.UBAfiuba
FACULTAD DE INGENIERÍA

Optimización con inteligencia artificial del despacho de riego solar en cultivos de pistacho

Ing. Tomás Corteggiano

Carrera de Especialización en Inteligencia Artificial

Director: Dr. Ing. Carlos Larisson

Jurados:

Mg. Ing. José Luis Hernández (UNRC)
Dr. Ing. David De Yong (UNRC)
Mg. Ing. Sebastián Tosco (UNRC)

Ciudad de Mendoza, abril de 2026

Resumen

La presente memoria describe el desarrollo de un sistema avanzado para la optimización del riego en una plantación de pistachos, con el fin de generar un cronograma de despacho de riego óptimo para una empresa agrícola. El sistema busca balancear objetivos claves como la uniformidad del riego, la eficiencia del bombeo y la sostenibilidad del acuífero mediante una función de costos unificada.

Para abordar la problemática, se empleó modelado físico, técnicas de inteligencia artificial y un algoritmo genético para resolver el problema de optimización asociado al despacho de riego.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se presenta una introducción general del trabajo. Se describe la problemática fundamental del riego solar y se analiza el estado del arte en las áreas claves del modelado de sistemas y su optimización. Finalmente, se expone la justificación de la solución propuesta y se delimitan los objetivos y el alcance del desarrollo.

1.1. Contexto y problemática del riego solar

La implementación de sistemas de riego 100 % solares es una solución tecnológica de amplio uso en el sector agrícola, particularmente en plantaciones que operan de forma autónoma (es decir, en modo *off-grid* [1]), donde la red eléctrica de distribución pública no está disponible o su conexión resulta económicamente inviable. Estas instalaciones se componen típicamente de una capa física, que cuenta con una bomba sumergible y válvulas de campo, y una capa de control y adquisición, que cuenta con un variador de frecuencia (VFD) y un PLC que comanda dicho hardware. En la operación estándar, esta arquitectura de control es robusta para ejecutar comandos, pero carece de la capa superior de optimización que define el plan de operación.

Esta ausencia de una capa de gestión inteligente traslada por completo la carga de optimización al operario. El despacho de agua se convierte en una consecuencia pasiva de la meteorología, siguiendo la curva de irradiación solar, o, en el mejor de los casos, sigue un plan estático (basado en temporizadores) definido manualmente por el ingeniero agrónomo. Este enfoque manual representa una carga significativa de horas de ingeniería y es inherentemente subóptimo, ya que no puede reaccionar a la variabilidad diaria de las condiciones hídricas o energéticas.

La necesidad de esta optimización inteligente se vuelve crítica en contextos agrícolas como el del cultivo de pistacho, en que los ciclos de inversión son largos y la planta requiere varios años para entrar en producción [2, 3, 4]. En regiones donde el agua es un recurso escaso o sujetas a sequías estacionales, una gestión deficiente del riego no solo reduce la eficiencia, sino que compromete la viabilidad y sostenibilidad a largo plazo de la plantación.

Por lo tanto, la problemática central que este trabajo aborda es esta falta de inteligencia operativa. La proliferación de la telemetría y los sensores de campo (IoT) en la agricultura moderna han generado un vasto potencial de optimización que,

en la mayoría de los sistemas de riego, permanece sin explotar [5]. Los controladores actuales se limitan a registrar estos datos o a reaccionar a umbrales simples, pero carecen de la capacidad de utilizarlos para una planificación proactiva.

El verdadero desafío que se resuelve en este trabajo es integrar la inteligencia artificial para dos tareas claves: primero, para transformar este flujo de datos en conocimiento predictivo (un modelo de alta fidelidad del sistema); y segundo, para actuar sobre dicho conocimiento. Se requiere un motor de optimización, en este caso, un algoritmo genético, que funcione como un buscador capaz de generar un cronograma de despacho óptimo a largo plazo. Esta capacidad de búsqueda y planificación proactiva, que balancea múltiples objetivos, es la pieza de gestión que falta en los sistemas actuales, que obligan a operar muy por debajo de su potencial de eficiencia.

1.2. Estado del arte: modelado y optimización

El problema central de este trabajo, generar un cronograma de despacho de riego óptimo para un sistema solar autónomo, no es un desafío singular, sino la intersección de dos conceptos técnicos que el estado del arte ha abordado históricamente por vías separadas:

1. El desafío del modelado: se requiere un simulador de muy alta fidelidad, un gemelo digital, que actúe como la función de evaluación del optimizador. Este modelo debe predecir con precisión la respuesta hidráulica (caudales, presiones, nivel freático) ante cualquier plan de riego candidato.
2. El desafío de la optimización: se necesita un algoritmo de búsqueda capaz de encontrar la combinación óptima de decisiones (qué válvula abrir y cuándo) dentro de un espacio de soluciones de dimensionalidad masiva, respetando un conjunto de restricciones físicas (energía, agua, presión) y un tiempo de cómputo estricto.

A continuación, se analiza el panorama de las soluciones existentes para cada uno de estos desafíos, identificando las limitaciones que motivan el desarrollo de este trabajo.

1.2.1. El modelado: consistencia vs. precisión

El algoritmo de optimización es, por naturaleza, agnóstico a la física del problema. Depende estrictamente de una función de evaluación [6] (en este caso, el gemelo digital) para cuantificar la calidad, o *fitness*, de un cronograma de riego candidato. El éxito de la optimización, por lo tanto, depende enteramente de la fidelidad de este modelo. El enfoque tradicional para construir esta función se basa en modelos de caja blanca o de primeros principios, como el estándar industrial EPANET [7], que resuelven las ecuaciones fundamentales de la mecánica de fluidos [8, 9]. Su fortaleza indiscutible es la consistencia física: al estar basados en leyes universales, son robustos y pueden generalizar su comportamiento a escenarios de operación novedosos [10]. Sin embargo, su debilidad es la incertidumbre en sus parámetros físicos. El rendimiento del modelo depende de una calibración precisa de variables estáticas (ej. rugosidad de tuberías) que son difíciles de medir y cambian con el tiempo [11]. Esto genera un error sistemático o residual entre la simulación idealizada y la realidad del campo.

Para solucionar esta brecha de precisión, en el extremo opuesto del espectro se encuentran los modelos puramente empíricos de caja negra (IA), que buscan aprender el comportamiento del sistema basándose únicamente en datos históricos de telemetría. La ventaja de este enfoque es la capacidad de alcanzar una alta precisión, capturando dinámicas complejas que el modelo físico ignora [12]. No obstante, esta solución es frágil: requiere volúmenes masivos de datos para cubrir todo el espacio operativo [13], no es robusta para extrapolar a escenarios no vistos y, crucialmente, no garantiza la consistencia física [13].

El panorama de las soluciones de modelado presenta, por lo tanto, un dilema fundamental: el ingeniero debe elegir entre la consistencia física de la caja blanca (sufriendo su imprecisión) o la precisión de la caja negra (sufriendo su falta de robustez). Esta dicotomía, junto con la alternativa conceptual de un modelo híbrido o de caja gris que busca unificar ambas [14, 15], se ilustra en la figura 1.1.

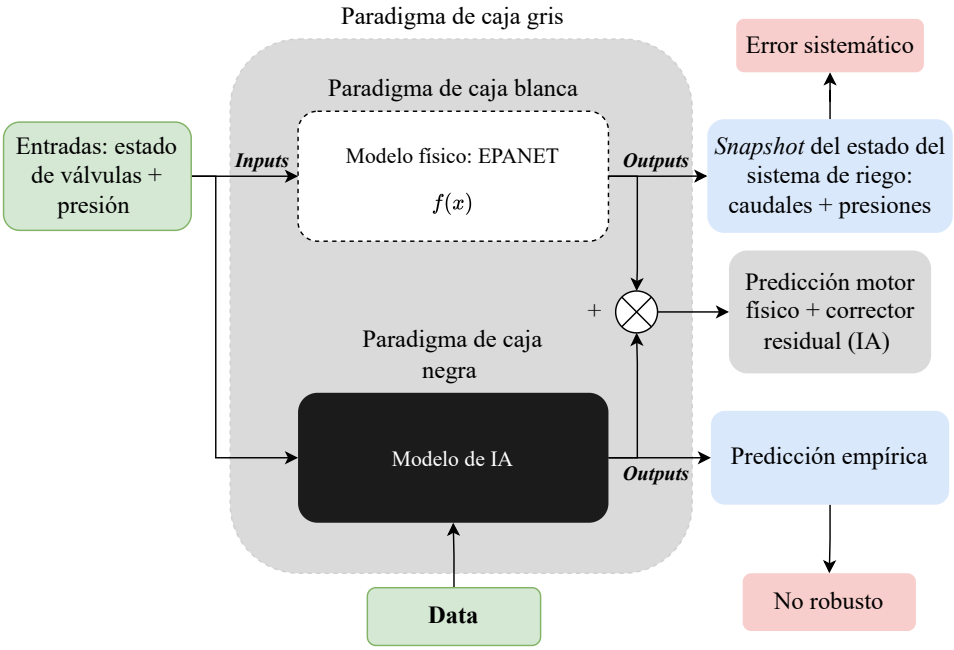

FIGURA 1.1. Diagrama conceptual de los paradigmas de modelado: caja blanca (física), caja negra (IA pura) y caja gris (híbrido).

1.2.2. Optimización: optimalidad vs. factibilidad

El segundo desafío fundamental radica en el propio algoritmo de búsqueda. La elección de la herramienta de optimización no es arbitraria, sino que está dictada por la naturaleza matemática inherente al problema del despacho de riego. Al formularlo, el problema se revela como un Problema de Programación Mixta-Entera No Lineal (MINLP) [16], una clasificación con profundas implicaciones computacionales.

