Universidad de Concepción
Departamento de Ingeniería Informática y Ciencias de la Computación

Diseño, Desarrollo e Implementación de un Sistema de Recomendación de Despacho de Recursos para el Combate de Incendios Forestales

Vicente Cser Muñoz

Memoria presentada para la obtención del título de Ingeniero Civil Informático

Patrocinante: Prof. Marcela Varas C
Comisión 1: Prof. Ricardo Flores H.
Comisión 2: Prof. Pedro Pinacho D.

Julio de 2025

Índice

1. Introducción

1.1 Antecedentes Generales del Problema

En Chile, los incendios forestales representan una amenaza creciente para los recursos naturales y la seguridad de las comunidades, un problema agravado por el cambio climático. Para una empresa como CMPC, cuya operación depende de plantaciones forestales, estos eventos constituyen un riesgo crítico tanto económico como ambiental. Episodios devastadores, como los de 2017 y 2023, han puesto en evidencia la vulnerabilidad de estas plantaciones, subrayando la necesidad de una gestión de emergencias robusta.

En este contexto, la Central de Incendios de CMPC desempeña un papel clave, y uno de sus desafíos más críticos es el despacho eficiente de recursos. Actualmente, este proceso se basa en gran medida en la experiencia acumulada de los despachadores, un enfoque que, si bien funcional, presenta limitaciones: inconsistencias derivadas de la subjetividad, falta de optimización en la asignación de recursos y una carga excesiva para despachadores inexpertos.

En un esfuerzo previo por abordar estas limitaciones, un equipo de la Universidad de Concepción creó un primer prototipo de un solver para la recomendación de despachos [7], que es el núcleo algorítmico utilizado en la presente memoria. Sin embargo, el desarrollo de dicho modelo enfrentó en su momento significativas complicaciones que definieron su alcance. En un inicio, el levantamiento de la lógica de negocio para alimentar el modelo fue un proceso complejo, que requirió cuantificar variables operativas clave como:

- La efectividad de combate y los costos de cada recurso.
- La estimación de la propagación del fuego y los tiempos de movilización.
- Las estrategias frente a múltiples focos y los protocolos de detección.

Las dificultades en el acceso a datos consolidados y un tiempo de desarrollo limitado llevaron a que el solver original se construyera sobre heurísticas que constituían aproximaciones funcionales de la realidad operativa. Este prototipo, aunque valioso a nivel algorítmico, no llegó a integrarse con los sistemas de CMPC, permaneciendo como un componente tecnológico aislado.

El presente proyecto de título retoma este desarrollo y se enfoca precisamente en superar el desafío que quedó pendiente: la integración de este solver en un sistema vivo y funcional. Por lo tanto, esta memoria no se centra en rediseñar las heurísticas internas del modelo, sino en construir la arquitectura tecnológica (API, interfaz de usuario, bases de datos en tiempo real) que permite que las recomendaciones del prototipo original puedan ser utilizadas, validadas y, eventualmente, mejoradas en un entorno operativo real. La solución propuesta es el puente que conecta un desarrollo académico previo con una necesidad industrial presente, sentando las bases para una gestión de emergencias basada en datos.

1.2 Solución Propuesta y Alcances

La solución propuesta consiste en un sistema diseñado para agilizar el despacho de recursos en la gestión de incendios forestales, respondiendo directamente a las necesidades operativas de la Central de Incendios de CMPC, una empresa papelera cuya sostenibilidad depende de la protección eficiente de sus plantaciones forestales.

Este sistema no solo busca agilizar los procesos operativos, reduciendo los tiempos de respuesta frente a emergencias, sino también aumentar la precisión y efectividad en la asignación de recursos en un contexto donde la rapidez, la adaptabilidad y la toma de decisiones basadas en datos son cruciales para minimizar pérdidas económicas y ecológicas.

Para responder a esta necesidad, la solución se estructura en tres componentes principales que operan de manera sinérgica. Estos combinan tecnologías modernas y enfoques computacionales avanzados con los sistemas existentes de CMPC, fortaleciendo la capacidad de combate frente a los incendios forestales y manteniéndose dentro de los alcances definidos para esta memoria de título.

Componentes del Sistema

1. **Servidor web con REST API:** El núcleo de comunicación del sistema es un servidor web desarrollado en Python con el framework FastAPI [4], elegido por su alto rendimiento, soporte nativo para operaciones asíncronas y facilidad para construir APIs REST robustas, escalables y bien documentadas. Este componente actúa como una capa de integración, recibiendo datos en tiempo real desde fuentes internas de CMPC (como bases de datos de recursos y sensores meteorológicos) y externas, procesándolos y distribuyéndolos entre los módulos del sistema. Su arquitectura modular y basada en estándares REST permite una interacción fluida entre la interfaz gráfica y el modelo de recomendación, y además facilita futuras integraciones con sistemas corporativos adicionales o herramientas de terceros. Por otra parte, FastAPI asegura una baja latencia en la comunicación, un factor crítico en escenarios de emergencia donde cada segundo cuenta y su diseño reduce la carga operativa en el servidor, garantizando estabilidad incluso bajo alta concurrencia de solicitudes.

2. **Interfaz gráfica:** La interfaz de usuario de la aplicación fue desarrollada con React [5], que es una biblioteca de JavaScript que permite crear interfaces de usuario para aplicaciones web y móviles. Esta incluye un mapa interactivo basado en la librería Leaflet [3], un formulario de ingreso de datos y una sección para visualizar las recomendaciones generadas por el sistema. Los despachadores pueden ingresar coordenadas geográficas manualmente o a través de clics en el mapa, lo que simplifica la entrada de datos en situaciones críticas. La elección de React garantiza una interfaz rápida y reactiva, capaz de actualizarse dinámicamente sin recargar la página, mientras que Leaflet aporta herramientas avanzadas de visualización geoespacial, como capas

personalizables y zoom de alta precisión, esenciales para priorizar la protección de plantaciones clave o infraestructura crítica de CMPC. Esta combinación mejora la usabilidad y reduce la curva de aprendizaje, incluso para despachadores con poca experiencia.

3. **Modelo de recomendación:** El núcleo analítico del sistema es un modelo de optimización basado en la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) [1], implementado en C++ para aprovechar su eficiencia computacional y capacidad para manejar cálculos intensivos en grandes volúmenes de datos. Este modelo procesa variables en tiempo real como la ubicación y severidad de los incendios, disponibilidad y capacidad de los recursos, condiciones meteorológicas (viento, humedad y temperatura) y prioridades estratégicas de CMPC, como la protección de plantaciones maduras o zonas cercanas a comunidades. Esta información es importante para la generación de asignaciones de los distintos recursos de combate. GRASP destaca por su capacidad de encontrar soluciones de alta calidad en tiempos reducidos, combinando una fase constructiva greedy con búsqueda local adaptativa, lo que lo hace ideal para entornos dinámicos donde las condiciones cambian rápidamente. La implementación en C++ asegura un rendimiento excepcional, entregando recomendaciones en fracciones de segundo incluso en escenarios con múltiples focos de incendio simultáneos, lo cual es un desafío frecuente para CMPC durante las temporadas críticas. Este enfoque supera las limitaciones de los métodos manuales, proporcionando consistencia y precisión sin depender exclusivamente de la experiencia humana.

Integración y Objetivos

El sistema se integra con los datos existentes de CMPC, que incluyen modelos de combustible, pendientes del terreno e información sobre recursos disponibles (ubicación de brigadas, equipos y vehículos). Esta integración asegura que las recomendaciones generadas por el sistema sean contextualizadas y relevantes, aprovechando al máximo la infraestructura de datos de la empresa. Los principales objetivos del sistema son:

- **Reducir tiempos de respuesta:** Al automatizar el análisis y la asignación de recursos, se minimizan los retrasos asociados con procesos manuales o herramientas obsoletas como las tablas de decisión simples usadas anteriormente.
- **Mejorar la precisión:** El uso de un modelo avanzado y datos en tiempo real permite asignaciones más efectivas, mejorando el uso de recursos y reduciendo el riesgo de errores humanos.
- **Apoyar a despachadores, especialmente los menos experimentados:** La interfaz intuitiva y las recomendaciones claras disminuyen la dependencia de conocimientos previos, haciendo que el sistema sea accesible tanto para usuarios novatos como para expertos.

Alcances del Proyecto

El proyecto abarca el diseño, desarrollo e implementación de un sistema funcional para el análisis y recomendación de recursos contra emergencias forestales. Esto incluye la construcción de los componentes técnicos esenciales, su integración con los sistemas existentes de CMPC, y la evaluación de su desempeño para operar en la central de incendios. El resultado final es una solución funcional y lista para su despliegue.

Un componente central de este sistema es el solver para generar recomendaciones de despacho. Este solver fue diseñado e implementado previamente por el equipo de la Universidad de Concepción en el marco del proyecto GESFIRE [9, 10] para CMPC en 2021 y se basa en la metaheurística GRASP. En el presente proyecto, se reutiliza su base teórica sin modificaciones sustanciales, pero se realiza una adaptación de sus componentes de entrada y salida. Esta adaptación permite la compatibilidad con el stack tecnológico actual, facilitando el uso de archivos JSON como interfaz de comunicación y asegurando una integración fluida con los demás módulos del sistema.

Además de la construcción tecnológica, el alcance del proyecto comprende la recopilación inicial de requisitos para definir la funcionalidad del sistema. Una vez desarrollado, el sistema fue puesto a prueba con los usuarios para verificar su correcto funcionamiento y la adecuación a las necesidades operativas identificadas. Para facilitar la transición hacia su uso, se realizó una demostración práctica del funcionamiento de la plataforma web con los usuarios. Este enfoque no solo busca superar las limitaciones de soluciones anteriores, sino que también sienta las bases para un nuevo estándar tecnológico en la gestión de incendios forestales en CMPC, con proyecciones para futuras mejoras y adaptaciones.

1.3 Objetivo General

Desarrollar un sistema integrado que optimice el despacho de recursos para combatir incendios forestales en la central de incendios de CMPC, combinando un modelo de recomendación, una REST API eficiente y una interfaz gráfica intuitiva, para asistir en la toma de decisiones en tiempo real, superar las limitaciones de métodos estáticos previos y mejorar la eficiencia operativa en términos de rapidez y precisión.

1.4 Objetivos Específicos

1. Integración de solver diseñado por Pinacho et al. [7]
2. Diseño e implementación de una interfaz gráfica intuitiva para ingresar coordenadas y timestamps, y visualizar recomendaciones.
3. Desarrollo de una REST API eficiente en Python para conectar la interfaz con el modelo de recomendación.

4. Integración del sistema con bases de datos de CMPC, asegurando coherencia y disponibilidad de datos.
5. Realización pruebas de rendimiento y usabilidad para garantizar robustez en escenarios reales.
6. Implementación de medidas de seguridad para proteger datos sensibles.
7. Entrega de un sistema funcional y documentado.

1.5 Metodología

El desarrollo del sistema se articuló en torno a un marco metodológico ágil, adaptado para un trabajo individual con un plazo acotado. Este enfoque se estructuró sobre tres pilares fundamentales: un sistema de gestión de tareas basado en Kanban, una práctica de desarrollo guiado por pruebas (TDD) para asegurar la calidad, y un proceso de trabajo dividido en fases lógicas. A continuación, se detalla cada uno de estos componentes.

Gestión del Proyecto: Kanban Personal con Historias de Usuario

Para la organización del trabajo se implementó una metodología de "Kanban personal con historias de usuario" (ver Anexo B). Se utilizó un tablero Kanban para visualizar y gestionar el flujo de tareas, las cuales se estructuraron como historias de usuario y se clasificaron en columnas de estado: "Por Hacer", "En Progreso" y "Completado". Este enfoque visual permitió una gestión flexible y un flujo de trabajo continuo, facilitando la priorización de funcionalidades según las necesidades emergentes del proyecto y los requerimientos de los stakeholders de CMPC.

Aseguramiento de la Calidad: Desarrollo Guiado por Pruebas (TDD)

Para garantizar la calidad y estabilidad del código, se adoptó la práctica de Desarrollo Guiado por Pruebas (TDD). Este enfoque consiste en un ciclo de tres pasos: primero, se escribe una prueba automatizada que define el comportamiento esperado de una funcionalidad (y que inicialmente falla); segundo, se escribe el código mínimo necesario para que la prueba pase; y tercero, se refactoriza el código para optimizarlo sin alterar su funcionalidad.

Esta práctica fue crucial para minimizar la introducción de errores y, especialmente, para trabajar sobre el modelo heredado, ya que permitió realizar modificaciones y mejoras con la confianza de que no se estaban rompiendo características existentes. Para la implementación de TDD se utilizaron las herramientas **Vitest** en el frontend y **Pytest** en el backend.

Fases del Desarrollo

El proyecto se estructuró en tres fases secuenciales que ordenaron el proceso desde la concepción hasta la validación final:

1. **Fase 1: Recopilación de Requerimientos y Datos** En esta etapa inicial se realizaron entrevistas y talleres con los stakeholders de CMPC para identificar las necesidades operativas y los requisitos técnicos del sistema. Se recopilaron datos geográficos clave (mapas de áreas forestales, modelos de combustible) y se analizó el flujo de trabajo de los despachadores para alinear el desarrollo con las realidades del terreno.
2. **Fase 2: Desarrollo e Integración Modular** Esta fase se centró en la construcción de los componentes del sistema. Se trabajó en módulos específicos (interfaz, servidor, modelo) de manera iterativa. Una tarea técnica clave dentro de esta fase fue la adaptación del modelo de recomendación legado. Originalmente, este no estaba preparado para comunicarse con una API, por lo que se modificó su lógica de entrada/salida para que pudiera procesar y generar datos en formato JSON, el estándar de comunicación de las APIs REST, garantizando una integración fluida con el backend.
3. **Fase 3: Pruebas y Validación con Usuarios** La última fase se dedicó a la validación exhaustiva del sistema. Además de las pruebas unitarias y de integración realizadas continuamente gracias a TDD, se llevaron a cabo sesiones de validación con usuarios reales de CMPC. En escenarios simulados, los despachadores interactuaron con la herramienta, proporcionando retroalimentación valiosa que permitió ajustar la interfaz y la presentación de las recomendaciones para maximizar su usabilidad y efectividad.

Esta metodología integral proporcionó una base sólida para el desarrollo del sistema, resultando en una solución funcional que equilibra rapidez, calidad y alineación con los objetivos estratégicos de CMPC.

1.6 Estructura del Informe

El informe se estructura de manera sistemática y ordenada, con el objetivo de presentar una narrativa coherente y lógica que aborde todos los aspectos relevantes del proyecto, desde su concepción hasta sus resultados finales. Está organizado en secciones claramente definidas que permiten al lector seguir un flujo progresivo de información, facilitando la comprensión tanto del problema abordado como de la solución propuesta y su implementación.

La primera sección se centra en detallar el problema, proporcionando un contexto claro sobre los desafíos operativos y técnicos que enfrenta la central de incendios de CMPC, así como las limitaciones de los enfoques previos que motivaron la creación de este sistema. Le sigue una exposición de la solución propuesta, donde se describen los objetivos del proyecto, sus características y la manera en que busca superar las deficiencias identificadas. Posteriormente, una sección dedicada al desarrollo del sistema profundiza en las decisiones metodológicas y tecnológicas adoptadas, explicando el proceso iterativo y las herramientas empleadas para

construir una solución funcional y adaptable. A esto se suma una parte enfocada en la evaluación, donde se analizan los resultados obtenidos, los criterios utilizados para medir el desempeño del sistema y las pruebas realizadas para validar su eficacia en escenarios reales o simulados. Finalmente, el informe cierra con las conclusiones y reflexiones, resumiendo los logros alcanzados, las lecciones aprendidas y las posibles líneas de mejora o expansión futura.

2. Marco Teórico

2.1 Introducción

Este capítulo establece los fundamentos teóricos en los que se basa el diseño e implementación del sistema. Con respecto a la gestión de emergencias forestales de CMPC, la rapidez, la precisión y la adaptabilidad son críticas, por esto, es esencial justificar las elecciones metodológicas que garantizan que la solución responda eficazmente a los desafíos operativos.

A lo largo de las siguientes secciones, se analizarán en detalle los conceptos clave que dan forma al proyecto, entre ellos:

- El patrón de arquitectura de software **Modelo-Vista-Controlador (MVC)**, que organiza la estructura de la aplicación.
- La metaheurística **GRASP** (*Greedy Randomized Adaptive Search Procedure*), que constituye el núcleo del motor de recomendación.
- La **arquitectura basada en contenedores**, que define la estrategia de despliegue y escalabilidad del sistema.
- Las **tecnologías específicas** seleccionadas para el desarrollo del frontend, backend y la gestión de datos.

El objetivo es proporcionar al lector el marco de referencia necesario para comprender las decisiones de implementación que se detallarán en capítulos posteriores, conectando la teoría con la solución práctica desarrollada para la central de incendios de CMPC.

Modelo de Recomendación

La optimización del despacho de recursos en escenarios complejos, como el combate de incendios forestales, pertenece a la categoría de problemas de optimización combinatoria NP-hard. Para abordar este tipo de problemas se utilizan **metaheurísticas**. Una de las más efectivas para problemas de asignación es **GRASP** (*Greedy Randomized Adaptive Search Procedure*).

GRASP es una **metaheurística iterativa** que construye una solución desde cero en repetidas ocasiones, explorando diferentes zonas del espacio de soluciones para evitar quedar atrapado en óptimos locales de baja calidad. Cada una de estas iteraciones se compone de dos fases fundamentales:

1. **Fase Constructiva:** En esta primera etapa, se construye una solución factible de manera incremental. En cada paso de la construcción, el algoritmo evalúa los posibles elementos a añadir a la solución mediante una función greedy que mide su aporte. Sin embargo, para no caer siempre en la misma solución "obvia", GRASP introduce un componente aleatorio y adaptativo: en lugar de elegir siempre el mejor elemento, crea

una lista restringida de candidatos con los elementos más prometedores. Luego, selecciona aleatoriamente un candidato de esta lista para añadirlo a la solución. Este proceso se repite hasta que se ha construido una solución completa y factible.
2. **Fase de Búsqueda Local:** Una vez obtenida una solución en la fase constructiva, se intenta mejorarla. La fase de búsqueda local explora sistemáticamente la "vecindad" de la solución actual, realizando pequeñas modificaciones (por ejemplo, intercambiando recursos asignados) con el objetivo de encontrar una solución vecina de mayor calidad. Este proceso de refinamiento continúa hasta que no se encuentra ninguna mejora posible en el vecindario.

El algoritmo GRASP repite este ciclo de construcción y búsqueda local múltiples veces, guardando siempre la mejor solución encontrada a lo largo de todas las iteraciones. La combinación de la aleatoriedad de la fase constructiva y el refinamiento de la búsqueda local permite al algoritmo encontrar soluciones de muy alta calidad para problemas de gran complejidad en tiempos computacionales eficientes.

Estándar CMPC

En CMPC existen varios estándares relacionados a la calidad y seguridad de sus aplicaciones. Entre ellas se pueden destacar las siguientes:

- Utilizan las tecnologías de Google, específicamente Google Cloud para desplegar sus aplicaciones.
- Los proyectos deben ser aprobados previamente por el equipo de ciberseguridad de la empresa.
- Si el proyecto fue aprobado, se otorgan los permisos de Google Cloud y las herramientas de desarrollo que se requiera utilizar.
- Existen herramientas que se pueden utilizar libremente, sin solicitar permisos, como todas las pertenecientes a Google Workspace.

Tecnologías

La arquitectura del sistema se basa en una selección de tecnologías modernas, escogidas para garantizar un alto rendimiento, escalabilidad y una clara separación entre las distintas responsabilidades de la aplicación. A continuación, se describen los conceptos fundamentales y, posteriormente, las herramientas específicas implementadas para cada uno.

Conceptos Tecnológicos Fundamentales

- **Frontend (Capa de Presentación):** Corresponde a toda la parte de la aplicación con la que el usuario interactúa directamente. Su responsabilidad es presentar los datos de manera clara e intuitiva y capturar las entradas del usuario (clics, formularios, etc.). En

este proyecto, es la interfaz visual que usan los despachadores para ver mapas, ingresar datos de incendios y recibir recomendaciones.

