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, Mantenimiento Industrial, 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 la calidad de las soluciones generadas y del tiempo de procesamiento requerido.

El objetivo central de este trabajo es desarrollar una herramienta que permita (1) liberar al equipo de programación de tareas manuales intensivas, optimizando su tiempo y esfuerzo y (2) reducir los errores en las labores de mantenimiento, especialmente aquellos derivados de programaciones deficientes.

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 en los equipos, 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 por sus siglas en inglés), 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 y 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, que permita (1) liberar al equipo de programación de tareas manuales intensivas, optimizando su tiempo y esfuerzo y (2) reducir los errores en las labores de mantenimiento, especialmente aquellos derivados de programaciones deficientes.

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.

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.

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.

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 auxiliares: En este proyecto, los equipos auxiliares 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.

2. Análisis y Descripción del Problema de Programación de Tareas

Con el objetivo de automatizar el proceso de programación de tareas en el rubro minero, se realizó un estudio que combinó el análisis de los documentos normativos del proceso con una encuesta dirigida al personal encargado de la programación. Este enfoque permitió comprender el funcionamiento operativo actual, identificar las principales barreras y desafíos, y validar la necesidad de una solución automatizada. Entre los hallazgos se destacan problemas como la alta incidencia de cambios de último minuto, la falta de herramientas para optimizar tareas y las dificultades en la asignación eficiente de recursos.

Este diagnóstico resalta la importancia de generalizar el proceso en un modelo matemático o computacional que permita implementar una solución basada en algoritmos de optimización. El objetivo final es abordar las restricciones identificadas, mejorar la eficiencia y reducir la carga operativa del equipo de programación. A continuación, se presenta una descripción detallada del proceso actual, sus elementos clave y las restricciones involucradas, como punto de partida para el diseño de la solución propuesta.

2.1. Descripción del proceso actual

El proceso actual de programación de tareas en el área de mantenimiento comienza con una etapa previa de planificación, que, aunque no forma parte del proceso de programación, proporciona los *inputs* esenciales, como las ventanas temporales iniciales para las tareas. A partir de esta información, el equipo de programación adapta las tareas a las restricciones reales de recursos mediante un ciclo semanal iterativo. A continuación, se describen en detalle estas etapas y su interacción.

2.1.1. 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.

2.1.2. Proceso de Programación

El equipo de programación recibe esta planificación inicial como *input* 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:

- **Generar el borrador inicial**: Crear una primera versión del cronograma, integrando las tareas planificadas y asignando recursos preliminares. Este paso suele representar una parte significativa del proceso, dado que establece la base para los ajustes posteriores.
- **Resolver conflictos de recursos**: Identificar y solucionar problemas relacionados con la disponibilidad de recursos, evitando sobreasignaciones y asegurando que cada tarea tenga los recursos necesarios para su ejecución.
- **Validar con otras áreas**: Consultar y coordinar con las áreas relacionadas para garantizar la viabilidad del cronograma, obteniendo el visto bueno sobre la distribución de tareas y recursos.
- **Realizar ajustes finales**: Incorporar cambios de última hora y ajustar los detalles finales del cronograma para garantizar su alineación con las necesidades operativas.

El cronograma 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.

2.1.3. Ciclo Semanal de Programación

El proceso de programación en el área de mantenimiento se organiza en un ciclo semanal, cuyo objetivo es garantizar que el cronograma sea factible, actualizado y aprobado por las áreas involucradas, alineándose con los requerimientos del negocio y la disponibilidad de recursos.

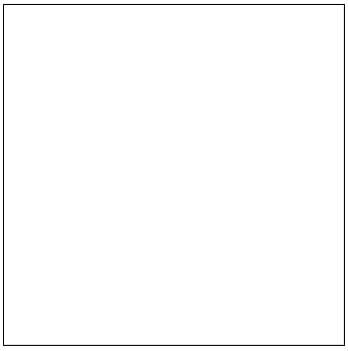

Figura 1: Flujo de Trabajo de la Solución.

Cada miércoles, el equipo de programación toma como base el cronograma generado en el ciclo de la semana anterior, que ya contiene actividades programadas para la semana actual. Este cronograma inicial es revisado y ajustado, incorporando nuevas tareas y actualizando los recursos disponibles. Con esta información, se genera un borrador del cronograma que abarca las próximas tres semanas.

El borrador se valida en conjunto con otras áreas para resolver conflictos de recursos y ajustar prioridades. Finalmente, cada viernes se emite el cronograma oficial, que servirá de base para el próximo ciclo. Este enfoque iterativo asegura que la programación sea dinámica y adaptable, optimizando la continuidad operativa y la coordinación entre áreas.

2.2. Diagnóstico del proceso actual

Como parte del levantamiento de información para comprender el estado actual del proceso de programación de tareas de mantenimiento, se realizó una encuesta dirigida al personal de programación de la empresa colaboradora. Los resultados permiten identificar aspectos críticos y validar la necesidad de herramientas automatizadas que optimicen la generación de cronogramas. Para ver el detalle de los resultados de la encuesta se puede consultar el Anexo A.

A continuación, se presentan los principales hallazgos y conclusiones derivados del diagnóstico del proceso actual:

- **Alta dedicación horaria**: 64 % de los encuestados invierte más de la mitad de su jornada semanal en generación del cronograma de actividades.
- **La creación del borrador inicial** es la fase más intensiva en tiempo, superando otras actividades como resolver conflictos o validar con otras áreas.
- **La no disponibilidad de materiales y repuestos** durante la ejecución de las mantenciones es el error más recurrente.
- **Entorno altamente dinámico**: El 90 % de los encuestados enfrenta cambios de último minuto al menos una vez por semana.
- **Problemas de coordinación con otras áreas** y **falta de colaboración** son barreras adicionales que dificultan la programación eficiente.

Los resultados refuerzan la hipótesis de que el proceso de programación podría beneficiarse significativamente de herramientas que automatizan partes clave del flujo de trabajo. En particular, una solución que automatice la creación del borrador inicial del cronograma y que permita incorporar cambios de último minuto sería altamente valorada. Se estima que una herramienta con estas capacidades podría reducir hasta en un 50 % el tiempo dedicado por los equipos de programación a

estas tareas, liberando tiempo para actividades de mayor valor estratégico.