La naturaleza Mixta-Entera (MI) surge de las variables de decisión: el estado de una válvula o bomba en un intervalo de tiempo es una decisión fundamentalmente binaria (0 para cerrado o 1 para abierto). A esto se suma la No Linealidad

(NL), que proviene directamente de la función de evaluación. El modelo hidráulico que predice el caudal y la presión, ya sea de caja blanca o negra, es una función compleja de la física de fluidos, no una simple suma algebraica.

La combinación de decisiones binarias (un espacio de búsqueda combinatorio) y una función de evaluación no lineal clasifica este problema en la categoría de complejidad NP-hard [17]. Esto significa que es computacionalmente intratable: el tiempo requerido para encontrar la solución óptima garantizada crece exponencialmente a medida que el problema escala (más válvulas o más intervalos). El espacio de soluciones a explorar es demasiado grande.

Esta realidad computacional genera un conflicto directo e irresoluble con las necesidades de un sistema de riego operativo. El estado del arte, si bien ofrece solvers exactos para MINLP que buscan el óptimo global, no puede garantizar el tiempo de convergencia.

Dado este compromiso entre la optimalidad y la factibilidad, la práctica establecida para problemas de *scheduling* (programación) y asignación de recursos NP-hard es el uso de metaheurísticas [18, 19]. Algoritmos como los algoritmos genéticos (AG) proponen un enfoque alternativo: en lugar de buscar una garantía de optimalidad global, ofrecen una garantía de tiempo de ejecución. Están diseñados para encontrar soluciones de alta calidad (casi óptimas) dentro de un presupuesto computacional fijo [20]. Para un problema de ingeniería en un entorno productivo, esta capacidad de obtener una solución factible en un tiempo acotado es un requisito operativo fundamental, lo que convierte a esta estrategia en un enfoque viable.

1.3. Motivación

La motivación de este trabajo es proponer una arquitectura de software unificada que resuelva los dos dilemas fundamentales identificados en el estado del arte. Esta solución se fundamenta en la hipótesis de que es posible lograr tanto la precisión en el modelado como la factibilidad en la optimización, sin sacrificar la robustez física ni el rendimiento operativo.

Ante el desafío del modelado, que obliga a elegir entre la consistencia de la caja blanca y la precisión de la caja negra, el objetivo es unificar la consistencia física de los primeros principios con la precisión empírica de los modelos de inteligencia artificial (IA).

Por otro lado, la optimización (MINLP) también requiere un enfoque específico. Si bien los solvers exactos son la herramienta fundamental para garantizar la optimalidad global, su tiempo de convergencia es, por la naturaleza NP-hard del problema, indefinido. Un cálculo que puede tardar horas o días no es una herramienta útil para la toma de decisiones agronómicas diarias. Por esta razón, se requiere un enfoque alternativo que priorice la velocidad de respuesta. Esta estrategia debe permitir al sistema encontrar soluciones de alta calidad (casi óptimas) dentro de un límite de tiempo, lo que resulta aceptable y práctico para las necesidades de la operación real.

1.4. Objetivos y alcance

Basado en la problemática y la motivación expuestas, el objetivo general de este trabajo es diseñar, implementar y validar el software que constituye la capa de gestión y optimización para un sistema de riego solar. El proyecto se enfoca en desarrollar los dos componentes técnicos fundamentales que responden directamente a los dilemas identificados en el estado del arte.

El primer objetivo específico es la creación de un gemelo digital híbrido. Esto implica implementar un simulador físico y, de forma crucial, desarrollar un modelo de inteligencia artificial cuya función es predecir y corregir el error residual de dicho modelo físico. Este componente servirá como la función de evaluación precisa y robusta para el optimizador.

El segundo objetivo específico es el diseño de un optimizador metaheurístico. Se implementará un AG capaz de navegar el complejo espacio de búsqueda MINLP. Este agente utilizará el Gemelo Digital Híbrido como su función de aptitud para generar cronogramas de despacho casi óptimos, priorizando la factibilidad temporal sobre la optimalidad exacta. El alcance de la implementación incluye también el desarrollo de la función de costos híbrida y unificada, que permite al agrónomo ponderar los objetivos de uniformidad, eficiencia y sostenibilidad del acuífero.

Es importante delimitar que el alcance de este trabajo es exclusivamente de software y se centra en el motor de decisión. El trabajo no incluye el diseño, provisión, instalación o mantenimiento de ningún componente de hardware (como bombas, sensores, PLCs o gateways). Asimismo, el desarrollo de la interfaz de usuario final o capa de presentación (dashboard) y el soporte operativo post-entrega quedan fuera del alcance de esta memoria.

Capítulo 2

Introducción específica

En este capítulo se presentan los fundamentos teóricos y las herramientas que sustentan la arquitectura de optimización propuesta. Se detalla el modelado híbrido de caja gris, la formulación del problema MINLP, la configuración del algoritmo genético y las métricas de evaluación empleadas.

2.1. Teoría del modelado híbrido: corrección de residuales

La simulación de sistemas hidráulicos presenta un compromiso entre la consistencia física y la precisión operativa. Mientras que los modelos de caja blanca (determinísticos) como EPANET garantizan coherencia física pero sufren sesgos por incertidumbre paramétrica, los modelos de caja negra (estocásticos) ofrecen alta precisión pero carecen de robustez física [21, 22].

Para balancear estas limitaciones, este trabajo implementa un modelado de caja gris (*gray-box*) mediante una arquitectura de corrección de residuales [23]. En este esquema, el modelo físico (f_{fisico}) aporta la línea base robusta, mientras que un modelo de inteligencia artificial (f_{AI}) predice y corrige el error sistemático o residual (R) del primero.

La formulación matemática de esta arquitectura híbrida se define en la ecuación 2.1:

$$f_{hybrid} = f_{fisico} + f_{AI}(\mathbf{X}) \tag{2.1}$$

donde el objetivo de entrenamiento para f_{AI} es el residual $R = Y_{real} - Y_{fisico}$.

2.1.1. Componente físico (f_{fisico})

El componente físico se desarrolla utilizando el motor de simulación EPANET [24, 25]. Este modelo se rige por leyes físicas claves que definen la complejidad del problema:

1. Pérdida de carga por fricción: se utiliza la ecuación de Hazen-Williams (ecuación 2.2) [26], una relación empírica que define la pérdida de carga (H_f) en una tubería.
$$H_f = \frac{10{,}67 \cdot L}{C^{1{,}852} \cdot D^{4{,}87}} \cdot Q^{1{,}852} \tag{2.2}$$

2. Comportamiento de emisores (goteros): el caudal de salida (q_e) se modela mediante la ecuación caudal-presión del emisor [27], una relación empírica de tipo potencia (ecuación 2.3) que vincula el caudal con la presión (p) en el nodo. En este trabajo se utiliza un exponente γ de 0,46, característico del

equipamiento instalado, introduciendo otra no linealidad.

$$q_e = C_d \cdot p^{\gamma} \tag{2.3}$$

2.1.2. Componente corrector (f_{AI})

El componente de corrección se implementa mediante un perceptrón multicapa (MLP). Este recibe como entrada el estado operativo del sistema (presión de bomba, nivel freático y configuración de válvulas) junto con la salida del modelo físico, aprendiendo a estimar la diferencia entre la simulación teórica y los datos reales medidos.

2.2. Formulación del problema de optimización

Una vez que se dispone de una función de aptitud f_{hybrid} de alta fidelidad, es necesario definir formalmente el problema de optimización.

2.2.1. Formulación matemática formal

El problema de optimización del despacho de riego consiste en encontrar el cronograma de apertura y cierre de válvulas que maximice un conjunto de objetivos, sujeto a las restricciones físicas de la red hidráulica y la disponibilidad de recursos. En tabla 2.1, se presenta una nomenclatura y formulación general del modelo.

TABLA 2.1. Nomenclatura del modelo de optimización.

Símbolo	Definición
$\mathbf{V}$	Matriz de decisiones de despacho (variable de decisión).
$v_{i,t}$	Variable binaria: 1 si la válvula i está abierta en el tiempo t, 0 si está cerrada.
$\mathcal{T}$	Horizonte de planificación (ej. 24 horas, discretizado en T intervalos).
$\mathcal{I}$	Conjunto de todas las válvulas (sectores de riego).
$f_{hybrid}(\mathbf{V})$	Función de aptitud.
$Q_{entregado,i}(\mathbf{V})$	Caudal total entregado al sector i bajo el plan $\mathbf{V}$.
$Q_{objetivo,i}$	Caudal objetivo para el sector i.
$P_{consumida,t}(\mathbf{V})$	Potencia consumida por la bomba en el tiempo t bajo el plan $\mathbf{V}$.
$P_{disponible,t}$	Potencia disponible pronosticada en el tiempo t.
$N_{recurso,t}(\mathbf{V})$	Nivel del recurso en el tiempo t bajo el plan $\mathbf{V}$.
N_{min}	Nivel mínimo de seguridad del recurso.
w_u, w_t, w_a	Ponderaciones estratégicas para los objetivos de la función.

2.2.2. Función objetivo ponderada

Un enfoque común para problemas multiobjetivo es formular una función de aptitud unificada. Esta función es una suma ponderada que puede incluir objetivos agronómicos, de eficiencia y de sostenibilidad. Esta se muestra en la ecuación 2.4:

$$\max_{\mathbf{V}} \left(w_u \cdot \mathcal{U}(\mathbf{V}) - w_t \cdot \mathcal{T}_{bombeo}(\mathbf{V}) - w_a \cdot \mathcal{S}_{acuifero}(\mathbf{V}) \right) \tag{2.4}$$

donde:

- $\mathcal{U}(\mathbf{V})$: una métrica de uniformidad, que se busca maximizar.
- $\mathcal{T}_{bombeo}(\mathbf{V})$: una métrica del tiempo de bombeo, que se busca minimizar.
- $\mathcal{S}_{acuifero}(\mathbf{V})$: una métrica del estrés sobre el acuífero, que se busca minimizar.

