Soluciones aproximadas: no garantizan la obtención de la solución óptima. Debido a su naturaleza aproximada, la calidad de la solución encontrada puede variar y no siempre es la mejor posible. Sensibilidad al conocimiento y experiencia: dependen del conocimiento y la experiencia previa para guiar la búsqueda. Si el conocimiento es incorrecto o insuficiente, las soluciones encontradas pueden ser sub óptimas o incorrectas. Posibilidad de quedar atrapado en óptimos locales: algunas heurísticas pueden quedar atrapadas en óptimos locales y no explorar todo el espacio de soluciones en busca de la solución óptima global. Dificultad de ajuste y optimización: en algunos casos, ajustar y optimizar las heurísticas puede ser un desafío, ya que pueden depender de múltiples parámetros y configuraciones que deben ser calibrados adecuadamente para obtener buenos resultados.

2.3 Algoritmos de programación entera y mixta

Los algoritmos de programación entera y mixta son técnicas utilizadas para resolver problemas de optimización en los que las variables de decisión deben tomar valores enteros

(programación entera) o una combinación de valores enteros y continuos (programación mixta). [5]

1. Programación entera: En la programación entera, todas las variables de decisión se restringen a tomar valores enteros. Esto agrega una complejidad adicional a la resolución del problema, ya que el espacio de búsqueda se vuelve discreto en lugar de continuo.
2. Programación mixta: La programación mixta permite que algunas variables de decisión tomen valores continuos y otras tomen valores enteros. Esto amplía la flexibilidad del modelo y permite una representación más precisa de algunos problemas.

Estos algoritmos son solo algunas de las técnicas utilizadas para resolver problemas de programación entera y mixta. La elección del algoritmo depende de la naturaleza del problema, la estructura del modelo y las restricciones específicas del dominio. La resolución de problemas de programación entera y mixta puede ser desafiante, ya que los problemas se vuelven más complejos y el espacio de búsqueda puede ser grande.

Ventajas y desventajas de la programación entera y mixta:

Ventajas	**Desventajas**
Flexibilidad en la modelización: modela problemas que incluyen tanto variables continuas como variables enteras, lo que proporciona una representación más precisa de muchos problemas del mundo real. Soluciones óptimas o cercanas a óptimas: en problemas donde las variables enteras representan decisiones discretas o restricciones específicas, la programación entera mixta puede encontrar soluciones óptimas o cercanas a óptimas. Capacidad de incorporar restricciones adicionales: permite agregar restricciones adicionales a las variables enteras, lo que permite tener en cuenta condiciones específicas y restricciones del problema. Soluciones más eficientes: aunque la programación entera mixta es más compleja que la programación lineal pura, los avances en algoritmos y técnicas de resolución han mejorado la eficiencia en la obtención de soluciones.	Mayor complejidad computacional: a medida que aumenta el tamaño y la complejidad del problema, la resolución puede volverse más lenta y requerir recursos computacionales significativos. Sensibilidad al tamaño del espacio de búsqueda: se basa en la búsqueda exhaustiva en un espacio de búsqueda discreto, lo que puede ser un desafío en problemas con un espacio de búsqueda grande. Problemas NP-duros: Algunos problemas se consideran NP-duros, lo que significa que no se ha encontrado un algoritmo eficiente para encontrar soluciones óptimas en todos los casos. Formulación y ajuste adecuado: la formulación del problema y el ajuste de los parámetros y restricciones pueden ser complejos. Una formulación incorrecta puede resultar en soluciones sub óptimas o inviables.

2.4 Problema de satisfacción de restricciones

Los problemas de satisfacción de restricciones (CSP, por sus siglas en inglés) es un tipo de problema en el campo de la inteligencia artificial y la programación centrado en encontrar soluciones que cumplan con un conjunto de restricciones predefinidas, las cuales son condiciones que las soluciones deben satisfacer para ser consideradas válidas, en algunos casos se pueden tener múltiples soluciones o incluso ninguna solución.

Se debe especificar un conjunto de variables, un dominio para cada variable y un conjunto de restricciones que limiten los posibles valores de las variables. Su objetivo es encontrar una asignación de valores que cumpla con todas las restricciones y suelen representarse como un grafo o una red de restricciones, donde los nodos representan variables y las aristas representan restricciones entre variables.

Su aplicación está dirigida a diversos campos como la planificación de horarios, diseño de circuitos electrónicos, asignación de recursos, logística y programación de tareas, entre otros. Se utilizan técnicas como la búsqueda heurística, la programación lineal y la programación por restricciones. [6]

Ventajas y desventajas de los CSP:

Ventajas	**Desventajas**
Modelo intuitivo: modela problemas de forma intuitiva donde se definen variables, dominios y restricciones que reflejan características y limitaciones. Flexibilidad de representación: representa una amplia gama de problemas con diferentes tipos de restricciones. Métodos de resolución eficientes: existen técnicas y algoritmos para resolver problemas de manera eficiente, encontrando soluciones óptimas en un tiempo razonable. Capacidad de encontrar soluciones óptimas: pueden buscar soluciones óptimas que cumplan con todas las restricciones establecidas. Escalabilidad: pueden manejar problemas de gran número de variables y restricciones, por lo que son adecuados para problemas complejos.	Complejidad computacional: puede aumentar su coste computacional a medida que aumenta el tamaño del problema. Múltiples soluciones o ninguna solución: puede haber problemas de múltiples soluciones lo que dificulta encontrar la mejor solución, y en algunos casos puede que no haya solución que satisfagan todas las restricciones. Formulación de restricciones: es necesario considerar todas las restricciones relevantes y expresarlas de manera correcta. En algunos casos, los problemas pueden requerir conocimiento y experiencia de un experto para definir las restricciones del modelo. Sensibilidad a cambios en el problema: pequeños cambios en el problema pueden conllevar a modificaciones significativas en las restricciones.

2.5 Entornos para desarrollo

Existen diferentes aplicaciones y entornos donde se pueden desarrollar soluciones al problema que se desea resolver empleando las herramientas estudiadas anteriormente. Los más populares son:

1. IDEs (Entornos de Desarrollo Integrado):
   - Visual Studio Code: editor de código ligero y altamente personalizable con soporte para una amplia gama de lenguajes. [7]
   - PyCharm: específico para Python que ofrece herramientas avanzadas para el desarrollo de aplicaciones en Python. [8]
   - IntelliJ IDEA: desarrollo de aplicaciones Java con soporte para otros lenguajes como Kotlin, Scala y Groovy. [9]
   - Eclipse: código abierto utilizado principalmente para desarrollo de aplicaciones Java, pero que también admite otros lenguajes a través de plugins. [10]

2. Editores de texto:
   - Sublime Text: ligero y altamente personalizable con soporte para muchos lenguajes y complementos. [11]
   - Atom: código abierto desarrollado por GitHub que es altamente personalizable y extensible. [12]
   - Vim: editor de texto modal disponible en la mayoría de los sistemas operativos Unix, conocido por su eficiencia y capacidad de personalización. [13]
   - Emacs: editor de texto extensible y altamente personalizable con soporte para una amplia gama de lenguajes de programación. [14]

3. Jupyter Notebooks: entorno interactivo basado en web que permite combinar código, texto explicativo y visualizaciones en un único documento. Es ampliamente utilizado en ciencia de datos y análisis exploratorio. [15]

4. Plataformas en la nube:
   - GitHub: plataforma de desarrollo colaborativo basada en la nube que permite alojar y compartir proyectos de código abierto. [16]
   - Google Colab: entorno de cuadernos basado en la nube que ofrece acceso gratuito a recursos de cómputo y bibliotecas de Python. [17]
   - Microsoft Azure Notebooks: servicio en la nube de Microsoft que proporciona cuadernos Jupyter para desarrollar y ejecutar código Python. [18]

Para la realización de este trabajo se ha decidido trabajar con Google colab porque es muy fácil e intuitivo de utilizar por cualquier persona. Es gratuito, no requiere configuración, posee acceso a GPUs sin coste adicional y permite compartir contenido fácilmente.

Google colab permite utilizar la biblioteca OR-TOOLS la cual es un paquete de software de código abierto para la optimización, optimizado para abordar los problemas más difíciles del mundo en el enrutamiento de vehículos, los flujos, la programación lineal y en números enteros, y la programación de restricciones, ganadora de la competencia de programación internacional con restricciones de cada año desde 2013. [19]

Además, OR-TOOLS puede emplear distintos solvers como:

- Gurobi: biblioteca comercial de optimización matemática que ofrece una amplia gama de algoritmos para resolver problemas de programación lineal, entera y cuadrática. Gurobi tiene interfaces en varios lenguajes, incluyendo Python, C++, Java y .NET. [20]
- CPLEX: biblioteca comercial de optimización matemática desarrollada por IBM. Proporciona algoritmos de optimización para resolver problemas de programación lineal, entera y cuadrática. CPLEX tiene interfaces en varios lenguajes, incluyendo Python, C++, Java y .NET. [21]
- SCIP: biblioteca de optimización de código abierto que se especializa en resolver problemas de programación entera y mixta. SCIP es altamente configurable y se utiliza en una amplia gama de aplicaciones. [22]
- GLPK: biblioteca de código abierto para resolver problemas de programación lineal (LP) y programación entera mixta (MIP). Es parte del proyecto GNU y está disponible de forma gratuita para su uso y desarrollo de aplicaciones. [23]
- GLOP: solver de programación lineal de código abierto desarrollado por Google, que forma parte de la biblioteca OR-Tools. Está diseñado para resolver problemas de programación lineal (LP) de manera eficiente y escalable. [24]

En algunos solvers como el GLOP, las variables binarias pueden tener valores en un rango de cero a uno (0-1) mientras que en otros como el SCIP los valores son estrictamente cero o uno.

Uno de los inconvenientes que posee OR_TOOLS es que la solución al modelo planteado solo será apreciable si hay una solución posible, esto quiere decir que, si la demanda definida no puede ser resuelta con la oferta existente, no se obtendrá una solución, y en consecuencia se usarán variables slack (variables de estiramiento) de las cuales se hablará más adelante.

3. Análisis del problema

Tal como se mencionó anteriormente, una smart city utiliza las tecnologías de información comunicación con el fin de mejorar la calidad de vida de las personas que habitan en ella. Toda ciudad tiene su propia demanda de necesidades basándose en distintos aspectos como su población, factores geográficos, climatológicos, económicos, políticos, sociales, etc.

Cualquier alcaldía o gobernación encargada de administrar dicha ciudad ha de procurar mantener dichas demandas satisfechas con los recursos que poseen o en su defecto, enfocarse en satisfacer la mayor cantidad de demandas existentes. Como es de esperarse, una de sus primordiales preocupaciones son el correcto funcionamiento de los servicios de públicos de asistencia como lo son hospitales, estaciones de policías y estaciones de bomberos y de segunda mano pudiéramos pensar en escuelas, personas encargadas de limpiar las calles, recolectores de desechos, transporte público, etc.

