Development of an Extensible Python Package for Modeling and Solving Production Scheduling Problems

Araceli Sarina[1] and Pablo A. Marchetti[1,2]

[1] Dpto. Ingeniería en Sistemas de Información, Universidad Tecnológica Nacional - Facultad Regional Santa Fe, Lavaisse 610, 3000 Santa Fe, Argentina
{asarina,pmarchetti}@frsf.utn.edu.ar
[2] Instituto de Desarrollo Tecnológico para la Industria Química (INTEC), UNL-CONICET, Güemes 3450, 3000 Santa Fe, Argentina

Abstract. This work describes the design and initial implementation of a Python package aimed at facilitating the construction and solution of optimization models for production scheduling problems. The motivation arises from the diversity of scheduling problems and the effort required to implement specific mathematical formulations for each case. The tool is built on Pyomo and uses an object-oriented design based on reusable components, separating the definition of the problem, operational features, objectives, formulations, data loading, and result presentation. This organization makes it possible to build instances, incorporate additional capabilities, and select compatible models within a common workflow. The package architecture, its main abstract and concrete classes, and the implemented extension mechanisms are described. As preliminary results, some available components for single-stage and multi-stage problems are presented, together with a usage example. Finally, future lines of work related to transformations between representations, problem decomposition, and integration with algorithmic solution strategies are discussed.

Keywords: production scheduling, mathematical optimization, software package, Pyomo

Desarrollo de un paquete extensible en Python para modelar y resolver problemas de programación de la producción

Resumen. Este trabajo describe el diseño y la implementación inicial de un paquete en Python cuya meta es facilitar la construcción y resolución de modelos de optimización para problemas de programación de la producción. La motivación surge de la diversidad de problemas de *scheduling*

y del esfuerzo requerido para implementar formulaciones matemáticas específicas para cada caso. La herramienta se apoya en Pyomo y utiliza un diseño orientado a objetos basado en componentes reutilizables, separando la definición del problema, las características operativas, los objetivos, las formulaciones, la carga de datos y la presentación de resultados. Esta organización permite construir instancias, incorporar capacidades adicionales y seleccionar modelos compatibles dentro de un flujo común de trabajo. Se describe la arquitectura del paquete, sus principales clases abstractas y concretas, y los mecanismos de extensión implementados. Como resultados preliminares, se presentan algunos componentes disponibles en problemas monoetapa y multietapa, junto con un ejemplo de uso. Finalmente, se discuten líneas futuras vinculadas con transformaciones entre representaciones, descomposición de problemas e integración con estrategias algorítmicas de resolución.

Palabras clave: programación de la producción, optimización matemática, paquete de software, Pyomo

1 Introducción

La programación de operaciones o *scheduling* es un proceso de toma de decisiones que cumple un rol central en la eficiencia de los procesos industriales y en el uso sustentable de los recursos productivos. En contextos cada vez más competitivos, la rentabilidad de una planta depende en gran medida de coordinar adecuadamente equipos, materiales y tiempos de producción, de modo de responder a la demanda de los clientes dentro de los plazos comprometidos y con el menor uso posible de recursos (Pinedo, 2008).

La definición de un problema de *scheduling* requiere caracterizar con precisión tanto la estructura del proceso como las decisiones operativas que deben representarse. Entre otros aspectos, pueden intervenir la receta o secuencia de producción, la topología de la planta, la selección y conectividad de los equipos, las políticas de almacenamiento intermedio, la transferencia de materiales, los tamaños y tiempos de procesamiento de los lotes, los tiempos de preparación o transición, la disponibilidad de recursos auxiliares, los patrones de demanda y los costos asociados a las tareas (Maravelias, 2021; Méndez et al., 2006).

Debido a esta amplia diversidad de configuraciones, en la literatura se han propuesto numerosas formulaciones y estrategias de solución para problemas de *scheduling* en entornos industriales (Méndez et al., 2006). En particular, los modelos de programación matemática mixta-entera lineal (MILP) y no lineal (MINLP) constituyen enfoques exactos capaces de representar restricciones operativas complejas y distintos criterios de desempeño. Para problemas de la industria de procesos, Maravelias (2021) sistematiza modelos y métodos de programación mixta-entera aplicables a distintos ambientes de producción, reforzando la relevancia de estas formulaciones como base para el desarrollo de herramientas computacionales. No obstante, su adopción práctica suele verse limitada por la necesidad de conocimiento especializado, los costos de desarrollo y el esfuerzo

requerido para adaptar cada formulación a un caso industrial específico (Harjunkoski et al., 2014).