- **Backend (Capa de Lógica y Servicios):** Es el motor invisible de la aplicación. Se ejecuta en un servidor y su función es centralizar la lógica de negocio, procesar los datos, comunicarse con la base de datos y comunicarse con el núcleo de optimización (solver). Expone una **API (Interfaz de Programación de Aplicaciones)** para que el frontend pueda solicitar y enviar información de manera segura y estructurada.
- **Base de Datos NoSQL:** A diferencia de las bases de datos relacionales tradicionales (que usan tablas con filas y columnas), las bases de datos NoSQL utilizan estructuras de datos flexibles, como documentos de tipo JSON. Son ideales para aplicaciones que requieren alta escalabilidad y cuyos esquemas de datos pueden evolucionar con el tiempo.
- **Sistemas de Información Geográfica (GIS):** Se refieren a las herramientas y formatos de datos diseñados específicamente para trabajar con información geoespacial. Permiten almacenar, analizar y visualizar datos geográficos como coordenadas, límites de terrenos (polígonos) o ubicaciones de recursos (puntos).

Tecnologías Específicas Implementadas

- **React (Frontend):** Para construir la interfaz de usuario se utilizó **React**, una biblioteca de JavaScript para crear interfaces dinámicas y componentizadas. Su principal ventaja es que permite construir la aplicación como un conjunto de piezas reutilizables, facilitando el mantenimiento y la escalabilidad. Para la visualización de mapas interactivos, se integró la librería **Leaflet**, una solución ligera y potente especializada en datos geoespaciales.
- **Tailwind CSS (Estilos Frontend):** Para el diseño visual de la interfaz, se empleó **Tailwind CSS**. Es un framework de CSS de tipo "utility-first" que permite aplicar estilos de forma rápida directamente en el código.
- **FastAPI (Backend):** El backend se implementó con **FastAPI**, un framework web moderno de Python. Fue elegido por su rendimiento, su soporte nativo para operaciones asíncronas (clave para manejar múltiples solicitudes sin bloqueo) y su capacidad para generar documentación de la API de forma automática.
- **Firestore (Base de Datos NoSQL):** Como solución de base de datos se utilizó **Cloud Firestore** de Google. Es una base de datos de documentos NoSQL, flexible y altamente escalable que opera en la nube. Es elegida por su sincronización en tiempo real y su buena integración con otros servicios.
- **Shapefile (.shp) (Datos GIS):** Para manejar la información geográfica estática de CMPC, como los límites de los predios y sus características (modelos de combustible), se utilizan archivos **Shapefile (.shp)**. Este es un formato vectorial estándar en la industria de los GIS para almacenar la forma y los atributos de elementos geográficos, permitiendo que el sistema realice cálculos espaciales precisos.

Factores Ambientales

El sistema considera una serie de variables ambientales y operativas que influyen directamente en la dinámica de los incendios forestales y en las decisiones de despacho. Estas variables se integran en el modelo de recomendación para garantizar que las soluciones propuestas sean contextualmente relevantes:

- **Temperatura (°C):** Determina la velocidad de combustión, ya que temperaturas más altas facilitan la ignición y el avance del fuego.
- **Velocidad del viento (m/s):** Afecta la propagación del incendio, acelerando las condiciones de viento fuerte y complicando su contención.
- **Dirección del viento (grados):** Define la trayectoria probable del fuego, permitiendo priorizar recursos en las zonas más amenazadas.
- **Humedad relativa (%):** Influye en la combustión, ya que si los niveles se encuentran bajos aceleran el fuego y si los niveles están altos dificultan la propagación.
- **IHCFM (Índice de Hidratación y Combustión de Material Fino):** Un índice compuesto que combina variables meteorológicas para estimar el riesgo de incendio, proporcionando una medida integral del peligro potencial.
- **Valor del fundo (USD):** Permite priorizar la protección económica, asignando recursos a zonas de mayor importancia estratégica o financiera para CMPC.
- **Pendiente del terreno.**
- **Modelo de combustible del terreno.**

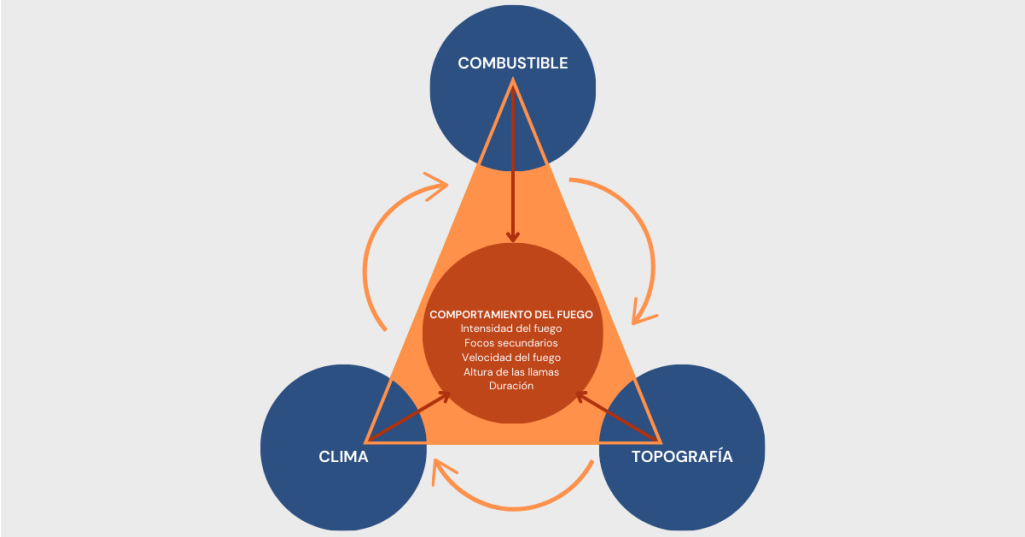


Figura 1. Comportamiento del fuego [6].

2.2 Estado del Arte en la Predicción y Combate de Incendios Forestales

La **predicción y combate de incendios forestales** es un campo crítico que ha visto avances significativos a nivel global, impulsados por la creciente frecuencia e intensidad de estos eventos. Existen sistemas importantes que cumplen funciones similares al objetivo de este proyecto, ayudando en la toma de decisiones al momento de predecir y combatir un foco de incendio.

A nivel internacional, destacan soluciones robustas como **Wildfire Analyst de Technosylva**. Este sistema es una solución SaaS en la nube que proporciona capacidades de predicción de la propagación de incendios forestales bajo demanda. Es fundamental para la respuesta operativa, el análisis de escenarios hipotéticos y la previsión del riesgo de incendios. Sus aplicaciones principales incluyen la predicción de riesgo, la simulación de propagación y la cuantificación del riesgo de cada foco activo. Otro referente importante es el sistema desarrollado por la empresa aeroespacial alemana **OroraTech**. Mediante fuentes satelitales integradas con cámaras infrarrojas, OroraTech ofrece un sistema de monitorización de puntos de calor en la tierra, complementado con un servicio de predicción del avance de estos focos.

El Contexto Chileno y la Necesidad de Soluciones Innovadoras

En Chile, sin embargo, este tipo de soluciones tecnológicas son notablemente escasas y, cuando están presentes, suelen basarse en simuladores con ciertas limitaciones. Un ejemplo es **Kitral**, una solución que, aunque valiosa para la modelización de escenarios hipotéticos, opera principalmente con datos históricos o preconfigurados. Esto restringe su capacidad para responder a condiciones en tiempo real o para generar recomendaciones dinámicas que se ajusten a la evolución inmediata de una emergencia. Además, estas herramientas locales a menudo requieren un nivel de conocimiento técnico avanzado, lo que las hace menos accesibles para usuarios sin formación especializada y no están diseñadas para lidiar con la variabilidad extrema de factores como el clima, la topografía o los recursos disponibles en el momento.

Es en este contexto donde surgen iniciativas nacionales de gran relevancia. Un proyecto chileno pionero en este ámbito es **"Cell2Fire"**, liderado por el académico Andrés Weintraub de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Este sistema, resultado de más de una década de investigación del grupo "Fire Management and Advanced Analytics" del Instituto Sistemas Complejos de Ingeniería (ISCI), busca ir más allá del mero combate de incendios, enfocándose en la prevención y la anticipación de sus impactos.

"Cell2Fire" se distingue por su capacidad de predecir dónde comenzarán los incendios con una certeza entre 85% y 90%, y anticipar su propagación en función de factores cruciales como la topografía, la vegetación, el clima y los vientos. Este enfoque se basa en modelos de

simulación avanzados y la combinación de datos históricos, imágenes satelitales, tecnología LiDAR (mediante drones) y técnicas de inteligencia artificial. Su trabajo se estructura en cuatro pilares clave: la adquisición de información, la predicción de igniciones (determinando probabilidades en celdas de 100x100 metros), la simulación de la propagación una vez declarado el incendio (analizando dirección, velocidad e intensidad), y la optimización de la ubicación de los cortafuegos. Además, este proyecto colabora activamente con entidades nacionales como CONAF, CORMA y SENAPRED, así como con especialistas internacionales de Canadá, Estados Unidos, España, Portugal e Italia, participando incluso en iniciativas europeas como Fire-Res.

La Propuesta de Valor del Presente Proyecto

A pesar de los avances representados por estas iniciativas, aún existe una brecha significativa en la disponibilidad de herramientas que sean al mismo tiempo precisas, en tiempo real y altamente usables para un público más amplio.

El presente proyecto busca llenar este vacío. No solo se centra en el procesamiento y análisis de información en tiempo real, utilizando datos actualizados para reflejar las condiciones del terreno y la meteorología, sino que también incorpora un diseño orientado a la usabilidad, con especial énfasis en apoyar a despachadores inexpertos. Esta característica es clave en contextos donde el personal disponible puede no contar con experiencia extensa, pero debe tomar decisiones críticas bajo presión.

Al proporcionar recomendaciones claras, adaptativas y basadas en la integración de múltiples variables como patrones meteorológicos, ubicación de recursos y prioridades, el sistema supera las limitaciones de las soluciones estáticas existentes. A diferencia de enfoques que dependen de datos históricos o preconfigurados, la propuesta se ajusta a la evolución inmediata de una emergencia. Además, su capacidad para reducir la dependencia de supuestos predefinidos y su flexibilidad para ajustarse a escenarios imprevistos lo posicionan como una herramienta clave, complementando y expandiendo el panorama actual. Este enfoque no solo eleva el estándar de respuesta ante emergencias, sino que también establece un precedente para el desarrollo de tecnologías adaptadas a las particularidades del contexto local, promoviendo una gestión más eficiente y accesible en un entorno operativo cada vez más complejo y demandante.

2.3 Arquitectura del Software

Para el diseño de aplicaciones modernas, se utilizan patrones y enfoques arquitectónicos que garantizan la modularidad, escalabilidad y mantenibilidad del software. A continuación, se describen dos de los conceptos fundamentales que conforman el diseño de este proyecto.

Patrón de Diseño Modelo-Vista-Controlador (MVC)

El **Modelo-Vista-Controlador (MVC)** es un patrón de arquitectura de software que separa la representación de la información de la interacción del usuario con ella. Su objetivo principal es organizar el código en tres componentes interconectados, cada uno con una responsabilidad clara, para facilitar el desarrollo y la gestión de aplicaciones complejas.

- **Modelo (Model):** Es el cerebro de la aplicación. Se encarga de gestionar los datos y la lógica de negocio. Es responsable de acceder y manipular la información (por ejemplo, consultando una base de datos) y de aplicar las reglas y cálculos definidos. El Modelo no tiene conocimiento directo sobre cómo se mostrarán los datos al usuario.
- **Vista (View):** Es la cara visible de la aplicación. Su única función es presentar los datos que le proporciona el Modelo en un formato específico para el usuario. Es, en esencia, la interfaz de usuario (UI). La Vista no contiene lógica de negocio; solo muestra información y captura las acciones del usuario (como clics o la entrada de texto).
- **Controlador (Controller):** Actúa como el intermediario entre el Modelo y la Vista. Cuando un usuario interactúa con la Vista, esta notifica al Controlador. El Controlador procesa la solicitud, se comunica con el Modelo para actualizar datos o solicitar información, y finalmente selecciona la Vista adecuada para responder al usuario.

Este patrón crea un flujo de comunicación claro: la Vista muestra los datos, el usuario actúa sobre la Vista, el Controlador gestiona esa acción, el Modelo procesa la lógica y los datos, y el ciclo se repite. La principal ventaja de esta separación es que permite que los equipos trabajen en paralelo (por ejemplo, un desarrollador en la lógica del Modelo y otro en el diseño de la Vista) y facilita la modificación de una parte sin afectar a las otras.

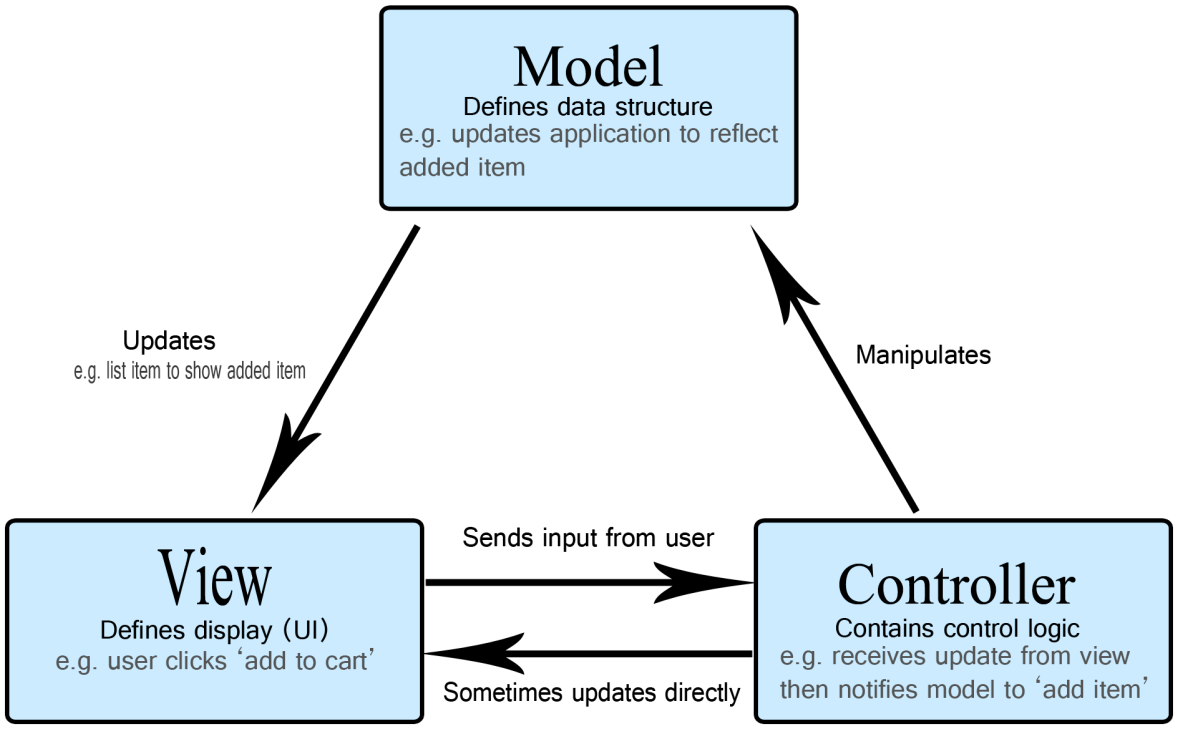


Figura 2. Diagrama de la arquitectura MVC [13].

Arquitectura Basada en Contenedores

La arquitectura de **contenedores** es un enfoque para el empaquetado y despliegue de software que permite que una aplicación se ejecute de manera fiable en diferentes entornos computacionales. Un contenedor es una unidad de software ligera, independiente y ejecutable que incluye todo lo necesario para funcionar: el código, el entorno de ejecución, las herramientas del sistema, las bibliotecas y las configuraciones.

La tecnología más popular para la gestión de contenedores es **Docker**. El proceso, conocido como **containerización**, consiste en encapsular la aplicación y sus dependencias en una "imagen" de contenedor. Esta imagen puede ser utilizada para crear instancias del contenedor en cualquier máquina que tenga un motor de contenedores instalado.

Los beneficios clave de esta arquitectura son:

- **Consistencia y Portabilidad:** Resuelve el clásico problema de "en mi máquina funciona". Al empaquetar todo junto, se garantiza que la aplicación se comporte de la misma manera en el entorno de un desarrollador, en un servidor de pruebas o en producción en la nube.
- **Aislamiento:** Cada contenedor se ejecuta en un entorno aislado, independiente de otros contenedores en la misma máquina. Esto significa que las dependencias de una aplicación no entrarán en conflicto con las de otra, mejorando la seguridad y la estabilidad.
- **Eficiencia y Escalabilidad:** Los contenedores son más ligeros que las máquinas virtuales tradicionales, ya que comparten el kernel del sistema operativo del anfitrión. Esto les permite iniciarse en segundos y consumir menos recursos, facilitando el escalado horizontal de una aplicación (es decir, ejecutar múltiples copias del mismo contenedor para manejar más carga).
- **Facilidad de Despliegue y Gestión DevOps:** Simplifica los ciclos de desarrollo y despliegue (CI/CD), ya que en lugar de configurar un servidor completo, solo se necesita ejecutar un contenedor. Este enfoque es la base de las arquitecturas de microservicios modernas.

2.4 Normativas, Estándares y Buenas Prácticas

El desarrollo del sistema se llevó a cabo siguiendo un conjunto de normativas, estándares y buenas prácticas que aseguran su calidad, interoperabilidad y seguridad, elementos esenciales para una herramienta destinada a operar en un entorno crítico como la gestión de incendios forestales de CMPC. Estas directrices no sólo garantizan que el sistema cumpla con requisitos técnicos y operativos, sino que también facilitan su mantenimiento, integración con otros sistemas y uso eficiente por parte de los despachadores. A continuación, se detallan los principales enfoques adoptados, explicando cómo se implementaron y su aporte al proyecto en términos de funcionalidad, confiabilidad y alineación con estándares reconocidos en la industria.

Coordenadas en Formato Decimal

Para garantizar la precisión y la compatibilidad en el manejo de datos geoespaciales, se adopta el formato de coordenadas en grados decimales (latitud y longitud), conforme a las especificaciones del sistema geodésico WGS84 (World Geodetic System 1984). Este estándar, ampliamente utilizado en aplicaciones de mapeo y sistemas de información geográfica (SIG), asegura que las coordenadas ingresadas por los usuarios o extraídas de las bases de datos de CMPC sean consistentes y compatibles con herramientas como Leaflet, utilizada en la interfaz gráfica. Además, facilita la interoperabilidad con sistemas externos, como servicios meteorológicos o plataformas de respuesta a emergencias, y reduce errores en la localización de incendios y recursos, aspecto crítico para la efectividad del despacho. No obstante, dado

que el personal de la central de incendios está acostumbrado al uso de coordenadas en formato de grados, minutos y segundos (DMS), la interfaz gráfica presenta las coordenadas en dicho formato. Internamente, estas se convierten a formato decimal antes de ser enviadas al backend, lo que permite mantener la compatibilidad técnica sin afectar la experiencia de los usuarios.

REST API con JSend

La API REST, desarrollada con FastAPI, sigue el estándar JSend para estructurar las respuestas enviadas al frontend y otros posibles consumidores. JSend es una convención que estandariza el formato JSON de las respuestas de una API, clasificándolas en tres tipos principales: éxito (success), fallo (fail) y error (error). Por ejemplo, una respuesta de éxito incluye un campo "status": "success" junto con los datos solicitados, mientras que un fallo indica problemas en la solicitud (como datos inválidos) y un error señala excepciones del servidor. Este enfoque mejora la claridad y predictibilidad de las interacciones entre el backend y el frontend, permitiendo a los desarrolladores y usuarios interpretar fácilmente los resultados y manejar excepciones de manera efectiva. Además, al adherirse a las mejores prácticas de diseño de APIs RESTful como el uso de métodos HTTP apropiados (GET, POST, etc.) y rutas descriptivas, se asegura una comunicación eficiente y alineada con estándares modernos de desarrollo web.

API Key para Seguridad

Para proteger el acceso a la API y garantizar que solo usuarios autorizados puedan interactuar con el sistema, se implementó un mecanismo de autenticación basado en claves API (API keys). Cada solicitud al servidor debe incluir una clave única generada para CMPC o sus despachadores, la cual es validada por FastAPI antes de procesar cualquier operación. Este enfoque de seguridad, alineado con las buenas prácticas de protección de APIs REST, previene accesos no autorizados y ataques maliciosos, como intentos de sobrecarga o extracción indebida de datos. Además, las claves se gestionan de manera segura, almacenándolas en variables de entorno y evitando su exposición en el código fuente, lo que refuerza la integridad del sistema frente a posibles vulnerabilidades. Este nivel de seguridad es particularmente relevante dado que el sistema maneja información sensible, como la ubicación de recursos y datos estratégicos de la empresa.