Para cubrir todas estas demandas de recursos por sectores se necesitan planes eficientes de asignación de recursos de forma que además se tenga un gasto económico reducido en sus posibilidades según del tamaño del problema que se afronte.

3.1 Solución propuesta

Para resolver la problemática planteada, se puede hacer uso de un modelo de aproximación matemática que intenta explicar parte de la realidad, considera algunos o todos los aspectos del escenario real. Lo primero que se necesita es conocer los elementos típicos de un modelo de optimización, expresado en la figura 8.

Figura 8. Esquema de creación de un modelo de optimización
Recuperado de: material docente y transparencias de poliformat de asignatura Calidad y Optimización

3.2 Parámetros (datos de entrada)

Antes de empezar a modelar se debe pensar qué se quiere optimizar y qué restricciones emplear recordando que este modelo representa una base como solución al problema planteado, pero que posteriormente puede usarse para casos más específicos, realizando adaptaciones en su código, así como pensar de que forma el usuario podrá introducir los datos al modelo.

Considerando que se va a trabajar con recursos, se ha dividido en dos áreas, recursos humanos y recursos físicos para definir características a cada uno y posteriormente ver similitudes entre ambos para asignarle a lo que se llama “recurso” propiedades comunes fáciles de entender:

Recurso Humano	Recurso Físico
• Ubicación • Nombre • Trabajo • Horas máximas de trabajo • Coste por hora	• Ubicación • Identificador • Tipo al que pertenece • Horas máximas que puede ser ocupado • Coste por hora

Una vez establecidas estas dos estructuras, y comprobada la similitud entre ambas, se procede a definir una clase genérica:

Recurso	
Ubicación	Nombre de la localización en donde se encuentra del recurso.
Identificador	Nombre que se le desee dar al recurso.
Tipo	Función principal del recurso.
Tipo Genérico	Función secundaria/genérica del recurso.
Horas	Máximo número de horas que se puede utilizar el recurso.
Coste por hora	Coste en euros del recurso por hora.

Se ha agregado el atributo Tipo genérico ya que es de utilidad cuando un recurso puede realizar dos tipos de trabajo, como ejemplo podemos tomar un médico traumatólogo, quien tiene como rol principal encargarse del área de traumatología pero que a su vez es un médico general.

Al momento de rellenar los datos, en el atributo Tipo se ha de colocar la función principal del recurso y en caso de que su trabajo o empleabilidad sea una rama o especialidad de otra, se podrá introducir su función general en el atributo Tipo genérico.

Para introducir los datos de demanda existentes, se ha definido una estructura bastante similar al de la oferta, donde los datos a introducir son los siguientes:

Demanda
• Ubicación • Tipo • Cantidad • Horas • Prioridad

Quedando el UML como se muestra en la figura 9.

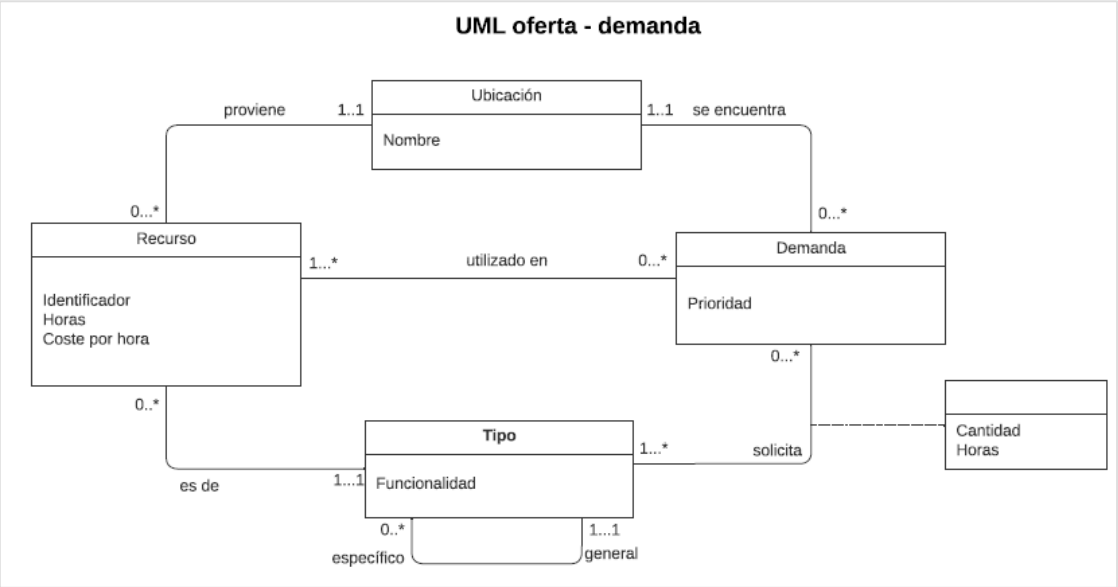

Figura 9. UML representativo de la problemática estudiada

3.3 Modelo matemático detallado

A continuación, se explica y detalla todo el modelo matemático planteado para la problemática definida, explicando que son las variables de decisión, cuáles se han creado, la definición de la función objetivo y la explicación de las distintas restricciones definidas.

3.3.1 Variables

Una vez conocidas las restricciones básicas del modelo se procede a definir las variables de decisión a emplear. Python cuenta con muchas opciones para esto:

- Variables binarias: son aquellas que pueden tomar valores ya sea de cero (0) o de uno (1); esta idea tan simple puede convertirse en una ayuda fundamental tanto para la modelación, como para la resolución de problemas.
- Variables no binarias: éstas pueden ser de tipo real o entero y son siempre continuas; su valor puede limitarse utilizando las restricciones y suelen ser útiles cuando queremos hablar de horas o cantidades de un mismo objeto.

Viendo que la problemática se centra en cantidad y horas; para la cantidad se emplearán variables binarias y para las horas se utilizarán variables enteras. Si se quisiera emplear medias horas (30 minutos) se puede ajustar la entrada de datos duplicando los tiempos, de manera que, si un recurso está disponible por una hora, se podrá sustituir por dos medias horas; lo mismo ocurriría para la demanda, siempre se deben trabajar en la misma escala temporal. Para cada recurso se definirá una variable entera de horas que puede trabajar en una ubicación para la cubrir la demanda de su propio tipo. Si se tienen cinco ubicaciones distintas, se define una

variable por ubicación y, si además el recurso posee un tipo genérico, se debe definir la misma cantidad de variables para este segundo tipo.

Ahora también se han de definir las variables slack, donde se define una por cada demanda existente de tipo no binaria y binaria como se ha hecho anteriormente. Estas variables no son espaciales o se definen de una forma distinta a lo indicado previamente, son iguales, solo que se encargarán de tomar los valores de las demandas que no puedan ser cubiertas para que el modelo pueda llegar a una solución. Para las horas faltantes se emplea de misma forma una variable entera y para la cantidad de personal faltante para cubrir dicha demanda una variable binaria. Estas variables de slack se han de incluir en la función objetivo, pero el coste por hora irá en función de las prioridades de las demandas.

$H_{id,ubi,tipo}$: número de horas que asiste el recurso id en la ubicación ubi con la empleabilidad tipo;

$$H_{id,ubi,tipo} \in Z \; ; \; H_{id,ubi,tipo} \geq 0$$

$T_{id,ubi,tipo}$: 1 si el recurso id asiste en la ubicación ubi con la función tipo, 0 en caso contrario;

$$T_{id,ubi,tipo} \in Z \; ; \; T_{id,ubi,tipo} \geq 0$$

M_{id}: 1 si el recurso id asiste al menos una ubicación, 0 en caso contrario;

$$M_{id} \in Z \; ; \; M_{id} \geq 0$$

$HS_{ubi,tipo}$: número de horas que no son cubiertas en la ubicación ubi con la función tipo;

$$HS_{ubi,tipo} \in Z \; ; \; HS_{ubi,tipo} \geq 0$$

$TS_{ubi,tipo}$: número de falta de recursos en la ubicación ubi con la función tipo;

$$TS_{ubi,tipo} \in Z \; ; \; TS_{ubi,tipo} \geq 0$$

$total\_gastado$: total gastado en euros generado por la asignación de los recursos;

$$Gasto\_Total \in R \; ; \; Gasto\_Total \geq 0$$

Parámetros

ID: identificador de cada recurso de la lista ofertada definida en la oferta.

Ubi: lista de las diferentes ubicaciones existentes en la demanda.

$Tipo$: función que cumple el recurso por su Tipo y Tipo Genérico obtenido de la oferta.

3.3.2 Función objetivo

Para definir la función objetivo se ha de considerar el coste que conlleva cubrir las horas de demanda por cada recurso, dicho cálculo sería:

$$costo\_hora_{recurso}\ x\ (horas\ trabajadas\ en\ total + (total\ translados - 1))$$

Adicionalmente, para asignarle importancia a las prioridades de las demandas, hay que generar una penalización para las horas que no sean cubiertas por falta de recurso, de forma que el solver decida cuál es la solución óptima minimizando el gasto que conlleva no cubrir horas de una ubicación A de prioridad X, por cubrir las horas de otra ubicación B de prioridad Y. De esta forma al ser las prioridades valores posibles del uno (1) al cinco (5), donde el menor valor es el más prioritario, se hará que el coste por hora de las horas no cubiertas aumente en función de la prioridad de dicha ubicación, haciendo que el coste de la prioridad 1 sea el más elevado y de la prioridad 5 el menor, quedando así:

- Prioridad 1: 100€ por hora
- Prioridad 2: 80€ por hora
- Prioridad 3: 60€ por hora
- Prioridad 4: 40€ por hora
- Prioridad 5: 20€ por hora

El coste mínimo por hora se ha definido en veinte euros (€20) que vendría siendo un sueldo superior en promedio al de ciertas especialidades como las de un médico cirujano. Hay que tener en cuenta cuales son los costes por hora establecidos para los recursos y observar cual sería el mayor de ellos. Una vez conocido, se define un coste superior a éste para la prioridad 5 y a partir de allí se definen nuevos costes para las siguientes prioridades.

$$costo\_prioridad_{ubicación}\ x\ (horas\_totales\_no\_cubiertas)$$

Quedando así la función objetivo como:

$$\sum_{id\ \in\ ID}\ \sum_{ubi\ \in\ Ubi} (H_{id,ubi,tipo} + T_{id,ubi,tipo}) - \sum_{id\ \in\ ID} M_{id} + \sum_{ubi\ \in\ Ubi} (costo\_hora\_prioridad_{ubi}\ x\ HS_{ubi,tipo})$$

3.3.3 Restricciones

Llegado a este punto se plantean cada una de las restricciones que el modelo debe cumplir para la resolución del problema y se conocen qué variables se definirán en el modelo. Es importante recordar que los recursos nunca son ilimitados y por ello debemos plantear restricciones que sean realistas pero que a la vez le permitan holgura al modelo.