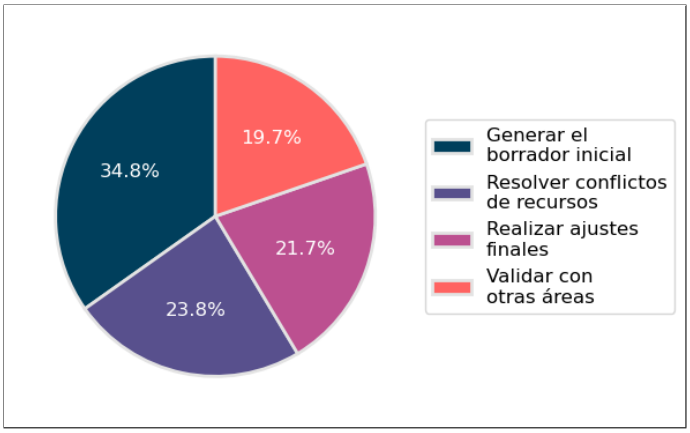

Figura 2: Distribución del tiempo de programación (Fuente: Elaboración propia en base a encuesta (Anexo A))

Si esta herramienta logra integrarse de manera efectiva con las fuentes de información de la empresa y se revisan los procedimientos para garantizar que estas fuentes proporcionen datos fidedignos, se estima que podría evitarse el 75 % de los errores en las labores de mantenimiento, los cuales están relacionados con la no disponibilidad de materiales, repuestos o equipos auxiliares.

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[1].

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.

En la Figura 3 se muestra un ejemplo de un RCPSP con 2 recursos y 10 tareas. En él se muestra que el Recurso 1 tiene una capacidad de 7 unidades mientras que el recurso 2 tiene una capacidad de 4 unidades. Dadas los requerimientos de recursos de cada tarea y las capacidades de cada recurso, en la figura se muestra una solución que logra un makespan de 12 unidades de tiempo.

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.

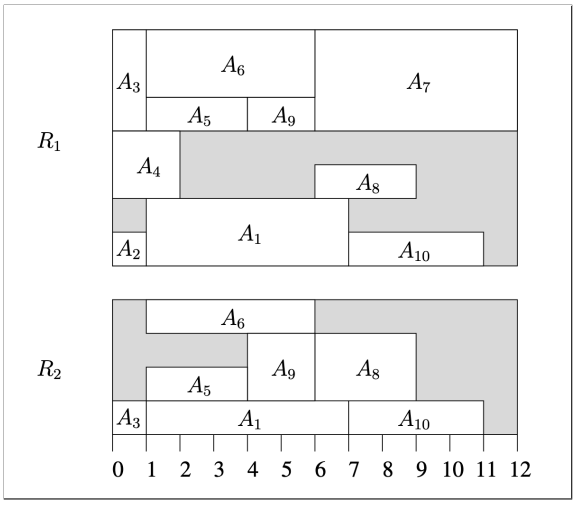

Figura 3: Ejemplo de RCPSP con 2 recursos y 10 tareas.

3.2. Algoritmos de Programación por Restricciones

La Programación por Restricciones (CP, por sus siglas en inglés) es una técnica ampliamente utilizada para resolver problemas combinatorios complejos, como los problemas de programación y planificación de actividades. En el contexto de este trabajo, CP es fundamental para abordar el *Problema de Programación de Proyectos con Restricciones de Recursos* (RCPSP), proporcionando un marco eficaz para modelar y resolver problemas que incluyen restricciones de tiempo y recursos.

Un problema de satisfacción de restricciones (*Constraint Satisfaction Problem*, CSP) se define como un conjunto de variables V , cada una asociada a un dominio de valores posibles D , y un conjunto de restricciones R que especifican las combinaciones válidas de valores entre las variables. La solución a un CSP es una asignación de valores a las variables que satisface todas las restricciones. Cuando se incorpora una función objetivo O para optimizar, el problema se denomina *Constraint Optimization Problem* (COP)[2].

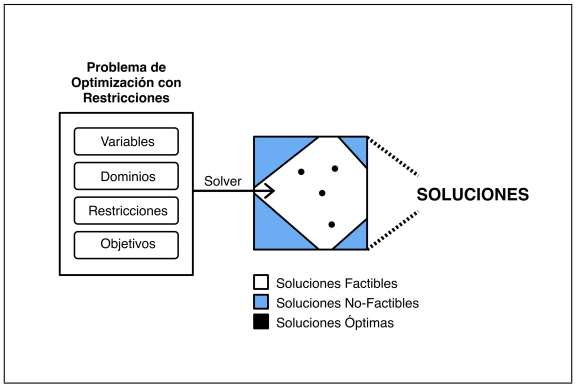

Figura 4: Esquema de Programación con Restricciones.

En el contexto del RCPSP, CP permite representar las actividades mediante variables de inicio, duración y fin. Las restricciones temporales, como las de precedencia, aseguran que el orden de ejecución sea respetado, mientras que las restricciones de recursos garantizan que no se exceda la capacidad disponible en ningún momento. Además, las ventanas temporales asociadas a cada tarea reflejan la disponibilidad de recursos y los plazos operativos.

La función objetivo de este tipo de problemas puede variar, siendo común la minimización del tiempo total de ejecución (*makespan*). La Programación por Restricciones permite, además, modelar estos problemas de manera declarativa, separando la especificación del modelo de los algoritmos utilizados para resolverlo. Esto facilita la extensión y modificación de los modelos, adaptándolos a las necesidades específicas sin comprometer la eficacia de los métodos de solución.

La flexibilidad de CP lo convierte en una herramienta poderosa para resolver problemas NP-Hard como el RCPSP. Permite integrar tanto restricciones complejas como funciones objetivo de optimización, asegurando soluciones de alta calidad. Esto la hace especialmente valiosa en contextos dinámicos, como la programación de actividades de mantenimiento en la industria minera, donde las condiciones y restricciones pueden cambiar constantemente.

