Pontificia Universidad Católica de Chile

Magister en Inteligencia Artificial

Desarrollo de un sistema de programación automatizada de tareas de mantenimiento en la industria minera

Emilio Bravo Maturana
Profesor Guía: Rodrigo Sandoval Urrich

Santiago, Chile.
Diciembre 2024

Título

Desarrollo de un sistema de programación automatizada de tareas de mantenimiento en la industria minera.

Autor

Emilio Bravo Maturana

Temática

Programación de Tareas, Algoritmos de Optimización, Resource-Constrained Project Scheduling Problem (RCPSP), Constraint Programming.

Resumen

El mantenimiento eficiente de equipos es crucial para garantizar la continuidad operativa en cualquier industria. Sin embargo, la programación manual de estas actividades es compleja, intensiva en tiempo y propensa a errores debido a las múltiples restricciones y variables involucradas. Esto genera una alta dependencia de personal especializado para llevar a cabo la programación de tareas.

Este proyecto tiene como objetivo principal automatizar el proceso de programación de tareas de mantenimiento, liberando al equipo de programación de trabajo manual intensivo y reduciendo significativamente la ocurrencia de errores. Para ello, el problema se modela como un **Problema de Programación de Proyectos con Restricciones de Recursos** (RCPSP, por sus siglas en inglés). Una generalización correcta del proceso permite utilizar herramientas de optimización para generar cronogramas que cumplan con todas las restricciones del problema, minimizando la función objetivo y mejorando la eficiencia general.

El enfoque busca diseñar un algoritmo que no solo encuentre soluciones óptimas, sino que también ofrezca rapidez y flexibilidad para adaptarse a cambios intempestivos. A través de un estudio detallado y pruebas exhaustivas, se evaluará el desempeño del modelo en términos de ahorro de tiempo, reducción del uso de recursos computacionales y calidad de las soluciones generadas, con el fin de garantizar una planificación eficiente y confiable.

Este texto enfatiza la automatización, la reducción de errores y la liberación del trabajo manual como objetivos centrales.

Índice

1. Introducción

1.1. Contexto

El mantenimiento de equipos es una actividad fundamental en cualquier industria que busca garantizar la continuidad operativa y la eficiencia en sus procesos productivos. Una planificación y programación adecuada de las actividades de mantenimiento no solo previene fallos inesperados, sino que también optimiza el uso de recursos y minimiza costos asociados a tiempos de inactividad. Sin embargo, la programación eficiente de estas actividades representa un desafío significativo debido a las múltiples restricciones y variables involucradas, como la disponibilidad de personal, recursos materiales, prioridades de tareas y la necesidad de evitar sobreasignaciones y tiempos muertos.

Tradicionalmente, la planificación del mantenimiento ha dependido de equipos humanos que, a pesar de su experiencia y conocimiento, enfrentan limitaciones en términos de tiempo y capacidad para manejar grandes volúmenes de datos y restricciones complejas. Este proceso manual es propenso a errores y puede no garantizar una solución óptima que maximice la eficiencia operativa. En este contexto, surge la necesidad de automatizar este proceso mediante el uso de algoritmos de optimización que puedan abordar de manera efectiva la complejidad inherente de la programación de mantenimiento.

El problema de programación de mantenimiento puede modelarse como un Problema de Programación de Proyectos con Restricciones de Recursos (RCPSP), un enfoque clásico en la investigación operativa para resolver problemas de planificación bajo condiciones de recursos limitados. En este modelo, las tareas de mantenimiento se consideran actividades con duraciones definidas, precedencias entre ellas y recursos compartidos, como personal o maquinaria, cuya disponibilidad es limitada. Las restricciones de precedencia aseguran que ciertas tareas no pueden comenzar hasta que otras hayan finalizado, mientras que las restricciones de recursos garantizan que las capacidades disponibles no sean excedidas en ningún momento.

El objetivo es encontrar una programación que minimice el tiempo total de ejecución (makespan) o que optimice el uso de recursos, lo que resulta en una planificación más eficiente y acorde con las demandas operativas. Modelar el problema de esta forma permite aplicar algoritmos de optimización avanzados, como algoritmos de Constraint Programming, para generar soluciones factibles y eficientes.

El presente proyecto tiene como objetivo principal proponer un algoritmo para automatizar la programación de actividades de mantenimiento, reduciendo tanto el tiempo dedicado al proceso como la ocurrencia de errores. Se evaluarán distintas funciones objetivo para determinar cuál entrega los mejores resultados en términos de eficiencia y calidad de las soluciones generadas. Para ello, se contará con el apoyo del área de mantenimiento de una empresa del rubro minero, quienes proporcionarán datos para realizar las pruebas y ofrecerán retroalimentación durante el desarrollo de la solución. Esta colaboración permitirá ajustar el modelo a las necesidades específicas y desafíos particulares del entorno operativo, asegurando que los resultados sean relevantes y aplicables.

1.2. Consideraciones

En el contexto de la empresa de del rubro minero, es importante contextualizar la terminología específica utilizada en la programación de tareas. A continuación, se definen algunos términos relevantes para facilitar la comprensión del problema.

Cronograma: El cronograma es la representación temporal de la programación de todas las

actividades a lo largo del tiempo. En este proyecto, el cronograma se materializa en una carta GANTT que muestra todas las tareas con su fecha de inicio, hora de inicio y duración. Es una herramienta clave para visualizar y gestionar la ejecución del mantenimiento.

Orden de Trabajo: Una orden de trabajo es un conjunto de tareas que se deben realizar para cumplir con un objetivo específico, como el mantenimiento de un equipo o vehículo. Estas órdenes de trabajo se ingresan en el sistema ERP de la empresa y sirven para coordinar y gestionar las actividades de mantenimiento o producción.

Tareas: Actividades individuales dentro de una orden de trabajo. Estas tareas pueden tener requisitos de precedencia, es decir, algunas deben completarse antes de que otras puedan comenzar. Cada tarea cuenta con una fecha de inicio más temprana (earliest date), una fecha requerida de finalización, una duración, una cantidad de trabajadores necesaria y recursos específicos que se requieren para su ejecución.

Criticidad: La criticidad clasifica las tareas según su nivel de prioridad, en una escala del 1 al 3. Las tareas con criticidad 1 son las más urgentes y deben programarse primero, ya que es grave si no se completan dentro de su ventana de tiempo. Las tareas con criticidad 2 y 3 son menos prioritarias y pueden permitirse más flexibilidad en su programación.

Recursos: Los recursos son los elementos limitados que se necesitan para ejecutar las tareas. En este problema, se identifican dos tipos principales de recursos: *Cuadrillas* y *Equipos compartidos*.