1) Toda demanda de total de horas de un tipo de recurso debe ser satisfecha para cada ubicación.

$$[demanda] \sum_{id \in ID} \left(H_{id,ubi,tipo}\right) + HS_{ubi,tipo} \geq \left(cantidad_{ubi,tipo}\ x\ horas_{ubi,tipo}\right)\ ; ubi, tipo \in demanda$$

2) Toda demanda de cantidad de un tipo de recurso debe ser satisfecha para cada ubicación.

$$[demanda] \sum_{id \in ID} \left(T_{id,ubi,tipo}\right) + TS_{ubi,tipo} \geq cantidad_{ubi,tipo}\ ; ubi, tipo \in demanda$$

3) Un recurso no puede trabajar más horas de las que requiere una demanda que asiste: un recurso no debe asistir más horas de las necesarias establecidas en el turno que indica la demanda para dicha ubicación.

$$[recurso, demanda]\ H_{id,ubi,tipo} \leq\ horas_{ubi,tipo}\ ; id \in recurso\ ;\ ubi, tipo \in demanda$$

4) Un recurso no puede asistir más ubicaciones de las indicadas en los parámetros de entrada: puede ocurrir en ocasiones que un recurso trabaje menos en una zona y que sea demandado en otra, por lo que puede ser factible transportar dicho recurso de una ubicación A a ubicación B para poder utilizarlo, pero a la vez se debe tomar en cuenta que el tiempo de ese traslado es tiempo que el recurso utiliza, por lo que se ha considerado que el tiempo de traslado de un punto cualquiera a otro es de una hora de trabajo de dicho recurso.

$$[recurso] \sum_{ubi \in Ubi} T_{id,ubi,tipo} \leq max\_ubicaciones\ ; id, tipo \in recurso$$

5) Un recurso no puede trabajar más horas de las que tiene previstas por contrato: ninguna persona por ley debería trabajar más horas de las que se estipulan en su contrato, así como una maquinaria no debe de utilizarse más horas de forma continua de lo que indica su manual.

$$[recurso] \sum_{ubi \in Ubi} H_{id,ubi,tipo} \leq horas_{id} + 1 - \sum_{ubi \in Ubi} T_{id,ubi,tipo} \; ; id, tipo \in recurso$$

6) Un recurso no puede cubrir dos demandas en un mismo lugar (su demanda por tipo y tipo genérico): debería utilizarse un recurso para cubrir una única demanda por ubicación; si se piensa en un doctor, esta persona no debería ejecutar el trabajo de dos a la vez, ya que se afecta la calidad de atención al ciudadano, así como los tiempos de respuesta a diferentes procedimientos, aunque si puede cubrir tanto demandas de su Tipo como de su Tipo Genérico.

$$[recurso, Ubi] \sum_{tipo} T_{id,ubi,tipo} \leq 1 \; ; id, tipo \in recurso \; ; \; ubi \in Ubi$$

7) Cálculo de gasto: esta restricción proporcionará el gasto real que conllevaría la asignación de los recursos elegidos por el modelo. Éste vendrá dado por:

$$costo\_hora_{recurso} \; x \; número\_horas\_trabajadas_{ubicación}$$

Donde el traslado inicial que sería de la ubicación origen del recurso a la ubicación uno no se considera como uso de sus horas de trabajo, por la cual no genera un gasto.

$$\sum_{id \in ID} \sum_{ubi \in Ubi} (H_{id,ubi,tipo} + T_{id,ubi,tipo}) - \sum_{id \in ID} M_{id} - Gasto_{Total} = 0$$

8) Si un recurso asiste >0 una demanda, su binaria correspondiente vale 1: restricción de enlace que asigna valor a las variables binarias $T_{id,ubi,tipo}$. Si un recurso asiste una demanda, la binaria será igual a 1. Este valor es utilizado para satisfacer las restricciones de cantidad y gasto total, así como la función objetivo.

$$[recurso, Ubi] \sum_{tipo} H_{id,ubi,tipo} \leq \max \_horas_{id} \; x \; T_{id,ubi,tipo} \; ; \; id, tipo \in recurso \; ; ubi \in Ubi$$

9) Si un recurso asiste más de una ubicación, su binaria correspondiente vale 1: esta restricción es la restricción de enlace que le da valor a las variables binarias M_{id} utilizadas para el cálculo del gasto. Gracias a esto se disminuye el gasto de traslado del punto origen del recurso a la primera demanda que asiste.

$$[recurso] \sum_{ubi \in Ubi} T_{id,ubi,tipo} \leq 3 \; x \; M_{id} \; ; id, tipo \in recurso$$

4. Implementación de la solución

Para el desarrollo e implementación de la solución en la herramienta Google Colab se han necesitado diversos elementos desde el objeto solver y librerías hasta la creación de diversos métodos, por lo que para una mejor comprensión se explica cada uno de los involucrados a continuación.

4.1 Solver

Debido a como se han planteado las restricciones del problema donde, si un recurso se traslada de una ubicación A a una ubicación B se considera que dicho traslado tiene como consecuencia una hora de utilización del mismo, se utilizará el solver SCIP para limitar los posibles valores de las variables binarias a valores enteros cero (0) o uno (1).

4.2 Librerías empleadas

- requests: permite el trabajo con peticiones HTTP.
- json: proporciona la posibilidad de convertir el contenido de los ficheros demanda.json y oferta.json para ser utilizados en el proyecto.
- time: otorga el método .time() para medir los tiempos de respuesta del modelo en los casos que se realizan de prueba y experimentación.
- ortools.linear_solver: módulo perteneciente a la librería pywraplp para la utilización del solver.
- IPython.display: ofrece herramientas para visualizar y mostrar contenido enriquecido en el entorno de la notebook de IPython.

4.3 Definición de clases

Se han diseñado dos clases para facilitar la creación de los objetos Recurso así como su almacenamiento, para la posterior creación de variables de decisión y manejo de recorridos iterativos para la generación de las restricciones definidas anteriormente en el punto 3.3.3.

- Recurso: posee los atributos num, ubicacion, ident, tipo, horas y coste. A su vez se le implementa el método *agrega_padre()* el cual recibe el nombre de la función genérica que pueda poseer el recurso si lo tuviese y la agrega.
  Para visualizar los valores se han diseñado dos métodos *imprimir_valores()* e imprimir*_num_ident()* para visualizar la correcta creación de los objetos.
- Recursos por tipo: estructura de datos de tipo lista en la cual se agruparán los recursos por tipo.

4.4 Obtención de parámetros

La introducción de parámetros se realizará mediante archivos .JSON ya que usa el formato UTF-8 y es bastante fácil de rellenar y a su vez de ser leído y utilizado en Python. Se crea el archivo oferta.json el cual tendrá la base de datos de todos los recursos que se poseen y otro archivo demanda.json con la base de datos de todas las demandas existentes. Una vez se hayan rellenado los datos se cargan en un repositorio de github público al cual luego se podrá acceder desde el proyecto para poder utilizar los datos.

Para la correcta lectura de los datos se emplea el método *requests.get()* el cual recibe como parámetros el URL de cada archivo.

4.5 API diseñada

ingresar_datos(oferta, demanda)

Recibe como parámetros las direcciones URL de los ficheros oferta.json y demanda.json, realiza obtención de datos y con el método .json() se obtienen los objetos de cada fichero para ser utilizados posteriormente en el modelo.

indicar_ubicaciones()

Solicita un número entero mayor a cero y sin decimales para el *max_ubicaciones*, en caso de que el número ingresado sea menor o igual a cero o decimal, volverá a solicitar el número hasta que cumpla con las características mencionadas.

rellenar_demanda()

Almacena todos los tipos y ubicaciones distintas que se encuentra en los parámetros de entrada de la demanda.

rellenar_oferta()

Almacena todos los tipos distintos encontrados en los datos de la oferta proporcionada.

rellenar_variables()

Esta función se encarga de crear y almacenar todas las variables de decisión necesarias para el modelo. Las variables que se crean son solo aquellas que tienen un posible uso, es decir, un recurso el cual su Tipo o Tipo Genérico no posea demanda no generará la creación de sus variables de decisión correspondientes.

Los recursos que si sean demandados se almacenarán y se generará una variable *binaria_resta* la cual será utilizada para el cálculo del gasto total.

Finalmente, agrega a las listas de slack una variable de falta de horas y una de falta cantidad para cada demanda habida en demanda.json.

función_objetivo()

Crea el objeto de tipo *solver* como minimización y agrega todas los coeficientes para el cálculo mediante el método *SetCoefficient(variable, coeficiente)*. Este método se utiliza entre dos a cuatro veces por recurso y ubicación en función de si el recurso puede asistir a la ubicación en función de si es demandado por Tipo y/o su Tipo Genérico, una para asignar las horas trabajadas (variable entera) y otra para indicar si el recurso trabaja en la ubicación demandada (variable binaria), junto a su coste por hora.

Posteriormente se agregan a la función objetivo los coeficientes de las variables de slack utilizando nuevamente el método *SetCoefficient(variable, coeficiente)*, donde esta vez los parámetros serán la variable slack de falta de horas cubiertas y falta de recursos según sea el caso, junto al coste por hora que viene dado por la prioridad de la demanda.

res_demanda_tipo()

Genera dos restricciones por demanda existente, una para las horas y otra para las cantidades de recursos que se solicitan. De cada demanda se obtiene su Tipo, Cantidad y Horas que solicita, se ubican todas las variables de cantidad de horas y binarias correspondientes a los recursos existentes para ese Tipo y se suman las variables de horas con la variable de slack de horas para definir la restricción. Se realiza la misma acción, pero esta vez sumando las variables binarias de cantidad con la de slack cantidad.

Si no existen recursos del mismo Tipo que requiere la demanda, la restricción solo se realiza solamente con las variables slack de horas y cantidad.

res_maxh_ubicaciones()

Define una restricción por recurso para cada demanda que pueda asistir. Se recorre cada demanda existente, se ubican los posibles recursos que son compatibles para asistir la demanda y se genera la restricción de que ninguno de ellos asista más horas de las que la demanda solicita.

res_max_ubicaciones()

Genera una restricción para cada recurso que pueda ser empleado porque su tipo y/o tipo genérico es demandado. Haciendo uso de la oferta utilizada se obtienen todas las variables binarias de cantidad generadas previamente y se suman estableciendo que la suma debe ser menor o igual al máximo de ubicaciones establecidas para los recursos.

res_max_horas()

Define una restricción por cada recurso que puede ser utilizado. Con la oferta utilizada se van buscando todas las variables enteras de horas definidas anteriormente, se suman y se indica que la suma debe ser menor o igual a la cantidad de horas de uso máximo indicado en los parámetros de entrada para ese recurso.

res_un_trabajo()

Igual que la función *res_max_horas()* se genera una restricción para cada recurso empleando la oferta utilizada por cada posible demanda que pueda asistir. Se obtienen la(s) variables binarias de asistencia (puede ser una o dos dependiendo de si posee un tipo y/o tipo genérico) para una ubicación A y se define que la suma de esta(s) debe ser menor o igual a uno

(1). Esto permite que el recurso pueda o no ser empleado en esta ubicación y en el caso de serlo garantiza que solo podrá cumplir una única función para esta ubicación.

res_enlace()

La función de este método es asignar un valor a las variables binarias de los recursos que refieren a si un recurso asiste (1) o no (0) una demanda, generando una restricción por recurso y demanda a la que puede asistir. Haciendo uso de la oferta utilizada se obtiene cada variable entera de horas, su binaria correspondiente y se define que el valor de la variable entera ha de ser menor o igual a multiplicación del límite de horas de uso del recurso por la variable binaria.