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 auxiliares, 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	4000001
ID Tarea	001
Descripción OT	Cambio de suspensión trasera lado izquierdo
Descripción Tarea	Retiro de suspensión trasera
Fecha Inicio Extrema	2024-11-29
Fecha Requerida	2024-12-30
Cuadrilla	BM001
Equipo Auxiliar	Alza Hombres Telescópico
Duración (horas)	6
Impacto	2
Cantidad de Trabajadores	4

Cuadro 1: Ejemplo de Tarea y sus atributos.

4.2. 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 auxiliares** son unitarios, por lo tanto no pueden asignarse a mas de una tareas simultaneamente.

4.3. 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.

4.3.1. 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 $R = \{R_0, R_1, \ldots, R_m\}$, 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 $I(R_j) = \{[a_1, b_1), [a_2, b_2), \ldots\}$, durante los cuales el recurso no está disponible.

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

Parámetros del modelo

- $T = \{T_0, T_1, \ldots, T_n\}$: conjunto de **tareas**.
- $R = \{R_0, R_1, \ldots, R_m\}$: 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.
- $G = \{G_0, G_1, \ldots\}$: conjunto de **grupos de tareas** con restricciones de precedencia.

4.3.2. 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**.

4.3.3. Restricciones

El modelo considera las siguientes 5 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:

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

4.3.4. Objetivo

El objetivo es minimizar una combinación del makespan global 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.

La función objetivo es:

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

Donde:

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

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

4.3.5. 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.

4.4. Detalle de Función Objetivo

Dado que en la práctica puede surgir la posibilidad de que ciertas tareas no puedan ser programadas debido a restricciones como la indisponibilidad de recursos, ventanas de tiempo incompatibles o conflictos en los intervalos prohibidos, es crucial considerar un mecanismo que penalice esta situación, de otro modo cada vez que hayan este tipo de incompatibilidades, el modelo no podrá encontrar una solución factible. Por ello, la función objetivo incluye un componente que penaliza la no programación de tareas. Esto convierte en factibles aquellas soluciones que no logran programar todas las tareas, pero minimiza esta situación tanto como sea posible.

Por otro lado, para equilibrar este aspecto con el objetivo de eficiencia general, también se incluye un segundo componente que penaliza la extensión total del tiempo necesario para completar todas las tareas programadas (makespan). Este enfoque asegura que el modelo busque simultáneamente programar tantas tareas como sea posible y minimizar el tiempo total requerido para su ejecución.

La función objetivo combina estos dos componentes mediante parámetros de ponderación α y β, que permiten ajustar la importancia relativa de cada criterio en la optimización:

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

Donde:

1. **Componente del Makespan ($\alpha \cdot$ makespan)**: Busca reducir la extensión global del horizonte temporal necesario para completar todas las tareas programadas, incentivando la eficiencia en el uso de recursos y tiempo.

2. **Componente de Penalización por Tareas no Programadas ($\beta \cdot \sum_{i=0}^{n} w_i \cdot (1 - x_i)$)**: Penaliza la no programación de tareas, asignando un peso mayor a aquellas con un impacto crítico. Aquí, w_i es el peso asociado a la tarea T_i, definido como:

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

   Este peso aumenta de manera cúbica con la criticidad (bajo valor de Impact_i) de la tarea, priorizando explícitamente las tareas más importantes.

Razonamiento detrás de la Ponderación

El uso de α y β permite al modelo adaptarse a diferentes prioridades operativas. Por ejemplo:

- Si α es significativamente mayor que β, el modelo favorecerá soluciones con makespan reducido, aun si algunas tareas críticas no se programan.
- Si β domina sobre α, el enfoque se orientará a maximizar la programación de tareas críticas, incluso si eso resulta en un makespan mayor.

De esta manera, la función objetivo ofrece un balance flexible que puede ser ajustado según las necesidades específicas del problema, garantizando una solución óptima que considere tanto la completitud de las tareas como la eficiencia global.

5. Flujo de Trabajo de la Solución

El flujo de trabajo de la solución se ha reorganizado en cuatro etapas secuenciales, cada una diseñada para estructurar y procesar la información de programación de manera eficiente. Estas etapas abarcan desde la ingesta y preparación de datos hasta la generación de un cronograma optimizado, maximizando la eficiencia computacional del solver. A continuación, se describen las cuatro 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. **Segmentación en Subconjuntos**: Para mejorar la eficiencia del solver, las tareas se agrupan en subconjuntos relacionados. Este proceso identifica 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.

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

4. **Post-Procesamiento**: Los resultados obtenidos del solver se consolidan y procesan para generar un cronograma unificado. Finalmente, se crean 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.

A continuación se presentan las subsecciones adaptadas a la nueva organización propuesta. Ahora el flujo de trabajo se compone de cuatro etapas principales: Pre-Procesamiento, Segmentación en Subconjuntos, Solver y Post-Procesamiento. Las secciones originales de *Input* y *Output* se han integrado en las etapas inicial y final, respectivamente, para reflejar la nueva estructura modular.

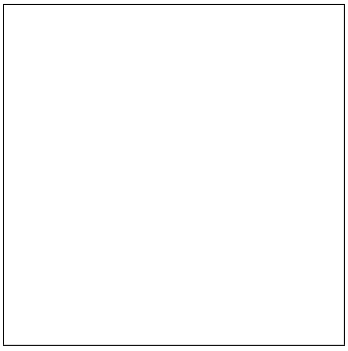

Figura 5: Flujo de Trabajo de la Solución.

5.1. Pre-Procesamiento

La etapa de Pre-Procesamiento constituye el punto de partida del flujo de trabajo, recibiendo y preparando la información procedente de diversas fuentes (como sistemas ERP, Microsoft Project u otros formatos externos). Esta fase garantiza que los datos sobre tareas, personal y equipos estén libres de inconsistencias, limpios y normalizados, facilitando así la labor posterior del solver.

