PROYECTO FINAL DE INGENIERÍA INDUSTRIAL

CREACIÓN DE UN SOFTWARE PARA PLANIFICACIÓN DE PRODUCCIÓN

Carlos Maximiliano Hense

47.213

Tutor: Lic. Francisco Fernando Villaverde

2011

DESCRIPTOR BIBLIOGRÁFICO

Creación de un Software para planificación de producción: Investigación de Operaciones, Optimización, Software.

Páginas: 75

ITBA

Año edición: 2011

A mi familia, por todo el apoyo brindado a lo largo de mi carrera

RESUMEN

En el marco del concurso organizado por la Association of Latin-Iberoamerican Operational Research Societies (ALIO) y Kimberly-Clark Corporation - Latin American Operations (KCC-LAO), a fines del año 2010 y principios del 2011, dos alumnos del ITBA desarrollaron un software que permite resolver un problema de planificación de producción. El presente proyecto busca explicar cómo se desarrolló el software que al ejecutarse realiza la carga de datos y parámetros, la ejecución del algoritmo de optimización, y la creación de archivos de salida con la solución alcanzada.

Se analiza detalladamente cuáles fueron los criterios de elección, por un lado de la herramienta elegida para el desarrollo del algoritmo que resolverá el problema planteado y, por otro lado, de la herramienta elegida para el desarrollo del código que hará del algoritmo un software. Además, se explica detalladamente la lógica de dicho código, qué inconvenientes surgieron durante su desarrollo y cómo se solucionaron. Luego se analiza su robustez, y finalmente se discuten las mejoras que se pueden realizar en el futuro.

SUMMARY

Participating in the contest organized by the Association of Latin-Iberoamerican Operational Research Societies (ALIO) and Kimberly-Clark Corporation - Latin American Operations (KCC-LAO), in late 2010 and early 2011, two ITBA students developed software to solve a production planning problem. This project seeks to explain how the development of the software that loads data, subsequently executes the optimization algorithm, and finally creates the output files of the solution.

The selection criteria for the tool to develop the algorithm that will solve the current problem on one hand and, on the other hand, for the tool that will develop the code that turns the algorithm into software, are analyzed in detail. The code's logic is explained thoroughly, indicating which problems popped-up during its development and how they were solved. After that, code robustness is analyzed and finally possible future improvements are discussed.

AGRADECIMIENTOS

A Andrés García Strauss, por su colaboración invaluable.

A Marcela Guardabassi, por su apoyo incondicional.

A Francisco Villaverde, tutor del proyecto y del trabajo final.

A José Jalil, por sus constantes aportes y apoyo en el desarrollo del proyecto.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El presente proyecto final de Ingeniería Industrial se basa en un trabajo realizado junto al alumno del Instituto Tecnológico de Buenos Aires (ITBA), Santiago Segarra, durante el final del año 2010 y principios del año 2011. La idea surge cuando la universidad promociona un concurso organizado por la "Association of Latin-Iberoamerican Operational Research Societies" (ALIO) y "Kimberly-Clark Corporation – Latin American Operation" (KCC – LAO) que por segunda oportunidad (la primera en el año 2009) promueven que alumnos universitarios, cursando una maestría o doctorado en distintas universidades de todo Latinoamérica, resuelvan un problema real de optimización de producción.

Una de las condiciones necesarias para poder inscribirse (ver Anexo I para conocer todas las reglas del concurso) era que cada grupo presente un tutor académico que sea, a su vez, profesor de dicha universidad. La persona elegida fue el Licenciado Francisco Villaverde, quien además es hoy el tutor del proyecto final de ambos alumnos.

El resultado final del proyecto fue la creación un software que permite resolver un problema de optimización de carga de línea de producción. Para poder lograrlo fue necesario utilizar 2 herramientas: una capaz de crear un algoritmo que resuelva el problema planteado y otra que desarrollará todo el entorno de dicho algoritmo con el objetivo de crear el producto final.

En el presente trabajo se realizará un análisis detallado sobre cuáles fueron los criterios que se usaron para la elección de ambas herramientas, cómo se creó a partir del algoritmo un software, cuál fue la lógica del código desarrollado, qué inconvenientes surgieron, cómo se solucionaron, cómo se sometió el código a prueba y cómo se lo puede mejorar en un futuro.

Separación de las tareas a realizar por cada integrante

Antes de comenzar con el análisis se explicará brevemente cómo se dividieron las tareas entre los alumnos para poder trabajar de forma más cómoda. Cuando en una empresa existe la necesidad de desarrollar un software, varias personas de distintas áreas forman parte del equipo. En este caso se pueden identificar tres partes principales: la primera es el usuario que tiene la necesidad y dará las características que el software debe presentar. En segundo lugar se encuentra el experto en optimización que será la persona encargada de desarrollar, verificar y mejorar el algoritmo que resolverá el problema

planteado en una herramienta específica de optimización; definirá además cuáles son los indicadores de "performance" para evaluar el algoritmo en cada uno de sus pasos de creación. Por último, se encuentra el ingeniero en software, quien hará del algoritmo el producto final: el software. También es el encargado de integrar toda la información, de crear los archivos a entregar y trabajará en conjunto con el experto en optimización en el desarrollo del modelo. En la figura 1 se puede observar lo descripto anteriormente:

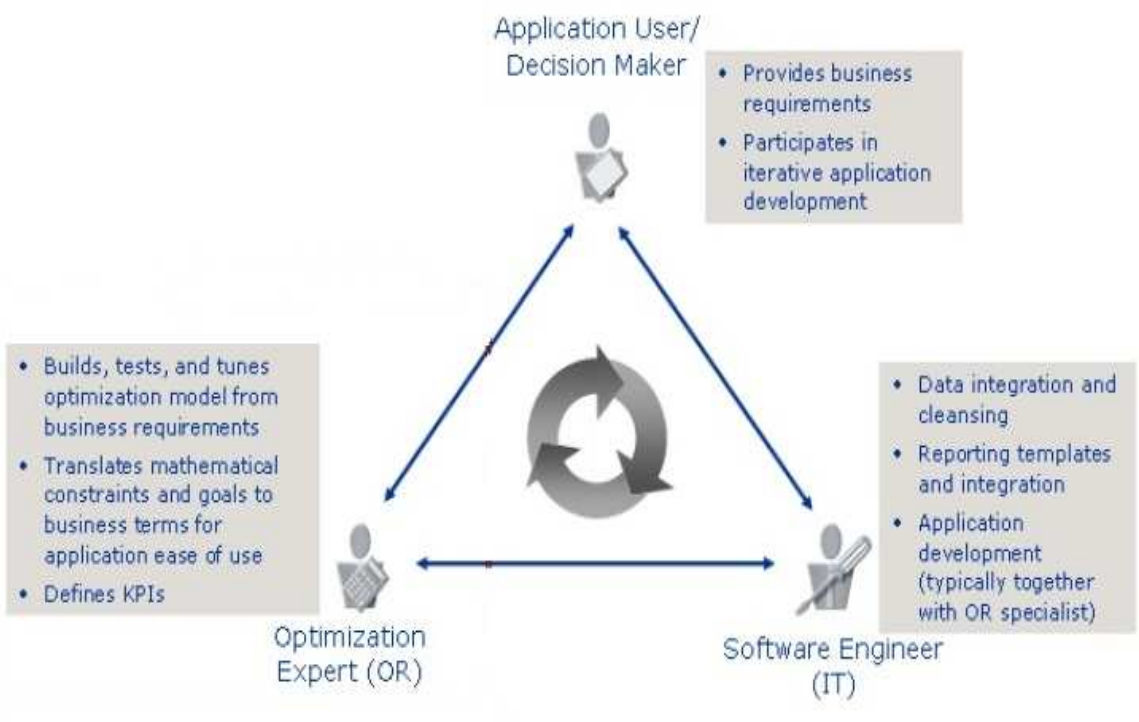


Figura 1: Organización del equipo para desarrollar un software [1]

En este trabajo, el usuario son los organizadores del concurso, el experto en optimización fue Santiago Segarra y el Ingeniero en Software fue Maximiliano Hense.

[1] ILOG. Student Workbook, 2007, pág. 15

CAPÍTULO 1: SELECCIÓN DE HERRAMIENTAS

SELECCIÓN DE LA HERRAMIENTA DE OPTIMIZACIÓN

Antes de comenzar con el análisis, podrán observar en el Anexo II un glosario con las definiciones de las palabras o conceptos que son específicos del ambiente informático y deben ser comprendidos para poder entender este trabajo. Una vez hecha esta aclaración se continúa con el desarrollo de esta sección. Para este concurso la organización presenta cuatro patrocinadores, que proveen licencias libres de software para optimización:

1- IBM: provee licencias académicas de Cplex 12.x, CP Optimizer 2.x y ODM 3.x OPL Development Studio 6.x.
2- Fair Isaac Corporation provee a cada equipo con una licencia académica o libre de FICO Xpress Optimization Suite 7.
3- OptTek provee a cada equipo con una licencia académica o libre de OptQuest Software
4- Maximal Software provee a cada equipo con una licencia académica o libre de MPL Software.

Nota: para mayores detalles dirigirse al Anexo III.

Es importante destacar que la organización no limita a los concursantes al uso de estos productos, sino que también se encuentra permitido el uso de cualquier herramienta de optimización que sea libre y su uso no requiera pago de licencias. Sin embargo, debido al poco tiempo disponible para la elección, solo se realizó un análisis detallado de las 4 herramientas propuestas por la organización del concurso. A continuación se realiza una breve introducción de cada una de ellas.

IBM ILOG CPLEX Optimizer

Como ya se mencionó, los organizadores del concurso proveían licencias académicas para varios productos de IBM: Cplex, CP Optimizer y OPL. *ILOG OPL development Studio* es un IDE (Ambiente de Desarrollo Integrado) que utiliza OPL (Lenguaje de Programación para Optimización) para crear y ejecutar modelos matemáticos, compuestos por funciones objetivos y restricciones, que interpretan información de un problema de optimización. Este software, a diferencia de los otros, presenta dos motores de optimización distintos: “Math programming” y “Constraint Programming”. Cplex es

la herramienta que se utiliza cuando tenemos que resolver problemas del tipo LP (programación lineal), MIP (programación lineal entera mixta) y QP (programación cuadrática), o sea, cuando utilizamos “Math programming”. Cuando los problemas son demasiados complicados para resolverlos utilizando los métodos anteriormente mencionados, por ejemplo, problemas de combinatoria considerables o problemas donde se busca gran detalle a la hora de planificar, se utiliza el segundo motor de optimización que ayuda a obtener buenas aproximaciones de la respuesta correcta en un lapso menor de tiempo.

ILOG Cplex provee soluciones robustas a problemas de optimización del mundo real a gran velocidad. Puede generar interfaces con distintos lenguajes como C++, C# y Java, lo que le da a los desarrolladores una variedad de caminos para poder trabajar. Cplex se puede utilizar en los siguientes sistemas operativos: AIX, HP-UX, Linux, Solaris y Windows.

Cuando se trata de un problema de programación lineal, uno puede elegir entre distintos algoritmos para poder resolverlo, como el método simplex (simple o compuesto), el algoritmo de barrera (o de punto interior) y el algoritmo de red.

En el caso que se deba resolver problemas de MIP, Cplex posee un algoritmo de “branch and cut” que presenta rasgos modernos como planos de cortes y heurística, para poder obtener soluciones enteras. Cuando se lo combina a este método con un “presolver” (por ejemplo utilizando el método simplex para dar una solución aproximada inicial) hace que Cplex sea una herramienta muy poderosa para resolver difíciles problemas de MIP.

Por último, Cplex tiene una opción que permite mejorar la performance si existe disponibilidad de núcleos en el procesador. Un detalle importante del programa es que a partir de la versión 11, incluye una nueva funcionalidad, donde la optimización de MIP es de forma paralela y se puede realizar de dos formas distintas: determinística u oportunística. La primera aprovecha el paralelismo para resolver los nodos del árbol “branch and cut”, produciendo una solución única e invariable, mientras que la segunda opción requiere menos sincronización entre “threads”, ofreciendo mejor performance en el cálculo de la solución.

Fair Isaac Corporation - FICO Xpress

Xpress es un conjunto de productos que incluye el lenguaje de programación de modelo matemático y una gran variedad de potentes motores de optimización. Hoy en día es

utilizado por clientes importantes como American Airlines, Avis, Nestlé, NFL, entre otros para crear soluciones rápidas y precisas a problemas con millones de variables y restricciones.

El optimizador permite resolver los siguientes tipos de problemas: lineal, lineal entero mixto, cuadrático y cuadrático entero mixto, entre otros. Existe un herramienta adicional llamada *Xpress-SLP solver* que se utiliza para resolver problemas que son no-lineales o no-convexos, pero este análisis escapa los límites del trabajo.