Explicación: para un recurso A, demanda B y tipo C con un máximo de cuatro horas de uso:

$H_{A,B,C} \leq 4\ x\ T_{A,B,C}$ donde si $H_{A,B,C} = 0 \rightarrow T_{A,B,C} = 0$ ó $T_{A,B,C} = 1$,

sí $H_{A,B,C} \geq 1 \rightarrow T_{A,B,C} = 1$

res_enlace_resta()

Este método hace una función similar al anterior, pero originando una única restricción por recurso de la lista. Suma todas las variables binarias creadas para el recurso por demanda que pueda asistir y hace que la suma sea menor o igual a la multiplicación de el máximo de ubicaciones indicado por la variable binaria M_{id} correspondiente. Esta variable binaria M_{id} será utilizada más adelante como se indicó anteriormente para el cálculo del gasto monetario total que conlleva la asignación de los recursos.

Explicación: para un recurso A y tipo C que puede asistir las demandas B y D, se tiene:

$T_{A,B,C} + T_{A,D,C} \leq max\_ubicaciones\ x\ M_{id}$,

donde si $T_{A,B,C} + T_{A,D,C} = 0 \rightarrow M_{id} = 0$ ó $M_{id} = 1$, sí $T_{A,B,C} + T_{A,D,C} \geq 1 \rightarrow M_{id} = 1$

res_gasto()

Realiza una función similar al de *función_objetivo()*. Suma cada variable $H_{id,ubi,tipo}$ con su propio $T_{id,ubi,tipo}$, le resta su variable M_{id} (para compensar el valor de traslado del punto origen del recurso a la ubicación de la primera demanda que asiste) y multiplica todo el cálculo anterior por el coste por hora del recurso *id* correspondiente. Finalmente, le resta la variable *total_gastado* y hace que la restricción sea igual a cero (0), haciendo así que ésta tome el mismo valor del coste de asignaciones y se pueda visualizar posteriormente.

Explicación: para un recurso A, tipo C que asiste la demanda B cuatro horas y E dos horas y otro recurso D, tipo F que asiste la demanda E siete horas, ambos de coste por hora igual a cinco (5), se tiene:

$\left(H_{A,B,C} + T_{A,B,C} + H_{A,E,C} + T_{A,E,C} - M_A\right) x\ costo_{hora\ A} + \left(H_{D,E,C} + H_{D,E,C} - M_D\right) x\ costo\_hora_D - total\_gastado = 0$

$(4 + 1 + 2 + 1 - 1)\ x\ 5 + (7 + 1 - 1)\ x\ 5 - total\_gastado = 0$

$(7)\ x\ 5 + (7)\ x\ 5 - total_{gastado} = 0 \rightarrow\ total\_gastado = 70$

func_resuelve()

Esta función ejecuta la resolución del modelo, muestra el valor de la función objetivo y proporciona el listado de todos los recursos que son utilizados, su ubicación, la función que realizan y la cantidad de horas correspondientes. Además, muestra el valor de la variable *total_gastado* que viene siendo el gasto total monetario que conlleva las asignaciones realizadas y todas las demandas insatisfechas indicando para cada una la cantidad de horas y recursos faltantes.

4.6 Resolución

Para ejecutar el modelo se deben ingresar las URL de la oferta y demanda, llamar a la función *ingresar_datos(oferta, demanda)* e *indicar_ubicaciones()*, crear el objeto *solver* que en este caso se ha elegido el SCIP e invocar a todas las funciones indicadas anteriormente. Finalmente, solo quedaría ejecutar el modelo y observar los resultados arrojados.

5. Pruebas de funcionamiento y resultados de ejecución

En este capítulo se presentan los distintos casos utilizados junto a los resultados obtenidos. Primero se realizan unas pruebas iniciales para la comprobación del correcto funcionamiento del modelo y luego se pasa al apartado de creaciones de datos aleatorios para la ejecución de distintos niveles de escalabilidad, mostrando los resultados obtenidos y realizando un análisis de estos, todas las pruebas se han realizado indicando que el parámetro de entrada *max_ubicaciones* es igual a tres (3).

5.1 Pruebas de funcionamiento iniciales

Para la comprobación del funcionamiento del modelo se realizan tres pruebas sencillas. La primera prueba consta de un solo recurso en la oferta como se puede observar en la figura 10 y cuatro demandas en ubicaciones y prioridades distintas, con el mismo tipo solicitado (figura 11).

Ubicacion	Identificador	Tipo	Tipo Generico	Horas	Coste
Benimaclet	Luis	Medico		8	7

Figura 10. Oferta prueba 1

Ubicacion	Tipo	Cantidad	Horas	Prioridad
Benimaclet	Medico	1	1	1
Orriols	Medico	1	1	2
Campanar	Medico	1	1	3
UPV	Medico	1	8	4

Figura 11. Demanda prueba 1

Se puede comprobar, tal como se observa en la figura 12, que el recurso no es asignado en más de tres ubicaciones, las horas totales de uso no son mayor a seis (6) porque se movilizará a dos ubicaciones distintas, además de la ubicación inicial haciendo un total de ocho horas y se comprueba que el total gastado por el uso de este recurso es de cincuenta y seis euros (€56).

```
El total gastado es: 56.0 euros

Luis es asignado a Benimaclet como Medico, horas: 1.0
Luis es asignado a Orriols como Medico, horas: 1.0
Luis es asignado a UPV como Medico, horas: 4.0

Demanda no cubierta en su totalidad: 2
Campanar tiene falta de horas en Medico: 1.0
Campanar requiere más Medicos para cumplir el minimo, en total: 1.0
UPV tiene falta de horas en Medico: 4.0
UPV requiere más Medicos para cumplir el minimo, en total: 1.0
```

Figura 12. Resultado de ejecución prueba 1

Adicionalmente se observa el listado de demandas no satisfechas indicando las horas y cantidades necesarias para satisfacerlas. En este caso el solver arroja como solución que antepone la satisfacción parcial de la demanda de la ubicación UPV antes que la de Campanar a pesar de las prioridades, porque tal como está hecha la función objetivo, es más económico cubrir cuatro horas de la prioridad 4 que vendrían siendo ochenta euros (€80) de no ser satisfechas, antes que cubrir solo una de prioridad 3 que representan sesenta euros (€60).

Se realiza una segunda prueba con la misma oferta enunciada en la figura 10 con una demanda diferente compuesta por dos ubicaciones distintas con una nueva solicitud de horas y cantidad, pero ambas del mismo tipo y prioridad como se aprecia en la figura 13.

Ubicacion	Tipo	Cantidad	Horas	Prioridad
Benimaclet	Medico	1	4	1
Orriols	Medico	2	6	1

Figura 13. Demanda prueba 2

Los resultados de esta ejecución se observan en la figura 14, que muestra cómo se ha asignado el recurso en ambas ubicaciones, siendo éste utilizado cuatro horas en Benimaclet y tres horas en Orriols. También indica que hay una falta de nueve horas en Orriols y que para cubrirla se necesitan en total dos recursos de este tipo ya que como máximo cada uno puede ser asignado por seis horas para esta demanda.

```
El total gastado es: 56.0 euros

Luis es asignado a Benimaclet como Medico, horas: 4.0
Luis es asignado a Orriols como Medico, horas: 3.0

Demanda no cubierta en su totalidad: 1
Orriols tiene falta de horas en Medico: 9.0
Orriols requiere más Medicos para cumplir el minimo, en total: 2.0
```

Figura 14. Resultado de ejecución prueba 2

Finalmente se realiza una prueba más completa de verificación, compuesta de ocho recursos divididos en los tipos “Médico”, ¨Policía”, “Cirujano”, “Bombero”, “Squad”. El recurso tipo “Cirujano” posee un tipo genérico “Médico” y el de tipo “Squad” tiene tipo genérico “Policía”, como se describe en la figura 15.

Ubicacion	Identificador	Tipo	Tipo Generico	Horas	Coste
Benimaclet	Luis	Medico		8	7
Benimaclet	Sergio	Bombero		8	5
Benimaclet	Alejandra	Medico		8	7
Benimaclet	Nicole	Cirujano	Medico	8	10
Blasco	Anna	Policia		8	5
Blasco	Gaby	Squad	Policia	8	8
Blasco	Nea	Bombero		8	5
Blasco	Alberto	Bombero		8	5

Figura 15. Oferta prueba 3

La demanda para esta ejecución está compuesta por dos ubicaciones y los mismos tipos que los de la figura 16.

Ubicacion	Tipo	Cantidad	Horas	Prioridad
Benimaclet	Medico	2	8	1
Benimaclet	Cirujano	1	2	1
Benimaclet	Bombero	1	8	2
Blasco	Cirujano	1	5	1
Blasco	Squad	1	8	4
Blasco	Policia	1	8	3
Blasco	Bombero	2	8	1

Figura 16. Demanda prueba 3

Como resultado se tiene una total asignación de los recursos satisfaciendo toda la demanda y el coste monetario es de cuatrocientos dieciséis euros (€416) como se muestra en la figura 17.

```
El total gastado es: 416.0 euros

Luis es asignado a Benimaclet como Medico, horas: 8.0
Sergio es asignado a Blasco como Bombero, horas: 8.0
Alejandra es asignado a Benimaclet como Medico, horas: 8.0
Nicole es asignado a Benimaclet como Cirujano, horas: 2.0
Nicole es asignado a Blasco como Cirujano, horas: 5.0
Anna es asignado a Blasco como Policia, horas: 8.0
Gaby es asignado a Blasco como Squad, horas: 8.0
Nea es asignado a Blasco como Bombero, horas: 8.0
Alberto es asignado a Benimaclet como Bombero, horas: 8.0

Demanda no cubierta en su totalidad: 0
No hay demanda pendiente
```

Figura 17. Resultado de ejecución prueba 3

5.2 Funcionamiento con datos generados aleatoriamente

Se procede a la toma de tiempos de ejecución del modelo basándose en tres niveles distintos de complejidad. Se ha diseñado una base de datos para la oferta y otra para la demanda, donde en cada nivel hay una cantidad distinta de datos:

- Nivel 1: 200 recursos y 20 demandas.
- Nivel 2: 400 recursos y 40 demandas.
- Nivel 3: 600 recursos y 60 demandas.