Las operaciones clave realizadas durante el Pre-Procesamiento incluyen:

- **Corrección de discrepancias**: Se revisan y ajustan datos en caso de encontrar inconsistencias. Por ejemplo, si una tarea presenta una fecha límite previa a su fecha de inicio permitida, se corrige automáticamente. Asimismo, se unifican los formatos de fechas, duraciones y demás campos relevantes.
- **Cuantización de ventanas horarias**: Las duraciones se expresan en intervalos de 15 minutos para trabajar con unidades enteras en el solver. De este modo, una tarea de una hora se representa como 4 unidades, y una tarea de 15 minutos como 1 unidad, optimizando la eficiencia en el modelado de restricciones temporales.
- **Estandarización en un formato unificado (JSON)**: Tras la limpieza y normalización, la información se organiza en un archivo JSON. Este archivo consolida datos sobre duración de las tareas, recursos requeridos, ventanas temporales y dependencias, garantizando un formato coherente que servirá de base para las etapas posteriores.

Una vez generado el archivo JSON estandarizado, se crean las estructuras de datos internas necesarias para el solver. A partir de dicho JSON, se construyen diccionarios que representarán las tareas y sus atributos relevantes:

- **Diccionario de tareas**: Contiene la duración, recursos y herramientas requeridas por cada tarea, junto con las variables clave del modelo (inicio, fin, entre otros).
- **Diccionario de ventanas de tiempo**: Especifica para cada tarea sus límites temporales (inicios tempranos, finales máximos), a fin de asegurar el cumplimiento de las restricciones temporales definidas.
- **Diccionario de capacidades de recursos**: Detalla la disponibilidad máxima de cada recurso, garantizando que no se sobrepase su límite durante la programación.
- **Diccionario de agrupaciones de tareas**: Define conjuntos de tareas que comparten recursos o secuencias de precedencia, facilitando la coherencia en la programación de actividades relacionadas.
- **Diccionario de intervalos prohibidos**: Registra los periodos en los que ciertos recursos

no están disponibles, asegurando que no se asignen tareas en dichos intervalos.

En conjunto, el Pre-Procesamiento sienta las bases para las etapas siguientes, generando un entorno de datos consistente y listo para ser segmentado y optimizado.

5.2. Segmentación en Subconjuntos

La Segmentación en Subconjuntos tiene como objetivo reducir la complejidad del problema, dividiendo el conjunto global de tareas en grupos más pequeños y manejables. Este proceso se basa en la identificación de dependencias entre tareas, recursos y cuadrillas, garantizando que aquellas actividades que comparten recursos o presentan relaciones críticas se mantengan dentro de un mismo subconjunto.

El método para llevar a cabo esta segmentación es el siguiente:

a. **Construcción del grafo**: Se modelan tareas, cuadrillas y equipos como nodos de un grafo no dirigido, añadiendo aristas entre estos nodos cuando comparten recursos o dependen unos de otros.

b. **Identificación de componentes conexas**: A través de algoritmos de componentes conexas, se detectan grupos de nodos interrelacionados. Cada componente conexa representa un subconjunto de tareas que deben mantenerse juntas para preservar la coherencia en el uso de recursos y las dependencias.

c. **Formación de subconjuntos**: Cada componente conexa determina un subconjunto de tareas que se tratará de manera independiente en el solver. Esto permite abordar el problema en etapas más pequeñas, reduciendo el número de variables y restricciones manejadas de forma simultánea.

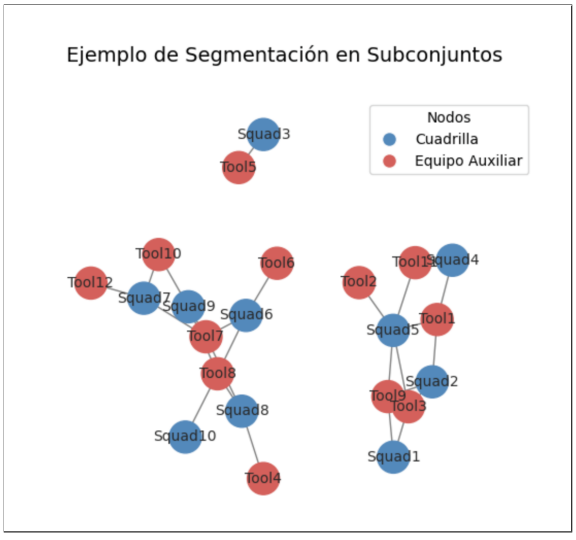

Figura 6: Ejemplo de segmentación a través de componentes conexas.

Gracias a este enfoque, la complejidad computacional disminuye, se mantiene la coherencia en el uso compartido de recursos y es posible resolver cada subconjunto en paralelo, optimizando el uso de recursos computacionales y disminuyendo el tiempo total de cálculo.

5.3. Solver

La tercera etapa del flujo de trabajo consiste en resolver cada uno de los subconjuntos de tareas utilizando el solver de Programación por Restricciones proporcionado por la librería *OR-Tools* de Google. Esta herramienta permite modelar problemas complejos de programación con restricciones, aprovechando las estructuras de datos preparadas en el Pre-Procesamiento.

Para mejorar el desempeño, se aplican estrategias de paralelización con librerías como `joblib`, ejecutando la resolución de múltiples subconjuntos de forma simultánea en diferentes hilos o núcleos de procesamiento.

Las restricciones se modelan de la siguiente manera:

- **Ventanas de tiempo**: Cada tarea se programa respetando sus límites temporales, utilizando variables de intervalo que delimitan su inicio y fin dentro de los intervalos permitidos.
- **Intervalos prohibidos**: Las tareas que requieren ciertos recursos se ajustan para no solaparse con periodos en los que dichos recursos no están disponibles, garantizando la factibilidad de la asignación.
- **Restricciones cumulativas de recursos**: Cada recurso tiene una capacidad máxima. El modelo asegura que el conjunto de tareas asignadas simultáneamente a un mismo recurso no exceda esta capacidad.
- **Precedencia en grupos de tareas**: Cuando existe una secuencia lógica de ejecución entre tareas (por ejemplo, en órdenes de trabajo), se incorporan restricciones de precedencia que aseguran el cumplimiento de dichas relaciones.

Para equilibrar la calidad de la solución y el tiempo de cómputo, se emplean parámetros como `max_time_in_seconds` y `relative_gap_limit`, que limitan el tiempo de búsqueda y la desviación aceptable respecto al óptimo teórico, respectivamente.

5.4. Post-Procesamiento

La fase de Post-Procesamiento consolida los resultados generados por el solver y los prepara en formatos finales útiles para su análisis, integración o visualización en herramientas externas.