Dos características de vital importancia para la resolución del algoritmo son que el "solver" tenga incorporado, entre otras cosas, la opción de trabajar utilizando el método simplex y el método MIP. Los creadores de esta herramienta destacan que el optimizador de *Xpress* provee rápidas y confiables implementaciones del método simplex para resolver problemas de programación lineal. Otras características son:

- Integración de un algoritmo de pre solución que ayuda a reducir el tamaño original del problema y del tiempo que lleva resolverlo.
- Parámetros automáticos para lograr una mejor performance y si se desea utilizarlo de forma manual, se pueden controlar todos los parámetros del proceso de optimización.

Por otro lado, el programa presenta un sofisticado algoritmo de "branch and cut" para resolver problemas de MIP y MIQP y, de esta forma, encontrar soluciones de forma rápida y de alta calidad. A este método también se encuentra integrada una herramienta de pre-solución que ayuda a reducir el tiempo y el tamaño del problema, como también varias estrategias de solución para que, de forma automática, ayuden a mejorar la calidad de solución del método. Otras características son:

- Resolución en paralelo para computadoras con varios núcleos.
- Utiliza heurística

El producto provee una variedad de interfaces con herramientas de programación; brinda las bibliotecas con los APIs (Interfaz de programación de aplicaciones) para C, C++, Java, .NET, VB6. El optimizador requiere versiones de Java partiendo de la 1.3 en adelante, .NET 1.1 en adelante para Windows 32 bit y .NET 1.2 en adelante para Windows 64 bit.

En lo que respecta al Hardware, la compañía realiza las siguientes sugerencias sobre el mejor entorno para crear un software:

- Para crear un programa en una desktop utilizar procesadores Intel i7 en adelante y para crearlo en un servidor, utilizar Intel Xeon X5570 en adelante.

- 2 GB de memoria en sistemas operativos de 32 bit y de 6 a 8 GB en sistemas operativos de 64 bit.

El programa también puede ejecutarse en plataformas Linux, Solaris, AIX y HP-UX. Los creadores remarcan que la performance del programa no se ve afectado por el sistema operativo de Windows que se tenga instalado: Windows XP, Vista, 7.

OptTek - OptQuest

OptQuest es una herramienta que facilita el desarrollo de aplicaciones basadas en la optimización de sistemas complejos que pueden o no simularse. El software es amigable para personas que poseen un conocimiento limitado en programación de procesos de optimización pero que necesitan trabajar con técnicas sofisticadas de dicho tema. El programa puede utilizarse directamente por analistas que buscan soluciones óptimas a negocios complejos y problemas de ingeniería. El motor del programa utiliza metaheurística, optimización matemática y componentes de red neural para poder encontrar la mejor solución a problemas de planeamiento. Es flexible, inteligente y fácil de usar. El motor de *OptQuest* también provee soporte para problemas de multi-objetivo de optimización (ejemplo: maximizar el rendimiento mientras se minimiza el riesgo).

El programa se encuentra disponible en .NET para Windows y Java para todas las plataformas. Es una programación orientada a objetos que permite el uso de C#, VB.net o cualquier lenguaje de programación .NET en el entorno de Windows. También existe una herramienta que se acopla al *OptQuest* que permite que se pueda programar en C y C++.

Otra opción del programa es que permite al usuario realizar varias evaluaciones en paralelo. De esta forma el tiempo para resolver el algoritmo es menor, ya que no necesita que una evaluación finalice para poder comenzar con la siguiente.

OptQuest se encuentra hoy en día embebido en más del 95% de todos los programas de simulación. Algunos ejemplos son:

- Monte Carlo Simulation
- Discrete Event Simulation
- Agent-based Simulation
- Project Portfolio Management

Maximal Software - Mathematical Programming Language

MPL emplea un lenguaje flexible que puede utilizarse para formular modelos de temas relacionados a distintas áreas de la compañía, como planeamiento de la producción, finanzas y distribución. Este software presenta una forma de operar distinta a los otros tres ya comentados. Incluye un lenguaje de modelo algebraico que permite al programador crear modelos de optimización a partir de ecuaciones algebraicas. El modelo se usa como base para generar una matriz matemática que se coloca directamente sobre el "solver" de optimización. Es por esto que los modelos desarrollados con *MPL* se pueden resolver utilizando muchos de los programas de optimización que se encuentran en el mercado. Es curioso que en dos de los programas que propone el concurso (*Cplex* y *Xpress*) se puedan correr los modelos desarrollados en *MPL*.

MPL presenta una arquitectura abierta (no se encuentra limitado a ningún "solver") lo que permite al programador elegir el "solver" que más le sirva para su proyecto. Una característica única del programa es que el "link" con el "solver" se realiza directamente a través de la memoria, por lo tanto al no haber archivos involucrados, la conexión es considerablemente más rápida y más robusta.

Es importante destacar que *MPL* puede manipular grandes cantidades de información e importar información directamente desde bases de datos u hojas de cálculo. Una vez que el modelo sea resuelto, también puede exportar las soluciones a una base de datos. Los modelos pueden ser embebidos en otras aplicaciones de Windows, lo que hace a *MPL* ideal para crear aplicaciones finales para usuarios.

Las características más importantes del lenguaje que utiliza *MPL* son:

- Separa la información de la formulación del modelo
- Importa información desde varias fuentes
- Uso de macros para repeticiones dentro del modelo
- Exclusión de partes del modelo utilizando condicionales
- Condiciones WHERE/IF para el manejo de casos especiales
- Ayuda al usuario con los mensajes de error

Un tema no menor es la memoria que utiliza el programa, ya que la memoria que exista en la computadora limita la velocidad de resolución del modelo. Generalmente, *MPL* utiliza entre 1 y 2 MB de memoria cada 10000 variables, lo que puede limitar el tiempo de creación y resolución de la matriz. Los creadores de *MPL* destacan que lograron que

el tiempo de creación de la matriz sea menor a 1 minuto en casos donde las matrices creadas son de gran tamaño, sin embargo el tiempo que puede llevar la resolución de la misma, independientemente de la cantidad de memoria que uno tenga, puede ser muy grande.

MPL fue diseñado para poder ejecutarse en múltiples plataformas; Windows es el sistema operativo más popular, pero también funciona en equipos HP 9000, IBM RS-6000, Sun SPARC y Silicon Graphics. Se destaca que modelos creados en una plataforma pueden ser leídos en otras, haciendo que *MPL* sea muy flexible.