Para las ubicaciones se han considerado los 19 distritos que componen la ciudad de Valencia, España[23] como se observa en la figura 18.

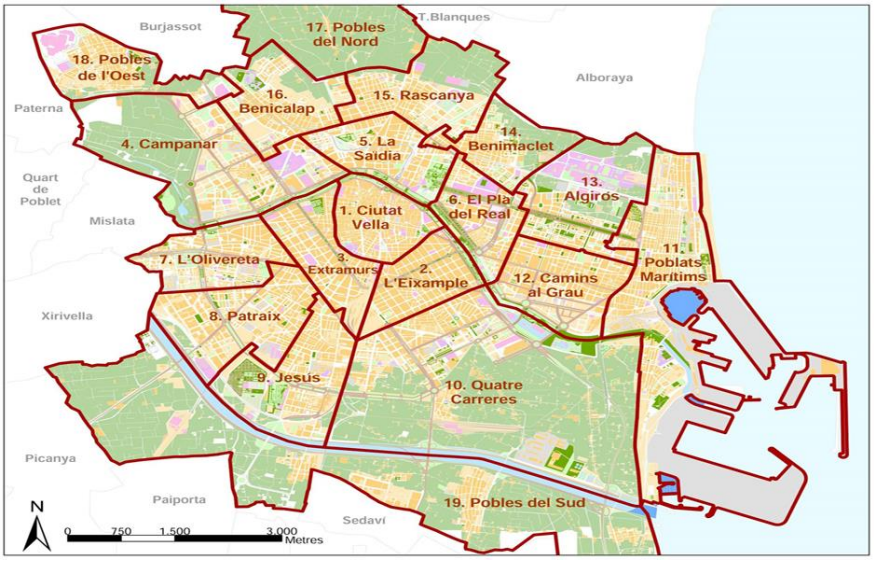

Figura 18. Mapa de distritos de Valencia, España
Recuperado de: https://calles.valenciaactua.es/los-19-distritos-de-valencia/

Para la realización de estas ejecuciones se han diseñado dos casos de prueba, a las cuales se le han generado sus datos 15 veces de forma aleatoria (5 por nivel para oferta y demanda) como se explica a continuación en el punto 5.2.1 y 5.2.2. Es importante recordar que este modelo no está limitado a estos datos, solo se han utilizado para poder generar casos de prueba pero que el usuario final va a poder utilizar sus propios datos con sus propias ofertas y demandas, sin necesidad de emplear las mismas ubicaciones, tipos, tipos genéricos, etc.

5.2.1 Generación de archivos de oferta

Los tipos posibles de oferta se han definido atendiendo a las especializaciones más comunes que pueden existir de una persona como Cirujanos, Traumatólogos, Bomberos Ayuntamiento, Comisarios, entre otros; así como de recursos físicos que pueden ser vehículos de asistencia de tipo Ambulancia Individual, Vehículo de asistencia Médica, Patrulla, Camión Bomba y muchos más, definiendo así una lista de dieciséis (16) tipos. Los tipos genéricos se han definido atendiendo a los tipos especializados descritos anteriormente: Médico, Bombero, Policía, Vehículo grande y Vehículo pequeño.

Para generar los archivos de oferta se ha desarrollado el archivo *generador_oferta.py*, el cual utiliza las ubicaciones, tipos y tipos genéricos definidos anteriormente y crea la lista de ofertas eligiendo de forma aleatoria una ubicación, tipo, horas (1-8) y coste (6-10). El tipo genérico se otorga correspondiendo al tipo que ha sido asignado al recurso al igual que su identificador, en el caso de ser un recurso humano se genera un nombre y de ser un vehículo una matrícula.

Para distinguir un recurso de otro, se ha priorizado que la combinación (ubicación, identificador, tipo) sea única para los recursos humanos, delimitando que no puede haber dos recursos de identificador, ubicación y tipo iguales. Para los vehículos el identificador (matrícula) es único por lo que pueden existir muchos vehículos de la misma ubicación y tipo siempre y cuando sean de diferentes identificadores.

5.2.2 Generación de archivos de demanda

Para la creación de los archivos de demanda se ha desarrollado el archivo *generador_demanda.py*, las ubicaciones son las definidas anteriormente y los tipos vienen de la combinación de los tipos y tipos genéricos del punto 6.1.

Este código selecciona de manera aleatoria una ubicación, tipo, cantidad (1-5), horas (1-8) y prioridad (1-5) en el cual para el archivo demanda generado la combinación de ubicación y tipo no puede repetirse para garantizar que todas las demandas serán distintas.

5.3 Comparación de tiempos de ejecución

Para la comparación de tiempos de ejecución se realiza un total de 25 ejecuciones del modelo (5 ofertas vs 5 demandas) para observar todas las combinaciones posibles de resultados. Hay que considerar que, al ser todos los datos de ofertas y demandas generados aleatoriamente, no se garantiza que todas las demandas sean satisfechas en su totalidad debido a la posible falta de cantidad de recursos, horas o inclusive tipos que son demandados pero que no son ofertados.

5.3.1 Resultados de ejecución de Nivel 1, 200 recursos – 20 demandas

Se observa en la figura 19 los resultados obtenidos de la resolución de cada oferta con cada demanda, donde se genera una media de 1.462 restricciones, 4,2 demandas no satisfechas en su totalidad y un tiempo de ejecución promedio de 0,325 segundos.

	NIVEL 1														
	demanda 1			demanda 2			demanda 3			demanda 4			demanda 5		
	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE
oferta 1	1.598	4	0,348	1.400	4	0,244	1.449	7	0,177	1.539	4	0,242	1.416	5	0,244
oferta 2	1.583	3	1,275	1.447	3	0,287	1.471	3	0,214	1.613	3	0,329	1.382	3	0,275
oferta 3	1.577	2	0,324	1.392	5	0,290	1.392	2	0,197	1.619	6	0,241	1.304	3	0,253
oferta 4	1.576	4	0,370	1.286	5	0,338	1.478	6	0,378	1.485	5	0,616	1.384	4	0,238
oferta 5	1.447	4	0,284	1.400	5	0,198	1.388	5	0,186	1.599	3	0,447	1.315	7	0,122

Leyenda	
NR	Número de restricciones
DNC	Demanda no cubierta en su totalidad
TE	Tiempo de ejecución

Promedio	
NR	1.462
DNC	4,2
TE	0,325

Figura 19. Tabla de Resultados de Ejecución del Nivel 1

Adicionalmente, si aumenta el número de restricciones no tiene por qué aumentar respectivamente el tiempo de resolución del problema ya que puede existir el caso de que hay menos tipos y/o cantidades de un recurso que es demandado y ofertado por lo que es más sencillo de realizar su asignación respecto a otros.

La figura 20 muestra el comportamiento de la demanda no cubierta para cada caso probado. Es apreciable que ninguna demanda ha sido satisfecha en su totalidad y era lo esperado ya que el propósito de generar los datos aleatoriamente era poder llevar las pruebas al caso más realista posible donde en el mundo real los recursos que existen no son ilimitados.

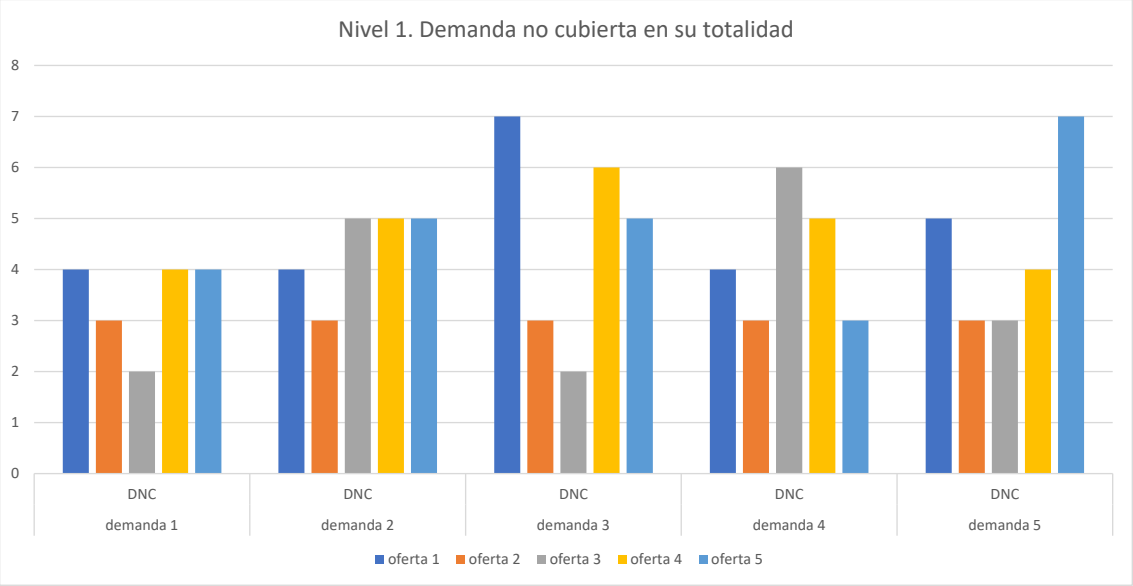

Figura 20. Gráfico de columnas agrupadas de Nivel 1. Demanda No Cubierta en su Totalidad

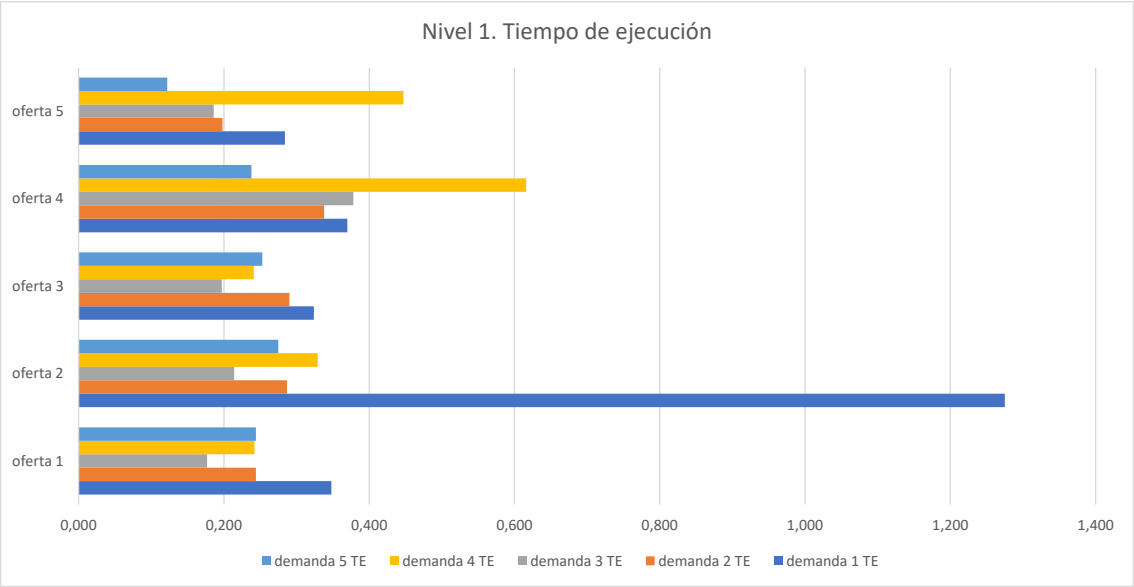

Figura 21. Gráfico de barras agrupadas de Nivel 1. Tiempo de Ejecución

En la figura 21 se pueden observar los distintos tiempos de ejecución de cada prueba, donde el tiempo de ejecución de la oferta 2 con demanda 1 es superior al resto con diferencia, siendo casi 1 segundo mayor al resto. Si adicionalmente se observa la figura 20, se puede apreciar como para las distintas ejecuciones de la oferta 2 con todas las demandas, el número de demandas no satisfechas en su totalidad es siempre 3, dando a entender que las diferencias de tiempo de ejecución puede ser debida a la relación de recursos los recursos que fueron ofertados y demandados, originando que las combinaciones de posibles asignaciones a considerar sean superiores aumentando así el tiempo de ejecución para definir la solución óptima.