Cuadrilla: La cuadrilla representa un grupo de trabajo asignado a una tarea específica dentro de la orden de trabajo. Cada cuadrilla tiene un número determinado de trabajadores y sigue un esquema de turnos específico, trabajando un número fijo de horas por día y un determinado número de días a la semana. La cuadrilla es responsable de ejecutar las operaciones asignadas dentro de su turno.

Equipos compartidos: En este proyecto, los equipos compartidos se refieren a los recursos que son utilizados por varias cuadrillas y que tienen una disponibilidad limitada. Estos equipos forman parte de los *Production Resource Tools* (PRT) y son críticos en la programación, ya que su uso debe gestionarse cuidadosamente para evitar conflictos y sobreasignaciones. Ejemplos típicos de estos equipos incluyen grúas, camiones pluma, *boom trucks*, *forklifts*, y alza hombres.

Turno: El turno define el esquema de trabajo de una cuadrilla. Por ejemplo, un turno 7x7 implica 7 días de trabajo seguidos por 7 días de descanso. Otros esquemas, como el 5x2A, 5x2B o 5x2C, representan 5 días de trabajo con 2 días de descanso, donde las letras A, B y C indican diferentes franjas horarias: A cubre las primeras 8 horas del día, B las siguientes 8, y C las últimas. Estos turnos permiten que las cuadrillas cubran las 24 horas del día de manera continua.

Planificación: La planificación es el proceso llevado a cabo por el equipo de planificadores, quienes establecen una ventana de tiempo en la que deben realizarse las tareas de mantenimiento. Este proceso se realiza considerando los requerimientos de las máquinas, los tiempos de los procesos industriales y otros factores relacionados con las operaciones del negocio. Es una etapa inicial que no contempla las restricciones de recursos, sino que establece un marco general para la ejecución de las actividades.

Programación: La programación es el proceso mediante el cual el equipo de programación toma la planificación inicial y organiza las tareas de manera detallada, asignando recursos específicos y fechas exactas. Durante esta etapa, se balancean los recursos disponibles para evitar sobreasignaciones y garantizar que todas las tareas puedan completarse dentro de sus ventanas de tiempo. La

programación precisa y eficiente es clave para minimizar errores y optimizar el uso de los recursos.

2. Descripción detallada y levantamiento del problema

Con el objetivo de automatizar el proceso de programación de tareas en el rubro minero, se llevó a cabo un estudio de los documentos que norman el proceso de programación de tareas en la empresa colaboradora. Este análisis permitió comprender en detalle cómo funciona la programación de tareas en el contexto operativo, y por qué resulta esencial generalizar dicho proceso en un modelo matemático o computacional. El propósito final es implementar una solución automatizada que utilice un algoritmo de optimización capaz de entregar resultados satisfactorios.

A continuación se presenta una descripción del proceso de programación de tareas en el área de mantenimiento de esta empresa del rubro minero, con el fin de identificar los elementos clave y las restricciones involucradas en el problema.

2.1. Proceso de Planificación

Previo al proceso de programación, el equipo de planificación es responsable de definir, con base en los requerimientos de las máquinas, los tiempos de los procesos industriales, y otros factores relacionados con el negocio minero, las ventanas temporales en las que deben ejecutarse las tareas de mantenimiento. Esto da como resultado una planificación inicial sobre la cual se basa el proceso de programación.

La planificación considera los objetivos estratégicos de la mina y prioriza el cumplimiento de los requerimientos operativos. Como resultado, se genera una planificación inicial que indica, para cada tarea, la ventana dentro de la cual debe ser realizada. Sin embargo, este proceso no contempla las restricciones de recursos (personal, equipos o herramientas) y asume la disponibilidad de estos en todo momento.

2.2. Proceso de Programación

El equipo de programación recibe esta planificación inicial como /textitinput y realiza un trabajo detallado para adaptarla a las capacidades reales de los recursos disponibles. Su labor principal es balancear los recursos, asegurando que no haya sobreasignaciones y que todas las tareas puedan ser ejecutadas dentro de sus respectivas ventanas. Esto incluye:

- **Balancear recursos**: Ajustar la programación para que ningún recurso esté sobreexigido y que todas las tareas cuenten con los recursos necesarios para su ejecución.
- **Asignar fechas y horarios específicos**: Situar con precisión cada tarea en un calendario, definiendo su inicio y fin en función de la disponibilidad de recursos y el cumplimiento de las ventanas definidas por la planificación.
- **Coordinar accesos**: Garantizar que las cuadrillas, equipos y espacios de trabajo estén disponibles y asignados adecuadamente para cada tarea.
- **Ajustar y validar la programación**: Consultar a las areas relacionadas para confirmar la viabilidad del cronograma y obtener su visto bueno.

El cronograma mínimo resultante detalla las tareas a realizar, la fecha y hora de inicio, la duración y los recursos necesarios. Este proceso también considera tareas en función de su criticidad, priorizando aquellas que tienen mayor impacto en la continuidad operativa. Este proceso también considera tareas en función de su criticidad, priorizando aquellas que tienen mayor impacto en la continuidad operativa.

2.3. Ciclo Semanal de Programación

El proceso de programación en el área de mantenimiento se organiza en un ciclo operativo semanal, que se repite de manera sistemática. Este ciclo tiene como objetivo garantizar que el cronograma generado sea factible, actualizado y aprobado por todas las áreas involucradas, de manera que se mantenga alineado con los requerimientos del negocio y la disponibilidad real de recursos.

El ciclo comienza cada miércoles, cuando el equipo de programación toma como base la programación generada la semana anterior. A partir de esta, se revisan las tareas previamente planificadas y programadas, incorporando las nuevas tareas ingresadas durante la semana, así como cualquier cambio en la disponibilidad de recursos. Con esta información, los programadores generan un borrador del cronograma que abarca tres semanas: la semana siguiente y las dos subsiguientes.

Durante este proceso, el borrador es iterado y validado en conjunto con otras áreas involucradas en el proceso de mantenimiento. Esta interacción tiene como objetivo verificar que las tareas estén correctamente distribuidas y que no existan conflictos en la asignación de recursos. Además, permite ajustar el cronograma en función de las prioridades o requerimientos emergentes.

El ciclo culmina cada viernes, cuando se emite el cronograma oficial que establece la programación para las tres semanas siguientes.

La semana siguiente, el equipo de programación retoma este proceso, partiendo de la base del cronograma generado el viernes anterior. Sin embargo, dado que continuamente ingresan nuevas tareas y los recursos disponibles pueden variar, el equipo revisa y actualiza el cronograma para reflejar estos cambios. Este enfoque iterativo garantiza que la programación se mantenga dinámica y adaptable a las condiciones operativas cambiantes, al tiempo que preserva la continuidad y consistencia en la ejecución de las actividades.