2.2.3. Restricciones del problema

La maximización está sujeta a un conjunto de restricciones físicas y operativas:

1. Restricción de potencia: la potencia consumida por la bomba no puede exceder la potencia disponible en ningún momento, de acuerdo a la ecuación 2.5.
$$P_{consumida,t}(f_{hybrid}(\mathbf{V})) \leq P_{disponible,t} \quad \forall t \in \mathcal{T} \tag{2.5}$$
2. Restricción de sostenibilidad (ej. nivel freático): el nivel del recurso no debe caer por debajo del límite de seguridad, según la ecuación 2.6.
$$N_{recurso,t}(f_{hybrid}(\mathbf{V})) \geq N_{min} \quad \forall t \in \mathcal{T} \tag{2.6}$$
3. Restricciones hidráulicas (implícitas): las leyes físicas de la red están encapsuladas dentro de la evaluación de la función f_{hybrid} .
4. Variables de decisión binarias: las decisiones de control de válvulas son binarias, como lo indica la ecuación 2.7.
$$v_{i,t} \in \{0, 1\} \quad \forall i \in \mathcal{I}, \forall t \in \mathcal{T} \tag{2.7}$$

2.3. Teoría de optimización: algoritmos genéticos

El algoritmo genético consiste en una búsqueda estocástica global basada en los principios de la evolución natural y la supervivencia del más apto [28, 29]. Este algoritmo es adecuado para navegar espacios de búsqueda complejos, no convexos y de alta dimensionalidad. Su uso acoplado a simuladores como EPANET para la optimización de redes de agua es una técnica documentada en la literatura [24, 25].

El AG opera sobre una población de cromosomas, donde cada cromosoma representa una solución candidata completa al problema de optimización.

2.3.1. Codificación del cromosoma (representación de la solución)

El cromosoma es la estructura de datos que codifica un plan de despacho de riego [30]. La elección de la codificación es fundamental para el rendimiento del algoritmo.

Se utiliza una codificación por valor binario (*value encoding*). El cromosoma es una cadena de valores (enteros, reales o binarios) donde cada gen representa un valor específico. Si el horizonte de planificación T se discretiza en N_T intervalos y hay N_V válvulas, un cromosoma C es un vector binario de longitud $N_V \times N_T$. Aunque

se almacena como un vector, conceptualmente representa una matriz donde cada elemento $v_{i,t}$ es un gen, como muestra la ecuación 2.8:

$$C = \begin{bmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,N_T} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,N_T} \\ \vdots & \vdots & \ddots & \vdots \\ v_{N_V,1} & v_{N_V,2} & \cdots & v_{N_V,N_T} \end{bmatrix} \tag{2.8}$$

donde $v_{i,t} \in \{0, 1\}$ representa el estado (cerrado/abierto) de la válvula i en el tiempo t.

2.3.2. Operadores evolutivos

Para garantizar la convergencia y diversidad de la población, se consideran los siguientes operadores:

- Selección por torneo: se eligen k individuos al azar y se selecciona el de mejor aptitud [31].
- Cruce uniforme: dado que el cromosoma es una matriz binaria sin dependencia posicional estricta, el cruce uniforme permite una recombinación eficiente de los genes. Cada bit del hijo se selecciona probabilísticamente de uno de los dos padres, promoviendo una mayor exploración del espacio de búsqueda [32].
- Mutación *bit-flip*: se aplica una mutación puntual donde un gen (estado de válvula) invierte su valor ($0 \to 1$ o $1 \to 0$) con una baja probabilidad. Esto introduce diversidad genética y previene el estancamiento en óptimos locales.

2.4. Herramientas computacionales y métricas

Para la materialización de los conceptos teóricos expuestos y la validación de los algoritmos de optimización, se selecciona un conjunto específico de herramientas de *software* y métricas de desempeño.

2.4.1. Bibliotecas y entorno de desarrollo

El desarrollo se fundamenta en el lenguaje de programación `Python`, seleccionado por su extenso ecosistema para la ciencia de datos y la ingeniería. Las bibliotecas principales se categorizan según su función en la arquitectura:

- Simulación hidráulica: para la implementación del componente físico (f_{fisico}), se utiliza la biblioteca *Water Network Tool for Resilience* (WNTR) [33]. Esta herramienta, basada en el motor estándar EPANET, permite la simulación hidráulica dinámica, facilitando la inyección de parámetros de control en tiempo de ejecución.
- Aprendizaje profundo: el componente corrector de residuales (f_{AI}) se construye sobre PyTorch [34]. Esta biblioteca ofrece la flexibilidad necesaria para diseñar arquitecturas de redes neuronales personalizadas y aprovecha la aceleración por hardware mediante tensores, lo cual es crítico para minimizar la latencia durante las miles de evaluaciones requeridas por el AG.

- Procesamiento numérico y datos: la manipulación eficiente de los cromosomas (matrices binarias) y las operaciones vectoriales del AG se realizan mediante NumPy, asegurando un bajo costo computacional en los operadores de cruce y mutación. Por su parte, Pandas se emplea para la gestión de series temporales y la ingeniería de características previa al entrenamiento.
- Gestión del ciclo de vida: para garantizar la reproducibilidad científica del modelo híbrido, se utiliza MLflow. Esta plataforma permite registrar sistemáticamente los hiperparámetros del modelo, las versiones de los datos de entrenamiento y las métricas de validación, asegurando la trazabilidad completa del experimento.

2.4.2. Métricas de evaluación

La evaluación del desempeño del sistema se aborda desde dos perspectivas complementarias: la precisión predictiva del gemelo digital híbrido y la eficacia del motor de búsqueda del AG.

Para cuantificar la capacidad del modelo híbrido (f_{hybrid}) de replicar la realidad y corregir el sesgo del modelo físico, se emplea principalmente la raíz del error cuadrático medio (RMSE) como se observa en la ecuación 2.9.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.9}$$

donde y_i representa el valor real medido y $\hat{y}_i$ la estimación del modelo. Adicionalmente, se utiliza el coeficiente de determinación (R^2) para medir la proporción de la varianza de los datos reales que es explicada por el modelo, validando su capacidad para capturar la dinámica del sistema.

Por otro lado, la calidad del proceso de optimización se evalúa mediante el monitoreo del costo de la mejor solución a lo largo de las generaciones, lo que permite verificar la convergencia del algoritmo.

Capítulo 3

Diseño e implementación

En este capítulo se describe la solución de software completa, detallando la arquitectura y la implementación técnica. La solución se centra en la integración de la simulación hidráulica y la inteligencia artificial para crear un gemelo digital híbrido. Finalmente, se explica cómo este gemelo se conecta a través de una interfaz de programación (API) con el motor de optimización para automatizar el despacho de riego.

3.1. Arquitectura general del software

El diseño del sistema de riego se basa en una arquitectura de cuatro capas: la capa física, la de control y adquisición, la de gestión y optimización, y la de presentación (ver figura 3.1). Este trabajo se enfoca exclusivamente en la capa de gestión y optimización, que es responsable de transformar los datos históricos y de telemetría en un plan de despacho de riego óptimo.

El software se implementó bajo una arquitectura de microservicios [35]. Esto garantiza la separación de responsabilidades y la escalabilidad horizontal de los dos componentes centrales: el entrenamiento continuo del modelo predictivo y su consumo operativo.

El diseño se basa en dos subsistemas, encapsulados en contenedores Docker para asegurar un entorno de ejecución aislado y consistente: el subsistema de entrenamiento y ciclo de vida (MLOps) y el subsistema de optimización y servicio. La vinculación entre ellos se establece a través de un registro central de modelos, lo que permite al motor de decisión consumir siempre la versión más precisa del gemelo digital híbrido, mientras el *pipeline* de MLOps opera de forma asíncrona.

3.1.1. Subsistema de entrenamiento y ciclo de vida (MLOps)

Este subsistema tiene como objetivo principal la sostenibilidad de la precisión del modelo predictivo. Mantiene el gemelo digital híbrido actualizado al ejecutar un ciclo de vida automatizado que mitiga el (*model drift*) [36]. Este flujo de trabajo, fundamental para un entorno de producción, está orquestado por *pipelines* de Apache Airflow [37].

La arquitectura comprende un *stack* de infraestructura para el manejo de los artefactos y la trazabilidad de los experimentos:

- Data Lake (MinIO): repositorio central de objetos que almacena los datos de telemetría históricos y los datasets limpios, actuando como la fuente única de verdad para el entrenamiento.

- Registro de Artefactos (MLflow): administra el ciclo de vida completo del modelo [38]. Registra los hiperparámetros, métricas y los artefactos binarios (el modelo y los transformadores de características) que son promovidos a producción para su consumo por el servicio de inferencia.
- Orquestación (PostgreSQL/Airflow): la base de datos PostgreSQL gestiona los metadatos y el estado de los *pipelines* de Airflow, asegurando la fiabilidad y reintentos del proceso de integración y despliegue continuo (CI/CD) [39].

El *pipeline* de entrenamiento se estructura en una secuencia de módulos lógicos de procesamiento de datos:

1. Módulo de ingesta y consolidación: responsable de la lectura y unificación de los datos de telemetría histórica del campo desde el Data Lake.
2. Módulo de limpieza: filtra los registros inconsistentes y asegura la calidad del *dataset*.
3. Módulo de ingeniería de características: genera las variables sintéticas necesarias para el modelo de corrección de residuales, como el número de actuadores activos.
4. Módulo de entrenamiento: ejecuta el entrenamiento. Si el nuevo modelo supera los umbrales de desempeño, es promovido al alias de `production` en el registro de modelos, lo que completa el ciclo de integración y despliegue continuo (CI/CD) [39].