5.3.2 Resultados de ejecución de Nivel 2, 400 recursos – 40 demandas

En la figura 22 se muestran los resultados de ejecución del nivel 2. Como consecuencia del aumento de recursos ofertados y demandas de un 100% respecto del nivel 1, se obtiene un promedio de 4.862 restricciones (aumento de 232,56%), 4,56 demandas no cubiertas en su totalidad y 5,774 segundos de ejecución (aumento de 1676,62%).

	NIVEL 2														
	demanda 1			demanda 2			demanda 3			demanda 4			demanda 5		
	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE
oferta 1	4.811	5	5,556	4.915	6	2,943	4.977	8	1,334	4.845	5	7,156	4.678	3	3,123
oferta 2	4.886	5	6,846	4.919	4	1,618	4.860	7	1,129	4.937	4	1,659	4.734	2	3,663
oferta 3	4.883	5	10,423	4.789	2	2,422	4.850	7	1,419	4.990	3	11,042	4.786	3	4,410
oferta 4	4.858	5	35,484	4.930	4	0,972	4.925	6	1,001	4.881	5	20,249	4.829	1	1,541
oferta 5	4.822	5	4,189	4.951	5	1,352	4.918	6	0,911	4.866	5	9,137	4.719	3	4,778

Leyenda	
NR	Número de restricciones
DNC	Demanda no cubierta en su totalidad
TE	Tiempo de ejecución

Promedio	
NR	4.862
DNC	4,56
TE	5,774

Figura 22. Tabla de Resultados de Ejecución del Nivel 2

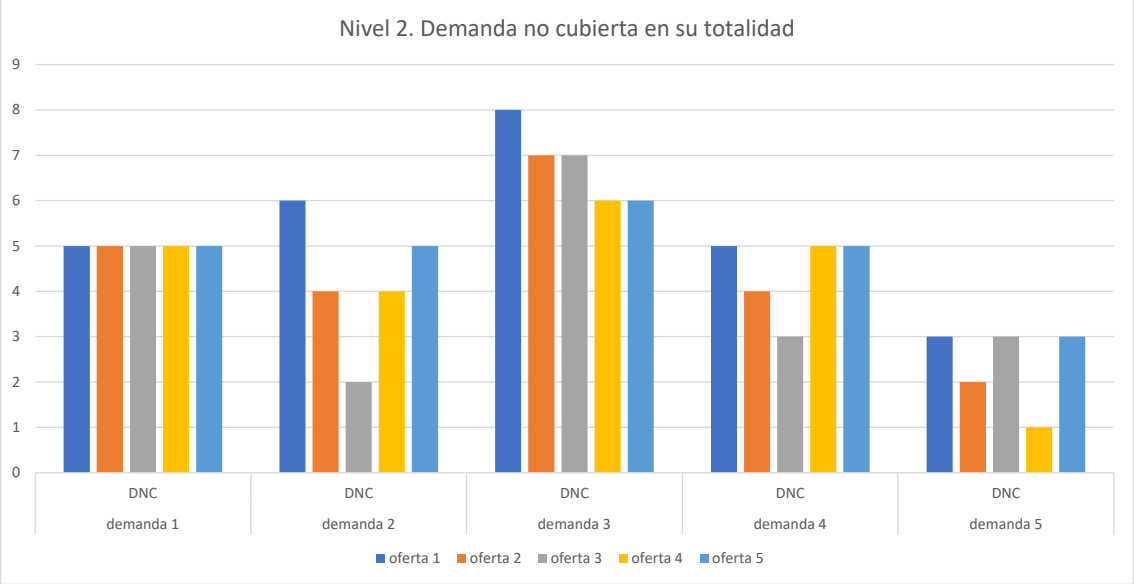

Figura 23. Gráfico de columnas agrupadas de Nivel 2. Demanda No Cubierta en su Totalidad

La figura 23 muestra un gráfico de barras para poder comparar los tiempos de ejecución obtenidos, el tiempo máximo fue de la oferta 4 con demanda 1 siendo de 35 segundos, superando por 15 segundos al segundo tiempo obtenido más grande por la oferta 4 y demanda 4. Analizando los demás tiempos obtenidos en la oferta 4 así como la cantidad de demandas no cubiertas en su totalidad mostradas en la figura 24, esta disparidad de valores obtenidos puede originarse por una similitud en los recursos ofertados y demandados, haciendo que las diferentes posibilidades para resolver el problema sean muchas, por lo que al solver le toma más tiempo realizar las asignaciones optimas como se explicó anteriormente.

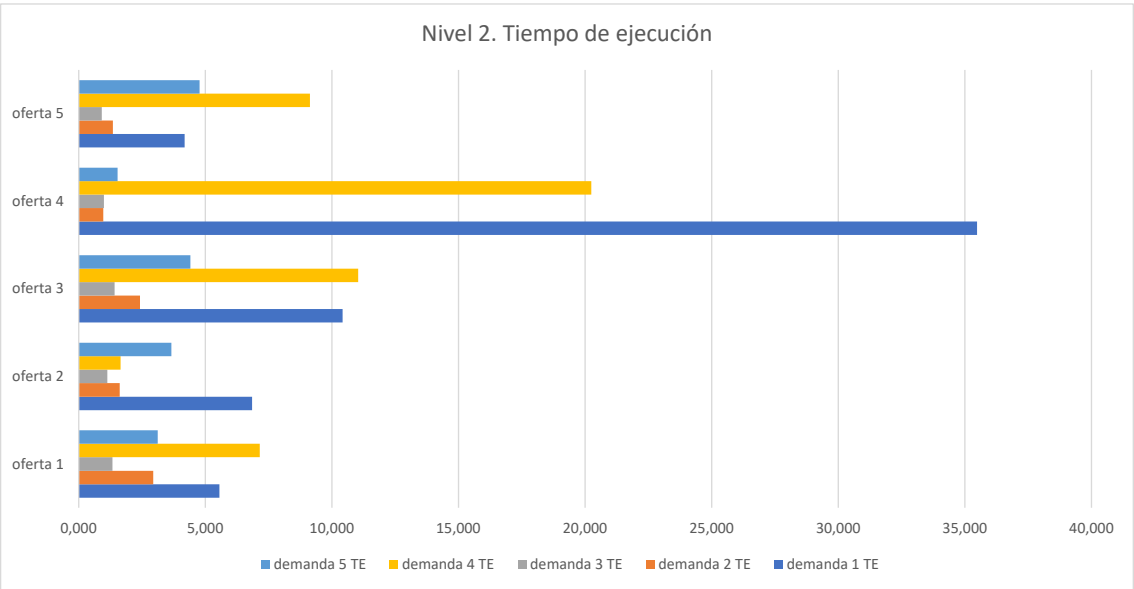

Figura 24. Gráfico de barras agrupadas de Nivel 2. Tiempo de Ejecución

5.3.3 Resultados de ejecución de Nivel 3, 600 recursos – 60 demandas

Como resultado de las ejecuciones del Nivel 3 se obtiene un promedio de 10.649 restricciones, 2,36 demandas no cubiertas en su totalidad y 145,881 segundos de ejecución como se observa en la figura 25.

	NIVEL 3														
	demanda 1			demanda 2			demanda 3			demanda 4			demanda 5		
	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE	NR	DNC	TE
oferta 1	10.599	5	19,348	10.915	0	216,512	10.047	1	14,534	10.683	2	91,519	11.001	2	257,910
oferta 2	10.616	4	402,947	11.017	0	70,963	10.016	1	13,675	10.734	3	70,400	11.156	1	81,458
oferta 3	10.593	8	43,772	10.909	1	122,723	9.781	6	32,599	10.713	3	201,411	11.279	1	41,882
oferta 4	10.574	4	8,177	10.808	1	288,567	9.934	2	8,029	10.654	4	143,175	10.978	3	172,695
oferta 5	10.754	3	314,826	10.811	0	283,070	10.102	1	153,256	10.585	2	473,906	10.960	1	119,679

Leyenda	
NR	Número de restricciones
DNC	Demanda no cubierta en su totalidad
TE	Tiempo de ejecución

Promedio	
NR	10.649
DNC	2,36
TE	145,881

Figura 25. Tabla de Resultados de Ejecución del Nivel 3

El tiempo de ejecución promedio ha aumentado significativamente respecto al de Nivel 2, debido a que la dificultad de resolución del modelo es mucho mayor. En el peor de los escenarios se podrían tener hasta 37.800 variables entre las variables binarias, enteras y de slack, lo que genera que las combinaciones de valores que debe evaluar el solver para la obtención de la solución óptima sea demasiado grande.

En la figura 26 se observa como la oferta 1, 2 y 5 satisfacen al 100% la demanda 2, mientras que en el resto de ejecuciones no se obtiene una satisfacción del 100%. Esto permite demostrar que incluso para una gran cantidad de datos que han sido generados aleatoriamente, el modelo realizado cumple su funcionalidad satisfaciendo completamente la demanda cuando es posible, mientras que en otros casos esto puede no ocurrir debido a la falta de recursos.