Este ciclo operativo es un componente clave del proceso de programación, ya que asegura que el cronograma se mantenga actualizado y alineado con las necesidades del negocio, mientras se minimizan los conflictos en el uso de recursos. Además, destaca la importancia de la coordinación entre áreas y la validación constante para garantizar un cronograma factible y efectivo.

2.4. Consideraciones al programar

El proceso de programación de actividades de mantenimiento implica que el programador tome las órdenes de trabajo pendientes y agende cada una de las tareas considerando su compatibilidad con los horarios y restricciones de otras actividades y de los recursos. Se deben tener en cuenta las siguientes consideraciones

- Cada **tarea** viene acompañada de información esencial, asignada en el proceso de planificación. Dentro de esta información tenemos la fecha de inicio extrema, la fecha requerida, la duracion, los recursos requeridos y la criticidad.
- Cada **cuadrilla** tiene su propio esquema de turnos, lo que implica que hay ventanas de tiempo en las cuales estos recursos no están diponibles.
- Cada **cuadrilla** tiene una capacidad determinada, que corresponde al número de trabajadores disponibles, lo que influye en la cantidad de tareas que pueden realizarse simultáneamente. Por ejemplo, si una cuadrilla tiene una capacidad de 10 trabajadores, podrá ejecutar 5 tareas que requieran 2 trabajadores cada una de manera simultánea.
- Los **equipos compartidos** son unitarios, por lo tanto no pueden asignarse a mas de una tareas simultaneamente.

3. Definiciones teóricas

En este apartado se presentan las definiciones y conceptos teóricos fundamentales para comprender el enfoque del proyecto. Se abordan el Problema de Programación de Proyectos con Restricciones de Recursos (RCPSP) y los algoritmos de Programación por Restricciones. Estos conceptos son esenciales para entender las metodologías utilizadas en la optimización de la programación de actividades de mantenimiento.

3.1. Problema de Programación de Proyectos con Restricciones de Recursos (RCPSP)

El RCPSP (*Resource-Constrained Project Scheduling Problem*) es un problema clásico en el campo de la investigación operativa y la gestión de proyectos. Consiste en programar un conjunto de actividades interrelacionadas, considerando restricciones tanto de precedencia entre tareas como de disponibilidad limitada de recursos. El objetivo principal es determinar el calendario óptimo que minimice la duración total del proyecto (*makespan*) o que optimice otro criterio relevante, como costos o utilización de recursos.

El RCPSP es conocido por ser un problema NP-Hard, lo que implica que no existe un algoritmo eficiente que pueda resolver todas las instancias del problema en tiempo polinomial. Esta complejidad se debe al crecimiento exponencial del número de posibles soluciones a medida que aumenta el tamaño del problema, es decir, el número de actividades y recursos involucrados. Por esta razón, los métodos exactos son viables solo para problemas de pequeña escala, mientras que para instancias más grandes se recurre a métodos heurísticos y metaheurísticos que proporcionan soluciones aproximadas en tiempos razonables.

Las principales características del RCPSP incluyen:

- **Restricciones de precedencia**: Algunas actividades no pueden comenzar hasta que otras hayan finalizado
- **Recursos limitados**: Los recursos necesarios para realizar las actividades (como mano de obra, equipos o materiales) tienen una disponibilidad limitada en cada periodo de tiempo.
- **Objetivo de optimización**: Generalmente, se busca minimizar el tiempo total del proyecto, aunque pueden considerarse otros objetivos como minimizar costos o equilibrar la carga de recursos.

El RCPSP es altamente aplicable en la programación de actividades de mantenimiento industrial, donde es necesario coordinar múltiples tareas con recursos compartidos y restricciones temporales.

3.2. Algoritmos de Programación por Restricciones

Los algoritmos de Programación por Restricciones (CP) son técnicas avanzadas de optimización que se utilizan para resolver problemas combinatorios complejos, como el Problema de Programación de Proyectos con Restricciones de Recursos (RCPSP). Estos algoritmos son especialmente efectivos debido a su capacidad para manejar de manera eficiente restricciones lógicas, aritméticas y de recursos.

La Programación por Restricciones es un paradigma en el que se definen variables, dominios y restricciones. Las variables representan los elementos desconocidos del problema, los dominios especifican los posibles valores que pueden tomar, y las restricciones limitan las combinaciones de valores que las variables pueden asumir simultáneamente. El objetivo es encontrar asignaciones de

valores a las variables que satisfagan todas las restricciones impuestas. Este enfoque es especialmente útil cuando las restricciones son numerosas y complejas, permitiendo modelar problemas de manera flexible y detallada.

En el contexto del RCPSP, los algoritmos de Programación por Restricciones ofrecen un marco poderoso para modelar y resolver el problema de manera eficiente. Se definen variables de inicio y fin para cada tarea, así como variables de intervalo que combinan inicio, duración y fin, facilitando el manejo de restricciones temporales y de recursos. Las restricciones de precedencia se establecen para asegurar que una tarea no pueda comenzar hasta que sus tareas precedentes hayan finalizado, lo cual se expresa mediante relaciones de desigualdad entre las variables de fin e inicio de las tareas involucradas.

Las restricciones de recursos se manejan mediante la restricción cumulativa, que garantiza que, en cualquier momento, la suma de los recursos consumidos por las tareas en ejecución no exceda la capacidad disponible. Esto es crucial en problemas donde los recursos son limitados y deben ser compartidos entre múltiples tareas. Además, se consideran las ventanas de tiempo para las tareas, imponiendo límites en las variables de inicio y fin según las fechas de inicio más tempranas y las fechas límite, lo que refleja la disponibilidad y los plazos específicos de cada tarea.

La función objetivo en estos problemas suele ser minimizar el makespan, es decir, el tiempo total del proyecto. Los algoritmos de Programación por Restricciones permiten integrar esta función objetivo en el modelo, buscando no solo soluciones factibles que satisfagan todas las restricciones, sino también optimizar este criterio para mejorar la eficiencia global del proyecto.

Una de las ventajas clave de los algoritmos de Programación por Restricciones es su eficiencia en el manejo de restricciones complejas, permitiendo resolver problemas NP-Hard como el RCPSP en tiempos razonables. Su flexibilidad permite incorporar fácilmente nuevas restricciones o modificar las existentes sin reestructurar todo el modelo, lo que es especialmente útil en entornos dinámicos donde las condiciones pueden cambiar. Además, son capaces de encontrar soluciones de alta calidad, óptimas o cercanas al óptimo, lo cual es esencial en contextos industriales donde la optimización de recursos y tiempos es crítica.