3.1.2. Subsistema de optimización y servicio

Este subsistema constituye el corazón operativo del sistema, implementando el motor de decisión encargado de resolver el problema de programación matemática. Su arquitectura es diseñada específicamente para superar el cuello de botella computacional que representa la simulación hidráulica secuencial. Para ello, se diferencian dos componentes acoplados mediante una estrategia de escalabilidad horizontal.

1. Capa de servicio de inferencia distribuida (*Snapshot API*): el diseño de arquitectura resuelve el desafío de rendimiento mediante la implementación de un clúster de microservicios de simulación. Cada unidad es un servicio RESTful asíncrono desarrollado con FastAPI [40], diseñado para ser ligero y sin estado (*stateless*) en cuanto a la petición, pero manteniendo en memoria el estado completo del gemelo digital híbrido (modelo físico WNTR más el modelo corrector de IA).

   Para lograr la aceleración requerida por el algoritmo genético, se implementa una estrategia de orquestación mediante Docker Swarm [41]. Esta tecnología permite instanciar múltiples réplicas del microservicio de simulación, distribuyéndolas dinámicamente a través de los nodos físicos disponibles en el clúster. El tráfico entrante es gestionado por una instancia de NGINX [42], que actúa como balanceador de carga, distribuyendo las miles de peticiones de evaluación generadas por el optimizador entre las distintas réplicas activas. Esta arquitectura permite paralelizar masivamente el cálculo de la función de costo, reduciendo el tiempo de búsqueda de forma lineal con respecto a la cantidad de recursos de hardware agregados.

2. Motor de optimización: el motor implementa la lógica metaheurística del AG. A diferencia de una implementación secuencial tradicional, este módulo dispone de una interfaz cliente asíncrona que aprovecha la concurrencia (*multithreading*) para enviar lotes de solicitudes de simulación simultáneas.

   Durante cada generación del AG, la evaluación de la aptitud de la población no se realiza individuo por individuo, sino que se distribuye a través de la red hacia el balanceador de carga. Esto permite que múltiples planes de riego sean simulados al mismo tiempo por las distintas instancias del servicio de inferencia, lo que permite aprovechar al máximo la capacidad de cómputo instalada y cumplir con los estrictos requisitos temporales.

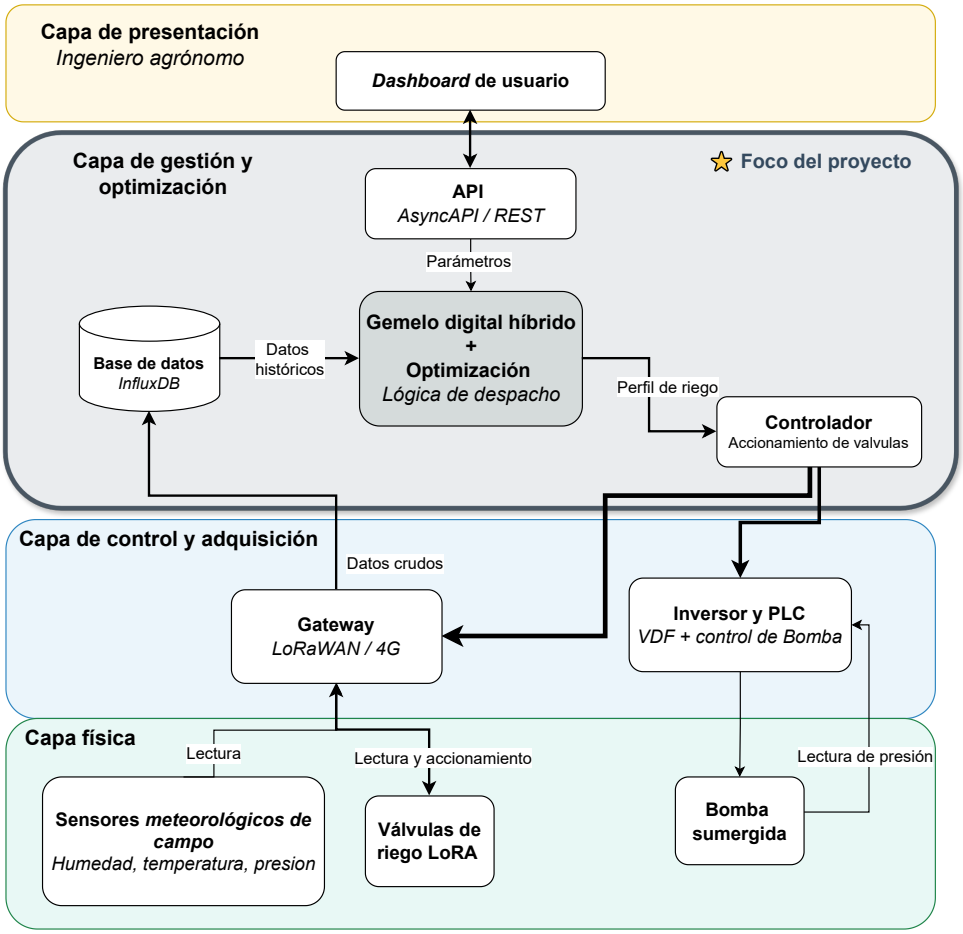

FIGURA 3.1. Diagrama de capas del sistema de riego, donde el foco del trabajo es la capa de gestión y optimización.

3.2. Implementación del gemelo digital híbrido

El gemelo digital se desarrolló bajo el paradigma de caja gris, fusionando un modelo hidráulico de primeros principios con un componente de aprendizaje profundo (*deep learning*). Este último tiene la función específica de predecir y corregir los errores residuales de la simulación física. Para asegurar la vigencia del sistema, la solución se integró en un ciclo de vida automatizado que ejecuta entrenamientos periódicos del modelo ante la llegada de nuevos datos de campo.

La figura 3.2 presenta el esquema conceptual del sistema implementado, ilustrando el flujo de información desde la ingesta de datos de campo hasta la generación del estado corregido del sistema.

A continuación, se detalla la ingeniería aplicada en cada uno de sus componentes constitutivos.

3.2.1. Construcción y parametrización del modelo físico

El componente físico (f_{fisico}) se implementó utilizando la biblioteca *Water Network Tool for Resilience* (WNTR) [33]. Se optó por definir la topología de la red mediante funciones de Python en lugar de utilizar archivos de configuración estáticos. Esta estrategia permite modificar las propiedades de la infraestructura, como la curva de la bomba o la longitud de los laterales, alterando directamente las variables del código fuente, lo cual elimina la necesidad de regenerar archivos externos ante cada cambio en los parámetros de diseño.

La red se modeló representando explícitamente los componentes hidráulicos principales: la fuente de agua subterránea, el grupo de bombeo, el cabezal de filtrado y la red de distribución. Para garantizar la fidelidad de la simulación base, se configuraron los siguientes parámetros constitutivos extraídos de las especificaciones técnicas:

- Modelo de fricción en tuberías: para la cuantificación de las pérdidas por fricción, se seleccionó la ecuación de Hazen-Williams. Se estableció un coeficiente de rugosidad adimensional $C = 130{,}0$, aplicado de manera uniforme a los tramos de conducción principal y a los laterales de riego, valor que corresponde a tuberías plásticas con un desgaste operativo estándar.
- Caracterización de la bomba: el comportamiento del sistema de bombeo se representó mediante la curva característica de altura manométrica-caudal ($H - Q$). Esta curva se definió mediante la interpolación de 16 puntos operativos, abarcando el rango completo desde la presión a válvula cerrada ($H \approx 124$ m) hasta el caudal máximo.
- Comportamiento de los emisores: se configuró un exponente de flujo $\gamma = 0{,}46$, propio del régimen turbulento en los emisores instalados, y un coeficiente de descarga base ajustado a las especificaciones del fabricante.

Sin embargo, la simulación individual de los miles de goteros presentes en el campo generaba una carga computacional incompatible con los tiempos de respuesta requeridos. Para solucionar este desafío, se implementó un algoritmo de agregación espacial en la función de generación de laterales. Este algoritmo discretiza cada lateral de riego en un número fijo de segmentos. En lugar de instanciar cada emisor físico, se asigna un emisor equivalente a cada nodo de discretización, cuyo coeficiente de descarga se calcula como la suma aritmética de los coeficientes de

los goteros reales contenidos en dicho segmento. Esta técnica reduce la dimensión de la matriz hidráulica que resuelve el motor WNTR, disminuyendo el tiempo de ejecución de la simulación de segundos a milisegundos.

Finalmente, la ejecución del modelo se configuró bajo el esquema de análisis dirigido por presión (PDA). A diferencia del modelo de demanda base (DDA), que fija el consumo independientemente del estado hidráulico, el esquema PDA calcula el caudal real entregado por los emisores como una función de la presión nodal instantánea.

3.2.2. Ingeniería de datos y modelo de corrección

El componente de inteligencia artificial (f_{AI}) se diseñó con el objetivo específico de predecir el residual (R), definido como la diferencia entre el caudal real medido y la estimación teórica del modelo físico base. Para garantizar la calidad de la información de entrenamiento, el flujo de datos inicia con una etapa de limpieza rigurosa en la que se aplican filtros para eliminar registros inconsistentes, tales como instantes donde los datos reportan caudales positivos y válvulas cerradas.

Sobre el conjunto de datos depurado, se ejecutó un proceso de ingeniería de características orientado a capturar las no linealidades del sistema hidráulico. A las variables base disponibles, la presión real medida en cabecera y el caudal simulado por el modelo físico, se sumaron variables sintéticas. Se generó el conteo total de válvulas activas (`valve_count`) mediante la suma de los estados binarios de los sectores, y se creó un término de interacción física (`presion_x_valve_count`) multiplicando la presión real por dicho conteo. Esta última variable actúa como un estimador de la impedancia hidráulica total del sistema, facilitando a la red neuronal la distinción entre variaciones de caudal debidas a la operación de la bomba y aquellas causadas por apertura y cierre de válvulas.

