

**Universidad Nacional Mayor de San Marcos**  
Universidad del Perú. Decana de América

**Facultad de Ingeniería de Sistemas e Informática**



## **Asignación Óptima de Recursos y Tareas en Metodologías Ágiles**

**Aplicación de Algoritmos Genéticos en Entornos SCRUM**

### **Integrantes:**

Salazar Garcia, Diego  
Arias Chumpitaz, Giovanni  
Arroyo Vasquez, Luis  
Lavaud Guevara, Jean

**Docente:** Claudio Arango

**Lima, Perú  
2025**

## Resumen

En el presente trabajo se explora el uso de algoritmos genéticos para resolver el problema de asignación óptima de tareas dentro de equipos ágiles. La metodología empleada permite balancear la carga entre desarrolladores, reducir tiempos de entrega y mejorar la gestión de dependencias entre tareas. Los resultados preliminares muestran una mejora significativa frente a enfoques tradicionales.

**Palabras clave:** *algoritmos genéticos, asignación de tareas, SCRUM, metodologías ágiles*

## Abstract

This paper explores the use of genetic algorithms to solve the problem of optimal task assignment within agile teams. The methodology allows for workload balancing, reduction of delivery times, and improved management of task dependencies. Preliminary results show a significant improvement compared to traditional approaches.

**Keywords:** *genetic algorithms, task allocation, SCRUM, agile methodologies*

### Sugerencias para esta sección

- Reemplaza los textos por un resumen real cuando finalices el informe.
- En español e inglés, resume: objetivo, metodología, resultados clave y conclusiones.
- Evita listas o fórmulas; usa prosa clara, breve y técnica.
- Las palabras clave deben estar relacionadas con los principales temas del proyecto.

# 1. Definición del Problema

En entornos ágiles como SCRUM, una de las dificultades más recurrentes es la asignación eficiente y equitativa de tareas entre los miembros del equipo de desarrollo. Esta asignación debe considerar múltiples factores simultáneamente, tales como la capacidad individual de cada desarrollador, la urgencia de las tareas, las dependencias entre actividades, las habilidades requeridas y los costos asociados al trabajo.

En la práctica, esta distribución suele realizarse de forma manual o semi-automática, lo que da lugar a diversos problemas: algunos desarrolladores terminan sobrecargados mientras otros están subutilizados, las tareas críticas o urgentes pueden quedar rezagadas, y las dependencias mal gestionadas provocan cuellos de botella en el flujo de trabajo del sprint. Además, si se ignora el nivel de habilidad requerido para cada tarea, pueden generarse asignaciones ineficientes que comprometen la calidad del producto o retrasan su entrega.

Este problema se vuelve aún más complejo al considerar que cada tarea puede requerir habilidades específicas en diferentes niveles, que la capacidad de los desarrolladores no siempre es uniforme (por ejemplo, algunos trabajan 30 horas por sprint, otros 40), y que el costo por hora de cada colaborador puede influir significativamente en el presupuesto del proyecto.

Frente a esta complejidad combinatoria y multifactorial, se requiere un enfoque de optimización inteligente que permita encontrar asignaciones que equilibren el uso de recursos, minimicen el tiempo total del proyecto, atiendan prioridades, y reduzcan costos. En este contexto, el uso de algoritmos genéticos se presenta como una alternativa viable para explorar eficientemente un espacio de soluciones altamente complejo.

## 2. Objetivos

### Objetivo General

Diseñar e implementar un sistema de asignación automática y óptima de tareas en entornos ágiles tipo SCRUM, utilizando algoritmos genéticos que consideren múltiples restricciones reales como carga de trabajo, habilidades requeridas, prioridades, dependencias y costos.

### Objetivos Específicos

- Balancear la carga de trabajo entre los desarrolladores, considerando sus capacidades horarias individuales.
- Asignar tareas a los desarrolladores cuyas habilidades coincidan en mayor medida con los requerimientos técnicos de cada tarea.