Comparación de las herramientas analizadas

En este apartado se compararán, utilizando la información anterior y adicional, distintos aspectos de las cuatro herramientas, con el objetivo de poder esclarecer cuál es la que más conviene utilizar para desarrollar el algoritmo de planificación de producción. Para facilitar la visualización del análisis, el mismo se mostrará en formato de tablas.

El primer aspecto a evaluar es el uso que presenta cada software:

	"Solver"	Modelizador	Ambos
ILOG CPLEX	x		
FICO Xpress	x	x	x
OptQuest	x		
MPL		x	x

Tabla 1.1 – Alcance de cada software

El segundo aspecto busca evaluar si es una aplicación independiente, si poseen bibliotecas para lenguajes orientados a objetos ("Object/Class Library"), bibliotecas de lenguajes de procedimientos ("Procedure/Callable Library"), si los códigos fuentes de cada una de las aplicaciones se encuentran disponibles (lo lógico es que no, ya que ninguno de los programas son aplicaciones "OpenSource"), y por último si se puede embeber dentro de otro programa ("Add-in to"):

	Aplicación independiente	"Procedure/Callable Library"	"Object/Class Library"	Código fuente	Add-in to
ILOG CPLEX	x	x	x		
FICO Xpress	x	x	x		
OptQuest	x	x	x		
MPL	x				*CPLEX, GUROBI, XPRESS, LINDO*, etc.

Tabla 1.2 – Otros alcances de cada software

El tercer aspecto es ver si el producto puede leer y crear hojas de cálculo y bases de datos:

	Leer hoja de cálculo	Escribir en hoja de cálculo	Leer base de datos	Escribir en base de datos
ILOG CPLEX	x	x	x	x
FICO Xpress	x	x	x	x
OptQuest	x	x	x	x
MPL	x	x	x	x

Tabla 1.3 – Capacidad de leer y escribir hojas de cálculo y bases de datos

Es importante destacar que en algunos programas se puede escribir el código para leer/crear dentro del mismo código de optimización (*OptQuest*), mientras que en otros se lo escribe en una herramienta externa (*Java*, *C#*) que invoque al optimizador (*Cplex, Xpress*).

El cuarto aspecto a evaluar es el formato de variables con el que puede trabajar:

	Enteras, binarias	Semi-continuas	Discretas	Lineales por tramos	Otras
ILOG CPLEX	x	x	x	x	
FICO Xpress	x	x	x	x	
OptQuest	x	x	x	x	
MPL	x	x	x	x	No lineales

Tabla 1.4 – Capacidad de trabajo con distintos tipos de variables

El quinto aspecto se refiere a los distintos métodos que puede utilizar el algoritmo para lograr una solución.

Para programación lineal:

	Primal Simplex	Dual Simplex	Interior Point
ILOG CPLEX	x	x	x
FICO Xpress	x	x	x
OptQuest	x	x	x
MPL	x	x	x

Tabla 1.5 – Métodos de resolución de problemas que utilizan programación lineal

Para MIP:

	Branch and Cut	Branch and Price	Heurística
ILOG CPLEX	x		x
FICO Xpress	x		x
OptQuest	x		x
MPL	x	x	x

Tabla 1.6 - Métodos de resolución de problemas que utilizan MIP

El sexto aspecto a comparar, es si los algoritmos tienen la capacidad de ser pre-resueltos (“presolve”) y si pueden realizar un análisis de inviabilidad.

	"Presolve"	Análisis de Inviabilidad
ILOG CPLEX	x	x
FICO Xpress	x	x
OptQuest	x	x
MPL	x	x

Tabla 1.7 – Características de los algoritmos de cada optimizador

El séptimo y último aspecto a comparar es con qué lenguajes de programación puede generar una interfaz:

	Lenguajes
ILOG CPLEX	.NET/C++/VB6/Java
FICO Xpress	.NET/C/C++/VB6/Java
OptQuest	.NET para Windows y Java para todas las plataformas. Con una herramienta adicional también se puede programar en C y C++.
MPL	Con la componente de biblioteca OptiMax, modelos de *MPL* pueden ser embebidos en aplicaciones finales para usuarios usando Visual Basic, VBA, C#, C/C++ y Java

Tabla 1.8 – Lenguajes que soporta cada optimizador

En referencia a las distintas plataformas donde se puede ejecutar cada optimizador, se destaca que para este concurso lo único que se necesita es que la herramienta cree una solución que pueda ser ejecutada en una máquina con Windows Vista con un procesador Intel Core 2, 1.83GHz y 2 GB de memoria RAM. Todas las herramientas pueden ser utilizadas en el entorno de cualquier sistema operativo de Windows moderno (XP, Vista, 7), por lo tanto este factor no influye a la hora de elegir que optimizador se debe utilizar.

Selección de la herramienta para la creación del algoritmo

Antes de realizar dicho análisis, se explicará una condición que debe presentar el algoritmo para poder cumplir con las condiciones de solución que impone la organización del concurso, ya que esto afecta directamente la elección de la herramienta. Las reglas piden que el problema se resuelva en un tiempo menor a 10 minutos. Cuando se comienza a analizar el problema con mayor profundidad, rápidamente uno se da cuenta, por un lado, que la cantidad de variables a tener en cuenta son muchas, y por otro lado, que existe una sola función objetivo a resolver. Es

importante aclarar que a medida que la cantidad de variables binarias sea mayor, el tiempo de resolución del algoritmo se incrementa de forma exponencial. Durante el desarrollo de la solución, rápidamente, se evidenció este problema, ya que no se llegaba a la solución pretendida en el tiempo estipulado debido a la gran cantidad de variables binarias con las que se trabajaba. Se comenzaron a realizar pruebas con el algoritmo desarrollado observando que las máquinas no trabajaban todos los productos posibles en cada época; empíricamente se determinó que se producen un máximo de 3 productos en cada máquina por época. Esto significa que la solución se calculará más rápido si el algoritmo solo procesa 3 productos por máquina por época en vez de todas las posibilidades (se obtiene la misma solución que si se procesaran todos los materiales por máquina por época). Por esta razón, es necesario que la herramienta que se elija realice una "pre-solución" al problema para que de esta forma, se establezcan cuáles son los 3 productos que deben producirse por máquina por época, logrando que el tamaño del problema sea menor (menos variables binarias) y consecuentemente el tiempo de solución disminuya.