Previo al entrenamiento, se aplicó una estrategia de saneamiento estadístico sobre la variable objetivo (el residual), eliminando valores atípicos (*outliers*) mediante el método del rango intercuartílico (IQR). Posteriormente, los datos se dividieron cronológicamente en subconjuntos de entrenamiento, validación y prueba, y se normalizaron las características numéricas utilizando un escalador estándar (StandardScaler). Es fundamental destacar que el escalador se ajusta exclusivamente con los parámetros estadísticos del conjunto de entrenamiento y luego se utiliza para transformar los conjuntos restantes, evitando así la fuga de información estadística (*data leakage*) hacia los datos de validación.

Finalmente, el modelo de corrección se implementó utilizando un MLP desarrollado con la biblioteca PyTorch [34]. La arquitectura de la red, compuesta por capas densas con funciones de activación no lineales, se optimizó para realizar la regresión del valor residual a partir del vector de características procesado. El proceso de entrenamiento minimiza la función de pérdida de error MSE e integra un mecanismo de parada temprana (*early stopping*) que monitorea el desempeño en el conjunto de validación, previniendo el sobreajuste y garantizando la robustez del modelo ante nuevas condiciones operativas.

3.2.3. Ciclo de vida y gestión de artefactos (MLOps)

La implementación del gemelo digital trasciende el código fuente. Se monta sobre una infraestructura de operaciones de aprendizaje automático (MLOps) orquestada por Apache Airflow [37]. Este sistema gestiona el flujo de datos desde su ingesta hasta el despliegue, estructurado en las siguientes etapas:

- Gestión de datos en capas: se utiliza un Data Lake basado en MinIO para almacenar los datos en tres estadios de refinamiento: crudos (*raw*), intermedios (*intermediate*) y procesados (*primary*). Esta arquitectura de capas permite la trazabilidad y la reproducibilidad de cualquier versión del *dataset*.
- Registro de experimentos: cada ejecución del *pipeline* de entrenamiento registra automáticamente las métricas de desempeño (RMSE, R^2) y los artefactos resultantes (el modelo serializado y los objetos escaladores) en un servidor de MLflow [38]. Esto permite analizar la evolución del desempeño del modelo a lo largo del tiempo.
- Validación y promoción automática: el *pipeline* incluye una etapa de decisión autónoma. Tras el entrenamiento, el sistema evalúa el nuevo modelo frente a un conjunto de prueba. Solo si el nuevo modelo supera en precisión al vigente (reducción del RMSE), se promueve automáticamente mediante el etiquetado de alias en el registro, quedando disponible inmediatamente para el servicio de inferencia.

3.2.4. Orquestación y ejecución en el servicio de inferencia

El servicio de inferencia (*Snapshot API*) es el componente que ejecuta el gemelo digital híbrido. Su función principal es coordinar la comunicación entre los modelos almacenados y las solicitudes de simulación que envía el optimizador. El código se diseñó para cargar todo lo necesario en la memoria RAM al iniciar el servidor, evitando así lecturas lentas de disco durante la operación.

Al arrancar, el servicio se conecta al registro de MLflow y descarga la versión del modelo de inteligencia artificial marcada como *production*. También inicializa el modelo físico WNTR. Esta preparación previa permite que el sistema responda a las peticiones de simulación mas rápidamente.

Cada vez que el optimizador solicita evaluar un plan de riego, el servicio ejecuta los siguientes pasos de forma secuencial:

1. Cálculo físico base: el servicio ejecuta primero el simulador WNTR. Este paso entrega un valor de caudal y las presiones para la configuración de válvulas solicitada.
2. Preparación de datos para la IA: el sistema toma los resultados físicos y calcula las variables adicionales que necesita la red neuronal, como la cantidad de válvulas abiertas y su interacción con la presión. Luego ajusta la escala de estos valores utilizando el artefacto guardado en MLFlow durante el entrenamiento.
3. Corrección del error: la red neuronal recibe los datos preparados y predice el error residual del modelo físico. El sistema suma este error a la predicción física inicial para obtener el caudal final corregido.

4. Respuesta: con el caudal corregido, el sistema estima cómo se distribuye el caudal residual entre los sectores activos y calcula cuánto desciende el nivel del acuífero usando un modelo de pozo específico. Estos valores finales se envían de vuelta al optimizador para que evalúe si el plan es bueno o malo.

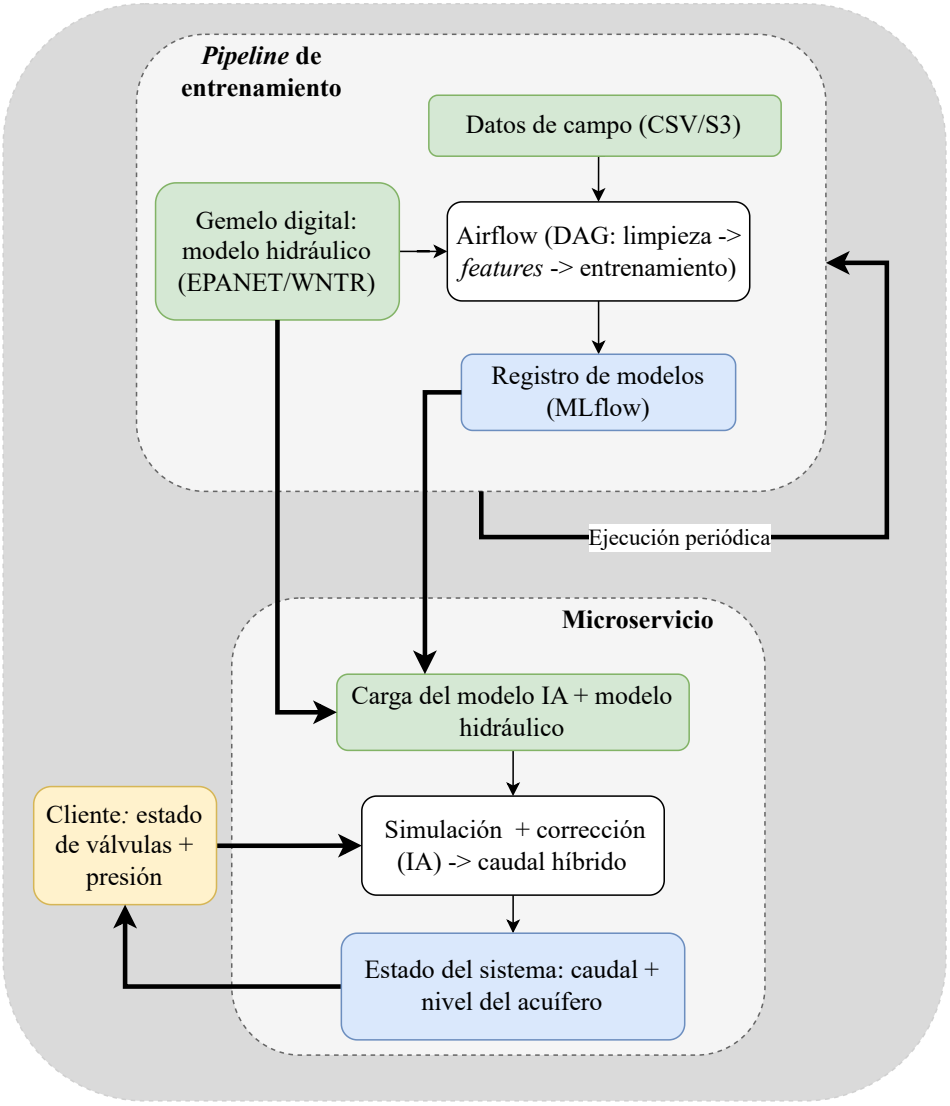

FIGURA 3.2. Esquema del gemelo digital híbrido: integración del pipeline de entrenamiento (MLOps), el modelo físico WNTR y el corrector de IA en el servicio de inferencia.

3.3. Implementación del optimizador de despacho (GA)

La implementación del motor de optimización representa el componente inteligente de decisión del sistema. Su diseño responde a la naturaleza combinatoria y no lineal del problema de programación de riego, donde los métodos tradicionales de programación lineal mixta-entera resultan computacionalmente inviables debido a la complejidad de las funciones hidráulicas subyacentes.

Se desarrolló un AG personalizado en el lenguaje Python, estructurado bajo el paradigma de programación orientada a objetos para garantizar la modularidad y el mantenimiento del código. Este motor integra lógica específica del dominio hidráulico directamente en la estructura de datos del cromosoma y en el ciclo de evaluación de aptitud. Esta integración permite reducir el espacio de búsqueda y acelerar la convergencia hacia soluciones factibles.

La figura 3.3 ilustra el flujo de control del sistema implementado, destacando la interacción con el clúster de simulación distribuida.