4. Aplicación de los conceptos teóricos al problema

Con el objetivo de automatizar el proceso de programación de tareas, se llevó a cabo una revisión de los documentos de procedimientos utilizados en la empresa además de entrevistas a personal clave. Este análisis permitió comprender en detalle cómo funciona la programación de tareas en el contexto operativo, y por qué resulta esencial generalizar dicho proceso en un modelo matemático o computacional. Esta generalización es clave, ya que proporciona la base necesaria para desarrollar una herramienta que permita automatizar el problema de programación de tareas de manera eficaz y eficiente.

4.1. Datos necesarios para el proceso de programación

El proceso de programación requiere como *input* el resultado del proceso de planificación, que define los parámetros esenciales para cada tarea. Esta información incluye una descripción general de la tarea, indicando a qué orden de trabajo corresponde, además de datos sobre la ventana de fechas en la que debe ejecutarse. También se especifica la duración de la tarea, su nivel de criticidad y los recursos necesarios para ejecutarla, como cuadrillas y equipos. El detalle de una tarea estándar con toda esta información se puede consultar en el Cuadro 1.

Además de la información proveniente del proceso de planificación, es necesario extraer datos adicionales desde el sistema ERP de la empresa. Esto incluye la disponibilidad de las cuadrillas, la cual se determina a partir de los esquemas de turnos que definen las ventanas de tiempo en las que estas pueden trabajar. También es fundamental considerar los equipos compartidos, cuya disponibilidad debe gestionarse cuidadosamente, ya que estos pueden ser utilizados por diferentes cuadrillas en momentos distintos.

La integración de estos datos asegura que el proceso de programación tenga toda la información necesaria para asignar las tareas de manera eficiente, respetando las restricciones operativas y de recursos disponibles.

Atributo	**Descripción**
ID Orden de Trabajo	4004934163
Descripción OT	Cambio de suspensión trasero lado izquierdo
ID Tarea	0050
Descripción Tarea	Retiro Suspensión Trasera
Fecha Requerida	2024-12-30
Fecha Inicio Extrema	2024-07-29
Duración (horas)	6
Impacto	2
Puesto de Trabajo	BM0BATM1
Cantidad de Trabajadores	4
Herramienta Requerida	Alza Homb Telescop Jlg 20 M

Cuadro 1: Ejemplo de Tarea Estándar

4.2. Generalización del Problema como RCPSP

Vamos a modelar el problema de programación de tareas como un problema de optimización con restricciones, específicamente como una versión del RCPSP. El objetivo es asignar tiempos de inicio a las tareas, maximizando la prioridad de las tareas programadas según su impacto, y respetando las restricciones de capacidad de los recursos, las ventanas de tiempo, los intervalos prohibidos y las posibles relaciones de precedencia entre tareas.

A continuación, se presenta una descripción general del modelo de programación de tareas con recursos limitados que servirá como base para la implementación de la solución.

4.2.1. Parámetros del Modelo

Tenemos los siguientes elementos:

- **Tareas**: Un conjunto $T = \{T_0, T_1, \ldots, T_n\}$, donde cada tarea T_i tiene:

  - Una duración $C(T_i) \in \mathbb{N}$.
  - Un conjunto de recursos requeridos $D(T_i) \subseteq R$.
  - Una cantidad de recurso requerido $q_i \in \mathbb{N}$.
  - Un impacto $\text{Impact}_i \in \mathbb{N}$, donde un valor más bajo indica mayor prioridad.
  - Una ventana de tiempo $W(T_i) = [\text{early}_i, \text{late}_i]$, que indica el tiempo más temprano y más tardío en el que puede comenzar la tarea.

- **Recursos**: Un conjunto $R = \{R_0, R_1, \dots, R_m\}$, donde cada recurso R_j tiene:

  - Una capacidad $P(R_j) \in \mathbb{N}$.
  - Un tipo $\text{Tipo}(R_j) \in \{0, 1\}$. Donde 0 indica que el recurso es una cuadrilla y 1 indica que es un equipo compartido.
  - Un conjunto de intervalos prohibidos $I(R_j) = \{[a_1, b_1), [a_2, b_2), \dots\}$, durante los cuales el recurso no está disponible.

- **Grupos de tareas**: Un conjunto $G = \{G_0, G_1, \dots\}$, donde cada grupo $G_k \subseteq T$ tiene restricciones de precedencia entre las tareas que lo componen.

Parámetros del modelo

- $T = \{T_0, T_1, \dots, T_n\}$: conjunto de **tareas**.
- $R = \{R_0, R_1, \dots, R_m\}$: conjunto de **recursos**.
- $C(T_i) \in \mathbb{N}$: **duración** de la tarea T_i.
- $D(T_i) \subseteq R$: **recursos requeridos** por la tarea T_i.
- $q_i \in \mathbb{N}$: **cantidad** de recurso que requiere la tarea T_i.
- $\text{Impact}_i \in \mathbb{N}$: **impacto** de la tarea T_i (prioridad inversa).
- $W(T_i) = [\text{early}_i, \text{late}_i]$: **ventana de tiempo** para la tarea T_i.
- $P(R_j) \in \mathbb{N}$: **capacidad** del recurso R_j.
- $\text{Tipo}(R_j) \in \{0, 1\}$: **tipo** del recurso R_j.
- $I(R_j) = \{[a_k, b_k)\}$: **intervalos prohibidos** del recurso R_j.
- $G = \{G_0, G_1, \dots\}$: conjunto de **grupos de tareas** con restricciones de precedencia.

4.2.2. Variables de decisión

Para modelar el problema, introducimos las siguientes variables de decisión:

- $x_i \in \{0, 1\}$: Indica si la tarea T_i es **programada** ($x_i = 1$) o no ($x_i = 0$).
- $S_i \in \mathbb{N}$: Tiempo de **inicio** de la tarea T_i, sujeto a $\text{early}_i \leq S_i \leq \text{late}_i - C(T_i)$ si $x_i = 1$.
- $E_i = S_i + C(T_i)$: Tiempo de **finalización** de la tarea T_i.
- $\text{Intervalo}(T_i) = [S_i, E_i)$: Intervalo de ejecución de la tarea T_i.
- $y_k \in \{0, 1\}$: Indica si el **grupo de tareas** G_k es programado ($y_k = 1$) o no ($y_k = 0$).