- Minimizar el tiempo total de ejecución del conjunto de tareas.
- Garantizar que las dependencias entre tareas se respeten en la planificación del sprint.
- Minimizar el costo total del proyecto considerando el costo por hora de cada desarrollador.
- Diseñar una función de evaluación que combine múltiples criterios mediante un sistema de pesos ajustables.

### 3. Alcance del Proyecto

Este trabajo aborda la implementación de un sistema de asignación automática de tareas en entornos SCRUM, considerando aspectos clave como capacidades horarias variables entre desarrolladores, habilidades técnicas diferenciadas, esfuerzo estimado de las tareas en horas, complejidad y prioridades. El sistema emplea un algoritmo genético como núcleo de optimización, buscando asignaciones que equilibren la carga, respeten dependencias y minimicen el costo total del proyecto.

El desarrollo se limita a la simulación de un equipo de desarrollo y no contempla integración directa con plataformas ágiles comerciales ni modelado de otros roles más allá de los desarrolladores. Tampoco se abordan requerimientos no funcionales como persistencia de datos, seguridad o despliegue en producción, ya que el enfoque es validar la viabilidad técnica del enfoque propuesto como un *proof of concept* adaptable a escenarios reales.

### 4. Marco Teórico

La gestión ágil de proyectos ha revolucionado la forma en que los equipos de desarrollo organizan su trabajo. En particular, el marco metodológico SCRUM propone una estructura iterativa e incremental para el desarrollo de productos, basada en ciclos de trabajo denominados *sprints* y en la colaboración constante entre los miembros del equipo (Schwaber & Sutherland, 2020). Uno de los principales desafíos dentro de SCRUM es la asignación eficiente de tareas a los desarrolladores, lo que implica considerar factores como la carga de trabajo, las habilidades requeridas, las prioridades y las dependencias técnicas entre actividades.

En entornos reales, dicha asignación suele realizarse manualmente por un *Scrum Master* o mediante juicio experto, lo cual puede introducir sesgos, sobrecarga en algunos miembros o secuencias ineficientes que retrasen el proyecto. Por ello, automatizar la asignación de tareas se vuelve una necesidad crítica para mejorar la eficiencia y la toma de decisiones en equipos ágiles (Masood et al., 2017). Como señala Maiello (2023), automatizar procesos dentro de metodologías ágiles puede representar una mejora significativa en términos de eficiencia y gestión operativa.

Una técnica destacada para abordar problemas de asignación con múltiples restricciones es el uso de **algoritmos genéticos** (AG). Estos forman parte de la computación evolutiva y se inspiran en los principios de la selección natural y la genética biológica propuestos por Darwin. Introducidos formalmente por Holland (1975), los algoritmos genéticos permiten buscar soluciones óptimas o casi óptimas en espacios de búsqueda complejos donde las técnicas tradicionales de optimización resultan ineficaces.

El funcionamiento de un algoritmo genético parte de una población inicial de soluciones aleatorias (denominadas *cromosomas*), que evolucionan mediante operadores como la selección, el cruce (*crossover*) y la mutación. Cada solución se evalúa mediante una función de *fitness*, la cual determina qué tan buena es en relación con el problema que se quiere resolver (Goldberg, 1989).

En el contexto de SCRUM, el cromosoma puede representar una asignación de tareas a desarrolladores. La función de fitness puede considerar criterios como el balance de carga, el cumplimiento de habilidades requeridas, el respeto por las dependencias y la minimización del costo total. Estudios como el de García Nájera y Gomez (2014) han demostrado que los algoritmos genéticos multiobjetivo son adecuados para resolver problemas complejos de planificación de proyectos de software, en los que se deben considerar simultáneamente restricciones como habilidades requeridas, precedencias entre tareas, tiempo de ejecución y costos.