El desarrollo de software para optimización puede analizarse en distintas capas, que cumplen roles complementarios. En una capa se ubican las bibliotecas y lenguajes generales de modelado algebraico, entre ellos Pyomo, AMPL, GAMS o AIMMS, que permiten expresar modelos de optimización de manera flexible e independiente de un dominio particular (AIMMS, s.f.; Bynum et al., 2021; Fourer et al., 2003; GAMS Development Corporation, s.f.). En otra capa se encuentran motores de resolución como Gurobi y CPLEX, y bibliotecas de optimización como Google OR-Tools, que proveen algoritmos y estructuras especializadas para resolver distintas clases de problemas (CPLEX, s.f.; Google OR-Tools, 2024; Gurobi Optimization, LLC, 2026). Estas herramientas resultan fundamentales para construir soluciones computacionales, aunque por sí solas no necesariamente organizan formulaciones reutilizables en un dominio, ni los flujos de datos y resultados asociados.

También existen herramientas más cercanas al dominio de *scheduling* y planificación. PyJobShop proporciona una interfaz en Python para formular problemas de *scheduling* mediante programación con restricciones, utilizando objetos del dominio como tareas, recursos, modos de procesamiento y precedencias (Lan & Berkhout, 2025). El paquete pyschedule ofrece una interfaz más acotada para problemas de asignación de tareas a recursos en horizontes discretos (pyschedule, s.f.). En el ámbito de la optimización basada en restricciones, Timefold Solver –continuación de OptaPlanner– aborda problemas como asignación de recursos, ruteo de vehículos, planificación de turnos (rostering) y scheduling (OptaPlanner, s.f.; Timefold, s.f.). En el ecosistema Python, OptaPy surgió como una interfaz para utilizar capacidades de OptaPlanner (OptaPy, s.f.). En conjunto, estas herramientas acercan el modelado al lenguaje del dominio, aunque suelen partir de una representación unificada del problema y de un motor de solución específico.

Este trabajo presenta resultados iniciales en el desarrollo de un paquete de software en Python para modelar y resolver problemas de *scheduling* mediante componentes flexibles y reutilizables. El enfoque utilizado en su construcción persigue un objetivo complementario. Mientras otras herramientas de programación de operaciones representan el problema mediante objetos del dominio, como tareas, recursos, máquinas, empleados, intervalos o asignaciones, en esta propuesta se busca mantener una cercanía explícita con las estructuras habituales de la programación matemática: conjuntos, parámetros, variables, restricciones y objetivos. Esta decisión es relevante cuando se pretende implementar, comparar o extender formulaciones MILP y MINLP provenientes de la literatura, ya que una misma clase de instancias puede admitir distintas representaciones matemáticas compatibles. El interés no es solamente describir una instancia para resolverla, sino alojar distintas formulaciones posibles dentro de una infraestructura común.

En este contexto, el paquete presentado se posiciona como una capa específica de dominio construida sobre Pyomo. El presente trabajo extiende una contribución preliminar en la que se presentó el diseño inicial y un ejemplo de

uso (Sarina, 2025). A diferencia del uso directo de Pyomo construyendo modelos ad hoc, la propuesta busca reutilizar estructuras comunes para la definición de problemas, características operativas, objetivos, formulaciones, componentes de modelo, datos de entrada y análisis de resultados. A diferencia de frameworks de *scheduling* basados principalmente en programación con restricciones o heurísticas, el paquete prioriza la convivencia de múltiples formulaciones matemáticas alternativas dentro de un mismo flujo de trabajo.

La herramienta está orientada a la investigación, desarrollo y experimentación con distintas representaciones y múltiples formulaciones alternativas del problema, pudiendo también utilizarse como componente de base para un sistema de planificación de planta, si se integra con una interfaz de usuario adecuada para operadores o planificadores de producción. En este artículo se describe su arquitectura actual, los componentes genéricos y concretos implementados, y ejemplos iniciales de uso.

2 Objetivo y alcances del paquete

El objetivo del paquete es ofrecer una infraestructura extensible para modelar y resolver familias de problemas de programación de la producción mediante formulaciones matemáticas. La herramienta está desarrollada en Python y se basa en el paquete Pyomo para el modelado matemático y resolución de los problemas. El objetivo abordado no es la resolución de una instancia particular de *scheduling*, sino la construcción de una capa de software capaz de organizar datos, problemas, características operativas, objetivos, formulaciones y resultados dentro de un flujo de trabajo común.

Desde el punto de vista del dominio, el paquete busca representar las distintas instancias de los problemas, por ejemplo compuestas por lotes o tareas, unidades de procesamiento, etapas, compatibilidades, tiempos de procesamiento y horizonte temporal. Sobre esta base pueden incorporarse características como fechas de entrega, tiempos de preparación, tiempos de transición dependientes de la secuencia, restricciones de recursos recursos renovables o no renovables (adicionales al equipamiento), tanques de almacenamiento, etc. Además, distintos criterios de optimización o funciones objetivo se acoplan como componentes adicionales. Estos elementos modifican la formulación matemática requerida, por lo que resulta necesario tratarlos como componentes combinables y reutilizables.