Variables de decisión

- $x_i \in \{0, 1\}$: Variable binaria que indica si la tarea T_i es **programada**.
- $S_i \in \mathbb{N}$: Tiempo de **inicio** de la tarea T_i.
- $E_i = S_i + C(T_i)$: Tiempo de **finalización** de la tarea T_i.
- $\text{Intervalo}(T_i) = [S_i, E_i)$: Intervalo de ejecución de la tarea T_i.
- $y_k \in \{0, 1\}$: Variable binaria que indica si el grupo G_k es **programado**.

4.2.3. Restricciones

El modelo considera las siguientes restricciones:

a. Ventanas de tiempo:

Cada tarea debe comenzar y finalizar dentro de su ventana de tiempo permitida si es programada:

$$x_i = 1 \implies \text{early}_i \leq S_i \leq \text{late}_i - C(T_i)$$

b. Restricciones de capacidad de los recursos:

Para cada recurso R_j, la suma de las demandas de las tareas que requieren el recurso en cualquier momento no debe exceder su capacidad:

$$\sum_{\substack{T_i \in T \\ R_j \in D(T_i)}} q_i \cdot \delta_{ij}(t) \leq P(R_j), \quad \forall t \in \text{Horizonte}$$

Donde $\delta_{ij}(t) = 1$ si $t \in [S_i, E_i)$ y $x_i = 1$; en caso contrario, $\delta_{ij}(t) = 0$.

El tipo de recurso afecta cómo se calcula la demanda:

- Si $\text{Tipo}(R_j) = 0$, se utiliza q_i como demanda.
- Si $\text{Tipo}(R_j) = 1$, cada tarea consume una unidad de capacidad ($q_i = 1$).

c. Intervalos prohibidos de los recursos:

Las tareas no pueden ser programadas durante los intervalos prohibidos de los recursos que requieren:

$$x_i = 1 \implies \forall R_j \in D(T_i), \forall [a_k, b_k) \in I(R_j) : \quad [S_i, E_i) \cap [a_k, b_k) = \emptyset$$

d. Restricciones de precedencia en grupos de tareas:

Para cada grupo G_k, si el grupo es programado, las tareas deben seguir una secuencia específica:

$$y_k = 1 \implies \forall T_i, T_{i+1} \in G_k : S_{i+1} \geq E_i$$

Además, las tareas dentro de un grupo se programan juntas o no se programan:

$$\forall T_i \in G_k : x_i = y_k$$

e. Consistencia de variables:

Si una tarea no es programada, sus variables de tiempo no tienen relevancia y pueden ser fijadas a cero para simplificar el modelo:

$$x_i = 0 \implies S_i = 0, \quad E_i = 0$$

4.2.4. Objetivo

El objetivo es minimizar una combinación de tres componentes: el makespan global, el makespan de los grupos y la penalización asociada a no programar tareas de mayor prioridad. Para esto, se define una función de peso para cada tarea basada en su impacto:

$$w_i = (\text{Impact}_{\max} + 1 - \text{Impact}_i)^3$$

Donde $\text{Impact}_{\max}$ es el valor máximo de impacto entre todas las tareas.

Tambien calculamos el makespan global como:

$$\text{makespan} = \max_{T_i \in T} E_i$$

El makespan de los grupos como:

$$\text{makespan}_{G_k} = \max_{T_i \in G_k} E_i$$

La función objetivo es:

$$\text{Minimizar } Z = \alpha \cdot \text{makespan} + \beta \cdot \sum_{k=0}^{|G|-1} \text{makespan}_{G_k} + \gamma \cdot \sum_{i=0}^{n} w_i \cdot (1 - x_i)$$

Donde:

- α: Peso asociado al makespan global.
- β: Peso asociado al makespan de los grupos de tareas.
- γ: Peso asociado a la penalización por no programar tareas según su impacto.

Este enfoque busca una solución que equilibre la minimización del makespan global, la duración de los grupos de tareas, y la priorización de tareas críticas según su impacto.

4.2.5. Resumen del Modelo

El modelo presentado integra múltiples aspectos del problema de programación de tareas con recursos limitados:

- **Ventanas de tiempo**: Asegura que las tareas se programen dentro de los intervalos permitidos.
- **Capacidad de recursos**: Garantiza que la demanda no exceda la capacidad disponible en ningún momento.
- **Intervalos prohibidos**: Evita la asignación de tareas durante periodos en los que los recursos no están disponibles.
- **Precedencia en grupos**: Mantiene el orden requerido entre tareas relacionadas.
- **Optimización basada en impacto**: Prioriza la programación de tareas más críticas según su impacto.
- **Minimización del makespan**: Incluye la minimización del makespan global y del makespan de los grupos.

5. Flujo de Trabajo

El flujo de trabajo de la solución está diseñado para estructurar y procesar la información de programación en cinco etapas secuenciales, asegurando una transición fluida desde los datos iniciales hasta los resultados finales. Estas etapas abarcan desde la ingesta de datos en su formato original hasta la generación de un cronograma optimizado y formatos exportables para su uso práctico. A continuación, se describen las cinco partes que componen el flujo de trabajo:

1. **Pre-Procesamiento**: Se toman los datos de las tareas desde distintas fuentes (ERP, Microsoft Project, u otros formatos) y se estandarizan, asegurando que los campos relevantes estén estructurados de manera uniforme.

2. **Input**: Los datos estandarizados se transforman al formato requerido por el solver, incluyendo duración de tareas, recursos, ventanas de tiempo y precedencias.

3. **Solver**: Se utiliza el modelo de CP de *OR-Tools* para resolver el problema aplicando restricciones y maximizando el cumplimiento de tareas según su prioridad.

4. **Output**: El solver genera un archivo JSON con el estado de programación de cada tarea, incluyendo tiempos de inicio, finalización y recursos asignados.

5. **Post-Procesamiento**: La salida del solver se procesa para generar un cronograma y archivos compatibles con herramientas como Microsoft Project, facilitando su integración en los flujos de trabajo.

Este flujo de trabajo modular garantiza flexibilidad y adaptabilidad, permitiendo procesar datos de diversas fuentes, optimizar la programación bajo restricciones complejas y generar resultados en formatos útiles para los usuarios finales. A continuación, se describen en detalle cada una de las etapas mencionadas.