Componentes Funcionales en React

En el desarrollo del frontend con React, se prioriza el uso de componentes funcionales sobre componentes de clase, siguiendo las buenas prácticas actuales de esta tecnología. Los componentes funcionales, combinados con el uso de Hooks (como useState y useEffect),

ofrecen un código más limpio, legible y fácil de mantener, además de aprovechar las optimizaciones de rendimiento que React ha incorporado en sus versiones recientes. Cada elemento de la interfaz, como el mapa interactivo o los paneles de información, se diseña como un componente reutilizable, lo que facilita la escalabilidad y la consistencia visual del sistema. Además, se siguen principios de diseño modular y se aplicaron herramientas como ESLint y Prettier para mantener un estilo de código uniforme y detectar errores potenciales durante el desarrollo.

Impacto de las Normativas y Estándares

La aplicación de estas normativas, estándares y buenas prácticas asegura que el sistema no solo cumpla con los requisitos funcionales, sino que también sea seguro, interoperable y fácil de mantener. Las coordenadas estandarizadas y la API con JSend garantizan una integración fluida con los sistemas de CMPC y una comunicación efectiva entre componentes, mientras que las API keys y los componentes funcionales refuerzan la seguridad y la calidad del código.

2.5 Antecedentes y Estudios Previos

CMPC empleaba originalmente una tabla de decisión simple como herramienta para la toma de decisiones. Sin embargo, esta metodología resulta insuficiente para adaptarse a las condiciones dinámicas y cambiantes del entorno operativo y estratégico de la empresa. La tabla, aunque útil en contextos estáticos o predecibles, carece de la flexibilidad y capacidad de análisis necesarias para responder eficazmente a variables complejas, imprevistos o escenarios en constante evolución. Esta limitación evidencia la necesidad de desarrollar un sistema más robusto, capaz de integrar datos en tiempo real, evaluar múltiples factores simultáneamente y ofrecer soluciones adaptativas. Fue precisamente esta carencia la que motivó la conceptualización y diseño del sistema actual, con el objetivo de superar las restricciones del enfoque anterior y optimizar los procesos de decisión en un contexto más exigente.

2.6 Conclusión

El presente capítulo ha establecido el marco conceptual y tecnológico que sustenta el sistema propuesto. Se ha analizado el patrón de diseño Modelo-Vista-Controlador (MVC) como la estructura fundamental para la separación de responsabilidades, la metaheurística GRASP como el motor de optimización para abordar problemas complejos de asignación, y la arquitectura basada en contenedores como el enfoque para garantizar un despliegue portable y escalable.

La selección de estos elementos no es arbitraria; en conjunto, conforman una base teórica sólida y coherente para el desarrollo de una aplicación robusta. El patrón MVC ofrece la modularidad necesaria para un sistema complejo, GRASP proporciona la inteligencia requerida para la toma de decisiones dinámicas, y la containerización asegura que la solución esté preparada para un entorno de producción moderno.

Con las bases teóricas y el estado del arte establecidos, el siguiente capítulo se centrará en la Descripción de la Propuesta. En esta sección, se presentará la solución general desarrollada, delineando su arquitectura principal y las funcionalidades clave que se implementaron. El objetivo es mostrar cómo los conceptos abordados se materializan en una herramienta concreta diseñada para responder a los desafíos presentados. En capítulos posteriores se abordarán los detalles técnicos específicos de cada componente y su implementación.

3. Descripción de la Propuesta

El objetivo principal de este proyecto es diseñar, desarrollar e implementar un sistema de recomendación para optimizar el despacho de recursos en la central de incendios forestales de CMPC. Actualmente, los despachadores dependen de una matriz de decisión estática, un enfoque tradicional que no logra abordar la complejidad y variabilidad de las emergencias forestales. Esta matriz no considera factores dinámicos como las condiciones meteorológicas (viento, temperatura, humedad), características geográficas detalladas (pendiente y tipo de combustible), ni la disponibilidad de recursos. Esto lleva a decisiones inconsistentes, especialmente entre despachadores con menos experiencia, lo que resulta en retrasos y un aumento del riesgo de pérdidas económicas y operativas para CMPC, afectando también a los equipos en terreno y a las comunidades cercanas.

Para superar estas limitaciones, se propone un sistema que automatiza y personaliza la toma de decisiones, ofreciendo recomendaciones adaptadas a cada situación. Este sistema se basa en un modelo de recomendación que, dada la alta complejidad computacional del problema de asignación de recursos en emergencias, utilizará un enfoque de aproximación para identificar soluciones de alta calidad en tiempos reducidos. El modelo analizará datos en tiempo real (coordenadas de incendios, timestamps, condiciones meteorológicas y disponibilidad de recursos) para sugerir asignaciones que maximicen la eficiencia y minimicen los riesgos. Esta propuesta moderniza el proceso de despacho, lo hace más accesible y confiable, incluso para despachadores novatos, al reducir la dependencia de la experiencia subjetiva y proporcionar una herramienta objetiva guiada por datos. La solución a implementar deberá considerar la interacción de varios componentes clave.

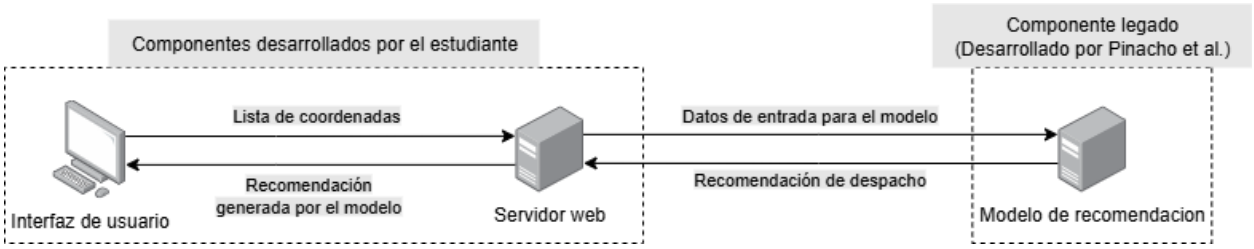


Figura 3. Diagrama MVC del sistema (elaboración propia).

El diseño de este sistema se basa en una arquitectura Modelo-Vista-Controlador (MVC), un patrón elegido por su capacidad para organizar componentes en capas diferenciadas pero interconectadas. Este enfoque garantiza modularidad, claridad en las responsabilidades y facilidad de mantenimiento, lo cual es ideal para un sistema complejo como este, enfocado en optimizar el despacho de recursos. La arquitectura MVC separa la lógica de negocio, la presentación de datos y la gestión de interacciones, asegurando que el sistema satisfaga las necesidades operativas y estratégicas de CMPC, desde el ingreso de datos críticos hasta la generación de recomendaciones precisas.

Este diseño prioriza la modularidad y la escalabilidad. Cada componente puede ser actualizado o expandido de forma independiente; por ejemplo, se podrían integrar nuevas fuentes de datos al modelo (como sensores IoT) sin impactar la vista, o mejorar la interfaz con funcionalidades adicionales sin alterar el controlador. La comunicación entre las capas sigue un flujo bidireccional: la vista envía solicitudes al controlador mediante peticiones HTTP (POST para ingreso de datos, GET para consultas), el controlador procesa estas solicitudes y consulta al modelo, y los resultados regresan a la vista en formato JSON para su renderizado. Este flujo asegura una interacción eficiente y sin fricciones, optimizada para las demandas en tiempo real de la gestión de incendios. A continuación, se detallan los componentes de esta arquitectura.

Modelo

El Modelo constituye la base lógica del sistema, gestionando la información y ejecutando los cálculos para generar recomendaciones. Esta capa encapsula las reglas de negocio y los datos, encargándose de su almacenamiento, procesamiento y administración.

Su componente central es el Solver, un modelo de recomendación desarrollado en C++. Este núcleo computacional se basa en la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) y su función es procesar variables ambientales y operativas, como temperatura, velocidad del viento, tamaño del fundo y estado de alerta para generar asignaciones de recursos. El Solver recibe datos del controlador, los analiza y envía las soluciones a la vista.

Su objetivo es determinar asignaciones que maximicen el patrimonio forestal preservado (valor económico y área protegida), mientras se minimizan los costos operativos de despliegue. En resumen, el Modelo actúa como el cerebro analítico del sistema, garantizando que las decisiones se sustenten en un procesamiento robusto y adaptado a la dinámica cambiante de los incendios forestales.

Vista

La **Vista** corresponde a la interfaz gráfica con la que los usuarios interactúan directamente. Su diseño se centró en la intuición y funcionalidad, buscando ser efectiva para despachadores con distintos niveles de experiencia en la central de incendios de CMPC. La interfaz, construida con **React** y optimizada con **Vite** para una compilación rápida, garantiza una experiencia de usuario responsiva y una actualización de la información sin demoras, lo cual es crítico en escenarios de emergencia donde la inmediatez es vital.

La Vista se compone principalmente de dos secciones para facilitar la interacción del usuario y la visualización de datos:

- **Formulario de Ingreso de Datos**: Esta sección es el punto principal de interacción para que el usuario proporcione la información necesaria al sistema. Para iniciar una consulta al modelo de recomendación, el despachador debe ingresar las coordenadas del foco

de incendio y el timestamp de cuándo ocurrió el incendio, así como el timestamp actual de la consulta. El sistema ofrece flexibilidad para la entrada de coordenadas, permitiendo a los usuarios ingresarlas manualmente en formatos decimales o de grados, minutos y segundos (DMS). El timestamp del incendio se ingresa manualmente, mientras que el timestamp de la consulta del usuario se registra automáticamente al momento de enviar la solicitud, simplificando el proceso operativo.

- **Sección de Resultados**: Tras procesar la información ingresada, el sistema presenta las recomendaciones generadas por el modelo en esta sección. Aquí, los usuarios pueden visualizar de manera clara y accesible las sugerencias específicas del algoritmo GRASP para el despacho de recursos, así como otros datos relevantes calculados por el sistema. Esta sección asegura que los despachadores tengan la información de salida consolidada para la toma de decisiones.

Controlador

El Controlador actúa como el intermediario central entre la vista y el modelo de recomendación. Su rol principal es gestionar la lógica de comunicación y coordinar las operaciones clave del sistema. La elección de la tecnología para su desarrollo se basó en la necesidad de alta velocidad, soporte para operaciones asíncronas, y la facilidad para construir APIs bien estructuradas, características fundamentales para asegurar una comunicación eficiente y un rendimiento óptimo, incluso en escenarios de alta demanda operativa.

Entre sus funcionalidades más destacadas se incluyen:

- Gestión de Solicitudes: Recibe las entradas del usuario (como coordenadas de incendios y timestamps) desde la vista a través de una API REST. Estas entradas son validadas rigurosamente utilizando Pydantic para asegurar la integridad y el formato correcto de los datos antes de ser enviadas al modelo GRASP para su procesamiento.
- Distribución de Resultados: Una vez que el modelo GRASP genera las recomendaciones de despacho, el controlador se encarga de formatear estos resultados (por ejemplo, a JSON) y enviarlos de vuelta a la vista. Esto garantiza que la interfaz pueda renderizarlos de inmediato, proporcionando una respuesta rápida al usuario.
- Integración de Datos: También es responsable de la conexión con bases de datos externas, extrayendo información actualizada y proporcionándola al modelo cuando se requiere para generar recomendaciones precisas.

Objetivos y Beneficios

El sistema busca alcanzar múltiples objetivos que abordan directamente los problemas identificados en el proceso actual de la central de incendios de CMPC. En primer lugar, mejorar los tiempos de respuesta mediante la automatización del análisis y la asignación de recursos, eliminando los retrasos inherentes a la matriz estática y permitiendo una acción más rápida y coordinada frente a los incendios. En segundo lugar, apoyar a los despachadores en

situaciones de alta presión, ofreciendo una herramienta visual e intuitiva que reduce la carga cognitiva y minimiza errores, especialmente entre despachadores con menor experiencia, e incluso disminuyendo potencialmente sus niveles de estrés. Además, a futuro, el sistema podría registrar datos detallados de cada emergencia, generando estadísticas valiosas para la planificación estratégica, tales como patrones de riesgo o la eficiencia de los recursos en diferentes escenarios, lo que podría ayudar a tomar decisiones informadas para la prevención y optimización de la respuesta.

Contexto y Relevancia

La relevancia de esta propuesta radica en su capacidad para llenar un vacío en la gestión de incendios forestales en CMPC. Actualmente, las soluciones tecnológicas avanzadas en este ámbito son escasas, y los sistemas existentes, como Kitral, no ofrecen la integración en tiempo real ni el soporte dinámico que este proyecto proporciona. Al aprovechar un modelo académico (cuya base teórica se detallará en secciones posteriores) y adaptarlo a un entorno operativo real, el sistema representa un avance práctico y una solución innovadora para CMPC. Su diseño modular y escalable asegura que pueda evolucionar con las necesidades de la empresa, integrando nuevas fuentes de datos o funcionalidades según sea necesario, mientras que su enfoque en la usabilidad lo hace accesible a una amplia gama de usuarios. En resumen, esta propuesta ofrece una solución integral que combina innovación, eficiencia y pragmatismo.

4. Detalle de la Propuesta

4.1 Diseño del Sistema

El diseño del sistema propuesto busca abordar de manera integral los desafíos mencionados, estructurándose en una serie de componentes interconectados que trabajan colaborativamente. Esta arquitectura modular está pensada para facilitar la integración entre los mismos componentes del sistema y con la infraestructura existente de CMPC. A continuación, se describen los elementos clave que conforman esta solución, cada uno con un rol específico para el procesamiento de información y la generación de recomendaciones.

El sistema está compuesto por los siguientes componentes principales:

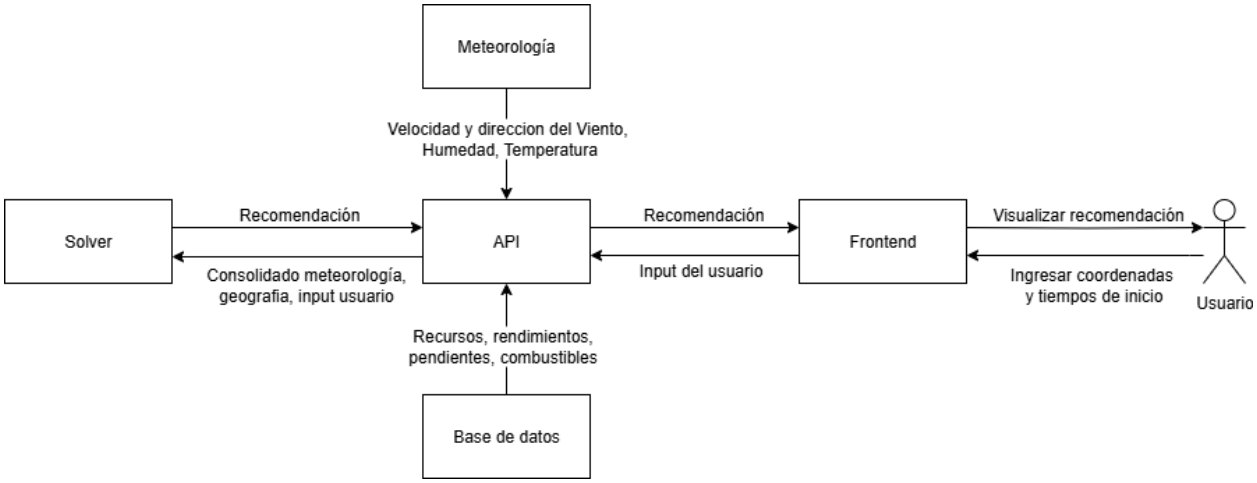


Figura 4. Diagrama del sistema (elaboración propia).

- El solver, encargado de recibir datos relevantes sobre el inicio de uno o varios incendios y devolver recomendaciones de asignación de recursos junto con métricas que proyectan el posible avance del incendio.
- Una API, diseñada para consolidar la información meteorológica, geográfica y entrada del usuario, y facilitar una integración fluida con la plataforma de despachos utilizada actualmente en CMPC, Gesfire. Esta API permitirá que Gesfire o un frontend independiente interactúen con el sistema.
- Un frontend, que constituirá el componente principal de interacción con el usuario. Podrá ser una aplicación móvil, una página web, una aplicación de escritorio, etc., y se encargará de comunicar los datos a la API y mostrar las recomendaciones generadas por el solver de manera visual e intuitiva.

- Meteorología, un servicio de datos meteorológicos en tiempo real que permitirá al sistema adaptarse a las condiciones climáticas actuales al momento de generar una recomendación, así como a las consultas específicas de los usuarios.
- Una base de datos, que almacenará datos específicos y relevantes sobre la temporada de incendios, los cuales, si bien pueden cambiar entre temporadas, permanecen estáticos o semi estáticos durante la misma. Estos datos incluyen:
    - Pendientes del territorio de interés: archivos raster o .shp que representan la pendiente del terreno afectado en una resolución determinada, permitiendo obtener un valor de pendiente a partir de una coordenada.
    - Combustibles: información similar a la pendiente, pero para los tipos de combustible presentes en el territorio, permitiendo determinar el tipo de combustible en una coordenada específica.
    - Tipos de recursos: la clasificación de los distintos recursos de combate disponibles para la temporada de incendios.
    - Costos de uso: los costos asociados al uso por hora y por despacho de cada tipo de recurso.
    - Rendimientos: la eficiencia estimada en metros de línea construida por hora, por tipo de recurso y según el tipo de combustible presente.
    - Recursos disponibles: información detallada sobre cada recurso, incluyendo coordenadas actuales, tipo, identificador único, disponibilidad (que se actualizará dinámicamente según la asignación a incendios), horas de trabajo acumuladas, entre otros.

En las siguientes secciones, se profundizará en el detalle de la implementación de estos componentes. Se presentarán las vistas del frontend, se explicará la integración del solver en el sistema y algunos detalles de su funcionamiento, y se detallarán las especificaciones de la API y la base de datos, delineando cómo cada uno contribuye a la funcionalidad global del sistema.

4.2 Frontend

La interfaz gráfica de usuario (GUI) es el componente central para la interacción con el sistema, permitiendo a los despachadores ingresar datos, visualizar recomendaciones en tiempo real y tomar decisiones informadas. Desarrollada con **React** para ofrecer una experiencia dinámica y reactiva, **Vite** para optimizar el desarrollo y compilación, y la librería **Leaflet** para la gestión de mapas geoespaciales, esta interfaz fue diseñada para ser intuitiva y eficiente, incluso para operadores inexpertos. Funciona como la "Vista" dentro de la arquitectura MVC del sistema, presentando la información procesada por el "Modelo" (GRASP y Firestore) a través del "Controlador" (FastAPI).

Las características principales de esta interfaz incluyen un formulario para la entrada de coordenadas de múltiples focos de incendio (en grados, minutos y segundos), junto con campos de timestamp para registrar la fecha y hora de inicio. Un mapa interactivo integrado con Leaflet permite a los despachadores seleccionar ubicaciones directamente, rellenando automáticamente los campos de coordenadas y simplificando el proceso. Tras el análisis del backend, un panel de información muestra resultados clave como el área quemada estimada, el tiempo proyectado de extinción y los detalles de los recursos asignados.

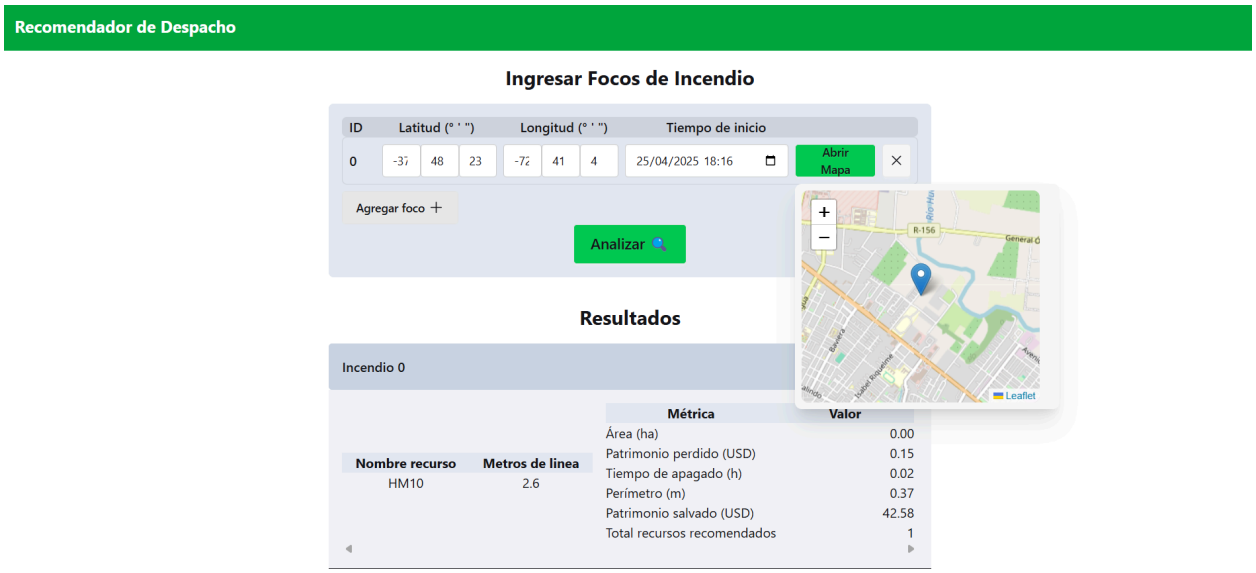


Figura 5. Frontend del recomendador de despacho (elaboración propia).