A su vez, existen antecedentes de problemas similares en la literatura, como el problema de asignación de personal (Staff Assignment Problem, SAP), en el que se busca distribuir empleados a tareas bajo múltiples restricciones y objetivos simultáneos. Peters et al. (2019) abordaron un caso real de este tipo en una firma de servicios profesionales, proponiendo un algoritmo genético multiobjetivo que superó en eficiencia y calidad de soluciones a métodos clásicos como la programación entera mixta (MIP). Su enfoque demostró que los algoritmos evolutivos pueden ofrecer soluciones altamente competitivas en contextos reales, incluso bajo limitaciones de tiempo y condiciones operativas complejas.

En ese sentido, los algoritmos genéticos ofrecen una herramienta flexible y poderosa para resolver problemas de asignación en contextos ágiles, ya que permiten incorporar múltiples objetivos, adaptarse a restricciones cambiantes y explorar soluciones más allá de la intuición humana. Su uso en el presente proyecto permite simular un entorno realista y automatizado para la planificación eficiente de sprints en equipos SCRUM.

## 5. Metodología

La presente sección detalla la secuencia de desarrollo de la solución propuesta, partiendo del análisis del problema hasta su implementación como prueba de concepto (PoC). Se presenta además un flujograma del algoritmo, la arquitectura conceptual del sistema y los aspectos técnicos empleados en la validación de los resultados.

### 5.1. Etapas del Desarrollo

El desarrollo del sistema se dividió en las siguientes etapas:

1. **Análisis del problema:** Se identificaron las principales dificultades en la asignación de tareas en entornos ágiles, especialmente bajo restricciones de habilidades, dependencias y tiempos.
2. **Modelado:** Se definieron entidades clave como *tareas*, *desarrolladores*, sus atributos y relaciones. Se eligió el algoritmo genético como técnica de resolución.
3. **Diseño e implementación:** Se programó el algoritmo genético en Python, integrando módulos para evaluación de soluciones (fitness), selección por torneo, cruza (crossover) y mutación. También se desarrolló un backend con Flask y un frontend con React.
4. **Validación:** Se ejecutaron simulaciones con datos ficticios para analizar la efectividad del sistema bajo distintas configuraciones.

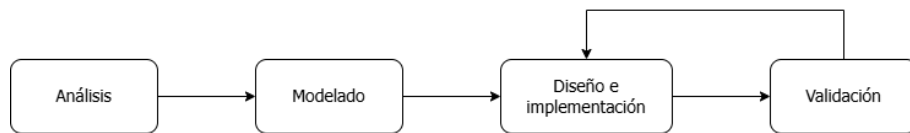


Figura 1: Resumen visual de las etapas del desarrollo del sistema.

## 5.2. Flujograma del Algoritmo

La Figura 2 muestra un flujograma general del proceso evolutivo del algoritmo genético implementado.