5.1. Pre Procesamiento

El pre-procesamiento constituye la primera etapa del flujo de trabajo y se encarga de recibir y preparar la información de entrada relacionada con las tareas, el personal y los equipos. Esta etapa es fundamental para garantizar que los datos sean consistentes, estén limpios y normalizados, permitiendo al solver trabajar de manera eficiente y confiable. Dependiendo de la fuente de los datos (por ejemplo, un sistema ERP o un archivo externo), se realizan las siguientes operaciones clave:

- **Corrección de discrepancias**: Los datos se revisan para detectar y corregir inconsistencias. Por ejemplo, si una tarea contiene una fecha requerida que es anterior a su fecha de inicio extrema, esta discrepancia se ajusta automáticamente. Asimismo, se corrigen formatos erróneos en campos como fechas y duraciones para garantizar su compatibilidad con las etapas posteriores.
- **Cuantización de ventanas horarias**: Para que el solver pueda trabajar con números enteros, las ventanas horarias se cuantizan en intervalos de 15 minutos. Esto significa que la duración de una tarea de una hora se representa como 4 unidades en el solver, mientras que una tarea de 15 minutos se representa como una unidad. Este proceso permite que las restricciones temporales sean manejadas de manera eficiente dentro del modelo de optimización.
- **Generación de formato estándar**: Una vez completadas las etapas de limpieza y normalización, los datos se estructuran en un archivo JSON. Este archivo contiene toda la información relevante, como la duración de las tareas, los recursos requeridos, las ventanas

horarias y las dependencias, en un formato compatible con la primera etapa del solver.
- **Segmentación en subconjuntos**: Con el objetivo de mejorar la eficiencia del solver, las tareas se agrupan en subconjuntos relacionados. Este proceso se basa en identificar conexiones entre tareas mediante relaciones compartidas, como recursos comunes o dependencias temporales. La segmentación permite particionar el problema en componentes más manejables, reduciendo la complejidad computacional.

En conjunto, el pre-procesamiento asegura que los datos de entrada cumplan con los requisitos técnicos del modelo de optimización, eliminando posibles inconsistencias y adaptando los datos a un formato uniforme y eficiente. Esta etapa es crucial para garantizar el éxito de las etapas posteriores en el flujo de trabajo.

5.1.1. Detalle Segmentación en subconjuntos

Para abordar el problema de programación de tareas de manera más eficiente, se implementó una estrategia de simplificación basada en la división del conjunto de tareas en subconjuntos o *subsets*. Este enfoque busca reducir la complejidad computacional y permitir un tratamiento más localizado de las restricciones y dependencias entre las tareas.

Desde el proceso de planificación, las tareas analizadas vienen previamente asignadas a cuadrillas específicas, que son los grupos de trabajo responsables de ejecutarlas. En principio, esta asignación podría sugerir que el problema se puede simplificar al optimizar las tareas cuadrilla por cuadrilla, de forma independiente. Sin embargo, esta estrategia no es viable debido a que ciertas tareas requieren de equipos compartidos.

Estos requerimientos generan interdependencias que impiden tratar las cuadrillas como sistemas completamente independientes. Por lo tanto, para obtener una solución factible y eficiente, es necesario considerar las tareas y sus dependencias de forma global.

La estrategia propuesta se basa en identificar estas relaciones entre tareas para determinar qué tareas están efectivamente conectadas entre sí, y así dividir el problema en subconjuntos más pequeños, pero manteniendo la coherencia de las dependencias. Para lograrlo, se sigue un proceso usando herramientas de teoría de grafos y algoritmos de componentes conexas, el cual se describe a continuación:

a. **Construcción del grafo:** Se representa el conjunto de tareas como un grafo no dirigido, donde:

   - Cada nodo representa una cuadrilla o un equipo.
   - Se añade una arista entre una cuadrilla y una herramienta si ambas están asociadas a la misma tarea.

   Este grafo modela las dependencias entre las cuadrillas y las herramientas, proporcionando una representación visual y analítica de las relaciones entre las tareas.

b. **Identificación de componentes conexas:** Se identifican las componentes conexas del grafo, que representan grupos de nodos interrelacionados. Cada componente conexa corresponde a un subconjunto de tareas que comparten cuadrillas, equipos o ambas. Esto garantiza que las tareas que están relacionadas por el uso de recursos compartidos permanezcan juntas.

c. **Creación de subconjuntos de tareas:** Para cada componente conexa identificada, se agrupan las tareas asociadas a las cuadrillas y herramientas involucradas en esa componente.

Estas tareas conforman un subconjunto, que puede ser tratado de forma independiente en el proceso de optimización.

Este método presenta ventajas clave que facilitan el proceso de programación. En primer lugar, al dividir el problema en subconjuntos más pequeños, se simplifica la resolución al reducir el número de variables y restricciones a manejar. Además, las dependencias importantes, como el uso compartido de herramientas, permanecen dentro de cada subconjunto, lo que asegura la coherencia de las relaciones críticas. Por otro lado, los subconjuntos independientes pueden resolverse en paralelo, lo que disminuye significativamente el tiempo total de cálculo. Finalmente, este enfoque destaca por su adaptabilidad, ya que permite identificar de manera clara las interdependencias en los datos, haciendo que el modelo sea más comprensible y modular.

5.2. Input

La etapa de *Input* se encarga de transformar el archivo JSON generado en la etapa de pre-procesamiento en las estructuras de datos requeridas por el solver. Estas estructuras están diseñadas para representar de manera eficiente la información de las tareas, recursos y restricciones.

A partir del archivo JSON, se generan los siguientes diccionarios principales:

- **Diccionario de tareas**: Incluye la duración, recursos y herramientas requeridos para cada tarea, y define las variables clave del modelo, como inicio y fin.
- **Diccionario de ventanas de tiempo**: Especifica los tiempos más temprano y más tardío en los que puede comenzar cada tarea, asegurando el cumplimiento de las restricciones temporales.
- **Diccionario de capacidades de recursos**: Representa la capacidad máxima disponible de cada recurso, asegurando que no se excedan sus límites durante la programación.
- **Diccionario de agrupaciones de tareas**: Define grupos de tareas relacionadas que comparten recursos o requieren precedencia, facilitando su coordinación.
- **Diccionario de intervalos prohibidos**: Identifica los tiempos en que un recurso no está disponible, evitando programaciones conflictivas.

La generación de estos diccionarios es un paso clave para preparar los datos de entrada al solver, permitiendo que toda la información de las tareas y restricciones esté estructurada de manera óptima para el proceso de optimización. Esta etapa asegura que los datos se integren correctamente en el modelo y se procesen de forma eficiente.

5.3. Solver

Para resolver el problema de RCPSP, se emplea la librería `OR-Tools` de Google, que proporciona un conjunto de herramientas y solucionadores para problemas de optimización combinatoria. En particular, se utiliza el solucionador de Programación por Restricciones (CP) de `OR-Tools`, que permite modelar y resolver problemas de programación con restricciones de manera eficiente y escalable.