Implementación Técnica

La interfaz fue desarrollada en React 19, utilizando Vite como herramienta de construcción para optimizar los tiempos de desarrollo y despliegue. Leaflet se integró para gestionar el mapa interactivo, aprovechando su compatibilidad con mapas base de OpenStreetMap y su capacidad para superponer marcadores dinámicos. La lógica central de la interfaz se encarga de gestionar el estado de los datos de entrada y las solicitudes al backend, asegurando una experiencia de usuario fluida y reactiva. Para una revisión más detallada de la implementación técnica y su arquitectura, la información completa se encuentra disponible en el repositorio del proyecto (ver Anexo B).

Diseño y Funcionalidad de la Interfaz

La interfaz se estructura para cumplir con los siguientes objetivos:

- **Ingreso de datos:** Los usuarios pueden registrar focos de incendio ingresando coordenadas (latitud y longitud en formato WGS84) y timestamps manualmente mediante campos de texto, o seleccionando ubicaciones directamente en un mapa interactivo.
- **Visualización de resultados:** Tras enviar los datos al backend con el botón "Analizar", la interfaz muestra las recomendaciones generadas por GRASP, incluyendo recursos asignados (identificadores, cantidades, costos) y métricas clave (tiempo de extinción, área quemada, etc.), presentadas en una tabla.
- **Interactividad:** El mapa Leaflet permite zoom y desplazamiento, facilitando la identificación precisa de las ubicaciones de los incendios y los recursos, mientras que los controles de entrada están diseñados para ser minimalistas y accesibles.

El diseño prioriza la simplicidad y la claridad visual, con un enfoque en la asistencia a usuarios de distintos niveles de experiencia, un objetivo clave del proyecto. Las vistas principales de la interfaz se ilustran en las figuras descritas a continuación.

Descripción de las Vistas

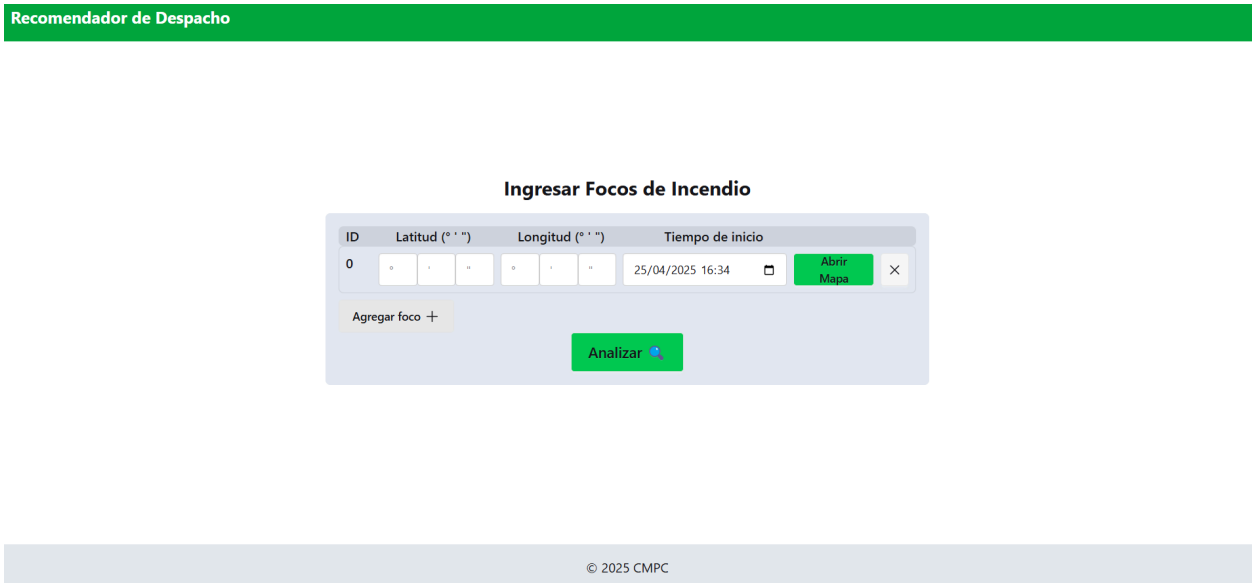


Figura 6. Interfaz Inicial del Recomendador de Despacho (elaboración propia). Muestra la pantalla principal con un título descriptivo, campos de entrada para digitar la latitud, longitud y timestamp, y botones para "Abrir Mapa", "Agregar Foco" y "Analizar". Esta vista, diseñada con un enfoque minimalista, facilita el ingreso de datos por parte de los despachadores, priorizando la accesibilidad y la claridad visual.

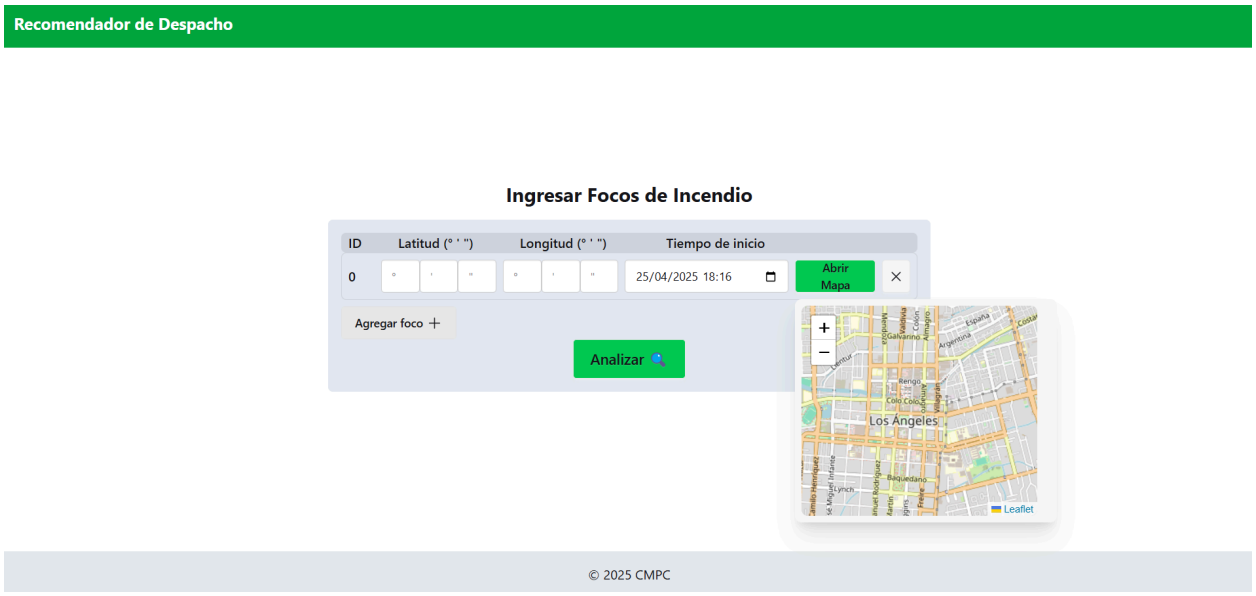


Figura 7. Mapa Interactivo para Selección de Coordenadas (elaboración propia). Se despliega el mapa Leaflet tras presionar "Abrir Mapa", centrado en una ubicación inicial representativa (Los Angeles, región de Biobío, Chile). Incluye controles de zoom y una capa de OpenStreetMap, permitiendo a los usuarios seleccionar coordenadas con un clic. Esta funcionalidad mejora la precisión geográfica y reduce errores en el ingreso manual.

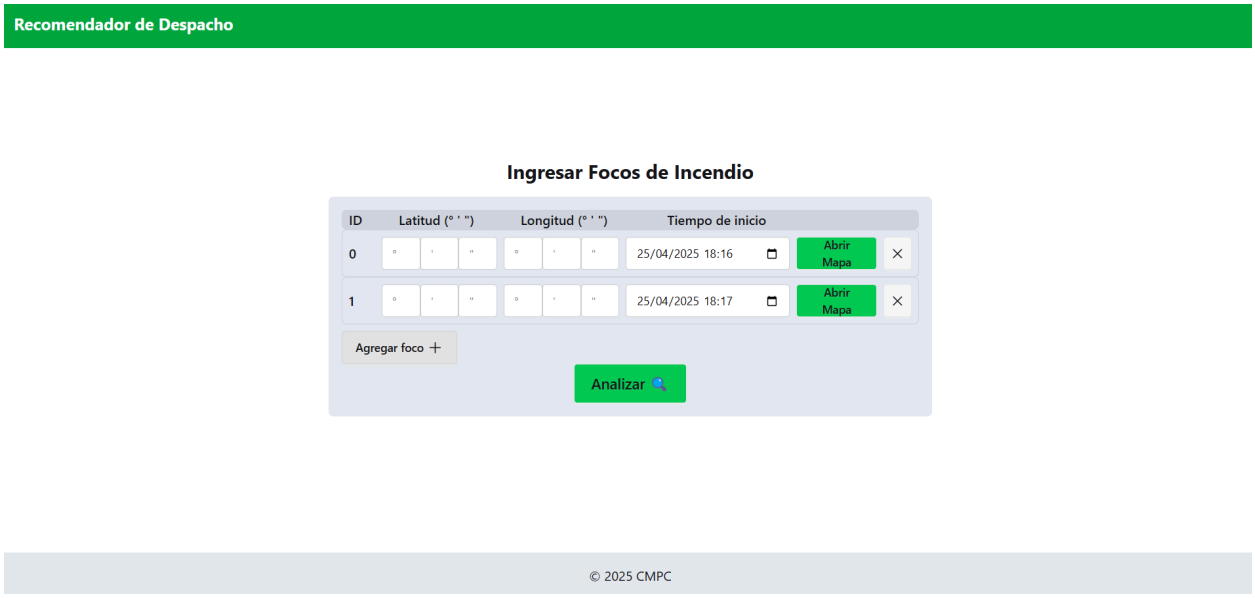


Figura 8. Agregar coordenadas y timestamp para nuevo foco (elaboración propia). Muestra la interfaz tras presionar "Agregar Foco", con los campos de entrada actualizados para registrar las coordenadas seleccionadas en el mapa (latitud y longitud) y un timestamp editable que indica la hora de detección del foco. Esta vista incluye un botón de eliminación para quitar el foco de la lista de análisis, ofreciendo a los despachadores una forma intuitiva de gestionar múltiples incidentes en tiempo real.

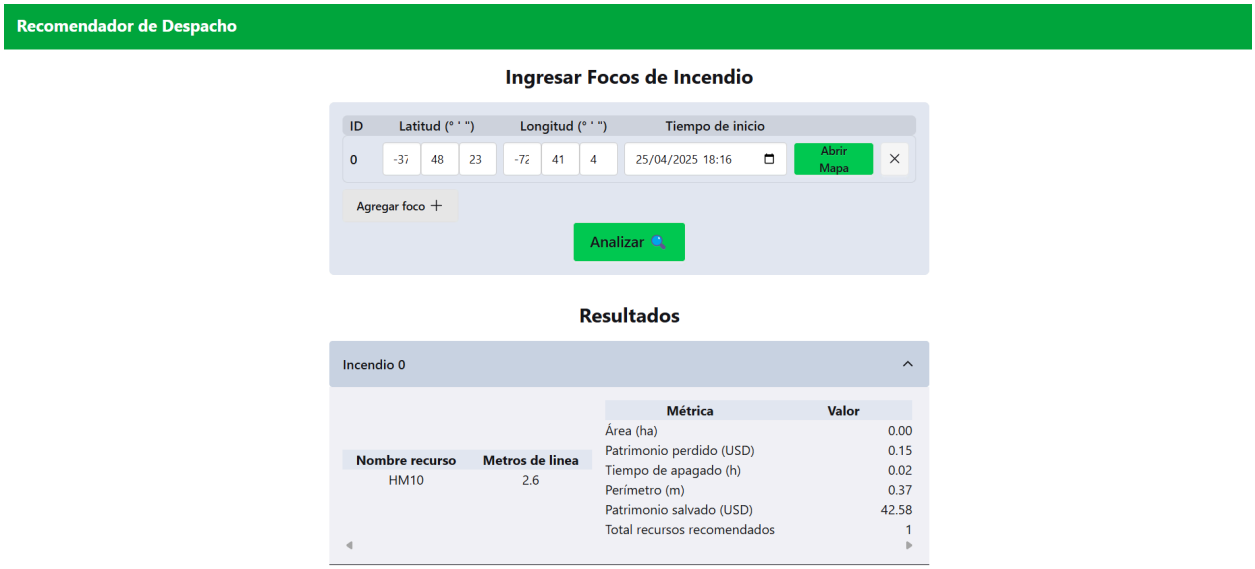


Figura 9. Resultados de las Recomendaciones tras Análisis (elaboración propia). Exhibe la pantalla posterior a presionar "Analizar", con una tabla que detalla las métricas de cada foco (tiempo de extinción, área quemada, perímetro, etc.) y los recursos asignados (ej. "DM1" con 200 metros de línea y costo de 2000 USD). Esta vista proporciona información de manera estructurada, apoyando la toma de decisiones en tiempo real.

4.3 Modelo de Recomendación

Adaptado específicamente para este proyecto a partir de un modelo previo proporcionado por la Universidad de Concepción a CMPC, se implementó en C++ para garantizar un rendimiento computacional eficiente, esencial en un contexto donde las decisiones deben tomarse en tiempo real. El modelo exhibe un comportamiento de tipo "anytime", lo que significa que puede ofrecer soluciones válidas en cualquier momento de su ejecución, con un límite de tiempo establecido, optimizando así la rapidez de respuesta sin comprometer la calidad de las recomendaciones. La integración con el sistema se logró mediante el uso de datos en formato JSON, procesados con la biblioteca nlohmann/json, facilitando una comunicación fluida con la REST API (FastAPI) y la interfaz gráfica (React), en línea con la arquitectura MVC adoptada.

Este modelo de recomendación basado en la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) procesa los datos recopilados para generar asignaciones de alta calidad de recursos. Toma como entrada las variables almacenadas en la base de datos, junto con datos en tiempo real como coordenadas de los focos de incendio y timestamps ingresados por los usuarios, analizándolos en conjunto con factores como pendientes, modelos de combustible y condiciones meteorológicas (temperatura, viento, humedad). Su diseño greedy y aleatorizado permite encontrar soluciones eficientes en un corto tiempo, adaptándose a las condiciones cambiantes de cada emergencia. La elección de C++ asegura un rendimiento óptimo, crucial para cálculos intensivos en escenarios donde la velocidad de respuesta puede determinar el éxito de la contención del fuego. El modelo no solo utiliza datos brutos, sino que también los enriquece con métricas calculadas, como el área potencialmente quemada y el tiempo estimado de extinción, que se devuelven como parte de las recomendaciones.

Funcionalidad del Modelo

El modelo GRASP combina una fase greedy, que genera soluciones iniciales basadas en criterios de prioridad (como la distancia de los recursos al foco), con una búsqueda local aleatorizada que refina estas soluciones para encontrar asignaciones de alta calidad. Este enfoque heurístico permite explorar un amplio espacio de posibilidades en tiempos reducidos, adaptándose a las condiciones dinámicas de los incendios forestales. La entrada del modelo incluye dos conjuntos de datos principales provenientes de CMPC y Open-Meteo:

- **Incendios:** Información detallada sobre cada foco activo:
    - *Coordenadas geográficas:* Latitud y longitud en formato WGS84 (grados decimales), para precisar la ubicación del incendio.
    - *Meteorología:* Temperatura (°C), velocidad del viento (m/s), dirección del viento (grados) y humedad relativa (%), que afectan la propagación y la intensidad del fuego.
    - *Geografía:* Pendiente del terreno (grados) y modelo de combustible (tipo y densidad de vegetación, ej. pino, eucalipto), que influyen en la dificultad de contención.

- **Recursos:** Detalles operativos de los recursos disponibles:
    - *Disponibilidad:* Cantidad y estado de brigadas, vehículos y equipos (ej. "disponible", "en uso"), obtenidos de la base de datos Firestore sincronizada con Google Sheets de CMPC.
    - *Costos:* Costo operativo por recurso en USD, incluyendo combustible y horas de trabajo.
    - *Rendimientos:* Capacidad operativa por modelo de combustible, como metros de línea de contención por hora o volumen de agua disponible, que mide la eficacia de cada recurso.

La salida generada por el modelo, también en formato JSON, especifica las asignaciones para cada incendio, acompañadas de métricas:

- *Recursos asignados:* Identificador del recurso (ej. "DM1"), cantidad asignada (ej. metros de línea) y costo asociado.
- *Métricas:* Tiempo estimado de extinción (horas), área quemada (hectáreas), valor del patrimonio perdido (USD) y valor del patrimonio salvado (USD).

Implementación Técnica

La implementación original del modelo no fue modificada. Sin embargo, se desarrolló una lógica propia para la lectura y escritura de datos, la cual se muestra a continuación. Esta fue implementada en C++ para integrarlo directamente con el código legado. Se utilizó la biblioteca nlohmann/json para el manejo de datos en formato JSON, debido a su simplicidad, eficiencia y compatibilidad con C++ moderno, lo que facilitó la lectura de entradas y la generación de salidas sin conversiones complejas. A continuación, se presenta un ejemplo simplificado (para más detalles, véase el código fuente en el Anexo A) que ilustra dicha lógica.

```
//Dependencias
#include <nlohmann/json.hpp>  // Biblioteca para manejo de JSON
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "solver.h"
#include "fire.h"

using json = nlohmann::json;  // Alias para simplificar la sintaxis

int main() {
    // inicialización del solver
    solver s;
```

```
    //Entrada
    // Lectura de la entrada JSON
    std::ifstream inputFile("input.json");
    if (!inputFile.is_open()) {
        std::cerr << "Error: No se pudo abrir el archivo de entrada" <<
std::endl;
        return 1;
    }
    json inputData;
    inputFile >> inputData;
    inputFile.close();
    //Procesamiento
    // Procesamiento de datos de incendios
    std::vector<Fire> firesData;
    for (const auto& fire : inputData["fires"]) {
        Fire f;
        f.lat = fire["lat"];
        f.lon = fire["lon"];
        f.temp = fire["temperature"];
        f.windSpeed = fire["wind_speed"];
        f.windDirection = fire["wind_direction"];
        f.slope = fire["slope"];
        ...
        firesData.push_back(f);

        std::cout << "Procesando incendio en (" << f.lat << ", " << f.lon
<< ")" << std::endl;
    }

    // Se repite el mismo proceso para los recursos
    ...

    // Se ejecuta la lógica de GRASP
    std::cout << "Generando recomendación" << std::endl;
    Recommendation recommendation = s.solve(firesData);
    //Salida
    // Generación de la salida JSON con recomendaciones
    json outputData;
    for (int i = 0; i < recommendation.assignments.size(); i++) {
        Assignment a = recommendation.assignments[i];
        auto& fireOutput = outputData["fires"][i];
        fireOutput["metrics"] = a.metrics;
```

```
        fireOutput["resources"] = a.resources;
    }

    // Recursos no usados
    outputData["notUsed"] = recommendation.notUsed;

    // Escritura de la salida JSON
    std::ofstream outputFile("output.json");
    if (!outputFile.is_open()) {
        std::cerr << "Error: No se pudo crear el archivo de salida" <<
std::endl;
        return 1;
    }
    outputFile << outputData.dump(4);  // Salida con indentación legible
    outputFile.close();

    std::cout << "Recomendaciones generadas exitosamente" << std::endl;
    return 0;
}
```

Análisis del Código

- **Dependencias:** Se utiliza <nlohmann/json.hpp> como la biblioteca principal para manejar JSON, aprovechando su diseño header-only y su interfaz intuitiva. El alias using json simplifica el código.
- **Entrada:** Los datos se leen desde el archivo input.json y se cargan en un objeto JSON. Estos datos incluyen los parámetros descritos previamente, cuyo formato específico se detalla a continuación.
- **Procesamiento:** Una vez que se ha leído el archivo de entrada, el modelo procesa la totalidad de los datos proporcionados utilizando la metaheurística GRASP, seleccionando los recursos en función de su afinidad con cada incendio. La afinidad de cada recurso se define mediante la fórmula: cantidad de metros de línea de contención que generan durante la primera hora de propagación del incendio dividido por el costo de combate, lo que permite priorizar la asignación de manera eficiente y optimizada según estos parámetros.
- **Salida:** La salida se genera en un objeto JSON que almacena los datos descritos previamente, siguiendo el formato detallado a continuación, y se guarda en el archivo output.json con un formato legible gracias a la función json.dump.