Partiendo del archivo JSON producido por el solver (que indica para cada tarea si fue programada y su hora de inicio), se procesan estos datos junto con la información original, produciendo:

- **Archivos XML para Microsoft Project**: Se genera un archivo compatible con Microsoft Project, conteniendo horas de inicio, finalización y duraciones, facilitando la incorporación del cronograma en flujos de trabajo existentes.
- **Archivo XLSX para Excel**: Se produce una carta Gantt en Excel, organizando las tareas en un cronograma visual y detallando inicio, fin, duración, así como los recursos asignados, permitiendo una interpretación rápida y clara.
- **Integraciones con APIs empresariales**: Si la organización cuenta con sistemas ERP u otros entornos de gestión, los resultados pueden integrarse automáticamente mediante APIs, asegurando la continuidad operativa y minimizando el trabajo manual.

El Post-Procesamiento garantiza que los resultados generados por el solver sean comunicados de forma clara y estén listos para su explotación por usuarios finales, herramientas de gestión o análisis posterior. De este modo, el flujo de trabajo se completa con un producto final útil y adaptable a las necesidades específicas de la organización.

6. Resultados y Análisis

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. Descripción del Dataset

El dataset utilizado en este proyecto fue proporcionado por el equipo de programación de la empresa colaboradora. Estos datos corresponden a tareas reales de mantenimiento para un período específico de tiempo y reflejan el contexto operativo y las restricciones enfrentadas por la empresa durante dicho período. El análisis del dataset proporciona una visión integral sobre la escala y la complejidad del problema de programación abordado.

Atributo	**Valor**
Número total de tareas	4.898
Número de órdenes de trabajo (OT)	1.055
Fecha mínima	2023/07/17
Fecha máxima	2023/08/03
Duración promedio de tareas (horas)	1,92
Número de cuadrillas	31
Número de equipos auxiliares	16

Cuadro 2: Resumen de las características principales del dataset.

El conjunto de datos incluye un total de 4.898 tareas, distribuidas en 1,055 órdenes de trabajo (OT). Estas tareas abarcan un rango de tiempo desde el 17 de julio de 2023 hasta el 3 de agosto de 2023, lo que representa un período de planificación de poco más de dos semanas. La duración promedio de las tareas es de aproximadamente 1.92 horas, indicando que la mayoría de las actividades tienen una extensión corta, aunque hay una variación considerable en las duraciones específicas.

En términos de criticidad, las tareas se clasifican en tres niveles de impacto, donde 1 representa las tareas más críticas, 2 aquellas con una prioridad intermedia y 3 las de menor prioridad. La distribución de tareas según su criticidad es la siguiente:

- **Criticidad 1 (Alta):** 666 tareas (13,6 % del total).
- **Criticidad 2 (Media):** 2.403 tareas (49.1 % del total).
- **Criticidad 3 (Baja):** 1.829 tareas (37,3 % del total).

El dataset también incluye información sobre los recursos disponibles para ejecutar las tareas. Estos recursos se dividen en dos categorías principales:

- **Cuadrillas:** El dataset considera un total de 31 cuadrillas, cada una con su respectivo esquema de turnos y capacidades específicas.
- **Equipos Auxiliares:** Hay 16 equipos auxiliares disponibles, los cuales deben compartirse entre las diferentes cuadrillas, lo que añade restricciones significativas al problema de programación.

El rango temporal y las características de las tareas se resumen en la Tabla 2, mientras que la distribución de tareas según su criticidad se ilustra en la Figura 7.

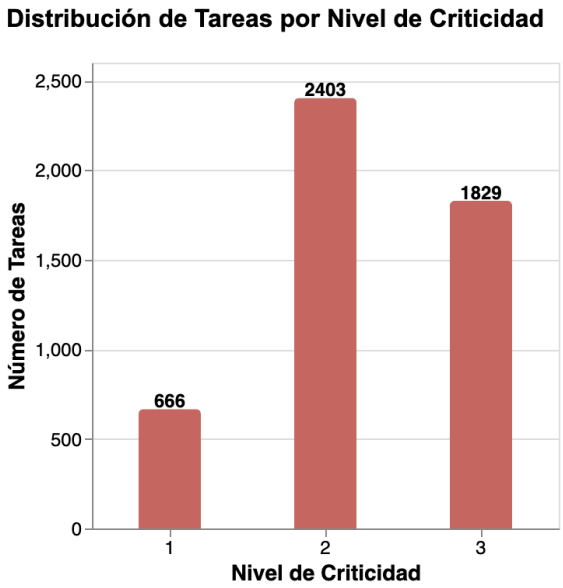

Figura 7: Distribución de tareas según su nivel de criticidad.

Este dataset proporciona una base detallada y realista para el análisis y la implementación del modelo de programación, ya que combina un gran volumen de tareas con restricciones complejas y una diversidad de recursos. Además, la alta proporción de tareas con criticidad 1 y 2 subraya la importancia de optimizar el uso de los recursos para garantizar la ejecución de las tareas más críticas dentro de sus ventanas de tiempo establecidas

6.2. Ajuste de Parámetros para la Función Objetivo

La función objetivo del modelo combina dos componentes clave: el makespan, representado por el parámetro α, y la penalización por tareas no programadas, representada por el parámetro β. Para determinar los valores más adecuados de α y β, se llevaron a cabo múltiples experimentos utilizando diferentes combinaciones de estos parámetros. El objetivo fue evaluar cómo cada configuración afecta el equilibrio entre la minimización del makespan y la cantidad de tareas programadas, especialmente aquellas con criticidad alta (impacto 1).

En estos experimentos, los resultados fueron analizados en función de métricas clave, incluyendo el número total de tareas programadas, la cantidad de tareas no programadas, el makespan obtenido, el tiempo de solución requerido y la cantidad de tareas no programadas según su nivel de impacto. Los resultados se presentan en dos tablas: la Tabla 3 resume las métricas generales, mientras que la Tabla 4 detalla la distribución de tareas no programadas según su impacto.