Para mejorar la eficiencia computacional, se implementa un enfoque de paralelización utilizando la librería *joblib* de Python. Este enfoque permite resolver los subconjuntos de tareas identificados en la etapa de pre-procesamiento de forma simultánea, aprovechando al máximo los recursos computacionales disponibles y reduciendo el tiempo total de cálculo.

Las restricciones se modelan de la siguiente forma:

- **Restricciones de Ventanas de Tiempo**: Para cada tarea, el modelo asegura que su tiempo de inicio esté dentro de su ventana temporal permitida. Estas restricciones se implementan mediante variables de intervalo opcional que habilitan o deshabilitan la programación de una tarea según su ventana de tiempo.
- **Restricciones de Intervalos Prohibidos**: Las tareas que requieren recursos específicos deben programarse fuera de los intervalos de tiempo en los que dichos recursos están ocupados o no disponibles. Esto se implementa mediante variables booleanas que indican si una tarea comienza o termina fuera de un intervalo prohibido, las cuales se imponen solo si la tarea es programada.
- **Restricciones Cumulativas para Recursos**: Para cada recurso, el modelo aplica una restricción cumulativa que asegura que el consumo de capacidad por parte de las tareas que lo utilizan no exceda su límite en ningún momento. Se especifica la demanda de cada tarea según el tipo de recurso (con capacidad de 1 unidad o más, dependiendo de la cuadrilla o del equipo adicional), y se controla el consumo de cada recurso con esta restricción.
- **Precedencia en Grupos de Tareas**: En los grupos de tareas (Órdenes de Trabajo), el modelo aplica restricciones de precedencia entre tareas específicas, asegurando que se respeten los órdenes de ejecución entre ellas si el grupo está programado. Esto se controla mediante la variable booleana, que activa las restricciones de precedencia dentro de cada grupo solo cuando el grupo completo ha sido asignado al cronograma.

La configuración del solucionador CP-SAT incluye dos parámetros clave, `max_time_in_seconds` y `relative_gap_limit`, que permiten limitar el tiempo de búsqueda y definir un margen de optimalidad aceptable, respectivamente:

- `max_time_in_seconds`: Se establece un límite de tiempo máximo para la búsqueda de soluciones, lo cual ayuda a evitar tiempos de procesamiento prolongados en caso de problemas complejos. En nuestro, el límite se ha configurado en 10 segundos.
- `relative_gap_limit`: Este parámetro permite especificar un *gap* relativo de optimalidad que representa el porcentaje aceptable de desviación de la solución encontrada respecto al óptimo teórico. En nuestro caso, se ha fijado en 10 %, lo cual permite una solución razonable en menos tiempo de procesamiento.

5.4. Output

La etapa de *Output* se encarga de procesar los resultados generados por el solver y estructurarlos en un formato JSON simple y claro. Este archivo JSON contiene, para cada tarea, información clave sobre su estado de programación y su tiempo de inicio, lo que facilita la interpretación de los resultados y su integración con las etapas posteriores del flujo de trabajo.

El archivo JSON generado incluye las siguientes variables por cada tarea:

- **Scheduled**: Indica si la tarea fue programada o no. Toma el valor de 1 si la tarea fue programada exitosamente y 0 si no pudo ser programada debido a conflictos de recursos o restricciones de tiempo.
- **Start**: Especifica la fecha y hora en la que comienza la tarea, en caso de que haya sido programada. Si la variable `Scheduled` es 0, entonces `Start` será `null`, indicando que la tarea no tiene una asignación en el cronograma.

Un ejemplo de salida en formato JSON es el siguiente:

```
{
```

```
    "TaskID_1": {
        "Scheduled": 1,
        "Start": "2023-07-01T08:00:00"
    },
    "TaskID_2": {
        "Scheduled": 0,
        "Start": null
    },
    ...
}
```

5.5. Post-Procesamiento

La etapa de *Post-Procesamiento* tiene como objetivo transformar los resultados del solver en formatos finales que sean compatibles con diversas herramientas de gestión y análisis. Esta etapa toma como entrada el JSON generado por el solver junto con los datos originales de entrada, y prepara archivos que consolidan toda la información relevante de las tareas y los recursos programados.

Durante esta etapa, se generan los siguientes outputs principales:

- **Archivo XML para Microsoft Project**: Este archivo contiene toda la información de las tareas programadas, incluyendo su hora de inicio, duración y hora de finalización. Es compatible con Microsoft Project, permitiendo que los resultados puedan integrarse fácilmente en esta herramienta para visualización y gestión avanzada.
- **Archivo XLSX para Excel**: Se genera una carta Gantt en formato Excel que organiza las tareas en un cronograma visual. Este archivo incluye detalles como:
  - Hora de inicio y hora de finalización de cada tarea.
  - Duración de las tareas en unidades temporales definidas.
  - Recursos asignados a cada tarea.
- **Integración con APIs o plataformas empresariales**: Los resultados también pueden ser enviados a las plataformas de gestión existentes en la empresa, como un ERP o sistemas similares, utilizando integraciones basadas en APIs. Esto permite que los datos programados sean automáticamente incorporados en el flujo de trabajo operativo de la organización.

6. Resultados

El desarrollo e implementación del modelo de optimización ha generado resultados prometedores, tanto en términos de la calidad del cronograma producido como en los beneficios operativos proyectados para el equipo de programación. A continuación, se presentan estadísticas clave y una evaluación de los ahorros potenciales derivados del uso de esta herramienta.

6.1. Mejoras en la duración total del cronograma

El análisis comparativo entre el cronograma generado manualmente y el producido por el solver muestra una reducción significativa en la duración total del proyecto (*makespan*). Con los datos del conjunto original, el *makespan* inicial era de 25,3 días, mientras que el modelo de optimización logró reducir este valor a 20,5 días, representando una mejora del 18,9 %. Este ahorro de tiempo no solo implica una mayor eficiencia en la ejecución de tareas, sino que también reduce el tiempo en que los equipos e instalaciones permanecen fuera de operación, impactando positivamente en los costos operativos.

Indicador	Valor
Makespan original	25,3 días
Makespan optimizado	20,5 días
Reducción en makespan	18,9 %

Cuadro 2: Reducción del *makespan* tras la optimización.

6.2. Ahorros estimados en tiempo del equipo de programación

Actualmente, el equipo de programación está compuesto por tres personas que dedican un promedio de 8 horas a la semana a la generación manual del cronograma. Con la implementación de esta herramienta, se estima que dicho tiempo podría reducirse considerablemente, ya que el solver automatiza gran parte del proceso, incluyendo la asignación de tareas y el balanceo de recursos. Esto representa un ahorro semanal estimado de 8 horas, que puede ser redirigido hacia otras actividades de mayor valor agregado, como la validación final o la coordinación con otras áreas.