El flujo previsto por la herramienta comprende la definición del problema, la incorporación de características y objetivos, la carga de datos, la validación de la instancia, la selección de un modelo compatible, la construcción del modelo Pyomo, la resolución mediante un solver externo y la presentación de resultados estructurados. Esta organización busca que las formulaciones y sus componentes puedan reutilizarse en distintos casos, evitando reescribir para cada variante las mismas tareas de carga, validación, construcción y análisis.

En la etapa de desarrollo presentada en este artículo, el énfasis no está puesto en cubrir exhaustivamente todas las variantes de problemas y formulaciones disponibles, sino en establecer una base extensible que permita incorporar nue-

vas formulaciones sin reescribir el flujo completo de datos, validación, construcción del modelo y análisis de resultados. Para validar esta base se implementan problemas y modelos concretos que permiten comprobar la integración de los componentes centrales y preparar la incorporación progresiva de nuevas familias de problemas y formulaciones.

3 Diseño y desarrollo

3.1 Herramientas utilizadas

El desarrollo del paquete se basa en el modelado y programación orientada a objetos (POO), con el propósito de separar responsabilidades y favorecer la reutilización de componentes. Python fue seleccionado como lenguaje de implementación por su integración con herramientas científicas, su expresividad y su amplia adopción en aplicaciones de ingeniería y optimización.

Para representar las formulaciones matemáticas se utiliza Pyomo, un paquete de Python que permite construir modelos de optimización de forma algebraica y vincularlos con distintos solvers. Esta elección resulta adecuada porque permite expresar modelos MILP y MINLP manteniendo una separación clara entre los datos del problema, la formulación matemática y el procedimiento de resolución. Pyomo ofrece numerosos complementos, incluyendo extensiones para trabajar con representaciones más abstractas como la programación disyuntiva generalizada (GDP) y sus reformulaciones Big-M o Hull. Sin embargo, esas alternativas quedan fuera del alcance en esta etapa y podrían evaluarse en desarrollos posteriores.

Además de Pyomo, se emplean paquetes adicionales para la lectura, transformación de datos y visualización de resultados (Pandas, Openpyxl, Matplotlib, entre otros). La resolución de los modelos se realiza mediante solvers compatibles con Pyomo; en los casos de estudio planteados se utiliza Gurobi, debido a su desempeño en problemas de programación mixta-entera lineal y a su disponibilidad para investigación académica.

3.2 Arquitectura y componentes principales

El paquete se organiza bajo el nombre `scheduling` y adopta una arquitectura modular orientada a sostener el flujo de trabajo descripto en la sección anterior. En la versión actual, los subpaquetes principales son `problems`, `features`, `models`, `components`, `io` y `visualization`. La Figura 1 resume esta organización y permite ubicar, antes de describir las clases, qué función cumple cada parte dentro del proceso de definición de problemas, carga de datos, construcción de modelos, resolución y análisis de resultados.

Desde esta perspectiva, el paquete puede entenderse como una cadena de responsabilidades. El subpaquete `problems` representa las instancias de problemas de programación de operaciones; `features` reúne características y objetivos que pueden incorporarse a dichas instancias; `models` contiene las formulaciones matemáticas disponibles; `components` aporta bloques reutilizables para construir

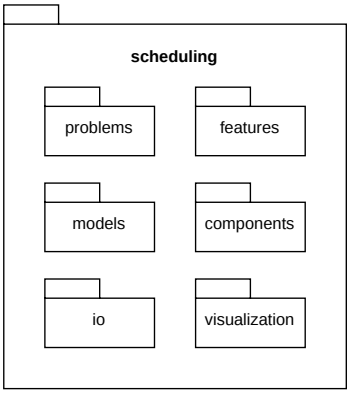

Figura 1. Organización general del paquete `scheduling`.