Formato de entrada y salida JSON

Entrada (input.json)

```
{
  timestamp,
  fires: [
    {
      id,
      lat,
      lon,
      temperature,
      humidity,
      windSpeed,
      windDirection,
      slope,
      vplFactor,
      rodalValue,
      cityDistanceMeters,
      builtLineLength,
      timestamp,
      fuelModel,
      incompatibilities
    },
  ],
  resources: [
    {
      id,
      type,
      workedHours,
      lat,
      lon,
      state,
      isGrouped,
      assignedFire,
      incompatibilities
    }
  ],
  performanceMatrix: [
    [...],
    [...],
    ...
  ],
  resourceCosts: {
```

```
    <type>: {
      transportUSD,
      hourUSD
    }
  }
}
```

Salida (output.json):

```
{
  fires: [
    {
      id,
      metrics: {
        extinguishedTime,
        area,
        perimeter,
        damage,
        savedDamage
      }
      resources: [
        {
          id,
          line,
          cost
        },
        ...
      ]
    }
  ],
  notUsed: [
    "...",
    ...
  ]
}
```

4.4 API REST y Base de Datos

El sistema implementa una arquitectura de datos diseñada para integrar información enviada en tiempo real al modelo de recomendación. Esta estructura permite vincular distintas fuentes internas utilizadas por CMPC, como hojas de cálculo, información de los recursos y costos asociados a su uso, dentro de un repositorio centralizado y accesible. A continuación se muestra el modelo lógico de la base de datos NoSQL utilizada, donde se destacan las colecciones que alimentan el sistema: Recursos, Rendimientos y Costos. Estas colecciones constituyen la base de conocimiento necesaria para que el modelo GRASP pueda generar recomendaciones contextualizadas y eficientes.

Base de datos

La base de datos del sistema está implementada en Firebase Cloud Firestore, una base de datos NoSQL en la nube de Google que ofrece almacenamiento escalable, sincronización en tiempo real y consultas eficientes. Esta plataforma funciona como el repositorio central de los datos del sistema, integrando información proveniente de distintas fuentes internas de CMPC.

La estructura de la base de datos se organiza en tres colecciones principales:

- Recursos: almacena información sobre la ubicación, disponibilidad, tipo de recurso y sus horas de trabajo acumuladas, entre otras.
- Rendimientos recurso–tipo de combustible: contiene datos sobre el desempeño esperado de cada tipo de recurso frente a los distintos tipos de combustible, permitiendo estimar su efectividad en función de las condiciones del terreno.
- Costos: contiene los costos asociados al uso de cada tipo de recurso, lo que permite integrar un criterio económico en la toma de decisiones durante la ejecución del modelo.

La colección de recursos se sincroniza automáticamente con hojas de cálculo en Google Sheets, herramienta ya utilizada por CMPC para la gestión operativa. Esta integración asegura que la base de datos se mantenga actualizada sin necesidad de cargas manuales frecuentes, minimizando errores y optimizando los tiempos de respuesta. Las colecciones de rendimientos y costos se alimentan desde fuentes internas específicas del modelo.

Gracias a esta arquitectura, el sistema puede operar con datos actualizados en tiempo real y también consultar información histórica cuando sea necesario, facilitando tanto el análisis retrospectivo como la ejecución eficiente del modelo GRASP.

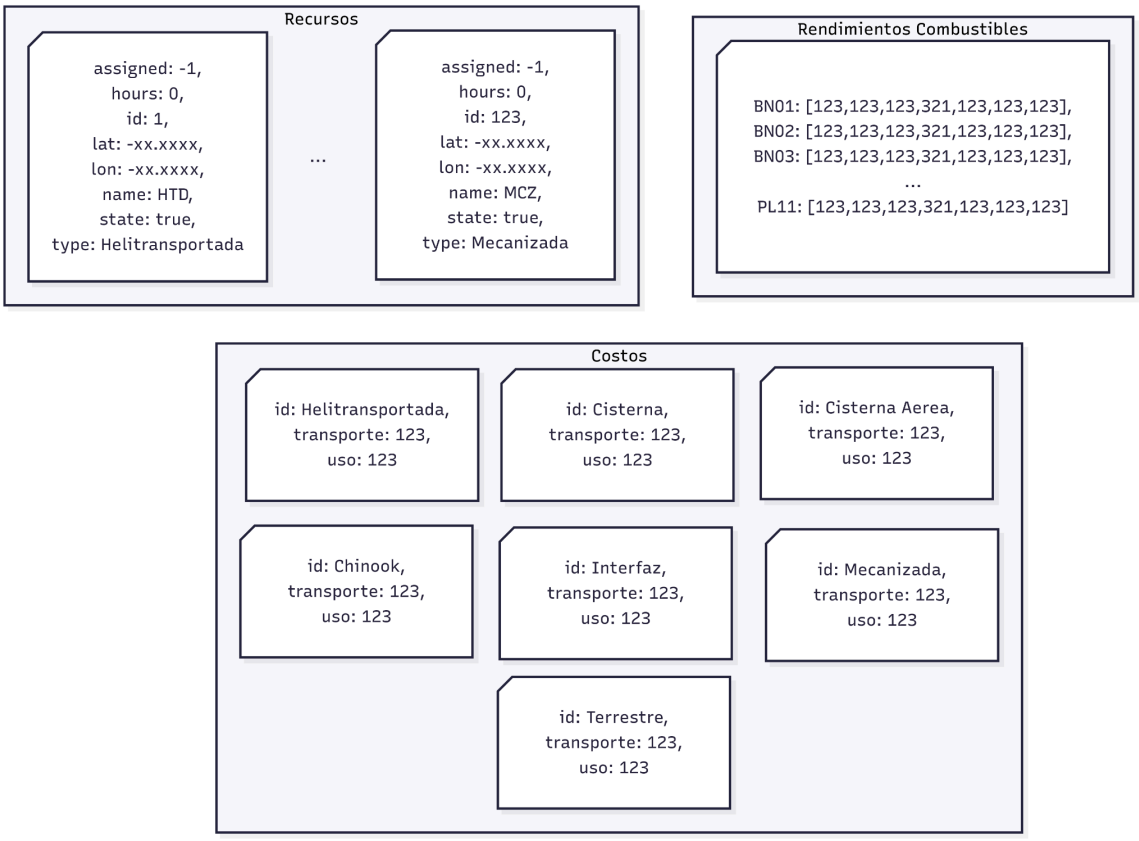


Figura 10. Modelo de base de datos NoSQL del sistema (elaboración propia).

REST API

El controlador, implementado como un servidor web basado en FastAPI (Python), es el intermediario que coordina las interacciones entre el modelo y la vista, gestionando el flujo de datos y las solicitudes del sistema. Sus funciones principales son:

- **Gestión de entradas:** Recibe los datos ingresados por los usuarios a través de la interfaz (coordenadas, timestamps) y los valida utilizando Pydantic, un componente de FastAPI que asegura la integridad y el formato correcto de la información antes de enviarla al modelo GRASP.
- **Integración de datos:** Consulta la base de datos Firestore para complementar las entradas del usuario con información adicional, como las condiciones del terreno o la disponibilidad de recursos, y las envía al modelo en un formato estructurado (JSON).
- **Distribución de resultados:** Una vez que GRASP procesa los datos y genera recomendaciones, el controlador las recibe, las formatea como respuestas JSON y las envía a la vista para su visualización. FastAPI, gracias a su soporte para operaciones asíncronas, permite manejar múltiples solicitudes simultáneamente, lo que es esencial en situaciones donde varios despachadores podrían estar operando al mismo tiempo. Además, el controlador implementa medidas de seguridad, como API keys, para

restringir el acceso a usuarios autorizados de CMPC, protegiendo los datos sensibles del sistema.

El desarrollo del servidor web y la API REST constituye el componente de control del sistema, encargado de gestionar la comunicación entre la interfaz gráfica, el solver de recomendación, la API de meteorología y las bases de datos internas de CMPC. Implementado en Python utilizando el framework FastAPI, el backend proporciona endpoints eficientes y seguros que facilitan la recepción de datos desde la interfaz, la integración de información dinámica y estática, y la invocación del solver, que utiliza el algoritmo GRASP, para generar recomendaciones en tiempo real. FastAPI fue seleccionado por su alto rendimiento, gracias a su soporte nativo para operaciones asíncronas y su capacidad para generar documentación automática de la API, lo que simplifica su uso y mantenimiento.

Diseño y Funcionalidad del Backend

El servidor web se estructuró en torno a un endpoint principal, /prediction/, que recibe solicitudes POST desde la interfaz gráfica con datos iniciales de los incendios (coordenadas y timestamps) y coordina el flujo de procesamiento para devolver recomendaciones optimizadas. Las principales funcionalidades del backend incluyen:

- **Recepción de datos:** Captura de coordenadas geográficas (latitud y longitud en formato WGS84) y timestamps enviados por los despachadores a través de la interfaz React.
- **Integración de datos externos:** Consulta de información meteorológica en tiempo real (temperatura, viento, humedad) desde Open-Meteo [3], datos geográficos (pendientes, modelos de combustible) desde archivos geoespaciales (.shp/.tiff) y disponibilidad de recursos (brigadas, vehículos) desde Firestore, sincronizado con Google Sheets de CMPC.
- **Invocación del modelo GRASP:** Preparación de una entrada JSON consolidada que combina todos los datos y ejecución del modelo compilado en C++ para obtener las asignaciones de alta calidad.
- **Respuesta al frontend:** Devolución de las recomendaciones en formato JSON, incluyendo recursos asignados y métricas (tiempo de extinción, área quemada, perímetro, entre otros), para su respectiva visualización.

El diseño asíncrono de FastAPI permite manejar múltiples solicitudes simultáneamente, una característica esencial para escenarios donde varios despachadores pueden operar al mismo tiempo en la central de incendios. Además, se implementan medidas de validación y seguridad para proteger los datos sensibles de CMPC, como la autenticación mediante API keys. Durante

la fase de cálculo de ETAs, se estima el tiempo de llegada de cada recurso a cada incendio, utilizando la distancia lineal entre ambos puntos y una velocidad promedio definida según el tipo de recurso.

Implementación Técnica

El backend se desarrolló en Python 3.9, utilizando FastAPI como framework principal y Pydantic para la validación estricta de los datos de entrada. El endpoint /prediction/ fue el núcleo de la API, diseñado para recibir una lista de incendios en formato JSON, procesarlos y devolver las recomendaciones generadas por GRASP (ver anexo A). A continuación, se presenta un fragmento del código implementado para este endpoint:

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List
import subprocess
import json
import requests
from google.cloud import firestore  # Cliente de Firestore
from time import datetime
import geopandas as gpd  # Para manejo de datos geoespaciales

app = FastAPI(title="API de Predicción de Incendios - CMPC")

# Modelo de datos para un incendio (entrada desde la interfaz)
class FireData(BaseModel):
    lat: float
    lon: float
    timestamp: datetime

# Modelo de datos para la solicitud completa
class PredictionRequest(BaseModel):
    fires: List[FireData]
    timestamp: datetime

# Función para obtener datos meteorológicos desde Open-Meteo
def fetch_meteo_data(lat: float, lon: float) -> dict:
    url =
f"https://api.open-meteo.com/v1/forecast?latitude={lat}&longitude={lon}&cur
rent_weather=true"
    response = requests.get(url)
```

```
    if response.status_code != 200:
        raise HTTPException(status_code=500, detail="Error al consultar
datos meteorológicos")
    data = response.json()["current_weather"]
    return {
        "temp": data["temperature"],
        "wind_speed": data["windspeed"],
        "wind_direction": data["windDirection"],
        "humidity": data["humidity"]
    }

# Función para obtener datos geográficos desde archivos locales
def fetch_geo_data(lat: float, lon: float) -> dict:
    gdf = gpd.read_file("data/geo/slope.shp")  # Archivo de CMPC
    point = gpd.points_from_xy([lon], [lat])[0]
    geo_info = gdf[gdf.contains(point)].iloc[0]
    return {
        "slope": float(geo_info["slope"]),
        "fuel_type": geo_info["fuel_type"]
    }

# Función para consultar recursos en Firestore
def fetch_resources_from_firestore() -> list:
    db = firestore.Client()
    resources = [
        {
            "id": resource.id,
            "lat": resource.get("lat"),
            "lon": resource.get("lon"),
            "type": resource.get("type"),
            "status": resource.get("status")
        }
        for resource in db.collection("resources").stream()
    ]
    return resources

@app.post("/prediction/")
async def get_prediction(request: PredictionRequest):
    try:
        # Obtener datos adicionales para el primer incendio (simplificación
para el ejemplo)
        meteo_data = fetch_meteo_data(request.fires[0].lat,
request.fires[0].lon)
```

```
        geo_data = fetch_geo_data(request.fires[0].lat,
request.fires[0].lon)
        resources = fetch_resources_from_firestore()

        # Preparar entrada consolidada para GRASP
        input_data = {
            "fires": [{
                "lat": fire.lat,
                "lon": fire.lon,
                "timestamp": fire.timestamp,
                **meteo_data,
                **geo_data
            } for fire in request.fires],
            "resources": resources
        }

        # Escribir entrada en archivo temporal para GRASP
        with open("input.json", "w") as f:
            json.dump(input_data, f)

        # Ejecutar el modelo GRASP y capturar salida
        process = subprocess.run(["./solver"], check=True,
capture_output=True, text=True)
        if process.returncode != 0:
            raise HTTPException(status_code=500, detail=f"Error en GRASP:
{process.stderr}")

        # Leer archivo de salida
        with open('output.json', 'r') as file:
            output_data = json.load(file)
            return output_data