Además, gracias a la capacidad del solver para realizar ajustes rápidos, el equipo podrá integrar tareas nuevas o modificar el cronograma de manera ágil, especialmente en situaciones donde surjan cambios inesperados en la planificación o disponibilidad de recursos. Esta funcionalidad es crucial para mantener la flexibilidad operativa y asegurar que el cronograma refleje las necesidades actuales en tiempo real.

6.3. Impacto en la reducción de errores humanos

Uno de los beneficios más significativos, aunque difícil de cuantificar en este momento, es la reducción de errores humanos. En los procesos manuales, es común que ocurran problemas como:

- Recursos doblemente agendados.
- Cuadrillas asignadas a tareas sin los recursos necesarios.
- Equipos programados en horarios donde no están disponibles.

Estos errores pueden generar retrasos en las operaciones de mantenimiento, aumentando el tiempo de inactividad de los equipos. En un entorno productivo, estos tiempos adicionales pueden traducirse en costos significativos debido a la falta de disponibilidad de equipos críticos. La automatización del proceso con un solver minimiza estas situaciones, ya que se asegura de cumplir todas las restricciones operativas de manera estricta.

6.4. Proyección de ahorros generales

Aunque no se puede calcular un valor exacto, la reducción en errores y el aumento en la eficiencia del proceso tienen el potencial de generar ahorros importantes para la empresa. Al minimizar los problemas operativos y los tiempos de inactividad no planificados, se reduce el impacto financiero asociado al mantenimiento no eficiente. Además, la capacidad de actualizar rápidamente el cronograma reduce la dependencia de procesos iterativos y manuales, haciendo que el equipo de programación sea más productivo y eficiente.

6.5. Pasos a seguir

La solución se desarrolla de manera modular, estructurándose en tres componentes principales: el backend, el frontend y la integración con las plataformas de la empresa. Esta división permite

organizar el flujo de trabajo de forma eficiente y facilita la colaboración con el equipo técnico de la empresa, asegurando que la solución se adapte adecuadamente a sus necesidades.

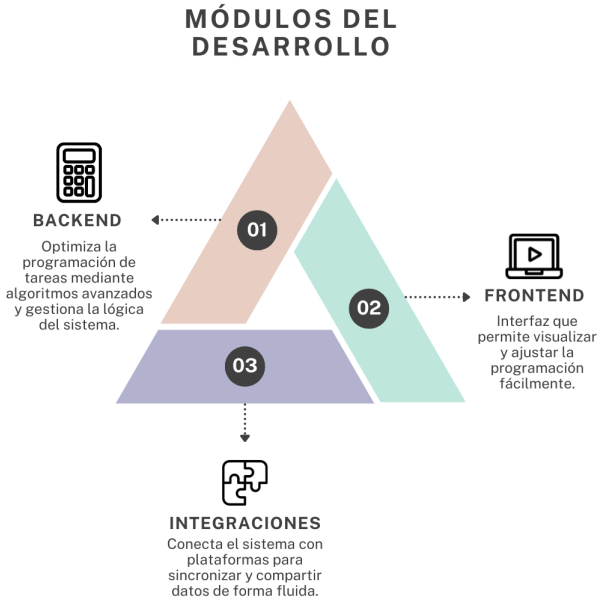

Figura 1: Módulos de Desarrollo de Solución

6.5.1. Backend

El backend representa el núcleo funcional de la solución y ejecuta el algoritmo de programación automatizada de tareas. Este algoritmo se selecciona en función de su capacidad para generar resultados de alta calidad que cumplan con las restricciones y optimicen la programación de tareas según los objetivos planteados. El backend recibe como entrada los datos de las órdenes de trabajo, que incluyen la información específica de cada tarea, tales como duraciones, recursos necesarios, precedencias y ventanas de tiempo.

Alojado en un servicio cloud de la empresa, el backend aprovecha la escalabilidad y el acceso seguro que este entorno proporciona. A partir de los datos ingresados, el backend genera como salida una programación detallada de tareas, especificando para cada una si fue programada y, en caso afirmativo, su hora de inicio. Este módulo se encarga de gestionar la lógica y optimización del proceso de programación, aplicando los algoritmos seleccionados, como CP-SAT, algoritmos genéticos u otros métodos heurísticos y metaheurísticos evaluados.

6.5.2. Frontend

El frontend funciona como la interfaz principal para los programadores de la empresa, quienes son los usuarios de esta solución. Su diseño se orienta a la facilidad de uso, permitiendo que los programadores carguen los datos de las órdenes de trabajo de manera sencilla. La carga de datos se puede realizar manualmente o mediante una integración con el sistema ERP de la empresa, de modo que los datos se incorporan automáticamente al sistema.

Una de las funciones clave del frontend es la visualización de la programación de tareas mediante

una carta Gantt interactiva, que permite a los usuarios visualizar y, si es necesario, ajustar manualmente la programación generada. Esta funcionalidad ofrece a los programadores la capacidad de evaluar y modificar las tareas programadas antes de aprobar la versión final de la programación. Una vez aprobada, esta información se envía a la plataforma de la empresa, notificando a todos los involucrados sobre la programación establecida.

6.5.3. Integración con Plataformas de la Empresa

El tercer módulo se encarga de la integración de la solución con las plataformas de la empresa. Este componente se desarrolla en colaboración estrecha con el equipo de TI de la empresa, asegurando que los formatos de entrada y salida de los datos sean compatibles con los sistemas existentes. La integración también abarca la configuración de conexiones seguras entre el backend y las plataformas corporativas, permitiendo un flujo de información bidireccional. De esta forma, los datos de las órdenes de trabajo y de la programación finalizada se comparten con el ERP y otros sistemas relevantes de la empresa, facilitando la comunicación y coordinación entre los distintos equipos.

6.6. Resumen de beneficios

En resumen, los principales beneficios obtenidos mediante la implementación del solver son los siguientes:

- Reducción del *makespan* en un 18,9 %, optimizando la duración total del cronograma.
- Ahorro estimado de 8 horas semanales en tiempo del equipo de programación.
- Mayor flexibilidad para realizar ajustes rápidos y manejar cambios intempestivos en el cronograma.
- Reducción en errores humanos, mejorando la precisión de la programación y minimizando los costos asociados a problemas operativos.

Estos resultados preliminares destacan el potencial del modelo para transformar el proceso de programación, haciéndolo más eficiente, preciso y adaptado a las necesidades dinámicas de la operación.