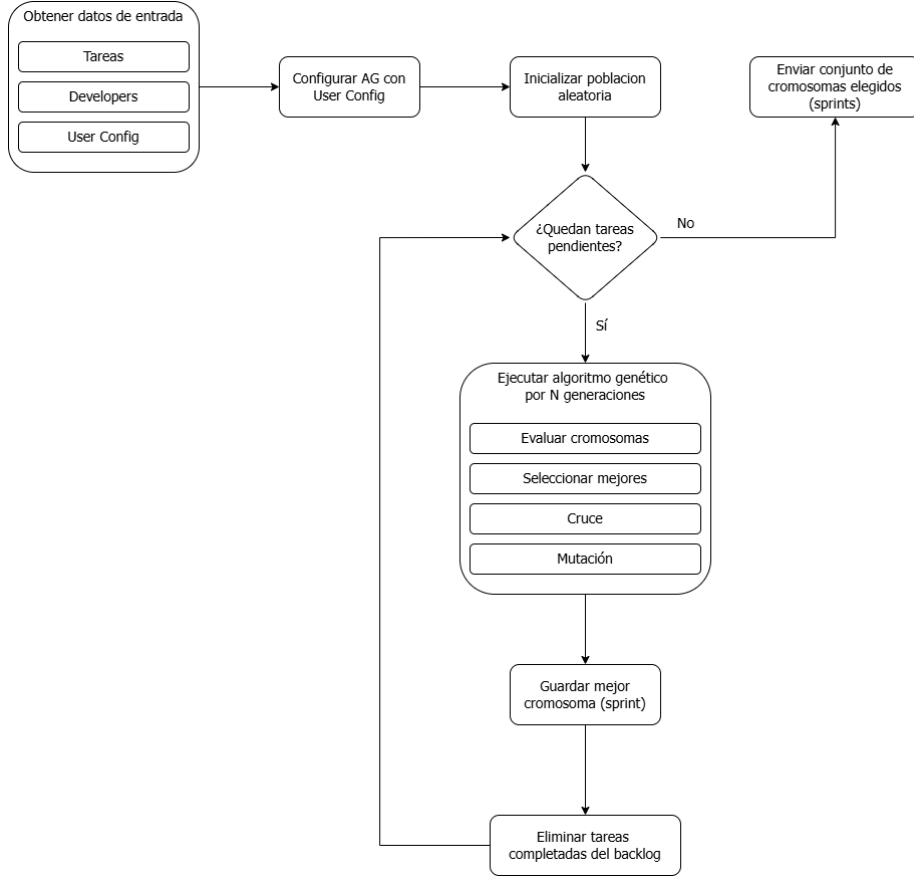


Figura 2: Flujograma general del algoritmo genético

## 5.3. Arquitectura del Sistema

La arquitectura general del sistema sigue un modelo cliente-servidor, compuesto por tres capas principales: el frontend, la API backend y el módulo del algoritmo genético. Cada componente cumple un rol específico y se comunican entre sí mediante peticiones HTTP.

- **Frontend:** Desarrollado con React y Vite, es la interfaz gráfica con la que el usuario interactúa para configurar parámetros del algoritmo y visualizar los resultados.
- **Backend/API:** Implementado con Flask en Python, actúa como intermediario entre el frontend y el algoritmo genético. Recibe las configuraciones del usuario y retorna los sprints generados.

- **Algoritmo Genético:** Encapsulado en el backend, este componente ejecuta la lógica evolutiva: inicialización de población, evaluación, selección, cruza, mutación y generación de nuevos sprints.

El siguiente esquema resume la interacción entre componentes:

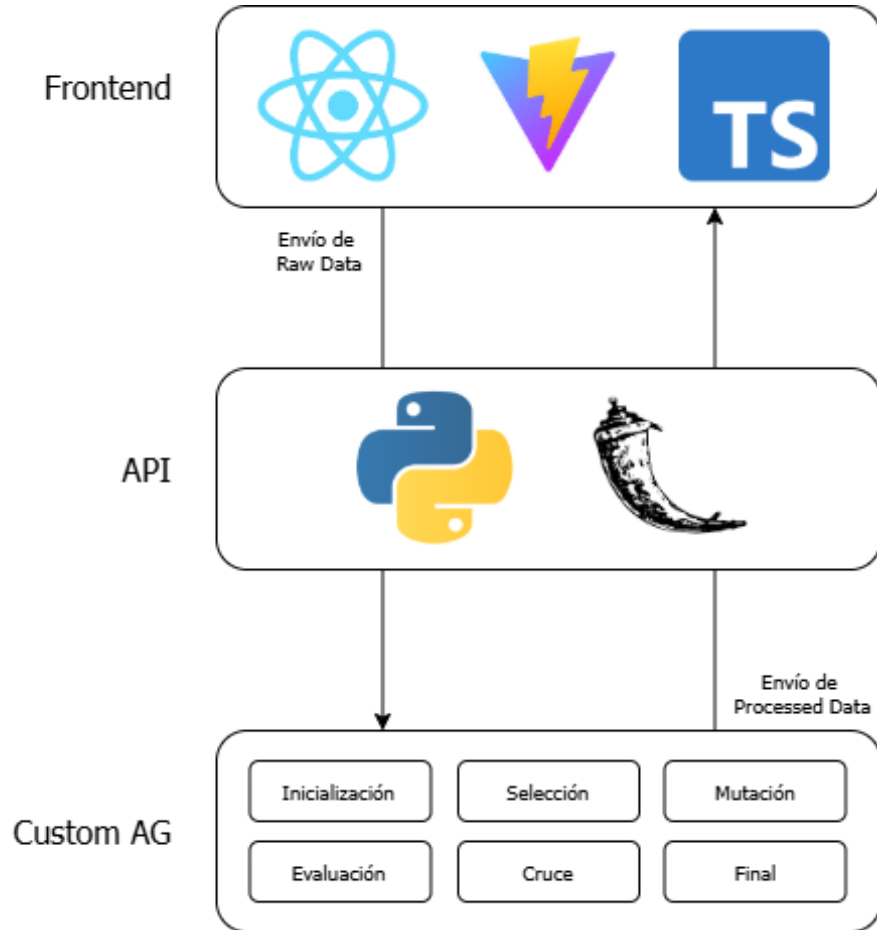


Figura 3: Arquitectura general del sistema.

#### 5.4. Validación mediante Pruebas Simuladas

Para validar el sistema, se utilizaron conjuntos de datos representativos de un equipo ágil ficticio. Las simulaciones incluyeron tareas con diferentes niveles de prioridad, complejidad y dependencias, así como desarrolladores con habilidades y capacidades diversas.

Aunque los datos de tareas y desarrolladores eran estáticos (predefinidos en el frontend), se realizaron múltiples experimentos variando los parámetros configurables del algoritmo genético (como el tamaño de la población, la tasa de mutación, la cantidad de generaciones, los pesos de la función objetivo, etc.). Esto permitió analizar el comportamiento del sistema bajo distintas condiciones.



Además, el desarrollo del algoritmo se llevó a cabo de forma incremental y validada. Cada módulo del algoritmo genético —como la inicialización, evaluación, selección por torneo, cruza y mutación— fue implementado y probado por separado. Posteriormente, se integraron de forma progresiva hasta alcanzar la versión final con todas las restricciones consideradas: cumplimiento de dependencias, balance de carga, coincidencia de habilidades, y penalización por tareas no asignables o desarrolladores sobrecargados.

Las métricas utilizadas para evaluar la efectividad del sistema incluyeron:

- **Tiempo total estimado del sprint** (*makespan*).
- **Balance de carga** entre desarrolladores.
- **Coincidencia de habilidades** entre tareas y asignados.
- **Cumplimiento de dependencias** entre tareas.
- **Costo estimado** de asignación basado en las horas de cada desarrollador.

Los resultados obtenidos demuestran que el sistema logra generar sprints viables y balanceados en tiempos razonables, y que responde de manera coherente ante variaciones en los parámetros evolutivos.

## 5.5. Herramientas y Tecnologías Utilizadas

El desarrollo del sistema se apoyó en una serie de herramientas y tecnologías modernas que facilitaron la implementación modular y colaborativa del proyecto. A continuación se detallan:

- **Lenguajes de programación:** Python para el backend y JavaScript/TypeScript para el frontend.
- **Frameworks y entornos:**
  - *Flask* para la creación de la API REST.
  - *React* y *Vite* para el desarrollo del frontend, con enrutamiento gestionado mediante React Router.
- **Control de versiones:** Git y GitHub fueron utilizados para el seguimiento del desarrollo y la colaboración en equipo.
- **Entorno de desarrollo:** Visual Studio Code fue el editor principal, junto con extensiones y herramientas como Thunder Client para pruebas de API.
- **Diagramación:** Draw.io fue empleado para la elaboración de flujogramas y diagramas de arquitectura.
- **Documentación:** La redacción del informe se realizó utilizando LaTeX, empleando una plantilla compartida mediante GitHub.