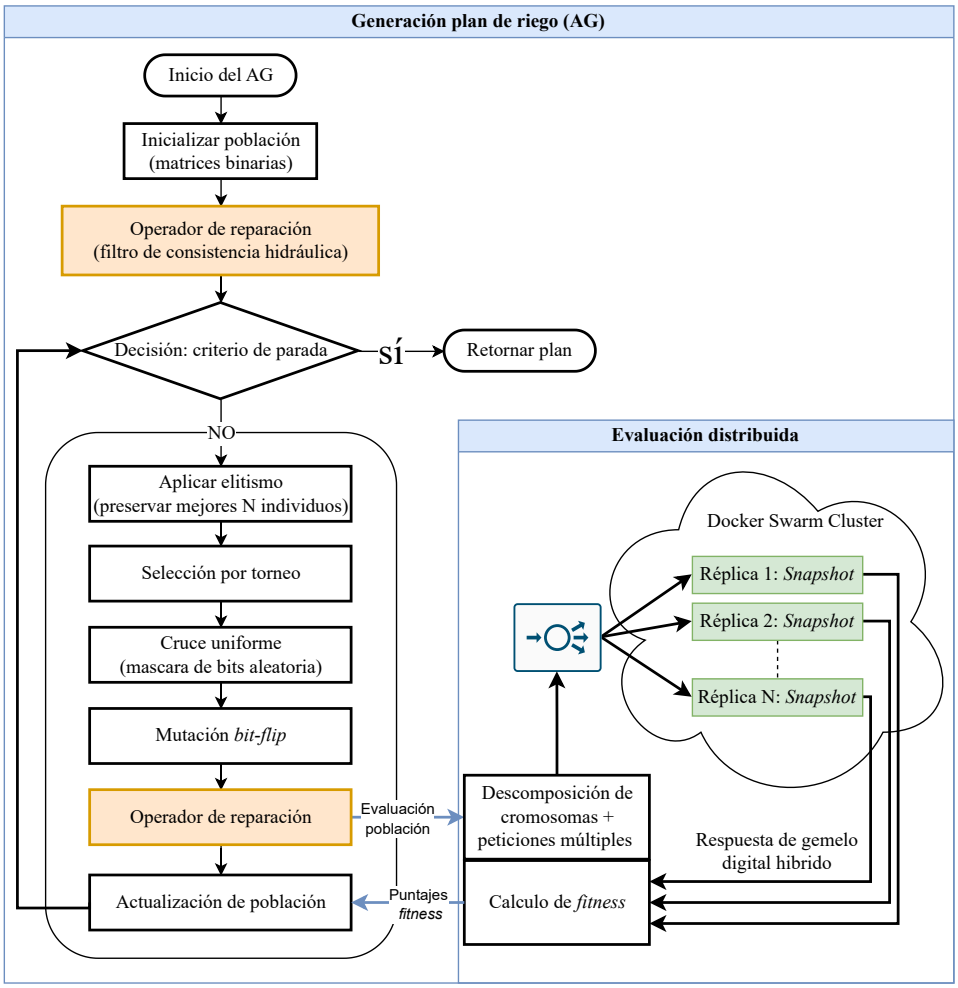

FIGURA 3.3. Diagrama de flujo del Algoritmo Genético Distribuido, destacando la etapa de evaluación paralela y el operador de reparación de restricciones físicas.

3.3.1. Representación matricial y codificación

El diseño de la estructura de datos para representar una solución candidata es fundamental para el rendimiento del algoritmo. Se optó por una codificación por valor binario (*value encoding*), implementada no como una lista lineal, sino como una matriz bidimensional utilizando tipos de datos de bajo nivel (`uint8`) de la biblioteca NumPy.

La matriz C, de dimensiones $N_{actuadores} \times N_{intervalos}$, modela el estado de todos los elementos activos del sistema:

- Dimensión espacial (filas): la primera fila (índice 0) representa el estado de la bomba principal. Las filas subsiguientes (índices 1 a N) corresponden a las válvulas electro-comandadas de cada sector de riego.
- Dimensión temporal (columnas: se define dinámicamente en función del horizonte de predicción. Para un horizonte típico de 24 horas con una resolución de 30 minutos, la matriz tiene 48 columnas.

Esta representación matricial permite aplicar operaciones vectorizadas para el cálculo de costos y la manipulación genética. El código 3.1 muestra la implementación de la función generadora, que instancia el cromosoma completo en una sola operación optimizada, garantizando una sobrecarga de memoria mínima incluso para poblaciones grandes.

```
def _create_individual(self) -> np.ndarray:
    """
    Genera una matriz de (N_actuadores x N_intervalos)
    con valores 0 o 1 (enteros de 8 bits).
    """
    return np.random.randint(0, 2,
                             size=(self.n_actuadores, self.n_intervals),
                             dtype=np.uint8)
```

CÓDIGO 3.1. Generación vectorizada del cromosoma como matriz binaria.

3.3.2. Manejo de restricciones y reparación

El espacio de búsqueda de cromosomas binarios aleatorios contiene una vasta cantidad de soluciones que son físicamente incoherentes. El caso más crítico es la configuración donde una o más válvulas de sector están abiertas ($v_{i,t} = 1$) mientras la bomba principal está apagada ($v_{0,t} = 0$). Evaluar estos individuos sería un desperdicio de recursos computacionales, ya que el simulador hidráulico arrojaría caudal cero o errores de convergencia numérica.

Para abordar esto, se implementó un mecanismo de reparación dentro del motor del AG. Este operador actúa como un filtro de consistencia hidráulica que se aplica inmediatamente después de la inicialización y tras cada operación de cruce o mutación.

La lógica aplica una operación de máscara booleana donde el estado de cada válvula en el tiempo t se multiplica por el estado de la bomba en ese mismo instante. Matemáticamente, esto asegura la condición:

$$v_{i,t} \leftarrow v_{i,t} \wedge v_{0,t} \quad \forall i > 0 \tag{3.1}$$

Esta técnica transforma el problema de una optimización con restricciones duras a una optimización sobre un subespacio factible, guiando la búsqueda hacia regiones productivas desde la primera generación.

3.3.3. Arquitectura de evaluación distribuida y cliente asíncrono

Uno de los desafíos técnicos más significativos en la optimización de sistemas físicos es el costo temporal de la función de evaluación (*fitness function*). En este sistema, evaluar un plan de riego implica simular su comportamiento hidráulico completo para determinar el caudal entregado y el consumo energético.

Para dimensionar la necesidad de una arquitectura escalable, consideremos un escenario estándar con una población de 50 individuos y un horizonte de 24 horas (48 intervalos). El número total de simulaciones hidráulicas (N_{sims}) requeridas por generación se calcula según ecuación 3.2.

$$N_{sims} = 50 \times 48 = 2{,}400 \text{ simulaciones/generación} \tag{3.2}$$

Si cada simulación tomara apenas 0.1 segundos, una ejecución secuencial demoraría 4 minutos por generación, resultando en horas de cómputo para converger.

Para resolver este cuello de botella, se diseñó una arquitectura de evaluación distribuida implementada en la clase `DigitalTwinClient`. Esta clase actúa como un proxy inteligente que desacopla el algoritmo genético del motor de simulación, gestionando la complejidad de la comunicación con el clúster de microservicios. Para ello, el cliente implementa un patrón de concurrencia a dos niveles para maximizar el rendimiento (*throughput*):

1. Sesiones persistentes: se instancia un objeto `requests.Session` que mantiene un *pool* de conexiones TCP abiertas (*Keep-Alive*) hacia el balanceador de carga. Esto elimina la latencia asociada al *handshake* TCP/IP en cada una de las miles de peticiones enviadas.
2. Paralelismo: el cliente descompone la matriz del cromosoma en N tareas de simulación independientes, correspondientes a las columnas (intervalos de tiempo) donde la bomba está activa. Utilizando la biblioteca `concurrent`, estas tareas se envían de forma asíncrona a un ThreadPoolExecutor.

Esta estrategia permite saturar la capacidad de procesamiento del clúster Docker Swarm, distribuyendo la carga de trabajo entre todos los núcleos disponibles en la infraestructura física. El cliente es responsable de reensamblar los resultados parciales asíncronos en una estructura de datos coherente para el cálculo del costo final.

3.3.4. Diseño de la función de aptitud

La función de aptitud es el componente que guía la búsqueda hacia soluciones que equilibren la eficiencia operativa con los objetivos requeridos por el agrónomo. El valor de aptitud para un individuo se calcula integrando los resultados parciales obtenidos de las simulaciones distribuidas.

El método de evaluación agrega los volúmenes de agua simulados para cada sector y calcula una suma ponderada del costo energético y las penalizaciones por desvíos hídricos. El diseño prioriza una penalización cuadrática ($L2$) sobre la diferencia entre el volumen de agua entregado y el objetivo (V_{target}).

3.3.5. Operadores evolutivos implementados

La mecánica de evolución de la población se rige por tres operadores estocásticos seleccionados específicamente para trabajar sobre la representación matricial binaria del problema. A continuación, se detalla la implementación lógica y el código fuente de cada uno.

Selección por torneo

Para seleccionar los padres de la siguiente generación, se implementó el método de selección por torneo. Como se observa en el código 3.2, este operador selecciona aleatoriamente un subconjunto de individuos de la población (definido por `tournament_size`) y determina el ganador basándose en el valor mínimo de aptitud. Esta implementación vectorizada utiliza NumPy para realizar el muestreo aleatorio y la comparación de índices, lo que resulta computacionalmente más eficiente que ordenar la población completa en cada generación.

```
def _selection_tournament(self, fitness_scores: np.ndarray) -> np.ndarray:
    """
    Selecciona un individuo usando seleccion por torneo.
    """
    # Elegir 'tournament_size' individuos al azar
    contenders_idx = np.random.randint(0, self.pop_size,
                                       self.tournament_size)

    # Obtener sus puntuaciones de aptitud
    contender_fitness = fitness_scores[contenders_idx]

    # El ganador es el que tiene la aptitud MAS BAJA (menor costo)
    winner_idx_in_contenders = np.argmin(contender_fitness)
    winner_global_idx = contenders_idx[winner_idx_in_contenders]

    return self.population[winner_global_idx]
```

CÓDIGO 3.2. Implementación de la selección por torneo.