    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error en el
procesamiento: {str(e)}")
```

Análisis del Código

- **Dependencias:** Se utilizan FastAPI para la API, Pydantic para la validación de modelos, subprocess para ejecutar el solver y bibliotecas adicionales como requests (Open-Meteo), google.cloud.firestore (Firestore) y geopandas (datos geoespaciales).
- **Modelos de datos:** FireData define la estructura de cada incendio recibido desde la interfaz, mientras que PredictionRequest encapsula la lista completa, asegurando que las solicitudes sean validadas antes de procesarse.
- **Funciones auxiliares:**
    - fetch_meteo_data: Consulta la API de Open-Meteo para datos meteorológicos en tiempo real.
    - fetch_geo_data: Lee archivos geoespaciales (.shp) de CMPC para obtener pendientes y tipos de combustible, usando geopandas para análisis espacial.
    - fetch_resources_from_firestore: Extrae recursos disponibles desde Firestore.
- **Endpoint /prediction/:** Recibe la solicitud, integra datos externos, escribe un archivo JSON para el solver, ejecuta el modelo y devuelve la salida. El uso de async aprovecha las capacidades asíncronas de FastAPI, y el manejo de excepciones mejora la robustez.
- **Ejecución del Solver:** Se utiliza subprocess.run para una ejecución más segura y simple, capturando la salida directamente como texto.

4.5 Integración del Sistema

La integración del sistema representa una fase crítica en el proyecto, donde todos los componentes desarrollados (interfaz gráfica, API, modelo de recomendación y bases de datos) se unificaron en una solución funcional y operativa, lista para ser utilizada por la central de incendios de CMPC. A continuación, se detalla el proceso de integración, las decisiones técnicas tomadas, los desafíos enfrentados y las soluciones implementadas para asegurar que el sistema funcione como una unidad cohesiva y eficiente.

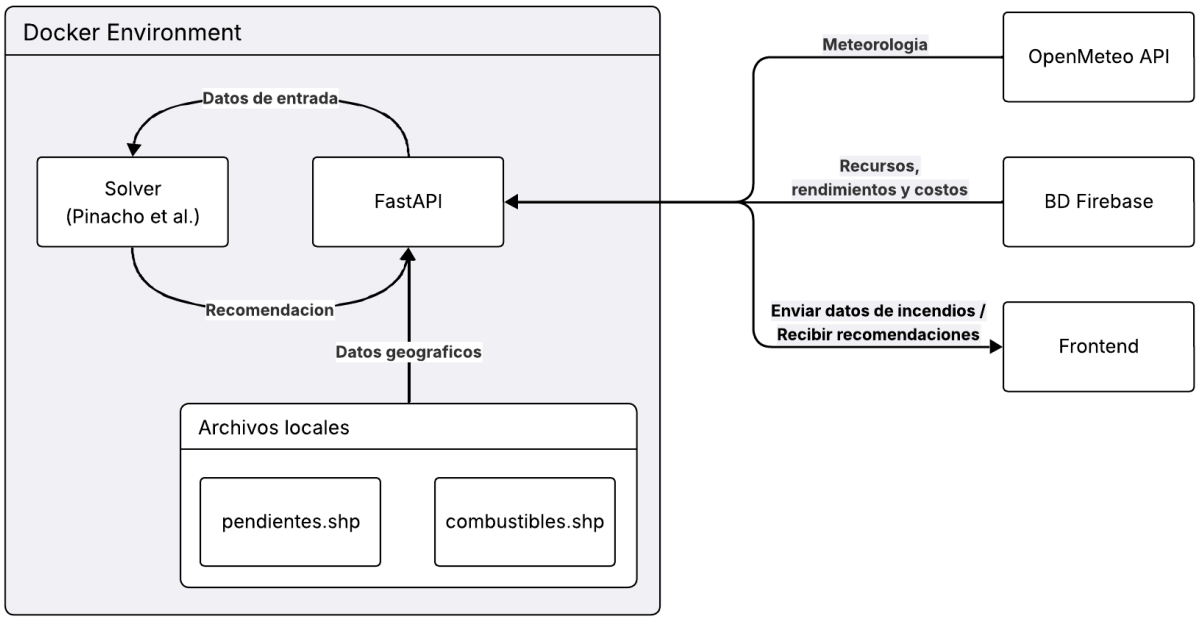


Figura 11. Diagrama de arquitectura de la solución (elaboración propia).

Despliegue del backend

El backend fue diseñado para ser desplegado en un servidor y presenta una arquitectura fácilmente containerizable. Los principales componentes son el modelo de recomendación, desarrollado en C++, y el servidor web implementado con FastAPI. Ambos módulos pueden ser containerizados y desplegados en plataformas de computación en la nube. Para este proyecto, se seleccionó Google Kubernetes Engine de Google Cloud Platform, dado que es el estándar operativo establecido en la empresa.

La comunicación entre el frontend y los componentes del backend se realiza a través de Cloudflare, utilizado como proxy y red de distribución de contenido (CDN), lo que permite acceso desde cualquier ubicación y brinda una capa de seguridad.

Conexión de Componentes

La integración implicó conectar la interfaz gráfica, la API y el modelo de recomendación en un flujo de trabajo continuo.

La interfaz gráfica fue desarrollada con React y desplegada como una aplicación web mediante Google Apps Script, lo que permitió que los despachadores accedieran a la plataforma desde cualquier navegador con conexión a internet, ya fuera en la central de incendios o en ubicaciones remotas. La comunicación con el backend se realiza a través de solicitudes HTTP al endpoint /prediction, utilizando la API Fetch de JavaScript. Para optimizar el rendimiento y el despliegue, se utilizó Vite, generando un paquete estático que es servido por la aplicación de Apps Script. Esta implementación permite restringir el acceso únicamente a los miembros de la organización CMPC, ya que el control de acceso está gestionado por Google Apps Script, lo que resulta adecuado dado que los usuarios previstos pertenecen exclusivamente a la subgerencia de protección forestal.

El backend fue desplegado como un servicio independiente, escuchando solicitudes en el puerto 8000. Actúa como el punto central de comunicación del sistema: recibe datos desde la interfaz (como coordenadas y parámetros del usuario), consulta la base de datos en Firestore para obtener información sobre los recursos disponibles, y envía estos datos al modelo de recomendación para su procesamiento. La integración con el modelo se realiza mediante un flujo bidireccional: FastAPI genera un archivo input.json, invoca el ejecutable C++ (./solver) y luego lee el archivo output.json para devolver los resultados al frontend en formato JSON. Este diseño permitió mantener el solver como un módulo independiente, optimizado en C++, mientras la API se encarga de la lógica de comunicación. Para proteger los datos sensibles de CMPC, como ubicaciones e inventarios, se implementó autenticación basada en API keys mediante un middleware en FastAPI. Además, se utilizó Pydantic para validar las solicitudes entrantes, garantizando que solo datos bien formados fueran procesados por el modelo, reduciendo así el riesgo de errores y mejorando la robustez del sistema.

El modelo de recomendación, desarrollado en C++ y compilado como un ejecutable, se integró al sistema como un microservicio independiente. Para facilitar la comunicación entre FastAPI y el modelo, se implementó un flujo de integración mediante un script intermedio en Python, que actúa como puente entre ambos componentes. Este script recibe los datos enviados desde la interfaz gráfica a través de la API, los escribe en un archivo input.json legible por el ejecutable C++, ejecuta el modelo y luego lee los resultados desde output.json, devolviéndolos a la API en formato JSON. Este enfoque permite mantener al solver como un módulo desacoplado y optimizado en rendimiento, aprovechando la velocidad de C++ para los cálculos intensivos, mientras que Python proporciona flexibilidad en la orquestación del flujo de datos. La biblioteca nlohmann/json fue utilizada para garantizar la compatibilidad del formato JSON con FastAPI y React, evitando transformaciones adicionales.

Infraestructura en la Nube y Acceso Remoto

El despliegue en la nube se configuró para maximizar la disponibilidad y accesibilidad del sistema. El servidor backend es accesible mediante un endpoint intermediario desplegado con un Cloudflare Worker, el cual se encarga de manejar las solicitudes CORS y enrutar el tráfico de forma segura hacia la API. Para reforzar la seguridad, el backend implementa una verificación de API key en cada solicitud, asegurando que solo usuarios autorizados puedan acceder al sistema. La base de datos en Firestore, alojada en los servidores de Google Cloud, se conecta al sistema mediante credenciales seguras gestionadas a través de variables de entorno en el contenedor de FastAPI, garantizando una sincronización en tiempo real sin comprometer la seguridad.

Desafíos y Soluciones

La integración presentó varios desafíos técnicos que fueron abordados con soluciones específicas:

- **Compatibilidad entre lenguajes:** La comunicación entre Python y C++ se llevó a cabo utilizando archivos JSON como medio de intercambio de datos. Para evitar colisiones entre los nombres de los archivos generados por distintas solicitudes, se implementó un sistema basado en la creación de un hash único derivado de la request específica de cada usuario, asegurando así que cada archivo tuviera una identificación distinta y permitiendo un manejo ordenado y sin conflictos de las múltiples solicitudes procesadas por el sistema.
- **Escalabilidad:** Para garantizar que el sistema pudiera soportar múltiples usuarios simultáneos, se optimizó el servidor FastAPI para manejar de manera concurrente la recolección de datos meteorológicos, geográficos y las consultas a la base de datos Firestore. Este enfoque permite procesar las solicitudes de forma paralela, eliminando la necesidad de esperar secuencialmente por cada tarea y evitando cuellos de botella en el servidor.
- **Seguridad:** La protección del sistema se implementó mediante el uso de API keys para autenticar las solicitudes, configuraciones CORS para restringir el acceso a dominios autorizados, y permisos de acceso en Google Apps Script para limitar el uso exclusivamente a los miembros de la organización CMPC, garantizando así un control efectivo contra accesos no autorizados.

Dentro de las tareas de integración, se abordó el manejo del sistema de coordenadas utilizado por la central de incendios de CMPC, que opera en formato de grados, minutos y segundos (DMS). Dado que las librerías empleadas para el procesamiento de datos geoespaciales trabajan en formato decimal (coordenadas cartesianas), fue necesario desarrollar un módulo en JavaScript que realiza la conversión de los datos ingresados por los usuarios al formato adecuado antes de ser transmitidos al backend.

5. Evaluación de la Propuesta

La fase de pruebas y evaluación del sistema desarrollado para la subgerencia de protección forestal de CMPC fue una etapa crucial e integral diseñada para garantizar su calidad, funcionalidad, desempeño y alineación con los objetivos establecidos: optimizar el despacho de recursos en emergencias forestales. Este proceso abarcó la verificación técnica de componentes individuales (modelo GRASP, interfaz gráfica y backend), la evaluación de su eficiencia operativa y seguridad, y la validación de su utilidad práctica en escenarios reales mediante pruebas con usuarios finales. Se emplearon múltiples enfoques rigurosos, como el Desarrollo Guiado por Pruebas (TDD), pruebas técnicas automatizadas, mediciones de desempeño y simulaciones realistas, complementadas con retroalimentación cualitativa de despachadores sobre usabilidad. A continuación, se detalla cada aspecto de esta fase, explicando las metodologías utilizadas, los resultados esperados y obtenidos, y cómo estas pruebas contribuyeron a optimizar el sistema, asegurando que fuera confiable, eficiente y fácil de usar antes de su despliegue final.

5.1 Desarrollo y Pruebas Técnicas (TDD)

El desarrollo del sistema se guió por la metodología de Desarrollo Guiado por Pruebas (TDD, por sus siglas en inglés: Test-Driven Development), una práctica que asegura la calidad del código desde las primeras etapas al escribir pruebas antes de implementar las funcionalidades. Este enfoque se aplicó tanto al frontend como al backend, utilizando herramientas específicas para cada componente:

Frontend (Vitest): Para la interfaz gráfica desarrollada en React, se empleó Vitest, una herramienta de pruebas diseñada para entornos basados en Vite (el sistema de compilación usado en el proyecto). Las pruebas unitarias verificaron el correcto funcionamiento de los componentes individuales, como el mapa interactivo de Leaflet, los formularios de ingreso de coordenadas y los paneles de visualización de recomendaciones. Por ejemplo, se probaron casos como la renderización correcta de marcadores en el mapa tras hacer clic, la renderización correcta de las métricas (área quemada, tiempo de extinción, etc.) y la respuesta de la interfaz ante entradas inválidas (coordenadas fuera de rango). Además, se realizaron pruebas de integración para asegurar que los componentes trabajaran juntos sin errores, como la comunicación fluida entre el formulario de entrada y la solicitud a la API. Vitest permitió ejecutar estas pruebas rápidamente durante el desarrollo, identificando y corrigiendo fallos de manera iterativa.

Backend (Pytest): En el servidor FastAPI, las pruebas se realizaron con Pytest, una biblioteca de Python ampliamente utilizada por su flexibilidad y potencia. Las pruebas unitarias evaluaron las rutas de la API REST, como los endpoints para recibir coordenadas, consultar la base de datos Firestore y enviar datos al modelo GRASP. Por ejemplo, se verificó que una solicitud POST con coordenadas válidas devolviera un JSON con el formato esperado (siguiendo el

estándar JSend) y que errores como una API key inválida generaran respuestas de "fail" o "error" adecuadas. Las pruebas de integración comprobaron la interacción entre FastAPI y el algoritmo despachador, asegurando que el puente de comunicación (script en Python que ejecuta el modelo en C++) funcionara correctamente y que los datos de entrada y salida se procesaran sin pérdidas. Este enfoque TDD garantizó una base de código robusta, reduciendo la probabilidad de regresiones y asegurando que cada nueva funcionalidad se integrara sin comprometer la estabilidad existente.

5.2 Evaluación del Algoritmo

El presente proyecto se centró en el desarrollo e integración del sistema completo, lo que implicó priorizar la construcción de nuevos componentes y la articulación de una solución integral. En este contexto, el modelo GRASP, fundamental para la generación de recomendaciones de despacho, ya había sido desarrollado y validado por el equipo "CPMC - Incendios - Equipo Solver". Sus pruebas se documentaron en un informe técnico con fecha del 6 de septiembre de 2021, donde se evaluó exhaustivamente el desempeño del algoritmo GRASP en comparación con heurísticas Greedy (basadas en distancia y afinidad) para la asignación de recursos en escenarios de incendios forestales.

Dada la robustez de estas validaciones previas y con el objetivo de optimizar el tiempo de desarrollo y la carga de trabajo, se optó por reutilizar la base teórica y lógica del solver GRASP sin introducir modificaciones en su comportamiento original. La intervención en este proyecto se limitó a la adaptación de sus interfaces de entrada y salida, migrando la comunicación a un formato JSON para alinearse con el stack tecnológico del sistema actual. Esta aproximación permitió aprovechar un componente ya probado y eficiente, manteniendo la modularidad del trabajo previo. Por consiguiente, no se consideró necesario replicar las pruebas ya realizadas al modelo GRASP, dado que su lógica interna no fue alterada y su funcionalidad ya estaba plenamente validada en el contexto original.

Metodología y Dataset

El equipo original diseñó un dataset representativo basado en datos reales y simulados:

Fuentes de Datos

- Distancias: Calculadas con OpenStreetMap API, con limitaciones en caminos internos de CMPC.
- Meteorología: Extraída de ERA5 (Copernicus), incluyendo humedad, temperatura, velocidad y dirección del viento.
- Recursos: Obtenidos de plantillas Excel de CMPC (tipos, bases, rendimientos, costos).

Generación de Escenarios

- Posiciones aleatorias de incendios dentro de fundos (polígonos Shapefiles).
- Parámetros como pendiente, uso de suelo y modelo de combustible derivados de rasters.
- Fechas aleatorias en temporada de incendios (enero-marzo, últimos 5 años).
- Recursos con probabilidad de inactividad y retrasos para aeronaves (15-20 minutos).
- Escenarios con incendios a medio combatir, ajustando ETAs y asignaciones iniciales con GRASP.

Los tests se ejecutaron en una máquina con Arch Linux (kernel 5.13.13), procesador Intel i7-9700 @ 3.00 GHz, y 16 GB DDR4, usando C++11 compilado con GCC 11.1.0 (-O3).

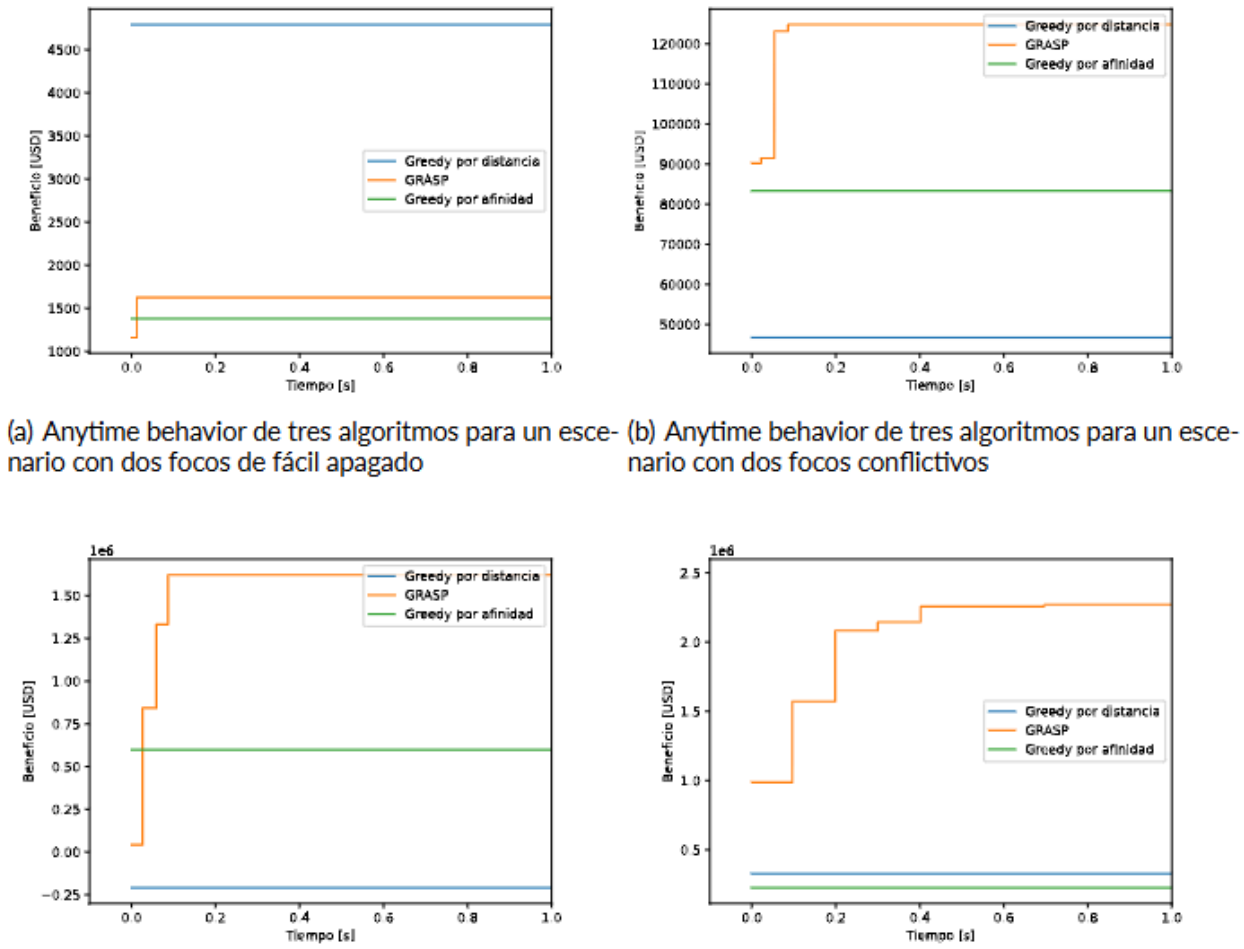
Resultados

Resultados después de 1 segundo de cómputo sobre los datasets utilizando Greedy por distancia (GPD), Greedy por afinidad (GpA) y GRASP. Para cada método se despliega la media $\bar{x}$ de los costos y para GRASP adicionalmente se despliega la desviación estándar σ.*Nota:* Valores negativos indican ahorros o ajustes contables específicos del modelo original.

Dataset	GpD	GpA	GRASP	
	$\bar{x}$ [USD]	$\bar{x}$ [USD]	$\bar{x}$ [USD]	σ [USD]
Escenario con 1 foco	195282.0	157426.0	2092810.0	7223.5
Escenario con 2 focos 1	4791.8	1380.8	3044.3	1773.2
Escenario con 2 focos 2	66.7	3635.9	3225.3	1269.5
Escenario con 2 focos 3	328.1	-2186.3	-4179.5	1049.7
Escenario con 2 focos con distintos inicios	46725.7	83332.4	121193.0	2124.4
Escenario con 3 focos 1	1250.0	-2707.5	-1065.3	2313.9
Escenario con 3 focos 2	4842.5	1032.8	638.0	2670.3
Escenario con 3 focos 3	5095.3	13135.3	17065.0	1412.4
Escenario con 3 focos con distintos inicios	71961.8	86461.9	103435.5	2476.4
Escenario con 4 focos 1	825.4	-7697.9	-8406.8	1935.7
Escenario con 4 focos 2	-209105.0	598644.0	1622470.0	91880.0
Escenario con 4 focos con distintos inicios	10593.1	7403.4	2647.2	2079.5

Dataset	GpD	GpA	GRASP	
	$\bar{x}$ [USD]	$\bar{x}$ [USD]	$\bar{x}$ [USD]	σ [USD]
Escenario con 1 foco	195282.0	157426.0	2092810.0	7223.5
Escenario con 5 focos	327674.0	225986.0	2460650.0	106550.5

Tabla 1. Comparativa de heurísticas [7].



(a) Anytime behavior de tres algoritmos para un escenario con dos focos de fácil apagado

(b) Anytime behavior de tres algoritmos para un escenario con dos focos conflictivos

(c) Anytime behavior de tres algoritmos para un escenario con cuatro focos

(d) Anytime behavior de tres algoritmos para un escenario con cinco focos

Figura 12. Comparativa de rendimientos entre algoritmos de optimización [7].

Conclusiones del Equipo Original

- GRASP supera a las heurísticas en escenarios complejos (>2 focos), explorando el espacio de estados de forma más exhaustiva.
- En casos simples, GpD o GpA pueden ser suficientes, pero GRASP es configurable para subsumirlas en sus primeras iteraciones.
- La variabilidad de GRASP (alta desviación en casos complejos) refleja su naturaleza metaheurística, ajustable según necesidades.

5.3 Usabilidad e Interfaz

La interfaz de usuario, desarrollada en React, fue evaluada considerando tanto la retroalimentación de los usuarios finales como métricas técnicas de rendimiento y accesibilidad. Esta evaluación tuvo como objetivo validar su usabilidad, alineación estética con las aplicaciones web existentes de CMPC, y eficiencia operativa en el contexto de la gestión de incendios forestales. A continuación, se detallan los aspectos clave de esta evaluación, incluyendo ajustes realizados y el análisis técnico derivado del informe Lighthouse del 4 de marzo de 2025.

Testimonio 1:

> “El trabajo realizado por Vicente representa un avance significativo en la incorporación de tecnologías aplicadas a la gestión del despacho de recursos desde la Central de Protección. En un contexto en que la inteligencia artificial desempeña un rol clave en la innovación y mejora continua, este desarrollo destaca por su capacidad para anticipar la evolución de un incendio, recomendar los recursos necesarios y estimar el tiempo requerido para controlarlo, ofreciendo así un valioso apoyo para decisiones rápidas y efectivas.
>
> Esta herramienta no solo reduce la carga cognitiva del despachador de la central de protección en situaciones de presión, agilizando la evaluación ante distintos escenarios de emergencia, sino que también abre oportunidades para seguir innovando y evolucionando su alcance y funcionalidades.
>
> Además, este desarrollo es una excelente plataforma para capacitar y entrenar al equipo de despacho, permitiendo comparar criterios y estrategias, lo que fortalece nuestra preparación ante situaciones reales.
>
> Sin duda, este recomendador es un aporte relevante que potencia e impulsa nuestra capacidad de respuesta y marca un prometedor primer paso hacia el futuro.”

— Cristhian Ovando, Product Owner de GesFire [10]

Interfaz responsiva

El sistema se probó en dispositivos típicos de la empresa: laptops Windows, teléfonos móviles y tablets. La interfaz React, optimizada para responsividad, funcionó correctamente en todos los casos, ajustándose a diferentes resoluciones.

Figura 13. Diseño responsivo del frontend (elaboración propia).

5.3.1 Ajustes Basados en Retroalimentación de Usuarios

Durante las pruebas con personal de CMPC, se recopiló retroalimentación valiosa que permitió identificar oportunidades para mejorar la intuitividad y usabilidad de la interfaz desarrollada en React. Aunque no se buscó alinear la estética con otras aplicaciones web de la empresa, los comentarios de los usuarios guiaron ajustes específicos para hacer el sistema más accesible y fácil de usar. Los cambios implementados incluyeron:

- **Botones más intuitivos:** Inicialmente, los botones solo usaban íconos, lo que generaba confusión entre los usuarios. Basado en el feedback, se reemplazan por botones con texto descriptivo (e.g., "Agregar Foco" en lugar de un simple ícono), facilitando la comprensión de su funcionalidad.
- **Mapa de selección de coordenadas:** Respondiendo a las necesidades expresadas por los usuarios, se añadió un mapa interactivo para seleccionar coordenadas directamente, mejorando la experiencia al permitir una interacción visual y práctica en lugar de depender únicamente de entradas manuales.
- **Ajustes de diseño y texto:** Se optimizaron los tamaños de texto para mejorar la legibilidad, se agregaron textos descriptivos que reflejan el estado de la aplicación (e.g., "Cargando datos" o "Recomendaciones generadas"), y se realizan ajustes generales en el diseño para hacerlo más intuitivo, como reorganizar elementos clave para un flujo de trabajo más natural.

5.3.2 Análisis Técnico con Lighthouse

El desempeño técnico de la interfaz fue evaluado mediante Lighthouse (versión 12.3.0), en un entorno emulado de escritorio (Chromium 133.0.0.0). El informe proporciona métricas clave de rendimiento, accesibilidad, mejores prácticas y SEO (Search Engine Optimization), complementando la evaluación cualitativa de los usuarios. Los resultados principales se resumen a continuación:

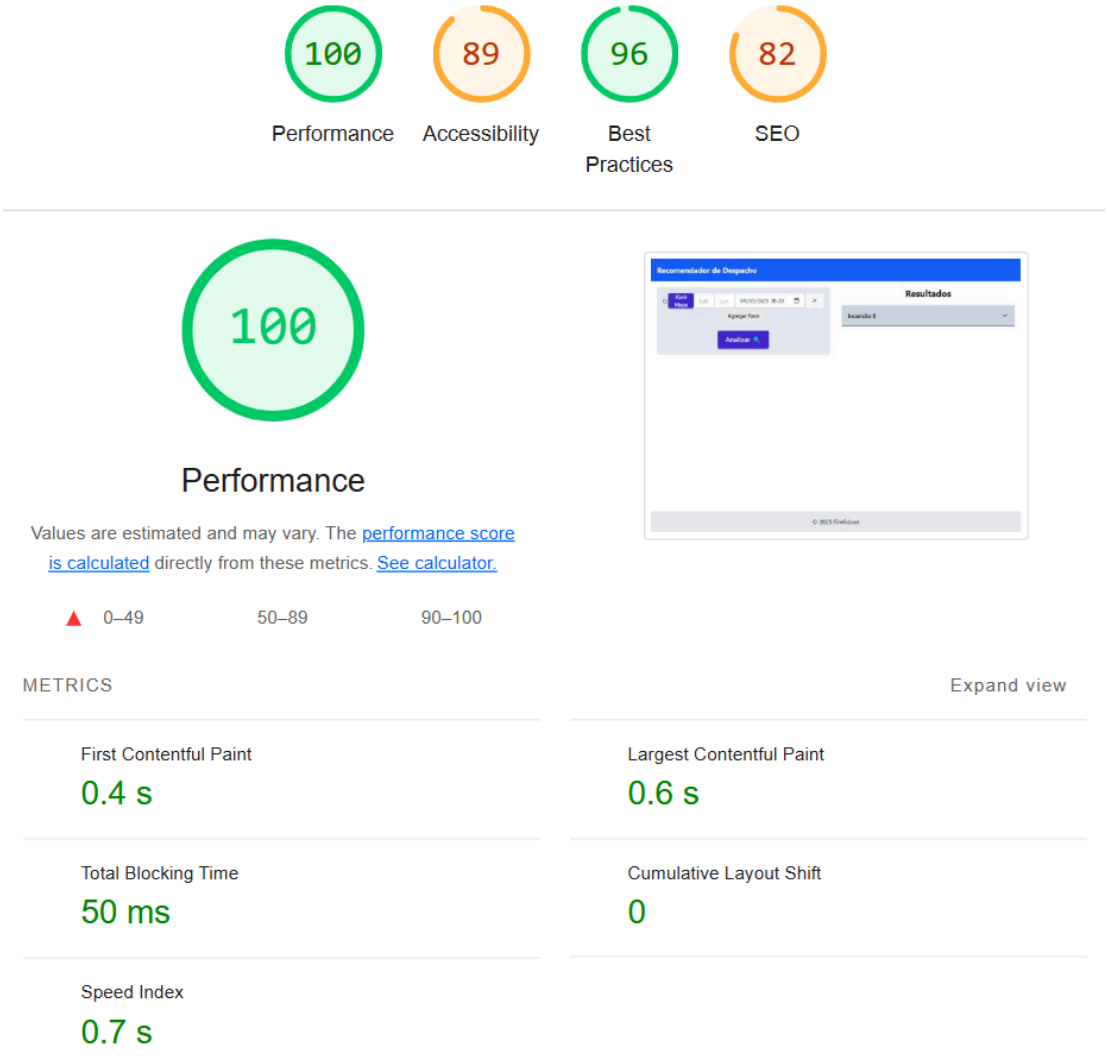


Figura 14. Puntuación de Google LightHouse (elaboración propia).

6. Conclusiones

El proyecto culminó en el desarrollo de un sistema integrado que agiliza el despacho de recursos para el combate de incendios forestales en la central de incendios de CMPC. Este sistema no solo aborda las deficiencias del enfoque anterior basado en una matriz de decisión estática, sino que también introduce una solución tecnológica que mejora la eficiencia operativa y la accesibilidad para los usuarios. A través de la integración de herramientas y tecnologías modernas (como el modelo de recomendación con la metaheurística GRASP, el servidor FastAPI y la interfaz gráfica en React), el sistema se posiciona como una herramienta robusta, intuitiva y adaptable, diseñada para responder a las demandas de un entorno de alta presión como las emergencias forestales. A continuación, se detallan los logros alcanzados, los desafíos superados, las lecciones aprendidas y las oportunidades identificadas para trabajos futuros, proporcionando una reflexión sobre el impacto y el potencial del proyecto.

Logros y Cumplimiento de Objetivos

El desarrollo del sistema propuesto permitió alcanzar de manera satisfactoria los objetivos establecidos al inicio del proyecto. Se logró diseñar, implementar y validar una herramienta funcional orientada a asistir en la toma de decisiones en el contexto de emergencias por incendios forestales. Desde el punto de vista técnico, se consolidó una arquitectura compuesta por GRASP (C++), FastAPI (Python) y React (JavaScript), que permitió combinar eficiencia computacional, comunicación fluida y una experiencia de usuario clara e intuitiva. La visualización geográfica interactiva, basada en Leaflet, junto con las métricas generadas (como área quemada estimada y tiempo probable de extinción), ofrecieron información de alto valor para los operadores.

Durante el proceso de validación, el sistema fue presentado a despachadores y a cargos de la gerencia de protección forestal, quienes manifestaron una evaluación positiva respecto al diseño, la utilidad y la usabilidad de la plataforma. En particular, la interfaz fue destacada por su accesibilidad, incluso para usuarios sin experiencia previa, lo que refuerza el cumplimiento del objetivo de desarrollar una herramienta intuitiva y de rápida adopción.

A pesar de los resultados positivos obtenidos en términos técnicos y de usabilidad, el sistema no fue finalmente incorporado en el flujo de trabajo operativo. Esta situación se debió a múltiples factores: la preferencia de los despachadores por trabajar con una sola plataforma centralizada; el ritmo de trabajo exigente que dificulta la incorporación de nuevas herramientas; la falta de costumbre para cambiar entre distintas aplicaciones; y la ausencia de permisos para integrar la herramienta a los sistemas internos, producto del hermetismo institucional en torno a sus tecnologías. Adicionalmente, las bases de datos entregadas por la empresa no estaban actualizadas con la temporada de incendios en curso, lo que afectó la precisión de las recomendaciones generadas y redujo la confianza en la herramienta.

No obstante, todos los objetivos del proyecto fueron cumplidos en términos de desarrollo, integración y validación técnica. El sistema demostró tener un alto potencial de impacto en la mejora del proceso de despacho, y constituye una base sólida para futuras implementaciones que, con una estrategia adecuada de integración y actualización de datos, podrían superar las barreras actuales. El trabajo realizado representa un aporte concreto tanto a nivel tecnológico como organizacional, al abrir nuevas posibilidades para el uso de sistemas inteligentes en la gestión de emergencias.

Desafíos Superados

El desarrollo del sistema enfrentó varios desafíos técnicos y operativos que requirieron soluciones específicas para garantizar su éxito. Un desafío significativo fue la sincronización de datos con Firestore bajo las restricciones impuestas por CMPC, como el acceso limitado a ciertas bases de datos internas y la necesidad de mantener la información actualizada desde Google Sheets. Se resolvió implementando un sistema de sincronización automática que actualiza Firestore cada 1 hora mediante un trigger de Google Apps Script basado en el tiempo, asegurando disponibilidad incluso con conectividad limitada. Además, la integración entre lenguajes (C++ para GRASP, Python para FastAPI, JavaScript para React) presentó dificultades de interoperabilidad, superadas mediante un puente de comunicación basado en scripts de Python que manejan la conversión de datos entre JSON y formatos nativos de C++.

Impacto y Beneficios Observados

El impacto del sistema trasciende los objetivos técnicos, ofreciendo beneficios operativos y estratégicos para CMPC. A nivel operativo, la reducción de tiempos de respuesta y el soporte a despachadores inexpertos optimizan el uso de recursos y minimizan las pérdidas forestales, alineándose con los intereses económicos y ambientales de la empresa. El sistema no solo agiliza las decisiones, sino que también reduce el estrés en situaciones críticas, un factor cualitativo difícil de medir pero esencial para el bienestar del personal. El diseño intuitivo (ilustrado en la sección 4.4, que muestran el mapa interactivo, el panel de métricas y el flujo de ingreso de datos) asegura que el sistema sea una herramienta práctica y confiable, lista para su implementación en entornos reales. Comparado con soluciones existentes en Chile, como Kitral, este sistema destaca por su enfoque en tiempo real y su accesibilidad, marcando una diferencia significativa en el contexto local.

Lecciones Aprendidas

El proceso de desarrollo y evaluación dejó varias lecciones valiosas, ampliando la experiencia de trabajo y permitiendo aprender de ellas. Entre el aprendizaje obtenido se puede destacar:

- **Acceso a bases de datos:** Por políticas de la empresa todas las personas que no fuesen de la unidad de data science de CMPC no tienen acceso a las bases de datos.

Esto genera dificultad al momento de obtener información, ya que esta se entrega en formato xlsx o en Google Sheets. Gracias a esto se utiliza la API REST de Firebase para conectar Google Sheets con Firestore.

- **Sistema de coordenadas utilizadas por la central de incendios:** En CMPC se utiliza el formato grados, minutos y segundos, esto dificulta el proceso de trabajo, ya que las librerías usan coordenadas cartesianas. Por esto se tuvo que programar una conversión en javascript que transforma la data ingresada por los usuarios y la envía en formato decimal.
- **Acceso a plataformas de desarrollo:** CMPC no entrega acceso a Google Cloud ni Firebase, por lo que se utilizan planes gratuitos para la realización de este proyecto adaptando el desarrollo a los límites que estos planes entregan.
- **Acceso a computadores:** El computador entregado para trabajar, no tiene permisos de instalación de programas. Por esto, las únicas opciones para desplegar frontend son Github Pages o Google Apps Script, se decide utilizar este último ya que también permite controlar el acceso a sólo personal de CMPC. Gracias a esto, se tuvo la oportunidad de desplegar aplicaciones interactivas utilizando únicamente Google Workspace y Google Apps Script.
- **Acceso a servidores:** La empresa no entrega acceso a servidores, por lo que como solución se utiliza el servidor de la universidad. Por esto se tomó la decisión de utilizar las tecnologías mencionadas anteriormente, pudiendo aprender acerca de alternativas gratuitas para desplegar la aplicación.
- **Usuarios finales:** involucrar a los despachadores de CMPC desde las primeras etapas es de gran relevancia para este proyecto, ya que su retroalimentación durante el desarrollo identificó mejoras críticas, como alertas más visibles y botones explicativos, que no se habrían considerado desde una perspectiva puramente técnica.
- **Integración de tecnologías (C++, Python, JavaScript):** Para realizar esto se tiene una planificación cuidadosa de la comunicación entre componentes, destacando la utilidad de estándares como JSON, APIs REST y así poder mantener la cohesión entre estos.
- **Metodologías ágiles (Kanban y TDD):** El uso de estas tecnologías resulta esencial para manejar los plazos ajustados y garantizar la calidad, reforzando la efectividad de estas prácticas en proyectos de alta exigencia.

Trabajos Futuros

Aunque el sistema satisface los objetivos actuales, se identificaron diversas oportunidades para su evolución y optimización en futuros desarrollos. A continuación, se detallan las principales líneas de trabajo propuestas:

- **Incorporación de Machine Learning:** Una dirección prometedora es potenciar las predicciones del sistema mediante aprendizaje automático. Por ejemplo, un modelo supervisado podría entrenarse con datos históricos de incendios de CMPC para predecir la propagación del fuego con mayor precisión, integrando variables como patrones climáticos estacionales, tendencias de combustibilidad del terreno y otros

factores contextuales. Esto complementaría las capacidades heurísticas de GRASP, elevando la exactitud de las recomendaciones.

- **Simulaciones Dinámicas de Propagación:** La implementación de simulaciones en tiempo real, permitiría visualizar en el mapa interactivo la evolución proyectada de un incendio según variables como el tiempo, el viento y el combustible disponible. Esta funcionalidad transformaría el sistema de una herramienta reactiva a una proactiva, facilitando la planificación preventiva y la toma de decisiones estratégicas.
- **Expansión a Dispositivos Móviles e Integraciones:** Se plantea desarrollar aplicaciones nativas para iOS y Android, mejorando la portabilidad y accesibilidad del sistema. Asimismo, la integración con plataformas como Gesfire o sistemas de monitoreo de incendios como Ororatech enriquecería las fuentes de datos y la interoperabilidad, fortaleciendo su utilidad operativa.
- **Caching del Lado del Servidor:** Implementar un sistema de caching con Redis reduciría la latencia en consultas frecuentes (e.g., inventario de recursos o datos meteorológicos), optimizando el tiempo de respuesta en escenarios de alta demanda.
- **Sistema de Logging:** Incorporar un sistema de registro histórico de las recomendaciones generadas asociadas al usuario, la cual permitiría analizar patrones de uso y decisiones pasadas, habilitando análisis de datos para mejorar el modelo y evaluar su efectividad a lo largo del tiempo.

Estas mejoras consolidarían al sistema como una solución integral y adaptable, alineada tanto con las necesidades operativas inmediatas como con los desafíos futuros del manejo de incendios forestales.

Reflexión Final

El proyecto culminó en el desarrollo de un sistema integral que optimiza el despacho de recursos para la gestión de incendios forestales en la central de incendios de CMPC, superando las limitaciones del enfoque anterior basado en una matriz estática y marcando un avance significativo en la eficiencia operativa y la accesibilidad para los usuarios. La integración efectiva del algoritmo de recomendación, FastAPI y React resultó en una solución robusta, intuitiva y adaptable, que agiliza los tiempos de respuesta y apoya a despachadores de todos los niveles, especialmente a los menos experimentados, mediante recomendaciones objetivas y una interfaz gráfica accesible. Los desafíos técnicos, como la integración con el solver para su ejecución en tiempo real y la sincronización de datos con Firestore, fueron abordados con soluciones innovadoras que fortalecieron el diseño final. Este sistema no solo reduce las pérdidas forestales al mejorar la contención en las etapas iniciales de un incendio, sino que también aporta valor estratégico al proporcionar métricas claras como área quemada y tiempo estimado de extinción, que incrementan la precisión y confianza en las decisiones. Con un impacto inmediato en las operaciones de CMPC y un potencial claro para evolucionar mediante machine learning, simulaciones dinámicas y optimizaciones técnicas como caching y logging. Esta herramienta se consolida como una solución escalable y práctica, preparada para enfrentar los retos actuales y futuros en el combate de emergencias forestales en un contexto de creciente complejidad ambiental.

7. Bibliografía

[1] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[2] Open-Meteo, "Docs," *Open-Meteo.com*. [En línea]. Disponible en: https://open-meteo.com/en/docs [Accedido: 22-abr-2025].

[3] Leaflet, "Leaflet — an open-source JavaScript library for interactive maps," *Leafletjs.com*. [En línea]. Disponible en: https://leafletjs.com/ [Accedido: 22-abr-2025].

[4] S. Ramírez, "FastAPI," *Tiangolo.com*. [En línea]. Disponible en: https://fastapi.tiangolo.com/ [Accedido: 22-abr-2025].

[5] Meta, "React," *React.dev*. [En línea]. Disponible en: https://react.dev/ [Accedido: 22-abr-2025].

[6] J. Peret, "¿Cómo se propaga el fuego? Factores clave detrás de su propagación," *Dabedan*, 25-sep-2024. [En línea]. Disponible en: https://www.dabedan.com/como-se-propagan-incendios-factores-clave-prevencion/ [Accedido: 22-abr-2025].

[7] P. Pinacho, R. Asín, D. Neira, A. Irribarra, e I. Osorio, *Algoritmo despachador de recursos para apagado de incendios*, 2021.

[8] K. Beck, *Test-Driven Development: By Example*, Boston, MA, USA: Addison-Wesley, 2002.

[9] A. Varas *et al.*, *Informe Asesoría Fortalecimiento de la Prevención y Gestión de Incendios*, 2022.

[10] Centro de Datos e Inteligencia Artificial UdeC, "Gesfire: el software que agiliza la toma de decisiones en incendios rurales," *CDIA UdeC*, 27-ene-2023. [En línea]. Disponible en: https://cdia.udec.cl/2023/01/27/gesfire-el-software-que-agiliza-la-toma-de-decisiones-en-incendios-rurales/ [Accedido: 28-abr-2025].

[11] Docker Inc., "Docker: Empowering App Development for Developers," *Docker.com*. [En línea]. Disponible en: https://www.docker.com/ [Accedido: 19-jul-2025].

[12] Universidad de Chile, "Cell2Fire: sistema predice dónde pueden comenzar los incendios, su comportamiento y cómo combatirlos," *U. de Chile*, feb. 2024. [En línea]. Disponible en: https://uchile.cl/noticias/213580/cell2fire-sistema-predice-donde-partiran-los-proximos-incendios [Accedido: 19-jul-2025].