Otro punto que se debe aclarar, ya que condiciona fuertemente la elección, es que tanto el experto en optimización como el ingeniero en software no tenían conocimientos previos de lenguaje de programación, pero si poseían conocimientos para poder desarrollar el algoritmo de planificación de producción.

Hechas estas dos aclaraciones, se explicará a continuación qué factores afectan directamente a la elección del software, finalizando con la creación de una matriz de decisión que dará como resultado la herramienta a elegir.

1- Calidad de la información recopilada: este punto es el más importante de todos, ya que ningún integrante del equipo conocía las herramientas (lenguaje, alcance, etc.) ni tenía conocimiento alguno de lenguajes de programación (C#, Java, etc.). Factor de ponderación 30/100
2- Capacidad de la herramienta de poder resolver el problema: en principio cualquier herramienta debería ser capaz de poder resolver el problema, ya que todas fueron propuestas por la organización. Sin embargo, las herramientas fueron creadas para cumplir un propósito principal y no necesariamente éste se condice con lo que se necesita de la misma. Factor de ponderación: 10/100
3- Dificultad del lenguaje para escribir el algoritmo: cada herramienta utiliza un lenguaje distinto para desarrollar el código. El programador siempre se

encuentra más familiarizado con algunos que con otros. Factor de ponderación: 15/100

4- Tiempo necesario para desarrollar el algoritmo: este factor es importante ya que existe una fecha para entregar el producto final y lleva tiempo aprender el lenguaje y desarrollar el algoritmo. Factor de ponderación: 20/100

5- Tiempo necesario para adquirir la herramienta: Una vez elegida la herramienta se deben pedir los permisos para poder comenzar a utilizarlas. No debería ser un problema en principio y por lo tanto no debería ser contemplado por el análisis, pero como existieron inconvenientes, entonces se le da un factor de ponderación de 10/100.

6- Contactos que puedan brindar ayuda sobre el optimizador: Como ya se mencionó anteriormente, nadie tenía conocimiento sobre el lenguaje de cada uno de ellos. El tiempo para entregar una solución es limitado y por lo tanto, conocer personas que nos podían brindar alguna solución a un posible problema es algo importante a tener en cuenta. Factor de ponderación: 15/100.

A continuación se muestra la matriz de toma de decisiones:

	Software	*ILOG Cplex*		*FICO Xpress*		*OptQuest*		*MPL*	
Factor	Ponderación	Aptitud	Nota	Aptitud	Nota	Aptitud	Nota	Aptitud	Nota
1- Calidad de la información	30	Excelente	10	Bueno	7	Regular	5	Regular	5
2- Capacidad de herramienta	10	Muy bueno	8	Muy bueno	9	Bueno	7	Bueno	6
3- Dificultad del lenguaje	15	Bueno	6	Regular	4	Regular	5	Regular	5
4- Tiempo de desarrollo	20	Bueno	6	Regular	5	Bueno	6	Regular	4
5- Tiempo para conseguir herramienta	10	Bueno	7	Bueno	7	Muy bueno	8	Malo	2
6- Posibles contactos	15	Muy bueno	8	Regular	4	Regular	4	Regular	4
	100	Total	780	Total	590	Total	555	Total	445

Tabla 1.9 – Matriz de decisión

El resultado es claro: acorde a los factores que se consideraron más importantes, ILOG Cplex es la herramienta que se debe utilizar para desarrollar el algoritmo de planificación de producción. Las principales razones fueron:

1- La calidad de la información que se pudo conseguir de dicha herramienta. Se consiguieron filminas de Power Point y varios tutoriales muy completos y de

lectura rápida, lo que facilita su aprendizaje. Buscando en foros de internet, también se encontró información muy útil.

2- Posibles contactos. La universidad nos facilitó el contacto del vendedor de IBM de dicha herramienta, quien a su vez nos contactó con uno de los desarrolladores. En reiteradas oportunidades nos ayudó con determinados temas relacionados al software.

Un dato no menor, es que Kimberly & Clark también utiliza esta herramienta en su centro de investigación operativa para crear sus algoritmos acorde a las necesidades que surjan.

SELECCIÓN DE LA HERRAMIENTA PARA CREACIÓN DE SOFTWARE

Tareas a realizar por el ingeniero en software

Antes de comenzar con el análisis de la segunda herramienta es necesario describir cuáles son las tareas que debe realizar el ingeniero en software. Éste será el responsable por el uso de la misma, teniendo en cuenta que su principal objetivo es brindarle soporte al experto en optimización para la creación del algoritmo, lo que implica que:

1- Se debe crear un primer archivo de datos en formato .dat (formato aceptado para lectura de datos a través del *Cplex*). Se deben leer 7 archivos .csv ("comma separated value") y volcar toda la información en un archivo de texto que luego se lo convierte en un archivo .dat. La organización brinda los datos de entrada en 8 archivos .csv: PPS_BOM.csv, PPS_DEMAND.csv, PPS_LEADTIME.csv, PPS_MACHINES.csv, PPS_MATERIALPROD.csv, PPS_MATERIALS.csv, PPS_PARAMETERS.csv y PPS_SETUP.csv. El experto en optimización pide que para este primer archivo de datos .dat, se carguen todos los archivos menos el PPS_SETUP y de los otros siete a algunos se les realicen las siguientes modificaciones:

   a- Que el valor de la 3er columna ("Duration") del archivo PPS_DEMAND.csv de cada fila sea remplazado por 30.

   b- Que el valor de la 3er columna ("Leadtime") del archivo PPS_LEADTIME.csv de cada fila sea remplazado por 0.

   c- Que el valor de la 4ta columna ("Nethours") del archivo PPS_MACHINES.csv de cada fila sea remplazado por 30.

d- En el archivo PPS_PARAMETERS.csv el valor de “Horizon” sea la suma de la duración de cada época en un ciclo (si hay 4 épocas por ciclo y cada época dura 30, entonces “Horizon” es igual a 120).

e- Los otros tres archivos .csv no se modifican.

2- La segunda tarea a programar es que el primer archivo .mod (archivo creado por el experto de optimización en el *Cplex*) muestre la mejor respuesta que obtuvo hasta el momento en que se lo finalice por haber transcurrido una cierta cantidad de tiempo definida por el equipo o cuando la tolerancia de diferencia relativa MIP sea menor al 5%. (parámetro que hay que predeterminar en el *Cplex*) Además se le debe especificar al programa cuál es la memoria máxima de uso que tiene (en este caso fue limitado por el concurso en 2GB de memoria RAM).