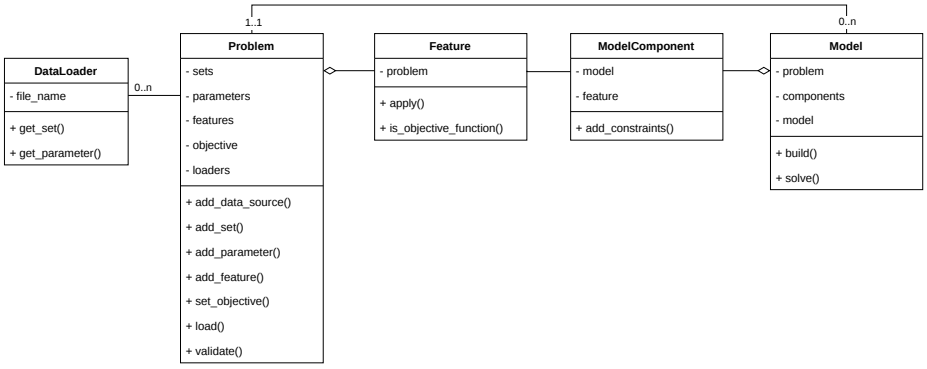

Figura 2. Clases abstractas principales del paquete `scheduling`.

esas formulaciones; `io` desacopla la lectura de datos del resto del paquete; y `visualization` concentra herramientas para interpretar la solución obtenida. Esta separación busca que el usuario pueda combinar problemas, datos, características y formulaciones sin reescribir el modelo completo para cada caso.

El segundo nivel del diseño está compuesto por las clases abstractas. Estas clases definen los contratos principales del sistema, que permiten separar la información del problema, la carga de datos, las características opcionales, la formulación matemática y los componentes asociados a las características. La Figura 2 resume estas relaciones. En este nivel la función más importante es establecer responsabilidades comunes para que luego puedan agregarse clases concretas sin modificar la estructura general del paquete.

Problem representa la instancia de trabajo sobre la cual se construirá una formulación. Su responsabilidad es reunir los conjuntos, parámetros, fuentes de datos, características y objetivo que describen el problema a resolver. Esta clase actúa como punto de entrada conceptual para el usuario: primero se define el

problema y luego se seleccionan los datos, las características y el modelo matemático apropiado.

Feature representa una característica operativa que el usuario declara sobre una instancia de `Problem`, como fechas de entrega, tiempos de preparación o tiempos de cambio dependientes de la secuencia. Además de identificar una capacidad del problema, cada característica define qué información necesita para poder aplicarse: durante la carga puede solicitar los parámetros requeridos y, durante la validación, verificar que esos datos estén disponibles y sean consistentes con los conjuntos de la instancia. En forma complementaria, **Objective** representa el criterio de optimización seleccionado y sigue el mismo patrón declarativo, manteniendo separado el objetivo de las demás características operativas.

Model representa la formulación matemática que se aplicará a una instancia de `Problem`. Su rol es transformar la información validada del problema en un modelo Pyomo que pueda ser resuelto por un solver. Además de construir las variables y restricciones propias de la formulación, el modelo instancia los componentes asociados con las características y el objetivo declarados. Esta asociación se realiza mediante una fábrica de componentes que traduce cada `Feature` u `Objective` registrado en uno o más `ModelComponent` compatibles con la formulación correspondiente.

ModelComponent encapsula la contribución matemática asociada con una característica u objetivo. Según el caso, un componente puede agregar restricciones, variables auxiliares u objetivos al modelo Pyomo, o proveer expresiones y funciones auxiliares utilizadas por otras restricciones de la formulación; por ejemplo, funciones para recuperar tiempos de preparación o de cambio según la unidad, la tarea o la secuencia. De esta manera, la lógica específica queda concentrada en bloques reutilizables y el usuario final no necesita manipular directamente las expresiones Pyomo.

Desde la perspectiva del usuario, el flujo resultante es declarativo: se crea el problema, se agregan características y objetivo al mismo, se cargan y validan los datos, y se selecciona un modelo que, al ser construido, instancia automáticamente los componentes correspondientes. Cuando una restricción particular aún no está contemplada por el paquete, el mecanismo de extensión previsto consiste en definir una nueva `Feature` u `Objective`, implementar el `ModelComponent` asociado y registrarlo para las formulaciones compatibles. Así, las restricciones particulares se incorporan como extensiones controladas de la arquitectura, no como instrucciones libres escritas por el usuario dentro del script de resolución.

DataLoader define la interfaz para obtener datos desde fuentes externas. Su incorporación separa el formato de entrada de la lógica de modelado, de modo que el paquete pueda leer conjuntos y parámetros desde planillas u otras fuentes sin modificar las clases que representan problemas o modelos. Una implementación concreta orientada a planillas Excel es **ExcelDataLoader**. En la versión actual, la lectura se basa en rangos nombrados, lo cual permite asociar regiones de la planilla con conjuntos y parámetros del modelo.