[13] Mozilla, "MVC," *MDN Web Docs*, [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Glossary/MVC [Accedido: 19-jul-2025].

[14] CMPC, "Compañía Manufacturera de Papeles y Cartones," *CMPC.cl*. [En línea]. Disponible en: https://www.cmpc.cl/ [Accedido: 19-jul-2025].

[15] Firebase, "Firebase: plataforma de desarrollo móvil y web," *Firebase.google.com*. [En línea]. Disponible en: https://firebase.google.com/ [Accedido: 19-jul-2025].

Anexos

Anexo A: Código Fuente (repositorios)

- **Frontend (React):** https://github.com/vcser/solver-front
- **Backend (Python):** https://github.com/vcser/solver-api
- **Modelo (C++):** https://github.com/vcser/solver-modelo

Anexo B: Historias de Usuario

Nombre	[H1] - Ingresar datos de un incendio
Descripción	Como despachador, Quiero ingresar las coordenadas y el timestamp de un incendio Para que el sistema pueda procesar estos datos y generar recomendaciones de despacho.
Criterios de aceptación	• Permitir ingresar una o más coordenadas manualmente • Permitir seleccionar en un mapa popup que autocompleta los valores con las coordenadas correspondientes • Validar que las coordenadas ingresadas sean geográficamente válidas • Ingresar un timestamp asociado a cada conjunto de coordenadas • Es posible cambiar el sistema de coordenadas con un botón • Se muestran la cantidad correspondiente de inputs para cada sistema de coordenadas • La app recordará la configuración del sistema de coordenadas preferido

Nombre	[H2] - Mostrar recomendaciones de despacho
Descripción	Como despachador, Quiero visualizar las recomendaciones de despacho generadas Para saber qué recursos asignar al incendio.
Criterios de aceptación	● Validar datos ingresados antes de enviar formulario ● Mostrar por cada recurso: ID, metros de línea, costo ● Incluir las métricas estimadas para el incendio (tiempo de respuesta, impacto estimado, etc.)

Nombre	[H3] - Integrar bases de datos meteorológicas
Descripción	Como sistema, Quiero integrar las bases de datos meteorológicas existentes Para que el sistema pueda considerar factores climáticos en sus recomendaciones.
Criterios de aceptación	● Acceder a la base de datos meteorológica open meteo en tiempo real ● Usar las variables climáticas relevantes (vel viento, temperatura, humedad, dir viento) ● Si open meteo no está disponible, reintentar con base meteorológica interna ● Respuesta correctamente formateada en jsend

Nombre	[H4] - Integrar bases de datos de recursos
Descripción	Como sistema, Quiero Integrar las bases de datos de los recursos de combate Para que el sistema pueda considerarlas en las recomendaciones.
Criterios de aceptación	● Al recibir una consulta por los recursos, enviar una lista de todos los recursos con sus propiedades: ○ ID ○ Tipo ○ Horas de trabajo ○ Latitud y Longitud ○ Estado (disponible/no disponible) ○ Incendio asignado ○ Está agrupado (debe ser 1) ○ Lista de ETAs por cada incendio ● Respuesta correctamente formateada en jsend

Nombre	[H5] - Integrar bases de datos geográficas
Descripción	Como sistema, Quiero Integrar las bases de datos geográficas existentes Para que el sistema pueda considerarlas en las recomendaciones.
Criterios de aceptación	● Recibir requests con coordenadas y devolver: ○ Pendiente ○ Modelo de combustible ○ Factor VPL ○ Valor por rodal ○ Distancia zona poblada más cercana ● Respuesta correctamente formateada en jsend

Nombre	[H6] - Administrar geografía
Descripción	Como Administrador Quiero actualizar archivos correspondientes a las bases de datos geográficas utilizadas por el modelo Para mantener actualizados los datos y que el modelo sea más preciso.
Criterios de aceptación	● Hay una pagina web en la que puedo cargar archivos ● Es posible indicar al sistema que funcion cumple el archivo en el modelo

Nombre	[H7] - Administrar recursos
Descripción	Como Administrador Quiero añadir, eliminar o modificar la lista de recursos de la temporada Para mantener actualizados los datos y que el modelo sea más preciso
Criterios de aceptación	● Hay una página web en la que puedo editar los recursos de la temporada

Nombre	[H8] - Recopilar datos a partir de coordenadas
Descripción	Como sistema, Quiero recopilar todos los datos necesarios para alimentar el modelo a partir de las coordenadas entregadas Para poder enviar esta consulta al modelo.
Criterios de aceptación	● A partir de coordenadas recuperar: ○ Datos geográficos ○ Datos recursos ○ Datos meteorológicos ● Construir entrada para el modelo

Anexo C: Informe Solver (Resumen)

Resumen técnico del modelo [7], incluyendo parámetros y funciones objetivo.

5 MODELO ABSTRACTO

En esta sección se define el modelo de mitigación de incendios desde una perspectiva matemática abstracta para luego presentar los algoritmos que siguen los principios aquí descritos. Para esto definimos parámetros y el Modelo General propuesto para el problema.

5.1 PARÁMETROS

Los parámetros que se pueden obtener ayudan a describir y entender los factores controlables y los que no son controlables. Estos los podemos dividir en:
Condiciones geográficas y de entorno:

- Accidentes geográficos Λ
- Tipo de Combustible Θ

En donde, tenemos que la composición geográfica es su combinación:

$$\mathcal{G} = \mathcal{G}(\Lambda, \Theta) \tag{9}$$

Recordemos que tanto Λ como Θ son rasters (o polígonos proyectables en un plano), por lo que la función $\mathcal{G}()$ se aplica sobre rasters.

Condiciones meteorológicas del momento de los incidentes:

- Temperatura T
- Presión P
- Velocidad (y dirección) del viento $\vec{\boldsymbol{v}}$
- Humedad $\mathcal{H}$

Estas las podemos agrupar en las llamadas variables meteorológicas:

$$\mathcal{M} = \mathcal{M}(T, P, \vec{\boldsymbol{v}}, \mathcal{H}) \tag{10}$$

Luego, dado un incendio, éste se puede caracteriza por:

- Posición (Latitud La y Longitud Lo)
- Vector de propagación $\vec{\boldsymbol{p}}$

El vector de propagación se logra obtener en función de:

$$\vec{\boldsymbol{p}} = \mathcal{I}(< La, Lo >, \mathcal{G}, \mathcal{M}) \tag{11}$$

la posición del incidente y las variables meteorológicas y geográficas de la zona colindante.

5.2 MODELO PROPUESTO

Para lograr mitigar los incendios que se concentren en un periodo de trabajo, se tiene en consideración diferentes recursos, los cuales podemos agrupar en un conjunto de recursos $\mathcal{R}$, contándolos a la vez, como recurso $r = 1, \ldots, R$, siendo $R = |\mathcal{R}|$ los distintos tipos de recurso.

El problema, por ende, consiste en encontrar una asignación $\mathcal{X} = \{x_{ri}\}_{r=1,\ldots,R;i=1,\ldots,n}$, en la que $\S_{\nabla\rangle}$ significa que el recurso r es asignado para combatir el incendio i. A la vez, necesitamos estimar cuánto tiempo será asignado al combate de cada incendio, es decir, $\mathcal{T} = \{td_{ri}\}_{r=1,\ldots,R;i=1,\ldots,n}$, en donde, td_{ri} corresponde al tiempo de combate del incendio i con el recurso r.

El problema se definirá como la conjunción de dos funciones objetivo, las dos expresadas en unidades monetarias, por un lado, y con miras al nuevo *ethos* de la empresa, lo que se quiere es maximizar el patrimonio salvado (proyección de incendio contra mitigación), y, por otro lado, la minimización de los costos asociados a esta asignación, es decir:

$$\text{máx} \quad B(\mathcal{X}, \mathcal{T}) + \text{mín} \quad C(\mathcal{X}, \mathcal{T}) \tag{12}$$

Donde B es el beneficio del patrimonio *salvado*, y C los costos incurridos en el esfuerzos de mitigación.

Los beneficios y costos se pueden disgregar como los costos individuales producidos por cada incendio i:

$$B(\mathcal{X}, \mathcal{T}) = \sum_{i=1}^{n} B_i^p(\mathcal{X}^i, \mathcal{T}^i) \tag{13}$$

$$C(\mathcal{X}, \mathcal{T}) = \sum_{i=1}^{n} C_i(\mathcal{X}^i, \mathcal{T}^i) \tag{14}$$

Para determinar el beneficio asociado a salvar el patrimonio dado un incendio (B_i^p) necesitamos aproximar el tiempo de mitigación de los incendios, por lo que necesitamos una función de determinación del tiempo que demora esto. Junto con ello, necesitamos una herramienta para predecir la expansión del incendio dado un tiempo estimado, denominado T^s. Para llevarlo a cabo, se utiliza como común denominador la formación de líneas de corta fuego, las cuales frenan el avance del fuego, pero que su construcción, de largo variable, depende de las características del incendio (propagación) y el tiempo en que se puede efectuar la respuesta efectiva. Para esto definimos las funciones L, P, T y A, como la extensión de línea generable, el perímetro del incendio (determinado por la isócrona), la función de tiempo de freno del fuego y el área (raster) de patrimonio quemada, dado un tiempo t y las características de propagación, respectivamente:

$$L = L(\vec{\boldsymbol{p}}, \mathcal{X}, \mathcal{T}, t) \tag{15}$$

La función longitud de línea L recibe las características del incendio (propagación), las asigna-

ciones y tiempos de respuesta y un tiempo dado t. Entrega el largo de la línea necesaria para el freno del (los) incendio(s) en estudio.

$$P = P(\vec{\boldsymbol{p}}, t) \tag{16}$$

La función del perímetro (isócrona) del incendio P recibe las características del incendio (propagación) y un tiempo dado t. Entrega el perímetro envolvente del incendio en el tiempo t.

$$T = T(L, P) \tag{17}$$

T es una función de estimación del tiempo de freno de un incendio que se crea en función de la línea L de freno y una función que determina el perímetro P.

$$A = A(\vec{\boldsymbol{p}}, T) \tag{18}$$

A recibe las características y tiempo de propagación y devuelve un área que determinar el patrimonio perdido por los incendios (función de aproximación del simulador).

Luego, podemos usar la ecuación anterior, pero dado nuestro tiempo de prueba T^s para calcular el área de expansión y hacer la resta específica, es decir:

$$B^p = A(\vec{\boldsymbol{p}}, T^s) - A(\vec{\boldsymbol{p}}, T) \tag{19}$$

Como se puede notar, está función entrega un tiempo pero a la vez:

$$t^{'} = T = T(L(\vec{\boldsymbol{p}}, \mathcal{X}, \mathcal{T}, t), P(\vec{\boldsymbol{p}}, t)) \tag{20}$$

Depende del tiempo t, por lo tanto, se debe hacer un cálculo, por ejemplo por bisección u otro método, para determinar el valor de un tiempo $t^{'}$ tal que el valor de línea realizable es, al menos, el necesario para abarcar un perímetro de fuego, dada la propagación $\vec{\boldsymbol{p}}$, siendo de este modo una solución factible, y por ende, B lograble.

Ahora, podemos reescribir el beneficio por patrimonio salvado del incendio i como:

$$B_i^p = A(\vec{\boldsymbol{p}}, T^s) - A(\vec{\boldsymbol{p}}, T) \tag{21}$$

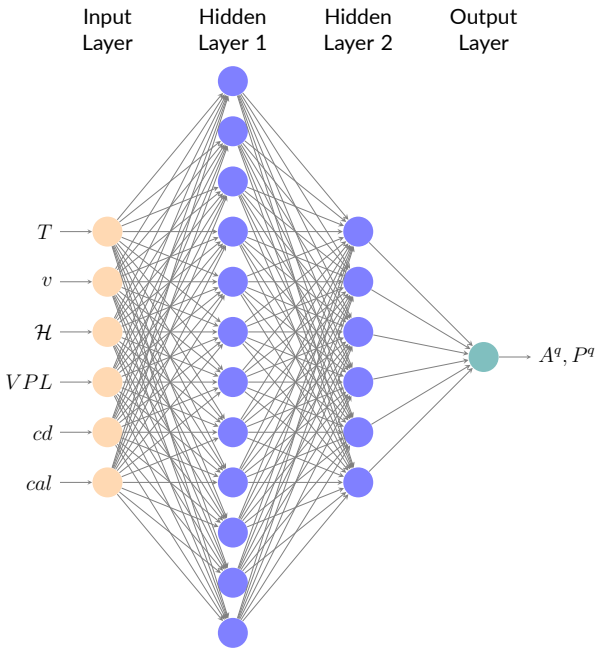
Finalmente, podemos reescribir el modelo propuesto como:

$$\text{máx} \sum_{i=1}^{n} B_i^p(A_i(\vec{\boldsymbol{p}}, T)) + \text{mín} \sum_{i=1}^{n} C_i^r(\mathcal{X}^i, \mathcal{T}^i) \tag{22}$$

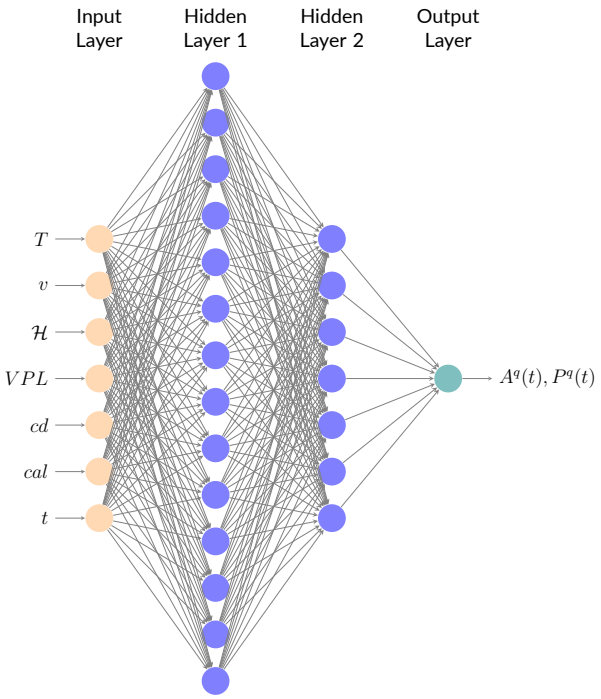
Sujeto a que solo se puede utilizar recursos asignados para mitigar el incendio i.

Para poder ocupar este modelo de forma eficiente en producción, es necesario que los parámetros sean alimentados con prontitud y que las funciones $B_i^p(A_i(\vec{\boldsymbol{p}}, T))$ sean calculadas rápidamente. Para este último punto se pueden obtener distintas aproximaciones (Discutidas en la Subsección 7.2.2):

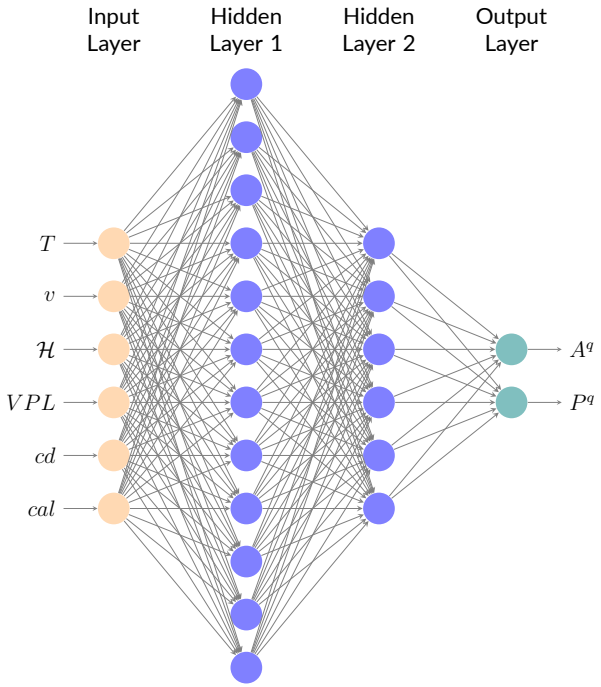
- Utilizar Kitral para crear una aproximación del área quemada
- Utilizar *Wildfire Analyst* para encontrar aproximaciones a la expansión
- Utilizar un algoritmo de *machine learning* (como redes neuronales) para la aproximación



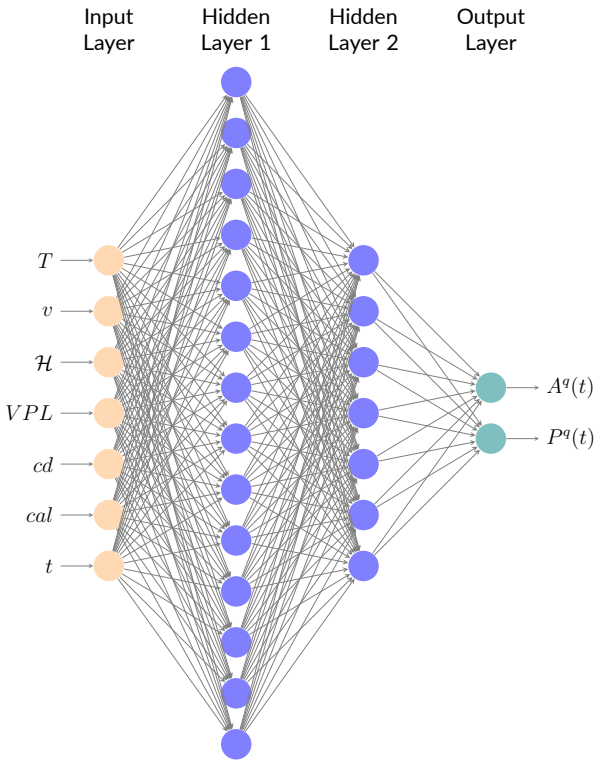
(a) Red neuronal propuesta 1



(b) Red neuronal propuesta 2



(c) Red neuronal propuesta 3



(d) Red neuronal propuesta 4

Figura 5.1: Redes neuronales propuestas.

6 ALGORITMOS

En esta sección desarrollamos los distintos enfoques algorítmicos implementados en el Solver. Una vez más, estos enfoques han sido desarrollados teniendo en mente restricciones computacionales fuertes sobre el tiempo de ejecución del algoritmo, buscando antes una solución buena rápidamente entregada que una óptima fuera de tiempo.

6.1 HEURÍSTICA

Para enfrentar problemas de optimización combinatoria de forma aproximada, unos de los primeros pasos corresponde a la elaboración de algoritmos heurísticos basados en el problema. Unas de las ventajas de esto, radican en que son algoritmos rápidos y que pueden ser considerados como componentes en la producción de posteriores soluciones metaheurísticas más avanzadas.

Con el objetivo de encontrar una solución inicial factible, se generan las siguientes heurísticas greedy:

- ***Greedy* por distancia:**

  Paso 1: Asignar prioridad a los incendios de acuerdo al patrimonio quemado en 6 horas sin control.

  Paso 2: Ordenar lista de incendios de mayor a menor prioridad e iterar **para cada** incendio i.

  Paso 3: Si los recursos disponibles no consiguen contener el incendio i, volver a *Paso 1* perturbando aleatoriamente la prioridad de los incendios.

  Paso 4: Ordenar lista de recursos disponibles de menor a mayor distancia al incendio i.

  Paso 5: Asignar recurso disponible más cercano al incendio i, siempre y cuando sea compatible con los recursos ya asignados y el incendio.

  Paso 6: Si el incendio i es contenido, volver a *Paso 2* y proseguir con el siguiente incendio.

  Paso 7: Si no quedan recursos disponibles y el incendio no está contenido, volver a *Paso 4* aplicando perturbaciones al orden de los recursos.

- ***Greedy* por afinidad:**

  Paso 1: Asignar prioridad a los incendios de acuerdo al patrimonio quemado en 6 horas sin control.

  Paso 2: Ordenar lista de incendios de mayor a menor prioridad e iterar **para cada** incendio i.

Paso 3: Si los recursos disponibles no consiguen contener el incendio i, volver a *Paso 1* perturbando aleatoriamente la prioridad de los incendios.

Paso 4: Ordenar lista de recursos disponibles de acuerdo a la afinidad, correspondiente a la cantidad de metros de linea de contención que generan durante la primera hora de propagación del incendio i dividido por el costo de combate.

Paso 5: Asignar recurso disponible más afín al incendio i, siempre y cuando sea compatible con los recursos ya asignados y el incendio.

Paso 6: Si el incendio i es contenido volver a *Paso 2* y proseguir con el siguiente incendio.

Paso 7: Si no quedan recursos disponibles y el incendio no está contenido, volver a *Paso 4* aplicando perturbaciones al orden de los recursos.

- ***Greedy* aleatorizado:**

  Paso 1: Asignar prioridad a los incendios de acuerdo al patrimonio quemado en 6 horas sin control.

  Paso 2: Escoger un incendio i al azar, de manera que aquellos de mayor prioridad tienen mayor probabilidad de ser escogidos.

  Paso 3: Si los recursos disponibles no consiguen contener el incendio i, volver a *Paso 1* perturbando aleatoriamente la prioridad de los incendios.

  Paso 4: Calcular para cada recurso disponible la afinidad, correspondiente a la cantidad de metros de linea de contención que generan durante la primera hora de propagación del incendio i dividido por el costo de combate.

  Paso 5: Escoger y asignar aleatoriamente un recurso al incendio i, siempre y cuando sea compatible con los recursos ya asignados y el incendio. Aquellos recursos con mayor afinidad tienen mayor probabilidad de ser escogidos.

  Paso 6: Si el incendio i es contenido volver a *Paso 2* y proseguir con el siguiente incendio.

  Paso 7: Si no quedan recursos disponibles y el incendio no está contenido, volver a *Paso 4* aplicando perturbaciones al orden de los recursos.

6.2 METAHEURÍSTICA

GRASP

Para encontrar una mejor solución durante la ejecución del programa, se utiliza la metaheurística *Greedy Randomized Adaptative Search Procedure* (GRASP), (Feo and Resende, 1995). En esta, se genera una solución *greedy*, particularmente con el *greedy* aleatorizado, y se mejora

esta solución a través de búsqueda local (Esto no considera los enfoques *greedy* explicados en la subsección anterior). Este comportamiento está ilustrado en el Algoritmo 1. La principal razón para utilizar GRASP se asocia a que es una de las metaheurísticas más ligeras y simples, lo cual facilita que disponga de distintas soluciones en un corto tiempo de ejecución.

Algoritmo 1 GRASP.

Input: t_{max} ▷ Tiempo máximo de resolución
repeat
  $sol = RandomizedGreedy(s)$
  $sol = Busqueda\_Local(s)$
  if $f(sol) > f(bestSol)$ **then**
    $bestSol = sol$
  end if
until $t \geq t_{max}$ ▷ t tiempo actual
Output: $bestSol$

COMPONENTE DE BÚSQUEDA LOCAL

Dada una solución S, su vecindad queda definida como aquellas soluciones que se consiguen al tomar un recurso ocioso en S y asignarlo a algún incendio, manteniendo factibilidad. Con esto, en un paso de búsqueda local se recorre la vecindad de la solución y se retorna la solución vecina que tenga mayor beneficio asociado.

6.3 FUERZA BRUTA

Para poder hacer un testeo conciso de la precisión de nuestros algoritmos heurísticos y metaheurísticos se ha desarrollado un algoritmo de fuerza bruta para testear la calidad de la solución de los algoritmos no exactos, con las instancias pequeñas presentes en nuestro dataset. En este, se evalúa cada posible asignación factible y se retorna aquella que tenga mayor beneficio.

Los Algoritmos de Fuerza Bruta (o *brute force* en inglés) evalúan todas las asignaciones de cada recurso a un incendio (o ninguno) y reportar la mejor. Si se tiene n focos y k recursos, la cantidad de asignaciones posible es $(n+1)^k$. Esto significa que, por ejemplo, si consideramos un foco y 100 recursos, hay que probar más de $1,2 * 10^{30}$ combinaciones, lo cual no es factible en términos computacionales. Tampoco si se considera un caso real que dispone de 50 recursos, aproximadamente.