## 6. Propuesta de Solución

La propuesta presentada consiste en un sistema interactivo orientado a entornos ágiles, que permite optimizar la asignación de tareas dentro de un equipo de desarrollo mediante el uso de un algoritmo genético multiobjetivo. La solución busca balancear la carga de trabajo, minimizar el tiempo total de ejecución y considerar factores clave como dependencias, habilidades requeridas y costos asociados.

El sistema se estructura en torno a una arquitectura modular que separa claramente la interfaz de usuario, la lógica de negocio y el motor de optimización. Esta separación permite una interacción clara y eficiente con el usuario, al mismo tiempo que encapsula la complejidad de la heurística evolutiva implementada.

En las siguientes subsecciones se detalla el modelo general del sistema, el diseño del algoritmo genético propuesto, y la forma en que los componentes se integran para generar una solución viable y extensible al problema de asignación de tareas.

### 6.1. Modelo General del Sistema

Como se representó anteriormente en la Figura 3, el sistema está compuesto por tres componentes principales: **Interfaz Gráfica**, **API** y **Núcleo de Optimización**.

- **Interfaz Gráfica:** Permite al usuario visualizar los resultados y configurar parámetros clave del algoritmo genético, como el tamaño de la población, número de generaciones, tasas de cruce y mutación, tamaño del torneo y los pesos de la función objetivo.
- **API:** Recibe las configuraciones del usuario y los datos estáticos de tareas y desarrolladores. Actúa como intermediario entre la interfaz gráfica y el núcleo del algoritmo, enviando y recibiendo información a través de endpoints definidos en Flask.
- **Núcleo de Optimización:** Implementado en Python, este módulo contiene la lógica completa del algoritmo genético. Se encarga de generar soluciones viables y optimizadas para la asignación de tareas, respetando restricciones de dependencias, habilidades, tiempos y costos.

Esta aproximación cliente-servidor con separación clara entre capas permite al usuario interactuar de forma sencilla con la aplicación, facilitando la configuración de parámetros y la visualización de resultados, todo mientras se mantiene encapsulada la complejidad interna del algoritmo genético multiobjetivo para la optimización de la asignación de tareas.

## 6.2. Algoritmo Genético Propuesto

El núcleo de optimización sigue un enfoque evolutivo, donde una población inicial de soluciones (cromosomas) se somete a iteraciones de evaluación, selección, cruce y mutación para mejorar la asignación de tareas.

- Se parte de una población aleatoria de asignaciones.
- Cada generación aplica operadores genéticos y se evalúan las soluciones.
- La mejor solución se toma como sprint generado.
- El proceso se repite hasta vaciar el backlog.

**Imagen:** (*opcional*) flujograma simple del algoritmo. Ej: pseudodiagrama basado en 'while tareas\_pendientes:'.

## 6.3. Representación de Soluciones

Cada cromosoma representa una posible asignación de tareas a desarrolladores. Esta representación codifica:

- Qué tareas se asignan a qué programadores.
- El orden en que deben completarse (respetando dependencias).
- El ajuste según las habilidades de cada desarrollador.

**Imagen:** (*opcional*) ejemplo visual de un cromosoma o tabla de asignaciones (opcional si ya lo describes en texto).

## 6.4. Evaluación de Soluciones (Función Objetivo)

La función objetivo pondera múltiples criterios para determinar la calidad de una solución:

- **Makespan (50 %):** duración del sprint.
- **Balance de carga (25 %):** varianza en horas por developer.
- **Coincidencia de habilidades (20 %):** afinidad entre habilidades requeridas y ofrecidas.
- **Costo (5 %):** según tarifas por hora.