Sobre estas abstracciones se construyen las clases concretas del paquete. La jerarquía de problemas actual, resumida en la Figura 3, parte de clases genera-

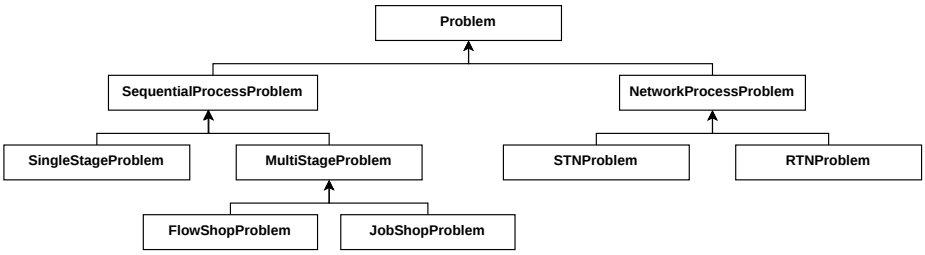

Figura 3. Jerarquía actual de problemas del paquete `scheduling`.

les y avanza hacia familias más específicas, de modo que la lógica común pueda compartirse entre variantes relacionadas. En particular, **SequentialProcess Problem** reúne elementos comunes de procesos secuenciales, como lotes, unidades y tiempos de procesamiento, y sirve de base para especializaciones monoetapa y multietapa.

La subclase **SingleStageProblem** representa procesos en los que cada lote requiere una única operación principal y puede asignarse a una de varias unidades disponibles. Por su parte, **MultiStageProblem** extiende el concepto previo a procesos compuestos por varias etapas. En este caso, cada lote debe atravesar una secuencia de operaciones y cada etapa cuenta con un conjunto de unidades compatibles. Esta clase permite representar cadenas de producción donde la coordinación entre etapas es parte central del problema. En particular, **Flow ShopProblem** y **JobShopProblem** son especializaciones que corresponden a familias clásicas de problemas de *scheduling* (Pinedo, 2008).

STNProblem y **RTNProblem** representan un segundo grupo de problemas, orientado a representaciones de estructuras de red empleadas en la programación de procesos. A diferencia de las clases secuenciales, estas representaciones describen relaciones entre tareas, estados o recursos, y se incorporan como una base conceptual para extender el paquete hacia procesos donde la representación secuencial no resulta suficiente.

De manera complementaria a la jerarquía de problemas, el subpaquete `models` organiza las formulaciones matemáticas disponibles y las bases previstas para nuevas formulaciones. Esta separación es importante porque el tipo de problema describe la instancia y sus datos, mientras que el modelo define cómo se representarán las decisiones y restricciones en términos matemáticos. En la arquitectura actual, representada en la Figura 4, se distinguen familias habituales en la literatura de programación de operaciones, incluyendo formulaciones de tiempo continuo, formulaciones de tiempo discreto, modelos basados en precedencia y modelos por ranuras de tiempo (Maravelias, 2021; Méndez et al., 2006).

ContinuousTimeModel agrupa formulaciones en las que los instantes de inicio y finalización de las tareas se representan mediante variables continuas. Este enfoque resulta adecuado cuando se busca mayor precisión temporal, evitando que los eventos estén restringidos a una grilla fija de periodos. Entre los modelos de tiempo continuo, **PrecedenceBasedModel** representa formula-

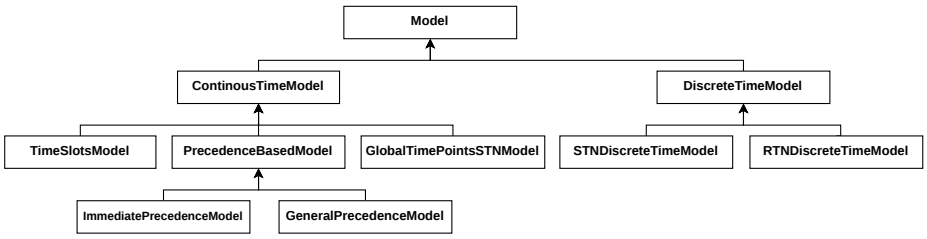

Figura 4. Jerarquía actual de modelos del paquete `scheduling`.

ciones que describen el orden relativo entre operaciones mediante relaciones de precedencia. Por su parte, los modelos con ranuras de tiempo, como **TimeSlotsModel**, incorporan posiciones o ranuras ordenadas para representar alternativas de asignación y secuenciación.

DiscreteTimeModel agrupa la base prevista para formulaciones en las que el horizonte temporal se divide en periodos de duración fija. Aunque esta discretización puede aumentar el tamaño del modelo, también simplifica la representación de ciertos tipos de restricciones y resulta natural en contextos donde las decisiones se toman en intervalos regulares. En la versión actual, esta clase ordena la extensión futura hacia modelos discretos concretos.