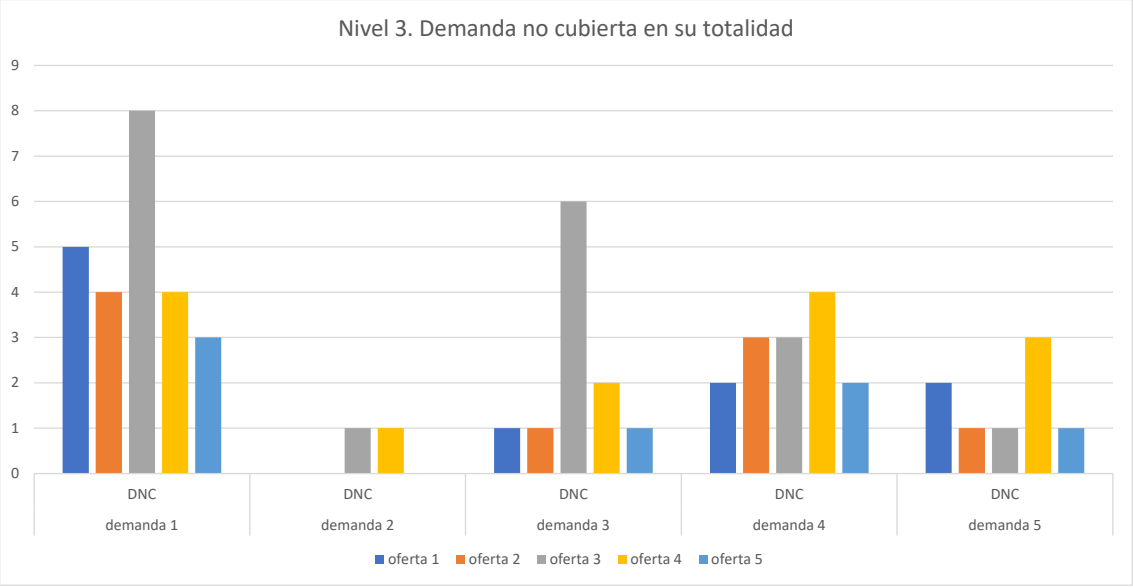

Figura 26. Gráfico de columnas agrupadas de Nivel 3. Demanda No Cubierta en su Totalidad

La figura 27 indica que los tiempos de ejecución de la oferta 2 con demanda 1 y oferta 5 con demanda 4 son superiores de forma notoria con respecto al resto, esto puede ser causado como se mencionó anteriormente, por la disparidad de tipos y/o cantidades de recursos ofertados y demandados haciendo que al modelo le tome más tiempo poder definir la asignación óptima de los recursos para cubrir la mayor cantidad de demanda.

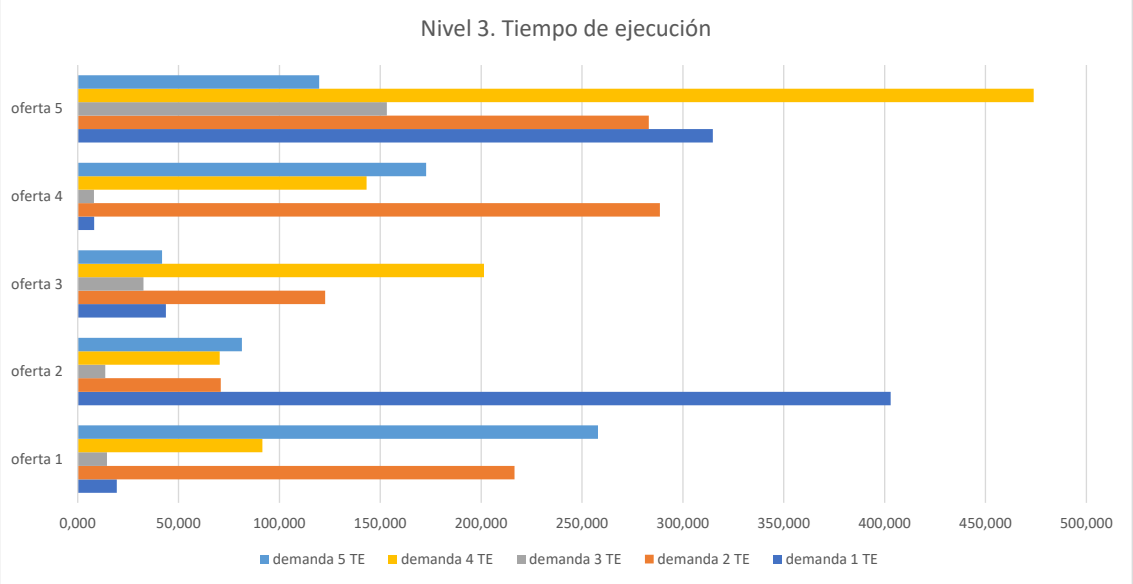

Figura 27. Gráfico de barras agrupadas de Nivel 3. Tiempo de Ejecución

Comparando los gráficos de las figuras 21, 24 y 27 se puede observar que, a pesar de que el nivel 3 posee el mayor tiempo de ejecución promedio, no significa que un problema con más restricciones conlleve a mayor tiempo de ejecución. Si se compara la ejecución de oferta 4 con demanda 3 de nivel 3 el tiempo de ejecución fue de 8,029 segundos con 9.934 restricciones, mientras que oferta 4 con demanda 1 del nivel 2 el tiempo de ejecución de 35,484 segundos con 4.858 restricciones; donde incluso en la prueba de nivel 3 se obtuvieron menos demandas no cubiertas que en la de nivel 2.

Comparando las tablas de las figuras 22 y 25 se puede observar que, en la ejecución de oferta 4 con demanda 3 de nivel 3, el tiempo de ejecución fue de 8,029 segundos con 9.934 restricciones, mientras que oferta 4 con demanda 1 de nivel 2, el tiempo de ejecución fue de 35,484 segundos con 4.858 restricciones. Sumando a esto la figura 23 y 26, donde se puede apreciar que en la prueba de nivel 3 se obtuvieron menos demandas no cubiertas que en la de nivel 2, se puede afirmar que a pesar de que el nivel 3 posee el mayor tiempo de ejecución promedio gracias a que se usan más datos, no significa que la una ejecución de un problema concreto con más restricciones conlleve a un mayor tiempo de ejecución, el tiempo de ejecución se verá afectado es por el número de combinaciones posibles que se pueden estudiar para obtener la solución óptima.

5.4 Alcance máximo

Para conocer cuál sería el alcance del modelo en la herramienta de Google Colab se ha generado un archivo oferta y demanda de 35000 y 3500 datos respectivamente con los archivos *generador_oferta2.py* y *generador_demanda2.py* y se ha lanzado a ejecución.

Google Colab solo cuenta con 12.7GB de memoria RAM en su plan de uso gratuito y la resolución de este caso consume más memoria RAM de la que se tiene, por lo que sería necesario cambiar a un plan de pago que posea mayor capacidad de memoria o ejecutar el modelo de forma local en un ordenador. Sin embargo, se obtienen más de tres millones de variables y de restricciones generadas como alcance máximo, como lo indica la figura 28.

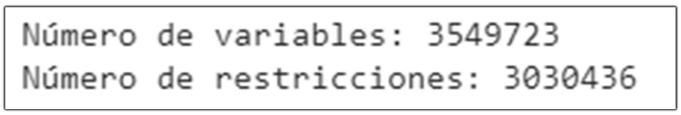

Figura 28. Resultado de ejecución de alcance máximo

6. Conclusiones

Una vez extraídos y analizados los resultados, se pueden definir las conclusiones. En este capítulo se contempla la finalización de este trabajo donde se indican todas las conclusiones obtenidas, se detalla la relación de éste con los estudios cursados en el Grado de Ingeniería Informática y finalmente se presentan los trabajos a futuro y mejoras considerados para mejorar este modelo, su empleabilidad y alcance.

6.1 Conclusiones obtenidas

El principal fracaso en el proceso transición y transformación de una ciudad hacia una smart city viene generado por los problemas de implementación y planificación. Hay que tener en cuenta aspectos técnicos, así como comunicacionales, geográficos, políticos, gerenciales e incluso sociales. Para cada caso hay que elaborar una planificación estratégica distinta de acuerdo con lo que se quiere realizar.

Para la realización de un buen modelo de optimización, primero es necesario conocer y comprender a fondo la problemática a resolver, investigar cuáles pueden ser las variables del modelo, qué se desea optimizar y que restricciones se deben diseñar para conseguir el correcto funcionamiento.

El lenguaje de programación Python permite el fácil uso de estructuras de datos como diccionarios clave–valor los cuales pueden ser esenciales para el desarrollo de un modelo de optimización.

Una de las dificultadas enfrentadas en este proyecto fue la creación de todo el modelo en sí, debido a como se debían realizar los diversos recorridos para la creación de variables, función objetivo y restricciones, ya que según fuese el caso un recurso podía generar más variables que otros, además de que las demandas podían no encontrarse originalmente ordenadas, por lo que no era posible segmentarlas.

Para un mayor número de variables y restricciones que pueda tener un modelo, su tiempo de resolución tiende a ser mayor, aunque puede haber casos en el que esto no sea así. El tiempo de ejecución también se ve afectado por las distintas posibilidades de resolución que puede tener el problema, así como los recorridos empleados en las definiciones de restricciones y de la función objetivo.

El entorno Google Colab es muy sencillo de utilizar y posee buenas prestaciones para la resolución de modelos de optimización que no requieran una extrema capacidad de cómputo. En caso de necesitar una capacidad de cómputo superior, es recomendable considerar otras herramientas para el desarrollo del modelo o pagar un plan de suscripción de Google Colab que posea las prestaciones requeridas.

6.2 Relación con los estudios cursados

Los estudios cursados en el Grado de Ingeniería Informática han sido de gran utilidad para la realización de este trabajo, en primer lugar, gracias a la evolución del pensamiento, ha

enriquecido la visión y capacidad de abstracción para encontrar soluciones y ha facilitado las herramientas necesarias para la resolución de problemas.

Ha sido de gran importancia la asignatura de Calidad y Optimización impartida en la rama de Sistemas de Información gracias a los conocimientos adquiridos en el lenguaje de Python y el desarrollo de modelos de optimización, al igual que ha sido determinante el aprendizaje en programación por las asignaturas Introducción a la Informática y Programación, Programación, Fundamentos de Sistemas Operativos y Sistemas Inteligentes.

En general el Grado cursado me ha otorgado todas las herramientas necesarias junto a la capacidad de abstracción para la resolución de los diferentes problemas y dificultades que se han ido presentando para la realización de este trabajo, sino para desempeñarme en mi carrera laboral.

6.3 Trabajos a futuro y mejoras

En esta versión se han conseguido todos los objetivos planteados inicialmente, sin embargo, enfocado en la mejora continua, escalabilidad, aplicabilidad y adaptabilidad del algoritmo, se pueden hacer mejorar para volverlo más eficiente, completo o incluso más específico, por lo que se proponen las siguientes mejoras:

1. Implantación del código en máquina local para resolver problemas de magnitudes mayores respecto a la cantidad de variables y restricciones. Sería necesario realizar modificaciones en las restricciones a nivel de estructura con el paso de parámetros para evitar problemas de compilación. El principal beneficio que proporciona es poder hacer ejecuciones del modelo a nivel local con un mayor poder de cómputo y disminución de velocidad en el tiempo de respuesta.

2. Desarrollo de una aplicación de escritorio o web para que el usuario cuente con una interfaz amigable, fácil de utilizar y comprender. De esta forma se extiende la oferta del servicio a toda persona, ente o institución que esté interesada en hacer uso del modelo, con una mejor interfaz gráfica.