**Imagen:** (*opcional*) fórmula matemática o pseudocódigo de evaluación.

## 6.5. Operadores Genéticos Personalizados

Los operadores han sido ajustados para respetar restricciones del problema:

- **Selección:** por torneo.
- **Cruce:** combinación parcial de asignaciones entre padres.
- **Mutación:** reasignación de tareas con baja probabilidad.
- **Corrección:** validación para evitar duplicados y tareas inválidas.

**Imagen:** no requerida, puede ser solo texto.

## 6.6. Interfaz y Configuración del Usuario

El sistema permite que el usuario ajuste los parámetros del algoritmo genético desde una interfaz web amigable:

- Población inicial
- Número de generaciones
- Tasa de mutación y cruce
- Tamaño del torneo
- Pesos para cada criterio de evaluación

**Imagen:** captura del frontend (panel de configuración).

## 6.7. Ejemplo de Ejecución

Se realizó una ejecución completa con el conjunto estático de tareas y desarrolladores. Se configuraron los parámetros del algoritmo y se generaron varios sprints:

- Se obtuvo un conjunto de sprints con carga balanceada.
- Las tareas respetaron todas sus dependencias.
- El makespan fue razonable para el conjunto dado.

**Imagen:** *opcional* captura de los resultados visualizados (por ejemplo, tabla con desarrolladores y tareas).

## 7. Conclusiones

El enfoque basado en algoritmos genéticos ha demostrado ser una herramienta eficiente para la asignación de tareas en entornos SCRUM. Se ha logrado un balance adecuado en la distribución del trabajo, respetando restricciones y optimizando el tiempo total del proyecto.

Este trabajo sienta las bases para aplicaciones reales en la industria del software, y abre la posibilidad de integrar técnicas similares en procesos más amplios de gestión ágil.

### Sugerencias para esta sección

- Resume los aprendizajes clave del trabajo realizado.
- Redacta al menos 3 conclusiones claras y bien separadas si es posible.
- Puedes mencionar limitaciones y oportunidades de mejora.
- Evita repetir el resumen o los objetivos textualmente.

## 8. Recomendaciones

A partir del análisis y la implementación realizada, se proponen las siguientes recomendaciones para futuras investigaciones o mejoras en sistemas SCRUM asistidos por algoritmos genéticos:

- Integrar herramientas de visualización para facilitar la interpretación de las asignaciones realizadas por el algoritmo.
- Explorar técnicas híbridas que combinen algoritmos genéticos con heurísticas basadas en reglas de negocio.
- Validar el modelo con datos reales de proyectos ágiles en diferentes industrias.
- Evaluar el impacto del algoritmo en la satisfacción y productividad del equipo.

### Sugerencia para esta sección

Formula recomendaciones específicas, factibles y que se deriven directamente de los resultados de tu propuesta. Considera aspectos como mejoras técnicas, escalabilidad o nuevas direcciones de investigación.

## Bibliografía

- García Nájera, A., & Gomez, M. (2014). A Multi-Objective Genetic Algorithm for the Software Project Scheduling Problem. *Lecture Notes in Computer Science*, 8857. [https://doi.org/10.1007/978-3-319-13650-9\\_2](https://doi.org/10.1007/978-3-319-13650-9_2)
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.
- Maiello, A. S. (2023). Task Optimization utilizing Digital Transformation Concepts-Automation Project Execution via AGILE Methodology.
- Masood, Z., Hoda, R., & Blincoe, K. (2017). Exploring workflow mechanisms and task allocation strategies in agile software teams. *Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22-26, 2017, Proceedings 18*, 267-273.
- Peters, J., Stephan, D., Amon, I., Gawendowicz, H., Lischoid, J., Salabarria, L., Umland, J., Werner, F., Krejca, M. S., Rothenberger, R., Kötzing, T., & Friedrich, T. (2019). Mixed Integer Programming versus Evolutionary Computation for Optimizing a Hard Real-World Staff Assignment Problem. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 29(1), 541-554. <https://doi.org/10.1609/icaps.v29i1.3521>
- Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. <https://scrumguides.org/scrum-guide.html>