3- Como el experto en optimización creó un vector de salida del primer archivo .mod, el cual se necesita para ejecutar el segundo archivo .mod., el ingeniero debe leer dicha variable y copiarla en el segundo archivo .mod para el correcto funcionamiento del algoritmo.

4- La cuarta tarea a realizar es el segundo archivo .dat, porque el experto requiere que luego se transcriba toda la información de los 8 archivos .csv originales en el segundo archivo .dat. Además pide que se realicen las siguientes modificaciones a algunos archivos:

a- Que el valor de la 3er columna (“Duration”) del archivo PPS_DEMAND.csv de cada fila sea remplazado por su entero redondeado para abajo.

b- Que el valor de la 3er columna (“Leadtime”) del archivo PPS_LEADTIME.csv de cada fila sea remplazado por su entero redondeado para arriba.

c- En el archivo PPS_PARAMETERS.csv el valor de “Horizon” sea la suma de la duración de cada época en un ciclo.

d- Que el valor de la 4ta columna (“Setup_Time”) del archivo PPS_SETUP.csv de cada fila sea remplazado por su entero redondeado para arriba.

e- Que el valor de la 5ta columna (“Setup-Cost”) del archivo PPS_SETUP.csv de cada fila sea remplazado por el cociente del valor de la 5ta columna y el valor de la 4ta columna.

f- Los otros cuatro archivos .csv no se modifican.

5- La siguiente tarea a realizar por el ingeniero es la creación de los archivos de salida. Según las reglas del concurso debían ser 3 archivos .csv con el siguiente formato:

   a- SEQUENCE_RANK.csv: en la primer columna debe estar la máquina, en la segunda columna el material que produce esa máquina y en la tercer columna la posición en la secuencia en la que se produce dicho material.

   b- MACHINE_RESULTS.csv: en la primer columna se indica la máquina, en la segunda el material, en la tercera la hora en la que se empieza a producir ese material en esa máquina, en la cuarta columna el tamaño del lote producido y en la quinta el tiempo de producción.

   c- COST_RESULTS.csv: en la primer columna se indica cuál es el beneficio total, en la segunda el nivel de servicio, en la tercera el costo de producción, en la cuarta el costo de inventario, en la quinta el costo por no cumplir con una orden y en la última el costo de penalización.

   Nota: se determinó junto al experto en optimización que el resultado de la ejecución del segundo archivo .mod serán los vectores necesarios para la creación de los tres archivos .csv de salida. La tarea del ingeniero fue leerlos del *Cplex* y colocarlos en su correspondiente archivo .csv.

6- La última tarea es sincronizar cada una de las tareas anteriores para que se realicen en su debido momento: primero debe crearse el primer .dat, luego ejecutarse el primer .mod, al finalizar este proceso se crea el segundo .dat que dará lugar a la ejecución del segundo .mod, finalizando con la creación de los archivos de salida.

Elección de la herramienta de programación

Una vez que se definieron las tareas a realizar por el ingeniero en software, se buscó la forma más simple para poder lograrla. Se probó hacer todas las tareas en la misma línea de comandos del Cplex, pero luego de unos días de leer tutoriales y probar, se llegó a la conclusión que no iba a ser posible. Para poder realizarlo se debía programar en los lenguajes que admitía el optimizador: C++, Java o C#.

En principio no habría ningún inconveniente con ninguno por parte de los organizadores, ya que la máquina donde iba a ser evaluada la solución poseía los "runtimes" para Java Virtual Machine 1.6 y .NET 2008. El problema que surgía era que el ingeniero en software no conocía ninguno de los lenguajes que se presentaban como opción. Investigando recolectó la siguiente información que le ayudó a tomar una decisión:

- C# es un lenguaje de programación orientado a objetos que fue desarrollado a partir de C++. Ambos lenguajes fueron desarrollados por Microsoft.
- Existe un programa de IBM (ODM) que usa como entorno de desarrollo Eclipse y dentro del mismo se encuentran, entre otros, el motor de *Cplex* y un motor para poder programar en Java. Java también es un lenguaje orientado a objetos.

C# es un lenguaje de más alto nivel que C++; una gran diferencia entre ambos es que el manejo de los recursos de la memoria está automatizado en C# y no en C++. Por lo tanto, C# es más fácil de usar y aprender, pero C++ ofrece mayor flexibilidad en cuanto al uso de la memoria (esto es beneficioso en aplicaciones que necesitan alta performance). Para los rangos de performance de este proyecto, es más conveniente el uso de C# por su facilidad a la hora de aprenderlo.

Para elegir entre C# y Java se realizó el siguiente razonamiento: Java tenía un punto a favor, ya que IBM tiene un programa que, mediante una plataforma en común, vincula el lenguaje Java con el *Cplex*; se consiguieron los manuales de dicho programa y además existe un contacto dentro de IBM que podía orientar al ingeniero en software. Por otro lado, si se desea programar desde cualquier lenguaje externo a una herramienta, la primera tarea a realizar es programar el código que ejecuta el *Cplex* desde la misma, lo que no es fácil y requiere un conocimiento avanzado de ambas herramientas para poder lograrlo. Por todo lo dicho, la herramienta elegida fue ODM, que tiene en un entorno de desarrollo, los motores de *Cplex* y Java vinculados.

Tras realizar la elección, se comenzó a programar un código que cumpliese con todos los requisitos anteriormente desarrollados. La primera actividad fue programar el primer archivo .dat (ver Anexo IV). Sin embargo, surgieron ciertas complicaciones que no se sabía solucionar. El primer inconveniente fue que no se sabía cómo compilar toda la solución en un archivo ejecutable y los manuales eran poco precisos al respecto; el segundo inconveniente era que no se podía determinar cuán flexible era la comunicación entre *Cplex* y Java, a pesar de estar vinculados a través de un mismo

entorno de desarrollo. Por estas dos razones y por cambios en la fecha de entrega de la solución se decidió investigar la opción de programar un "link" entre el *Cplex* y Microsoft Visual C# 2010, ya que en esta última herramienta es muy fácil crear un archivo ejecutable y, por otro lado, una vez superada la barrera de comunicación entre programas, la programación del código para cumplir con los requisitos iba a ser más simple. Al comprobar que esta conexión era posible, se optó por esta herramienta.