Cruce uniforme matricial

Dada la estructura matricial del cromosoma, donde no existe una dependencia espacial fuerte entre intervalos de tiempo distantes, se optó por el cruce uniforme. El código 3.3 detalla la lógica de este operador. Se genera una máscara aleatoria de bits con las mismas dimensiones que el cromosoma. La descendencia se construye aplicando operaciones lógicas a nivel de bits (*bitwise operations*). Esta técnica asegura una mezcla profunda del material genético, permitiendo recombinar patrones de riego exitosos independientemente de su ubicación temporal.

```
def _crossover_uniform(self, parent1: np.ndarray,
                       parent2: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
    child1, child2 = parent1.copy(), parent2.copy()

    if random.random() < self.crossover_rate:
        # 1. Crear mascara aleatoria de 0s y 1s
        mask = np.random.randint(0, 2, size=parent1.shape, dtype=np.uint8)

        # 2. Crear la mascara inversa
        mask_inv = 1 - mask
```

```
11
12          # 3. Aplicar mascaras para crear los hijos
13          child1 = (parent1 * mask) | (parent2 * mask_inv)
14          child2 = (parent2 * mask) | (parent1 * mask_inv)
15
16      return child1, child2
```

CÓDIGO 3.3. Implementación del cruce uniforme con operaciones de máscara.

Mutación bit-flip

Para mantener la diversidad genética y evitar el estancamiento en óptimos locales, se aplicó un operador de mutación por inversión de bit. El código 3.4 muestra cómo el algoritmo recorre la matriz del cromosoma y, sujeto a una baja probabilidad de mutación (P_m), invierte el estado binario del gen ($1 - x$). Es importante notar que, tras este operador, el individuo resultante es procesado nuevamente por la función de reparación descrita en la sección 3.3.2 para garantizar la coherencia física del sistema.

```
def _mutation_bit_flip(self, chromosome: np.ndarray) -> np.ndarray:
    mutated_chromosome = chromosome.copy()
    for i in range(self.n_actuadores):
        for t in range(self.n_intervals):
            if random.random() < self.mutation_rate:
                # Inversion de bit: 1 -> 0 o 0 -> 1
                mutated_chromosome[i, t] = 1 - mutated_chromosome[i, t]
    return mutated_chromosome
```

CÓDIGO 3.4. Lógica de mutación bit-flip.

Estrategia de elitismo

Finalmente, para asegurar la convergencia monótona del algoritmo y evitar la pérdida de las mejores soluciones encontradas debido a la estocasticidad de los operadores genéticos, se implementó una estrategia de elitismo. En cada generación, antes de aplicar la selección y el cruce, el algoritmo identifica los N mejores individuos de la población actual (típicamente los 2 mejores) y los copia directamente a la siguiente generación sin modificaciones. Esto garantiza que la calidad de la mejor solución nunca disminuya a lo largo del proceso evolutivo.

3.4. Implementación de funciones de costo y penalizaciones

La función de costo, o función de aptitud, representa la traducción matemática de los objetivos del negocio y las restricciones físicas del sistema. Su implementación es crítica, ya que define el gradiente de búsqueda que guía al algoritmo genético a través del vasto espacio de soluciones posibles. En este trabajo, se implementó una función de evaluación compuesta que agrega múltiples objetivos competitivos, eficiencia energética, cumplimiento agronómico y sostenibilidad, en un único valor escalar.

La lógica de evaluación reside en el método `evaluate_schedule` de la clase cliente del gemelo digital. Este método actúa como un agregador de resultados, procesando los datos devueltos por el clúster de inferencia para cuantificar el desempeño global del plan de riego.

3.4.1. Costo base de eficiencia operativa

El primer componente de la función objetivo busca minimizar el uso de recursos operativos. Dado que el sistema es alimentado por energía solar fotovoltaica, el costo no es necesariamente monetario, sino temporal y de desgaste de equipos. Se definió la eficiencia en términos de tiempo de bombeo acumulado, calculada según la ecuación 3.3:

$$C_{eficiencia} = \sum_{t=1}^{T} v_{0,t} \cdot \Delta t \tag{3.3}$$

donde $v_{0,t}$ representa el estado binario de la bomba en el intervalo t (1 si está encendida, 0 si está apagada), y Δt corresponde a la duración del intervalo de tiempo discretizado (en horas).

Esta formulación incentiva al algoritmo a encontrar la solución más compacta posible, aquella que entrega el volumen de agua requerido en la menor cantidad de horas de operación, lo cual maximiza la vida útil del equipo de bombeo y libera ventanas de tiempo para mantenimiento o contingencias.

3.4.2. Penalización agronómica cuadrática

El objetivo primordial del sistema es satisfacer la demanda hídrica del cultivo. Sin embargo, en un escenario de recursos limitados, es común que no se pueda satisfacer la totalidad de la demanda de todos los sectores simultáneamente. El desafío de implementación consistió en definir cómo penalizar estos déficits para promover una distribución justa del agua.

Se descartó el uso de una penalización lineal o diferencia absoluta en favor de una penalización cuadrática (norma L2), definida en la ecuación 3.4:

$$P_{volumen} = \sum_{i=1}^{N} (V_{sim,i} - V_{target,i})^2 \tag{3.4}$$

donde N es el número total de sectores de riego, $V_{sim,i}$ es el volumen acumulado simulado para el sector i, y $V_{target,i}$ es el volumen objetivo definido por el agrónomo para dicho sector.

El código 3.5 presenta la implementación de esta lógica, donde se observa cómo el algoritmo itera sobre los resultados de volumen acumulado por sector y eleva al cuadrado la diferencia respecto al objetivo antes de sumarla al costo total.

```
# Logica de penalizacion L2 (Cuadratica)
penalizacion_hidrica_cuadratica = 0.0
desviacion_hidrica_absoluta_m3 = 0.0

for sector_name, target_m3 in self.targets_m3.items():
    # Obtener el volumen total simulado para el sector
    achieved_m3 = total_volume_per_sector.get(sector_name, 0.0)

    # Calcular el desvio
    desviacion = (achieved_m3 - target_m3)

    # Penalizacion cuadratica: castiga desproporcionadamente
    # los grandes desvios
    penalizacion_hidrica_cuadratica += (desviacion ** 2)

    # Metrica auxiliar para reporte
    desviacion_hidrica_absoluta_m3 += abs(desviacion)

# Calculo final de aptitud
penalizacion_total = self.w_hidrico_quad *
    penalizacion_hidrica_cuadratica
aptitud_total = costo_eficiencia + penalizacion_total
```

CÓDIGO 3.5. Cálculo de la penalización hídrica cuadrática.

Esta decisión de diseño tiene un impacto profundo en el comportamiento del optimizador. Al elevar el error al cuadrado, un déficit grande en un solo sector se penaliza mucho más severamente que varios déficits pequeños distribuidos en varios sectores. En consecuencia, el algoritmo prioriza naturalmente soluciones que equilibran el riego entre parcelas, evitando sacrificar completamente un sector para beneficiar a otro, lo cual se alinea con las mejores prácticas agronómicas para mantener la uniformidad del cultivo.

3.4.3. Manejo de restricciones físicas y ambientales

Además de los objetivos de eficiencia y volumen, el sistema debe respetar restricciones operativas estrictas. La implementación aborda estas limitaciones mediante el método de funciones de penalización, transformando violaciones de límites físicos en costos adicionales que degradan la aptitud del individuo.

En primer lugar, la operación de la bomba está condicionada por la curva de generación solar. El servicio de inferencia verifica en cada intervalo si la potencia requerida por el punto de operación hidráulico excede la potencia disponible pronosticada. Si se detecta una violación o déficit de potencia, el simulador retorna un estado de fallo. En el lado del cliente, se implementó una lógica que asigna un valor de aptitud infinito a ese intervalo ante un fallo energético. Esto actúa como una barrera suave que empuja rápidamente a la población del algoritmo genético lejos de las configuraciones que intentan operar la bomba fuera de las horas de radiación solar efectiva.

Simultáneamente, para proteger el recurso hídrico subterráneo, se estableció un límite máximo de descenso del nivel dinámico del pozo. El gemelo digital estima este descenso en función del caudal total extraído en cada instante. La función de costo implementa una penalización dinámica para esta variable, expresada en la ecuación 3.5:

$$P_{acuifero} = \sum_{t=1}^{T} \text{máx}(0, N_{min} - N_{sim,t}) \times w_{sostenibilidad} \tag{3.5}$$

donde N_{min} es el nivel mínimo de seguridad del acuífero (profundidad máxima permitida), $N_{sim,t}$ es el nivel dinámico simulado en el tiempo t, y $w_{sostenibilidad}$ es un factor de ponderación que ajusta la severidad de la penalización.

Esta implementación permite que el algoritmo explore soluciones cercanas al límite operativo del pozo, lo que maximiza la extracción permitida pero crea un fuerte gradiente de rechazo en cuanto se compromete la seguridad del acuífero.

3.4.4. Agregación final y ponderación

El valor final de aptitud, o *fitness*, devuelto al motor genético es la suma ponderada de todos los componentes descritos, como muestra la ecuación 3.6:

$$Fitness = C_{eficiencia} + (w_q \cdot P_{volumen}) + (w_s \cdot P_{acuifero}) + (w_p \cdot P_{potencia}) \tag{3.6}$$

donde w_q, w_s y w_p son los coeficientes de ponderación para los objetivos de volumen, sostenibilidad del acuífero y cumplimiento de potencia, respectivamente.

En la implementación actual, se asignó un peso preponderante a la penalización por volumen (w_q), estableciendo que la satisfacción de la demanda de riego es la prioridad jerárquica superior, seguida por la sostenibilidad del acuífero y, finalmente, la eficiencia operativa.

Finalmente, cabe destacar que la arquitectura del software expone estos parámetros como variables de configuración externa. Tanto los pesos de la función de costo, como los volúmenes objetivo por sector, el horizonte de planificación y la discretización temporal, constituyen las entradas principales del sistema de gestión. Estos valores están diseñados para ser definidos por el usuario a través de una interfaz de consola o gráfica, lo que otorga al ingeniero agrónomo el control total para ajustar la estrategia de optimización y ejecutar el plan de riego según las necesidades específicas del cultivo y las condiciones operativas del momento.