La configuración a, donde $\alpha = 1$ y $\beta = 0$, prioriza exclusivamente la minimización del makespan. Sin embargo, este enfoque no logra programar ninguna tarea, lo que resulta inaceptable, ya que no se cumple el objetivo principal del modelo. Por otro lado, la configuración b ($\alpha = 0$, $\beta = 1$) busca maximizar la cantidad de tareas programadas, logrando 4854 tareas en total. Aunque esta configuración presenta un makespan elevado (1072 unidades de tiempo), consigue programar todas las tareas de impacto 1, lo cual es un resultado favorable para las tareas más críticas.

La configuración c ($\alpha = 1$, $\beta = 1$) equilibra ambos objetivos, pero a costa de una mayor cantidad

Conf.	α	β	Tareas Programadas	Tareas no Programadas	Tiempo de Solución (s)	Makespan
(a)	1	0	0	4.898	3,43	0
(b)	0	1	4.854	44	4,35	1.072
(c)	1	1	4.658	240	8,42	492,75
(d)	1	2	4.831	67	9,95	516,25

Cuadro 3: Resultados generales obtenidos con diferentes configuraciones de α y β.

de tareas no programadas (240). Si bien su makespan es menor que el de b, su penalización por dejar tareas críticas fuera del cronograma lo hace menos atractivo.

Finalmente, la configuración d ($\alpha = 1$, $\beta = 2$) logra un balance ideal entre los objetivos de programación y makespan. Con 4.831 tareas programadas, esta configuración deja fuera solo 67 tareas, manteniendo el makespan en un nivel razonable de 516,25 unidades de tiempo. Además, se asegura que todas las tareas de impacto 1 son programadas, mientras que las tareas de impacto 2 y 3 no programadas son reducidas al mínimo posible.

Conf.	Impacto 1	Impacto 2	Impacto 3
(a)	666	2.403	1.829
(b)	0	8	36
(c)	21	59	160
(d)	0	12	55

Cuadro 4: Distribución de tareas no programadas según su impacto.

Se seleccionó la configuración d debido a su capacidad para satisfacer las prioridades del modelo. Aunque el tiempo de solución es el más alto entre las configuraciones evaluadas (9,95 segundos), este es un compromiso aceptable dado que garantiza un equilibrio óptimo entre la minimización del makespan y la programación de tareas críticas. Sin embargo, este tiempo de solución podría representar un desafío operativo en escenarios más grandes. Para abordar este problema, se propone utilizar la segmentación en subconjuntos, una técnica que divide el problema en partes más manejables, reduciendo así el tiempo de cálculo necesario para resolver cada subconjunto de tareas.

Para ilustrar el efecto de los diferentes valores de α y β, se incluye la Figura 8, que muestra el trade-off entre tareas programadas y el makespan en función de las configuraciones evaluadas. Este gráfico resalta cómo la configuración d logra un equilibrio favorable en comparación con las demás opciones.

En conclusión, la selección de $\alpha = 1$ y $\beta = 2$ se justifica por su desempeño superior en las métricas clave del modelo, alineándose con las prioridades operativas y estratégicas de la empresa. La implementación de la segmentación en subconjuntos permitirá optimizar aún más el tiempo de solución, asegurando que el modelo pueda escalar eficientemente a problemas más complejos.

6.3. Resultados de la Segmentación en Subconjuntos

La segmentación en subconjuntos permitió dividir el conjunto total de tareas en 14 subconjuntos más manejables, con el objetivo de reducir el tiempo de procesamiento del modelo. En la Tabla 5

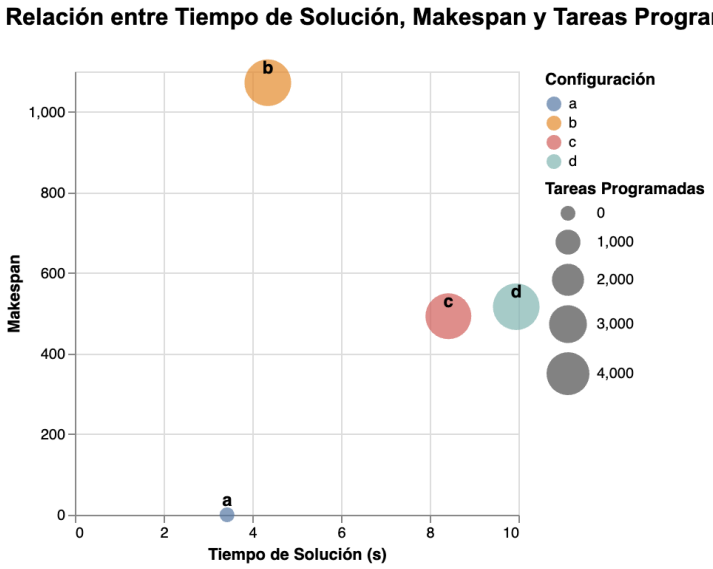

Figura 8: Análisis de las configuraciones de α y β.

se presentan los resultados obtenidos, incluyendo el número de tareas en cada subconjunto, su porcentaje relativo con respecto al total (redondeado sin decimales) y el tiempo de solución requerido para resolver cada uno. Este análisis se llevó a cabo en un servidor virtual con las siguientes especificaciones:

- **vCPU/s:** 4 vCPUs.
- **RAM:** 8192 MB.
- **Almacenamiento:** 75 GB NVMe.
- **Sistema Operativo:** Ubuntu 24.10 x64.

Subconjunto	Número de Tareas	% del Total	Tiempo de Solución (s)
S1	2.598	53 %	4,34
S2	674	14 %	0,65
S3	319	7 %	0,43
S4	254	5 %	0,12
S5	198	4 %	0,01
S6	136	3 %	0,01
S7	122	2 %	0,00
S8	110	2 %	0,01
S9	98	2 %	0,00
S10	87	2 %	0,00
S11	84	2 %	0,01
S12	78	2 %	0,00
S13	72	1 %	0,00
S14	68	1 %	0,00

Cuadro 5: Resultados de la segmentación en subconjuntos.

