Índice general

III

Índice general

Índice de figuras

v

Índice de figuras

- 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).

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

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.

2.4. Herramientas computacionales y métricas 11

- 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.

3.3. Implementación del optimizador de despacho (GA)

3.4. Implementación de funciones de costo y penalizaciones

20 Capítulo 3. Diseño e implementación

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.

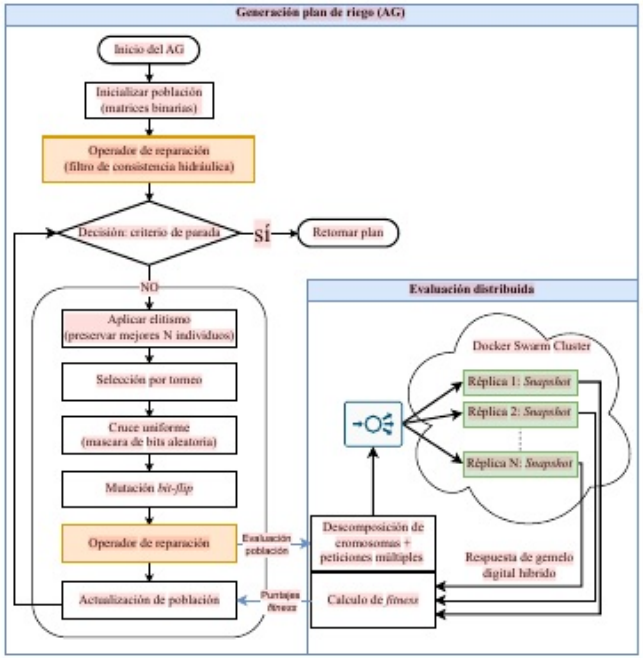

FIGURA 3.3. Diagrama de flujo del Algoritmo Genético Distribuido, destacando la etapa de evaluación paralela y el operador de reparación de restricciones físicas.

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

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}).

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

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
```

```
        # 3. Aplicar mascaras para crear los hijos
        child1 = (parent1 * mask) | (parent2 * mask_inv)
        child2 = (parent2 * mask) | (parent1 * mask_inv)

    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.

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).

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:

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

[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.

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.

[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/.

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.