CAPÍTULO 2: CREACIÓN DEL SOFTWARE

LÓGICA DEL CÓDIGO DESARROLLADO

Una vez elegida la herramienta de programación y definidas las tareas a realizar, se procedió con el desarrollo del código. En este apartado se explicará la lógica del mismo para poder construir, a partir de un algoritmo que desarrolló el experto en optimización, un software que cumpla con todos los requisitos del cliente.

Lo primero que se buscó fue tratar de ejecutar de forma exitosa cualquier modelo de *Cplex* desde el C#. Una vez que se superó la barrera, buscando en tutoriales el código que permitía realizarlo, se procedió a desarrollar el código en torno al algoritmo para cumplir con las características pedidas del producto.

Las clases en C# se guardan en archivos .cs. Se pueden definir varias clases en un solo archivo .cs, pero por un tema de organización y prolijidad generalmente se escribe una clase por archivo .cs. En total se crearon 7 archivos .cs que contenían la totalidad del código:

1- Program.cs
2- GeneralInfo.cs
3- CsvHandler.cs
4- DatHandler.cs
5- PostprocessingDataHandler.cs
6- CommandLine.cs
7- Timer.cs

Antes de comenzar con la explicación de cada uno de ellos, es necesario explicar que para poder programar las tareas necesarias, se necesitan las bibliotecas (archivo .dll) que contienen las clases para que el C# pueda ejecutar el *Cplex* en el momento que se desea. Las mismas se obtuvieron en la página web del desarrollador del optimizador (IBM) y se agregaron como referencias del proyecto junto a las bibliotecas que ya utiliza el programa por defecto:

- System
- System.Core
- System.Data
- System.Data.DataSetExtensions
- System.Xml

- System.Xml.Linq
- Ilog.Cplex
- Oplall

Para consultas sobre la sintaxis del código o cualquier característica del C#, dirigirse al tutorial oficial de C# de Microsoft Developer Network.

Program.cs

Inicialmente hay que definir cuáles son las clases de las librerías que hay que utilizar para poder desarrollar el código en este archivo. Para incluirlas en el código de manera más cómoda, se escribe la sentencia *using* seguida de un "namespace" (todas las clases pertenecen a cierto "namespace", en este caso las clases necesarias se encuentran en los "namespace" ILOG.CP, ILOG.CPLEX, ILOG. OPL e ILOG.Util). Es importante aclarar que las clases que se son creadas por el ingeniero de software pertenecen al "namespace" KCC_Contest. A continuación podrán ver el diagrama de bloques correspondiente a esta clase, mostrando los métodos que se definen en ella, junto al código desarrollado y sus correspondientes explicaciones:

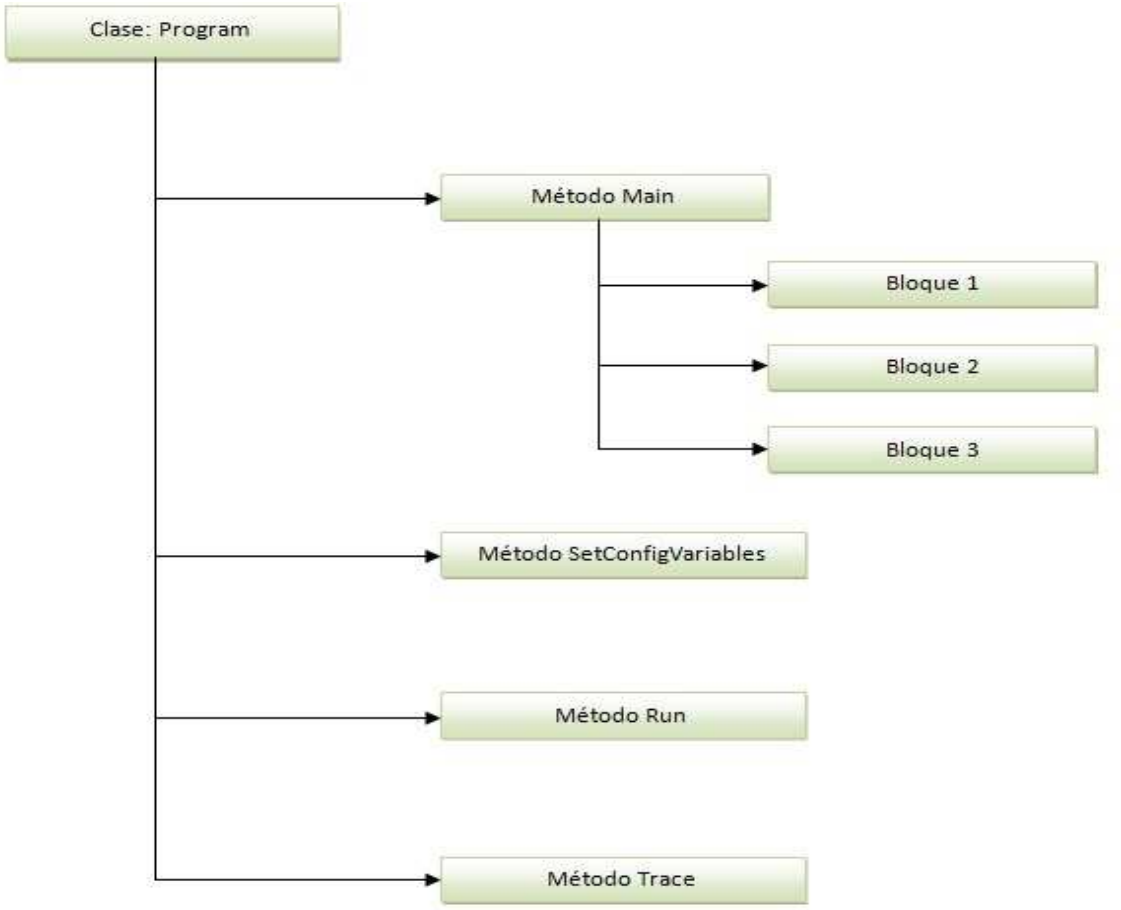


Figura 2.1: Diagrama de bloques de la clase Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using ILOG.Concert;
using ILOG.CP;
using ILOG.CPLEX;
using ILOG.OPL;
using ILOG.Util;
using System.Globalization;
```

En la clase *Program* se definieron cuatro métodos: *Main*, *Run*, *SetConfigVariables* y *Trace*. Es importante explicar que el método *Main*, por defecto, es el primero que se llama cuando se ejecuta el archivo compilado (.exe). A continuación se explica el desarrollo de dicho método:

```
namespace KCC_Contest
{
    class Program
    {

*/
El método contiene 3 bloques, cada uno de ellos rodeado de un "try catch"
(función que sirve para atrapar las excepciones).
/*

public static void Main(string[] args)
   {
     int status = 127;

*/
Primero se crea una instancia de la clase Timer, para medir el tiempo que lleva
la resolución del primer modelo
/*

            _globalTimer = new Timer();

*/
PRIMER BLOQUE:
   1- Primero llama al método SetConfigVariables (que se explicará luego de este
      método).
```

2- Luego llama al método *GenerateFirstDatFile* (que se explicará más adelante).