La segmentación produjo un subconjunto principal, $S1$, que contiene 2.598 tareas, representando

el 53 % del total. Este subconjunto, al abarcar más de la mitad de las tareas, requirió el mayor tiempo de solución (4,34 segundos). Los otros 13 subconjuntos contienen entre 68 y 674 tareas cada uno, con porcentajes significativamente menores del total. El segundo subconjunto más grande, $S2$, con 674 tareas (14 %), presentó un tiempo de solución de 0,65 segundos, destacando la eficiencia del enfoque modular al abordar subconjuntos de menor tamaño. Los subconjuntos más pequeños, desde $S5$ hasta $S14$, contienen menos del 5 % de las tareas totales cada uno y se resolvieron en tiempos cercanos a cero segundos.

El procesamiento de los subconjuntos se llevó a cabo de manera paralela, aprovechando las capacidades del servidor. Como resultado, el tiempo total de solución se igualó al máximo tiempo de solución de todos los subconjuntos, es decir, 4,34 segundos. Este enfoque demostró la ventaja de dividir el problema en partes independientes y resolverlas simultáneamente, reduciendo considerablemente el tiempo total necesario para obtener los resultados.

El análisis de los resultados muestra que más del 50 % de las tareas están concentradas en un único subconjunto ($S1$), mientras que los demás subconjuntos están distribuidos de manera desigual, con tamaños decrecientes. Esta concentración sugiere que ciertas tareas comparten fuertes conexiones o restricciones comunes, explicando la formación de un subconjunto dominante. Aunque $S1$ tiene un impacto significativo en el desempeño general, el tiempo total de cálculo resulta eficiente y adecuado gracias al enfoque de paralelización.

En conclusión, la segmentación en subconjuntos demostró ser una estrategia efectiva para mejorar el tiempo de solución del modelo, permitiendo resolver problemas complejos en tiempos razonables. El tiempo total de solución se igualó al del subconjunto más grande (4,34 segundos), gracias al procesamiento paralelo.

7. Conclusiones y Pasos a Seguir

Los resultados obtenidos muestran que el modelo desarrollado cumple con las expectativas de la empresa colaboradora, proporcionando una solución eficiente y satisfactoria para la programación automatizada de actividades de mantenimiento. Se logra un cronograma que cumple con las condiciones y restricciones definidas, en tiempos de procesamiento acotados y manejables.

El modelo logra un equilibrio entre la minimización del makespan y la programación de tareas críticas. La segmentación en subconjuntos se revela como una estrategia efectiva para reducir el tiempo de cálculo, permitiendo resolver problemas complejos en tiempos razonables y escalables.

Para nuestro dataset de 4.898 tareas, el modelo logra programar el 98,6 % de las tareas, dejando solo 67 tareas no programadas. Además, se garantiza la programación de todas las tareas de impacto 1, lo que refleja la priorización de las actividades más críticas. El makespan obtenido es de 516,25 horas, lo que indica una distribución eficiente de las tareas en el tiempo disponible.

Con base en una encuesta realizada al personal de programación de la empresa colaboradora, se evidenció que el proceso actual de programación de actividades podría beneficiarse significativamente de herramientas que automatizan partes clave del flujo de trabajo. En particular, una herramienta que automatice la creación del borrador inicial del cronograma y permita la incorporación fluida de cambios de último minuto, ajustes derivados de conflictos de recursos y validaciones con otras áreas, sería altamente valorada.

Se estima que una herramienta con estas características podría **reducir hasta en un 50 % el tiempo que el equipo de programación dedica a estas tareas**. Esto optimizaría la asignación

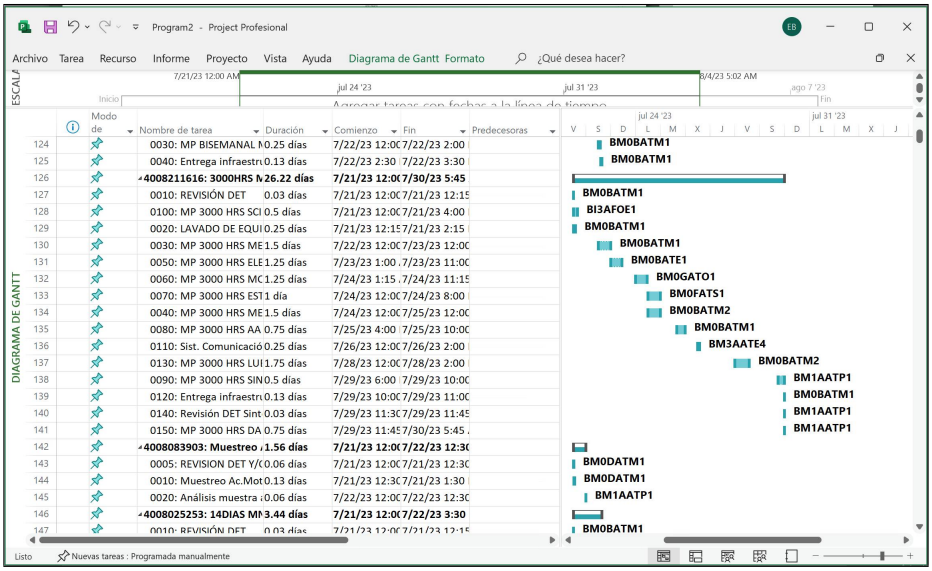

Figura 9: Captura de Cronograma Resultante en MS Project

de su tiempo, permitiéndoles enfocarse en actividades estratégicas.

Para garantizar el éxito de esta herramienta, será fundamental su integración con las fuentes de información de otras áreas de la empresa. En particular:

- Conexión con el área de planificación para acceder a información clave sobre las tareas: requerimientos de recursos, ventanas de tiempo y restricciones operativas.
- Integración con el sistema de bodega para garantizar la disponibilidad de materiales y repuestos.
- Conexión con los datos de disponibilidad de equipos y cuadrillas.

Además, será necesario revisar y ajustar los procedimientos internos de estas áreas para garantizar que la información sea precisa y confiable. Si se logra implementar una herramienta que integre estas funcionalidades y asegure la calidad de los datos, se estima que **los errores en las labores de mantenimiento derivados de una programación deficiente podrían reducirse en un 75 %**.

7.1. Pasos a Seguir

Es importante mencionar que el desarrollo del modelo y su implementación en un entorno operativo real requiere de una serie de pasos adicionales. A continuación, se detallan las recomendaciones para la implementación exitosa de la solución propuesta.

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.