Desde el punto de vista del usuario, el flujo de trabajo conecta estas capas en una secuencia relativamente directa: se define una instancia de problema, se asocian datos de entrada, se incorporan características y un objetivo, se selecciona una formulación compatible, se construye el modelo Pyomo y se invoca el solver. La salida se organiza en objetos de resultado que resumen las tareas programadas, el valor de la función objetivo, el estado de resolución y otra información útil para el análisis posterior.

En la gestión de datos, el uso de rangos nombrados en Excel permite desacoplar la estructura física de la planilla de los nombres utilizados por el paquete. Además, el cargador contempla variantes frecuentes de nomenclatura y representaciones alternativas para algunos subconjuntos, lo que facilita adaptar planillas existentes al formato requerido por el modelo. Antes de construir la formulación, las instancias se validan para verificar la presencia de los conjuntos y parámetros requeridos, la consistencia de claves y la compatibilidad básica entre lotes, unidades y etapas.

Finalmente, los resultados se representan mediante estructuras específicas del paquete. Cada tarea programada contiene información sobre lote, unidad y tiempos de inicio y finalización, mientras que el resultado global reúne la lista de tareas, el desempeño del solver y el valor objetivo obtenido. Esta información puede consultarse en forma tabular, exportarse para análisis con Pandas y visualizarse mediante diagramas de Gantt, lo que conecta la formulación matemática con una salida interpretable para el usuario.

4 Resultados preliminares

4.1 Componentes implementados

Los resultados preliminares se presentan desde la perspectiva del desarrollo del paquete y de su utilización por parte de un usuario. Por este motivo, el foco no está puesto en comparar tiempos de resolución ni en evaluar la eficiencia computacional de las formulaciones, sino en mostrar que la arquitectura propuesta permite construir problemas, declarar características, seleccionar formulaciones y obtener resultados mediante componentes reutilizables.

La validación considerada en esta etapa se entiende como una verificación funcional del flujo de trabajo implementado. Para ello se utilizan ejemplos que parten de planillas de entrada, cargan y validan los datos, construyen una formulación compatible, invocan un solver externo y transforman la solución en resultados estructurados. Estos recorridos cubren casos monoetapa y multietapa, incluyendo una formulación de precedencia general y una formulación multietapa basada en ranuras.

De manera complementaria, el desarrollo cuenta con pruebas automatizadas de carácter unitario y de integración. Estas pruebas verifican la lectura desde Excel mediante rangos nombrados, la validación de problemas, la selección activa de lotes, la construcción de modelos monoetapa y multietapa, el modelo por ranuras de tiempo, la incorporación de características y objetivos, y la extracción de resultados tabulares y gráficos.

En la versión actual del paquete se implementaron componentes en las capas principales descriptas en la sección anterior. La Tabla 1 resume estos avances y la función que cumple cada grupo dentro del flujo de modelado. Un aspecto crítico es la separación entre las características declaradas por el usuario y los componentes del modelo, que se materializa mediante la fábrica de componentes descripta previamente. A partir de las características y el objetivo asociados con la instancia, el modelo instancia los `ModelComponent` registrados para la formulación seleccionada. Esto permite que las restricciones, variables auxiliares, objetivos o expresiones necesarias queden encapsulados en componentes reutilizables, manteniendo estable la interfaz pública utilizada en los ejemplos.

4.2 Casos de estudio

Para verificar el funcionamiento integrado del paquete se prepararon ejemplos de uso que implementan distintas combinaciones de problemas, características y formulaciones. Estos ejemplos no se proponen como instancias de referencia para comparar desempeño, sino como recorridos completos que muestran cómo el usuario puede pasar desde datos en planillas de cálculo hasta un modelo resuelto y una salida interpretable.

Por un lado, un ejemplo inicial corresponde a una planta batch monoetapa con equipos paralelos. En este caso se utiliza `SingleStageProblem` junto con una formulación de precedencia general para problemas monoetapa. El ejemplo

Tabla 1. Componentes implementados en la versión actual del paquete.

Capa	Componentes	Función principal
Problemas	Problemas monoetapa y multietapa secuenciales; bases de diseño para extensiones secuenciales y de red.	Representar la instancia a resolver, sus conjuntos, parámetros y validaciones estructurales.
Características y objetivos	Fechas de entrega, tiempos de preparación dependientes de unidad, tiempos de cambio dependientes de secuencia, makespan, earliness y tardiness totales, según formulación compatible.	Declarar capacidades opcionales y criterios de optimización sin modificar la clase del problema.
Formulaciones	Modelos de precedencia para casos monoetapa y multietapa; modelo de tiempo continuo basado en ranuras.	Transformar la instancia en un modelo Pyomo compatible con las características seleccionadas.
Entrada y validación	Lectura desde Excel mediante rangos nombrados, alias de parámetros y validación de claves.	Separar el formato de datos de la lógica de modelado y detectar inconsistencias antes de construir el modelo.
Resultados	`ScheduleTask`, `ScheduleResult`, salida tabular, `DataFrame` y diagrama de Gantt.	Entregar una solución interpretable y reutilizable para análisis posterior.

permite declarar fechas de entrega fijas, tiempos de preparación dependientes de la unidad y el objetivo de minimización de *makespan*.