3. Facilitar la introducción de parámetros permitiendo la lectura de archivos Excel (.xlsx), SQL (.sql), archivos de texto (.txt), entre otros. Para realizar este cambio se tendría que modificar la función *ingresar_datos* la cual permita la introducción de cualquier extensión de archivo mencionada anteriormente, con el fin de darle mayor alcance al modelo, permitiendo al usuario además de ingresar los datos en un formato .json a que puede hacer uso de los datos que ya tenga creados previamente en otro archivo.

4. Extensión y mejora de la clase Recurso con el fin de poder generar una jerarquía de tipos más completa que no se encuentre limitada a un Tipo y Tipo Genérico, sino que pueda ser una jerarquía de N niveles. Permite que si un recurso posee una especialización de empleabilidad muy específica no se vea limitado por ésta, sino que pueda tener más usos gracias a su jerarquía de niveles por el que está compuesto. Para esto habría que modificar el código para que se puedan ingresar todas las jerarquías existentes e incluso se pudiera eliminar de los parámetros de entrada de datos el atributo Tipo Genérico ya que con solo el Tipo modificando las restricciones de satisfacción de demandas.

5. Diseño de una restricción que favorezca la asignación de un recurso de ubicación A a su misma ubicación en caso de ser posible, para disminuir el número de traslados. Para ello habría que crear una nueva función que genere esta restricción la cual para un recurso de ubicación A se observen las posibles ubicaciones a las que pueda asistir y favorezca su coste de uso en la asignación, por ejemplo, en su totalidad a una demanda de la misma ubicación A. Esto promueve que los recursos se trasladen lo menos posible, reduciendo el tiempo de traslados y haciendo más cómoda la distribución, además al reducir los traslados se disminuye la afluencia en el uso de vías públicas, se minimiza el uso de transporte, ayudando también a mejorar el congestionamiento vehicular y disminuyendo la generación de agentes contaminantes para el medio ambiente, entre otros.

6. Modificar el modelo para que se puedan definir demandas por turnos de trabajo en un día, semana o mes, así como por bloques de horas, definiendo jornadas matutinas, vespertinas y nocturnas. El beneficio de este cambio es poder darle al modelo mayor empleabilidad, permitiendo ajustarse a escenarios más específicos o limitados que se puedan tener según las necesidades del usuario.

7. Permitir que una demanda de ubicación A y tipo B pueda solicitar una cantidad de recursos por X tiempo y otra cantidad por Z tiempo. Para esto habría que realizar modificaciones en las funciones de generación de variables, función objetivo y restricciones cambiando los recorridos de datos. Esto le otorgará más flexibilidad al usuario a la hora de definir las demandas que posee y como quiere que sean cubiertas.

8. Desarrollo de una función que genere un archivo de salida con todas las asignaciones de recursos ordenado por ubicación con los respectivos identificadores, uso/empleabilidad, tiempo de asignación y costes de cada recurso empleado. Así el usuario podrá conocer más a detalle el resultado generado, realizar informes si lo desea de estructuras de coste, compartir los resultados obtenidos con otras personas interesadas y mucho más.

7. Referencias

[1] Colaboradores de Wikipedia. (2023). Ciudad inteligente. Wikipedia, la enciclopedia libre. Recuperado 20 de junio de 2023, de https://es.wikipedia.org/wiki/Ciudad_inteligente#:~:text=En%20Comunidad%20Digital%2C%20Enrique%20Ruz,IBM%20bautizar%C3%ADa%20como%20Smart%20City

[2] Ventajas y desventajas de las Smart Cities | BBVA Suiza. (2021, 21 octubre). BBVA.CH. Recuperado 20 de junio de 2023, de https://www.bbva.ch/noticia/ventajas-y-desventajas-de-las-smart-cities/

[3] Caparrini, F. S. (s. f.). *Problemas de Búsqueda y Planificación - Fernando Sancho Caparrini*. Recuperado 20 de junio de 2023, de http://www.cs.us.es/~fsancho/?e=33

[4] Brito, J. Ramírez, M. Izquierdo, P. *Heurística* [Instituto Universitario Politécnico] Heurística. Recuperado 20 de junio de 2023, de https://bibliotecaiztapalapauin.files.wordpress.com/2018/07/heurc3adstica.pdf

[5] Tutoriales, G. (2016). Qué es la Programación Entera. *Gestión de Operaciones*. https://www.gestiondeoperaciones.net/programacion-entera/que-es-la-programacion-entera/

[6] Caparrini, F. S. (s. f.). *Problemas de Satisfacción de Restricciones - Fernando Sancho Caparrini*. Recuperado 20 de junio de 2023, de http://www.cs.us.es/~fsancho/?e=141

[7] Flores, F. (2023, 13 abril). Qué es Visual Studio Code y qué ventajas ofrece. *OpenWebinars.net*. Recuperado 20 de junio de 2023, de https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/

[8] *PyCharm: el IDE de Python para desarrolladores profesionales, por JetBrains*. (2021, 2 junio). JetBrains. Recuperado 20 de junio de 2023, de https://www.jetbrains.com/es-es/pycharm/

[9] *IntelliJ IDEA – the Leading Java and Kotlin IDE*. (2021b, junio 1). JetBrains. Recuperado 20 de junio de 2023, de https://www.jetbrains.com/idea/

[10] *Eclipse IDE: Principales caraterísticas*. (2021, 6 enero). EducaciónIT. Recuperado 20 de junio de 2023, de https://blog.educacionit.com/2014/01/16/eclipse-ide-principales-carateristicas/

[11] *Sublime Text - the sophisticated text editor for code, markup and prose*. (s. f.). Recuperado 20 de junio de 2023, de https://www.sublimetext.com/

[12] *IDE Atom: ¿Qué es y cuáles son sus Características?* (s. f.). Recuperado 20 de junio de 2023, de https://open-bootcamp.com/aprender-programar/tipos-de-ide-atom

[13] Equipo editorial de IONOS. (2023). Vim, un editor de textos basado en Linux con una amplia funcionalidad. *IONOS Digital Guide*. Recuperado 20 de junio de 2023, de https://www.ionos.es/digitalguide/servidores/herramientas/editores-linux-como-editar-codigo-con-vim/

[14] EcuRed. (s. f.). *Emacs - EcuRed*. Recuperado 20 de junio de 2023, de https://www.ecured.cu/Emacs

[15] *Project Jupyter*. (s. f.). Home. Recuperado 20 de junio de 2023, de https://jupyter.org/

[16] *Hello World - GitHub Docs*. (s. f.). GitHub Docs. Recuperado 20 de junio de 2023, de https://docs.github.com/en/get-started/quickstart/hello-world

[17] *Google Colaboratory*. (s. f.). Recuperado 20 de junio de 2023, de https://colab.research.google.com/?hl=es

[18] *Notebooks en Microsoft - Visual Studio*. (2021, 21 diciembre). Visual Studio. Recuperado 20 de junio de 2023, de https://visualstudio.microsoft.com/es/vs/features/notebooks-at-microsoft/

[19] *OR-Tools | Google for Developers*. (s. f.). Google for Developers. Recuperado 20 de junio de 2023, de https://developers.google.com/optimization?hl=es-419

[20] *Gurobi Optimizer - Gurobi Optimization*. (2022, 5 noviembre). Gurobi Optimization. Recuperado 20 de junio de 2023, de https://www.gurobi.com/solutions/gurobi-optimizer/

[21] *IBM Documentation*. (s. f.-b). Recuperado 20 de junio de 2023, de https://www.ibm.com/docs/es/icos/12.8.0.0?topic=mc-what-is-cplex

[22] *SCIP*. (s. f.). Recuperado 20 de junio de 2023, de https://www.scipopt.org/

[23] *GLPK - GNU Project - Free Software Foundation (FSF)*. (s. f.). Recuperado 20 de junio de 2023, de https://www.gnu.org/software/glpk/

[24] *Sudoku, Linear Optimization, and the Ten Cent Diet*. (2014, 30 septiembre). Recuperado 20 de junio de 2023, de https://ai.googleblog.com/2014/09/sudoku-linear-optimization-and-ten-cent.html

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**		X		
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**	X			
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.	X			
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**	X			

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Desde el momento que decidí el área donde quería desarrollar mi TFG tenía claro en mi mente que quería trabajar en un proyecto de amplio alcance, utilizable en todo ámbito y por cualquier ente, replicable, que estuviese fundamentado en lograr eficiencia y que su objetivo final impactara positivamente el bienestar social y la calidad de vida de la población.

El TFG realizado tiene estrecha relación con los Objetivos de Desarrollo Sostenible (ODS) de salud y bienestar, trabajo decente y crecimiento económico, ciudades y comunidades sostenibles, alianzas para lograr objetivos, y en menor medida con otros objetivos ya que trata del desarrollo de una herramienta para la asignación eficiente de recursos dentro del ámbito de smart cities, pero que a la vez, su versatilidad le permite ser empleado en distintos fines y por múltiples entes, tanto por personas, empresas privadas, instituciones públicas, empresas de prestación de bienes y servicios, entre otros.

El objetivo de este trabajo fue desarrollar un algoritmo que ayudara a satisfacer las necesidades de la población (demandas) mediante los recursos disponibles (ofertas), minimizando el tiempo de respuesta y promoviendo el ahorro del gasto monetario, con el propósito de hacer un uso eficiente de los recursos y del dinero. Producto del ahorro en el gasto monetario, se persigue promover su utilización en proyectos que impacten positivamente el bienestar social de la población, bien sea a través de la inversión en material educativo, nuevas escuelas y centros de enseñanza, planes para combatir la pobreza, desarrollo de nuevas tecnologías, mejoras en la movilización reduciendo así la emisión de agentes contaminantes al medio ambiente, nuevos centros de asistencia social y de salud, desarrollo de áreas verdes, mejoramiento en la calidad de los servicios públicos, reducción de impuestos municipales, mejora de salarios y beneficios y muchas más ideas que refuercen la transformación del pensamiento gubernamental, empresarial, ciudadano e individual a una cultura de desarrollo sostenible y sustentable.

Aunque existan ciertos objetivos que en este momento no tienen relación directa con este trabajo, no significa que no se puedan tomar en cuenta, pues una de las características fundamentales de una smart city es que su filosofía va de la mano con la energía renovable y el saneamiento del agua, por lo que si una comunidad autónoma, en este caso Valencia, decide aplicar este algoritmo para disminuir su gasto público e invertir el dinero sobrante en proyectos para el saneamiento de agua, se tendría una relación directa con estos objetivos. Igualmente ocurre con los objetivos hambre cero o energía asequible y no contaminante, pues se pueden desarrollar proyectos relacionados utilizando el excedente monetario, aplicando el mismo modelo de algoritmo desarrollado para la asignación de los recursos disponibles y distribuirlos de forma eficiente.