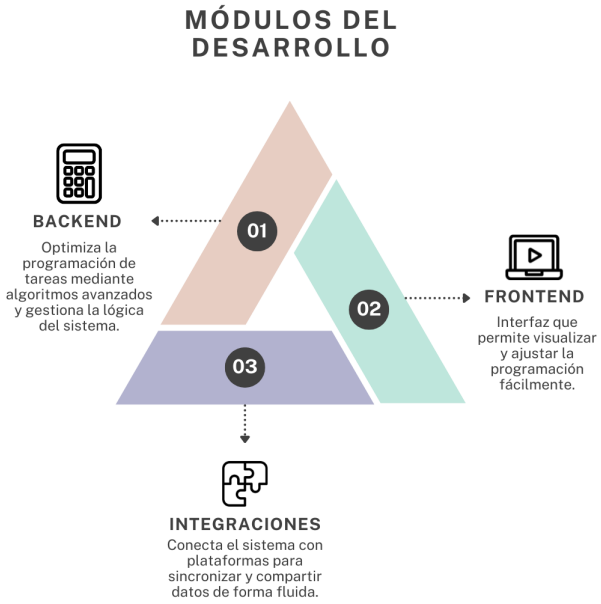

Figura 10: Módulos de Desarrollo de Solución

7.1.1. 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.

El backend se implementará en Python, utilizando **FastAPI** como framework para exponer los endpoints REST necesarios. Para el motor de optimización, se empleará **OR-Tools** para resolver el problema mediante Programación por Restricciones (CP-SAT) u otros métodos heurísticos evaluados previamente. El almacenamiento de datos se manejará con **PostgreSQL**, alojado en un entorno cloud escalable.

El servicio será desplegado en un clúster de **Kubernetes** en **Azure Kubernetes Service (AKS)**, aprovechando su capacidad de escalabilidad automática y alta disponibilidad. El contenedor del backend se construirá utilizando **Docker**, con un pipeline de integración continua configurado en **Azure DevOps** para automatizar pruebas y despliegues.

7.1.2. Frontend

El frontend funcionará como la interfaz principal para los programadores de la empresa, quienes son los usuarios de esta solución. El diseño estará enfocado en la facilidad de uso, permitiendo que los programadores carguen los datos de las órdenes de trabajo de manera sencilla. La carga de datos podrá realizarse de manera manual o mediante integración con el sistema ERP de la empresa.

El frontend será desarrollado en **React.js**, utilizando librerías como **React-Gantt-Chart** o **D3.js** para generar gráficos interactivos de Gantt que permitan la visualización y ajuste manual de los cronogramas generados. Además, el frontend permitirá a los usuarios exportar el cronograma a formatos compatibles con *Microsoft Project* y Excel.

Este módulo será desplegado como una aplicación web estática en **Azure App Service**, conectándose con el backend a través de los endpoints expuestos por FastAPI. La seguridad de la comunicación se garantizará mediante certificados SSL proporcionados por **Azure Key Vault**.

7.1.3. Integración con Plataformas de la Empresa

El tercer módulo se encargará de la integración con las plataformas existentes de la empresa. Este módulo se desarrollará en estrecha colaboración con el equipo de TI de la organización para garantizar la compatibilidad de los formatos de entrada y salida con los sistemas actuales, incluyendo el ERP.

La integración será bidireccional, permitiendo:

- La importación automática de datos sobre órdenes de trabajo, disponibilidad de cuadrillas y recursos desde el ERP.
- La exportación del cronograma final a los sistemas existentes para su validación y distribución.

Se configurarán conexiones seguras mediante protocolos estándar como *OAuth 2.0* y *HTTPS*. Además, el uso de **Azure API Management** facilitará el monitoreo y la administración de las APIs involucradas.

7.2. Arquitectura de la Solución

La arquitectura propuesta combina herramientas modernas de desarrollo y gestión de infraestructura para garantizar escalabilidad, disponibilidad y seguridad. En la Figura 11 se detalla la arquitectura general de la solución, implementada en **Microsoft Azure**.

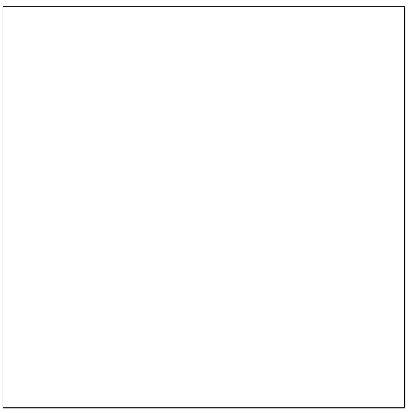

Figura 11: Arquitectura Propuesta de la Solución

- **Backend:** Contenedorizado con **Docker** y desplegado en un clúster de **Azure Kubernetes Service (AKS)**, con una base de datos **PostgreSQL** alojada en **Azure Database for PostgreSQL**.

- **Frontend:** Aplicación web estática desplegada en **Azure App Service**, integrada con el backend mediante endpoints REST.
- **Integración:** APIs gestionadas mediante **Azure API Management**, conectadas con sistemas corporativos y almacenamientos adicionales en **Azure Blob Storage**.
- **Seguridad:** Certificados SSL y claves gestionadas en **Azure Key Vault**.
- **Monitoreo:** Métricas y logs gestionados mediante **Azure Monitor**.

7.3. Próximos Pasos

- Realizar un plan detallado de desarrollo basado en la arquitectura propuesta, incluyendo estimaciones de tiempo para cada módulo.
- Implementar pruebas unitarias y de integración para garantizar la calidad del software en todas las etapas de desarrollo.
- Capacitar al equipo de la empresa en el uso de la solución, asegurando una transición fluida hacia el nuevo sistema.
- Monitorear el desempeño del sistema una vez implementado, recolectando retroalimentación para futuras mejoras.

7.4. Conclusión Final

El modelo desarrollado representa un avance significativo en la automatización de la programación de actividades de mantenimiento, con el potencial de optimizar el flujo de trabajo, reducir errores y mejorar la eficiencia operativa. La implementación de los pasos a seguir garantizará que esta solución pase de ser un prototipo a una herramienta integrada y funcional que genere un impacto tangible en las operaciones de la empresa colaboradora.