Por otro lado, un segundo grupo de ejemplos trabaja con una planta multietapa (`MultiStageProblem`), donde las unidades están distribuidas por etapa y cada lote debe atravesar una secuencia de operaciones. Sobre esta instancia se prueban dos formulaciones: un modelo de precedencia general multietapa y un modelo con ranuras de tiempo. Además, se incorpora un caso específico con tiempos de transición dependientes de la secuencia. En estas instancias se utiliza la selección activa de lotes para resolver subconjuntos pequeños de la instancia, lo cual facilita probar nuevas combinaciones de componentes durante el desarrollo.

4.3 Utilización de la herramienta

El Listado 1 muestra un ejemplo del flujo de uso para un caso multietapa. La secuencia de instrucciones refleja la interfaz pública del paquete: se crea el problema, se asocia una fuente de datos, se agregan parámetros, se declaran características y objetivo, se cargan y validan los datos, se construye una formulación compatible, se resuelve el modelo con un solver externo y se trabaja con los resultados.

```
from scheduling import constants
from scheduling.models import MultiStageGeneralPrecedenceModel
from scheduling.problems import MultiStageProblem

excel_file_name = "ejemplo-planta-batch-multietapa.xlsx"

problem = MultiStageProblem()
problem.add_data_source(excel_file_name)
problem.add_parameter("time_horizon", 80)

problem.add_feature(constants.SEQUENCE_DEPENDENT_CHANGEOVER_TIMES)
problem.set_objective(constants.MAKESPAN)

problem.load()
problem.validate()

model = MultiStageGeneralPrecedenceModel(problem)
model.build()

result = model.solve(solver="gurobi", options={})
result.print_tasks()
result.plot_gantt(show=True)
```

Listado 1. Ejemplo de script de usuario para un modelo multietapa de precedencia general.

Este ejemplo muestra que la formulación matemática queda encapsulada detrás de clases específicas del paquete. El usuario no necesita construir manualmente las variables y restricciones de Pyomo para cada caso, sino declarar las características y el objetivo del problema, y seleccionar el modelo apropiado. Con ediciones localizadas del mismo script, podría agregarse `constants.FIXED_DUE_DATE` como característica adicional, cambiar el criterio de optimización a `constants.TOTAL_EARLINESS`, o reemplazar la formulación instanciada por `TimeSlotsModel`. La interfaz común permite expresar estos cambios a nivel de problema y modelo, mientras que las ecuaciones concretas generadas permanecen encapsuladas dentro de la implementación.

La resolución, si finaliza con éxito, devuelve un resultado estructurado disponible también como atributo del modelo. El objeto `ScheduleResult` reúne las tareas programadas, el valor de la función objetivo, el estado del solver, la condición de terminación, cotas y tiempos reportados por el solver cuando están disponibles.

Finalmente, las tareas extraídas de la solución se representan mediante objetos `ScheduleTask`, que registran lote, unidad, tiempos de inicio y finalización, etapa cuando corresponde, y duraciones asociadas a preparación o cambio. A partir de esa estructura, el usuario puede imprimir una tabla, obtener un `DataFrame` de Pandas o generar un diagrama de Gantt. Esta salida completa el recorrido de uso del paquete: los datos de entrada se transforman en una formulación de optimización y luego en una representación utilizable del *schedule* obtenido.

5 Conclusiones y trabajos futuros

Los avances presentados muestran la utilidad potencial de una herramienta orientada a construir modelos matemáticos para problemas de programación de la producción mediante componentes reutilizables. En lugar de desarrollar una implementación aislada para cada caso, el paquete permite describir una instancia, incorporar características operativas, seleccionar una formulación y obtener resultados estructurados siguiendo un flujo de trabajo común. Esta organización facilita implementar y evaluar rápidamente alternativas de modelado para un problema dado, lo cual resulta valioso tanto en actividades de investigación como en aplicaciones industriales.

Desde el punto de vista del desarrollo tecnológico, el paquete constituye una base para acumular capacidades propias en modelado, implementación y resolución de problemas de *scheduling*. La separación entre problemas, características, formulaciones, componentes de modelo, datos de entrada y resultados permite incorporar progresivamente modelos de la literatura y compararlos dentro de una infraestructura común. Esta capacidad es relevante no solo para producir resultados académicos, sino también para futuras actividades de transferencia, donde suele ser necesario adaptar formulaciones a datos, restricciones y criterios específicos de cada organización.