Capítulo 4

Ensayos y resultados

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

4.1. Entorno y banco de pruebas

4.2. Pruebas del gemelo digital hibrido

4.3. Caso de estudio: simulación de despacho

Capítulo 5

Conclusiones

Todos los capítulos deben comenzar con un breve párrafo introductorio que indique cuál es el contenido que se encontrará al leerlo. La redacción sobre el contenido de la memoria debe hacerse en presente y todo lo referido al proyecto en pasado, siempre de modo impersonal.

5.1. Resultados obtenidos

5.2. Trabajo futuro

Bibliografía

[1] Albert Sagala y col. «Implementation of Portable Off-Grid Solar Water Pump for Irrigation Systems». En: *Jurnal Mantik* (2022). URL: https://www.researchgate.net/publication/358920031_Implementation_of_Portable_Off-Grid_Solar_Water_Pump_for_Irrigation_Systems.

[2] Giuseppina Mandalari y col. «Pistachio Nuts (Pistacia vera L.): Production, Nutrients, Bioactives and Novel Health Effects». En: *Plants* (2021). DOI: 10.3390/plants11010018. URL: https://doi.org/10.3390/plants11010018.

[3] Lidia Núñez y col. «Pistachio Phenology and Yield in a Cold-Winter Region of Spain: The Status of the Cultivation and Performance of Three Cultivars». En: *Horticulturae* (2024). DOI: 10.3390/horticulturae10121235. URL: https://doi.org/10.3390/horticulturae10121235.

[4] R. Kanber. «Growth, yield and periodicity of pistachio under different irrigation and fertilization regimes». En: *Water and nutrient management for slope agriculture and horticulture*. Options Méditerranéennes. CIHEAM - Mediterranean Agronomic Institute (proceedings / technical paper). Disponible en: https://om.ciheam.org/ (consulta la edición original para la paginación exacta). CIHEAM / Options Méditerranéennes, Series B, Studies y Research, 2003.

[5] MDPI. *Internet of Things-Based Automated Solutions Utilizing Machine Learning for Smart and Real-Time Irrigation Management: A Review*. https://www.mdpi.com/1424-8220/24/23/7480. Accedido: 3 de septiembre de 2025. 2024.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989. ISBN: 978-0201157673.

[7] Lewis A. Rossman. *EPANET 2: Users Manual*. Water Supply and Water Resources Division, National Risk Management Research Laboratory. U.S. Environmental Protection Agency (EPA). Cincinnati, OH, USA, 2000. URL: https://www.epa.gov/water-research/epanet.

[8] Huy Truong. *DITEC-WDN: A Large-Scale Dataset of Hydraulic Scenarios across Multiple Water Distribution Networks*. https://arxiv.org/html/2503.17167v2. 2025. arXiv: 2503.17167v2.

[9] M. E. T. et al. «Calibration via Multi-period State Estimation in Water Distribution Systems». En: *World Environmental and Water Resources Congress 2017*. ASCE, 2017.

[10] Barbara Rakitsch Maja Rudolph Stefan Kurz. *Hybrid Modeling Design Patterns*. https://arxiv.org/pdf/2401.00033. 2024. arXiv: 2401.00033.

[11] G. P. G. de Oliveira y col. «Calibration Model for Water Distribution Network Using Pressures Estimated by Artificial Neural Networks». En: *Procedia Engineering*. Vol. 186. Elsevier, 2017, págs. 434-441.

[12] Jie Li y col. «Physics-based and data-driven hybrid modeling in manufacturing: a review». En: *Tsinghua University Press* (2024).

[13] Meng-Han Tao et al. «A comprehensive review of physics-informed deep learning: Applications, advancements, and challenges». En: *The Innovation* 5.6 (2024), pág. 102548.

[14] Wikipedia contributors. *Grey box model — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Grey_box_model. Accedido: 3 de septiembre de 2025. 2024.

[15] Henrik Madsen. «Grey Box Modelling of Hydrological Systems With Focus on Uncertainties». Tesis doct. Technical University of Denmark (DTU), 2011.

[16] Christodoulos A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. New York, NY: Oxford University Press, 1995. ISBN: 9780195095295.

[17] Phil Husbands. *Genetic Algorithms for Scheduling*. Inf. téc. Accedido: 24 de octubre de 2025. University of Sussex, 1996.

[18] À. Corominas y J. M. de la Fuente. *GENETIC ALGORITHMS FOR SHOP SCHEDULING PROBLEMS: A SURVEY*. Inf. téc. Accedido: 24 de octubre de 2025. Universitat Politècnica de Catalunya, 2000.

[19] A. S. M. M. Jaber y S. M. B. M. Shapiai. «Performance Comparison of Metaheuristic Optimization Algorithms in Solving Production Scheduling Problems». En: *Journal of Intelligent Computation and Technology* 1.1 (2021), págs. 1-11.

[20] M. B. M. Kamar y M. A. M. Ali. «A comparative study of metaheuristics algorithms based on their performance of complex benchmark problems». En: *Decision Making: Applications in Management and Engineering* 6.1 (2023). Accedido: 24 de octubre de 2025.

[21] William Braham. *White, Black, and Gray-Box Modelling*. University of Pennsylvania. 2025. URL: https://www.design.upenn.edu/work/white-black-and-gray-box-modelling.

[22] M. Rügner y col. «Physics-based machine learning predictions despite or in combination with scarce data». En: *PMSA* 1.1 (2021). URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC8069582/.

[23] S. E. W. T. M. H. van der Meer. «Hybrid Modelling by Machine Learning Corrections of Analytical Model Predictions towards High-Fidelity Simulation Solutions». En: *ResearchGate* (2021). URL: https://www.researchgate.net/publication/350813252_Hybrid_Modelling_by_Machine_Learning_Corrections_of_Analytical_Model_Predictions_towards_High-Fidelity_Simulation_Solutions.

[24] Y. Tao y col. «Multi-objective optimization of water distribution networks based on non-dominated sequencing genetic algorithm». En: *PLoS ONE* 17.11 (2022). URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0277954.

[25] Various. «Optimization of Water Distribution Networks Using Genetic Algorithm Based SOP–WDN Program». En: *MDPI* 14.6 (2022). URL: https://www.mdpi.com/2073-4441/14/6/851.

[26] Larry W. Mays. *Water Distribution Systems Handbook*. McGraw-Hill, 2001. Cap. Hydraulic Principles of Pipe Flow.

[27] J. Keller y D. Karmeli. «Trickle Irrigation Design Parameters». En: *Journal of the Irrigation and Drainage Division* 101.IR3 (1975), págs. 275-291.

[28] Various. *Genetic Algorithms for Combinatorial Optimization Problems*. Inf. téc. Ghent University, 2005. URL: https://backoffice.biblio.ugent.be/download/1147278/1147528.

[29] Various. «A Review of Timetable Scheduling System Using Genetic Algorithm». En: *International Journal of Trend in Research and Development* 6.1 (2019). URL: https://www.researchgate.net/publication/392508312_A_Review_of_Timetable_Scheduling_System_Using_Genetic_Algorithm.

[30] Various. «Variable-Length Chromosome Genetic Algorithm for Network Scheduling Problem». En: *PMC* (2021). URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC8228978/.

[31] Various. «A Comparative Review Between Various Selection Techniques In Genetic Algorithm For Finding Optimal Solutions». En: *ResearchGate* (2022). URL: https://www.researchgate.net/publication/364952362_A_Comparative_Review_Between_Various_Selection_Techniques_In_Genetic_Algorithm_For_Finding_Optimal_Solutions.

[32] Various. «Truncation Selection Operator for Enhancing the Performance of Wireless Sensor Networks». En: *MDPI* 11.1 (2022). URL: https://www.mdpi.com/2079-9292/11/1/28.

[33] Taylor L. Shaw y col. «WNTR: A Water Network Tool for Resilience». En: *Journal of Computing in Civil Engineering* 32.2 (2018), pág. 04017086. DOI: 10.1061/(ASCE)CP.1943-5487.0000720.

[34] Adam Paszke y col. «PyTorch: An Imperative Style, High-Performance Deep Learning Library». En: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, págs. 8024-8035.

[35] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015. ISBN: 9781491950357.

[36] Thomas Schlachter y Felix Sager. «Hybrid modeling of physical systems using neural networks for residual correction». En: *Journal of Process Control* 96 (2020), págs. 1-11. DOI: 10.1016/j.jprocont.2020.10.002.

[37] The Apache Software Foundation. *Apache Airflow Documentation*. https://airflow.apache.org/docs/. Consultado el 27 de octubre de 2025. 2024.

[38] Matei Zaharia y col. «MLflow: A Platform for the Machine Learning Lifecycle». En: *Proc. of the 2nd International Workshop on Systems for ML* (2018). Presentado en SysML 2018. URL: https://www.mlflow.org/.

[39] Kathryn Holovaty y Matthew Kopecky. «MLOps: Principles and Practices». En: *Gartner Research Report* (2020). Informe de arquitectura sobre el ciclo de vida de modelos en producción.

[40] Sebastián Ramírez. *FastAPI Documentation*. Framework web moderno y de alto rendimiento para construir APIs con Python. 2024. URL: https://fastapi.tiangolo.com/.

[41] Docker Inc. *Docker Swarm Mode Overview*. Herramienta de orquestación nativa para gestionar clústeres de Docker. 2024. URL: https://docs.docker.com/engine/swarm/.

[42] F5, Inc. *NGINX Documentation*. Servidor web y balanceador de carga de alto rendimiento. 2024. URL: https://nginx.org/en/docs/.