Como trabajos futuros, se propone ampliar la herramienta en tres direcciones principales. En primer lugar, se buscará incorporar mecanismos para transformar o vincular distintas representaciones de problemas, incluyendo formulaciones secuenciales y representaciones de red como STN y RTN. En segundo lugar, se estudiará la posibilidad de dividir instancias complejas en subproblemas, permitiendo aplicar modelos o componentes diferentes a cada parte del problema. En tercer lugar, se avanzará en la integración de los modelos y componentes implementados dentro de estrategias algorítmicas de resolución, combinando formulaciones exactas, descomposición, enfoques híbridos y procedimientos específicos según la estructura de cada caso.

Finalmente, si bien el paquete se orienta inicialmente a problemas de *scheduling*, varios de sus componentes abstractos y su flujo de trabajo general son aplicables a otros dominios de optimización. Las ideas de declarar problemas, agregar características, asociar componentes de modelo, cargar datos desde fuentes externas y devolver resultados estructurados no dependen exclusivamente de la programación de operaciones. Por este motivo, una línea de trabajo por explorar consiste en extraer los elementos centrales hacia una capa más general, sobre la cual `scheduling` pueda funcionar como subpaquete especializado dentro de una herramienta de optimización más amplia.

Agradecimientos. Los autores de este trabajo desean agradecer a la Universidad Tecnológica Nacional por la financiación recibida a través del PID UTN SIECFE0010230TC.

Referencias bibliográficas

AIMMS. (s.f.). *AIMMS Documentation*. Consultado el 26 de junio de 2026, desde https://documentation.aimms.com/

Bynum, M. L., Hackebeil, G. A., Hart, W. E., Laird, C. D., Nicholson, B. L., Siirola, J. D., Watson, J.-P., & Woodruff, D. L. (2021). *Pyomo – Optimization Modeling in Python* (3.ª ed.). Springer.

CPLEX. (s.f.). *IBM ILOG CPLEX Optimizer*. Consultado el 26 de junio de 2026, desde https://www.ibm.com/products/ilog-cplex-optimization-studio

Fourer, R., Gay, D. M., & Kernighan, B. W. (2003). *AMPL: A Modeling Language for Mathematical Programming* (2.ª ed.). Duxbury Press.

GAMS Development Corporation. (s.f.). *General Algebraic Modeling System (GAMS)*. Consultado el 26 de junio de 2026, desde https://www.gams.com/

Google OR-Tools. (2024, 28 de agosto). *Scheduling Overview*. Consultado el 26 de junio de 2026, desde https://developers.google.com/optimization/scheduling

Gurobi Optimization, LLC. (2026). *Gurobi Optimizer Reference Manual*. Consultado el 26 de junio de 2026, desde https://docs.gurobi.com/current/

Harjunkoski, I., Maravelias, C. T., Bongers, P., Castro, P. M., Engell, S., Grossmann, I. E., Hooker, J., Méndez, C., Sand, G., & Wassick, J. (2014). Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, *62*, 161-193.

Lan, L., & Berkhout, J. (2025). PyJobShop: Solving Scheduling Problems with Constraint Programming in Python. https://doi.org/10.48550/arXiv.2502.13483

Maravelias, C. T. (2021). *Chemical Production Scheduling: Mixed-Integer Programming Models and Methods*. Cambridge University Press.

Méndez, C. A., Cerdá, J., Grossmann, I. E., Harjunkoski, I., & Fahl, M. (2006). State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers & Chemical Engineering*, *30*, 913-946.

OptaPlanner. (s.f.). *OptaPlanner*. Consultado el 26 de junio de 2026, desde https://www.optaplanner.org/

OptaPy. (s.f.). *OptaPy: An AI Constraint Solver for Python*. Consultado el 26 de junio de 2026, desde https://github.com/optapy/optapy

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems* (3.ª ed.). Springer.

pyschedule. (s.f.). *pyschedule: Resource Scheduling in Python*. Consultado el 26 de junio de 2026, desde https://github.com/timnon/pyschedule

Sarina, A. (2025). Desarrollo y Resultados Preliminares de un Paquete de Optimización de la Producción en Python. *AJEA (Actas de Jornadas y Eventos Académicos de UTN). JIT 2024 – Jóvenes Investigadores Tecnológicos*, (AJEA 55), 372-378. https://doi.org/10.33414/ajea.1941.2025

Timefold. (s.f.). *Timefold Solver Documentation*. Consultado el 26 de junio de 2026, desde https://docs.timefold.ai/timefold-solver/latest/introduction