```
/*

try
    {
       SetConfigVariables();

       List<string[]> overrideCols = new List<string[]>();
       overrideCols.Add(new string[] { "PPS_DEMAND.csv", "2",
       GeneralInfo.epochDurationFixValue.ToString() });
       overrideCols.Add(new string[] { "PPS_LEADTIMES.csv", "2", "0" });

       DatHandler.GenerateFirstDatFile(GeneralInfo.csvFolderPath,
       GeneralInfo.datFilePath, overrideCols, null, new List<string>() {
       "PPS_SETUP.csv" }, true);
    }
      catch (System.Exception ex)
        {
          Console.WriteLine(ex.Message);
          Console.ReadLine();
          return;
        }

*/
```

SEGUNDO BLOQUE

Este código ejecuta el modelo de *Cplex*, llamando al método *Run*. En la línea que dice new *CommandLine*, los argumentos que se pasan son la ubicación del archivo .mod y del .dat.

```
/*

try
     {
      OplRunSample oplrun = new OplRunSample();
      string[] test = Environment.GetCommandLineArgs();

      oplrun._cl = new CommandLine(GeneralInfo.modFilePath, GeneralInfo.datFilePath);
      oplrun._timer = new Timer();
      status = oplrun.Run();
     }
           catch (ILOG.OPL.OplException ex)
           {
```

```
                Console.WriteLine(ex.Message);
                status = 2;
            }
            catch (ILOG.Concert.Exception ex)
            {
                Console.WriteLine(ex.Message);
                status = 3;
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex.Message);
                status = 4;
            }

*/
TERCER BLOQUE
En la línea que dice "double timeLimit =...", calcula el tiempo que le queda para ejecutar el segundo .mod.
Llama al método ChangeSolutionModLine, que abre el segundo archivo .mod, y en la línea donde se indica el tiempo límite, cambia el valor que figura por el recién calculado ("timeLimit")
En la línea que dice "new CommandLine", los argumentos que se pasan son la ubicación del archivo .mod y del .dat.
Ejecuta nuevamente el modelo de Cplex pero pasando como argumento la ubicación del segundo .mod y .dat (llamando nuevamente al método Run)
/*

try
     {
       OplRunSample oplrun = new OplRunSample();
       string[] test = Environment.GetCommandLineArgs();

       double timeLimit = GeneralInfo._globalTimeLimit - _globalTimer.getAbsoluteTime();
       DatHandler.ChangeSolutionModLine(GeneralInfo.secondModFilePath,    GeneralInfo.tilimFlag,
       "\tcplex.tilim  =  " + timeLimit.ToString(CultureInfo.InvariantCulture.NumberFormat)
       + ";");

       isFirstMod = false;

       oplrun._cl = new CommandLine(GeneralInfo.secondModFilePath, GeneralInfo.secondDatFilePath);
       oplrun._timer = new Timer();
       status = oplrun.Run();
     }
```

```
            catch (ILOG.OPL.OplException ex)
            {
                Console.WriteLine(ex.Message);
                status = 2;
            }
            catch (ILOG.Concert.Exception ex)
            {
                Console.WriteLine(ex.Message);
                status = 3;
            }
            catch (System.Exception ex)
            {
                Console.WriteLine(ex.Message);
                status = 4;
            }

            Environment.ExitCode = status;
            Console.WriteLine("--Press <Enter> to exit--");
            Console.ReadLine();
```

El segundo método que se explicará es el *SetConfigVariables* que es llamado por el método *Main* en el primer bloque.

```
*/
El método abre el archivo de configuración de la aplicación KCC_Solution_Config.txt (creada por el Ingeniero en software), para leer las ubicaciones en el disco de los archivos .mod y .csv (el software tiene que ser lo suficientemente flexible, ya que los datos de entrada con los cuales evaluarán la performance son desconocidos para los participantes), guardando los datos leídos en variables de la clase estática GeneralInfo.
/*

private static void SetConfigVariables()
      {
        TextReader reader = new StreamReader(GeneralInfo.configFileName);

        string line;

        while ((line = reader.ReadLine()) != null)
          {
            if (line.Contains("modFilePath"))
```

```
            GeneralInfo.modFilePath = line.Replace("modFilePath=\"", "").Replace("\"", "");
          else if (line.Contains("secondModFilePath"))
            GeneralInfo.secondModFilePath = line.Replace("secondModFilePath=\"",
            "").Replace("\"", "");
          else if (line.Contains("csvFolderPath"))
            GeneralInfo.csvFolderPath = line.Replace("csvFolderPath=\"", "").Replace("\"", "");
        }
    }
```

Otro método que invoca el método *Main* y está definido en esta clase, es el método *Run*. La mayor parte de las líneas de código de este método fueron extraídas del tutorial de *Cplex*, realizándole algunas modificaciones para ajustarlo a las necesidades de nuestro problema. A continuación se lo muestra y explica:

```
*/
Este es el método que permite ejecutar el modelo. Al final hay un llamado al
método ProcessData(isFirstMod, opl), que recibe como argumentos el resultado de
la ejecución del modelo (en la variable opl) y una indicación de si se trata del
primer o segundo archivo .dat (ya que en ambos casos hay que realizar acciones
diferentes)
/*

int Run()
    {
     int status = 127;
     OplFactory oplF = new OplFactory();
     if (_cl.CompileName != null)
      {
       OplCompiler compiler = oplF.CreateOplCompiler();
       StreamWriter ofs = new StreamWriter(_cl.CompileName, false);
       OplModelSource modelSource = oplF.CreateOplModelSource(_cl.ModelFileName);
       compiler.Compile(modelSource, ofs);
         ofs.Close();
         Trace("Compile");
         return 0;
        }
      if (_cl.ModelFileName == null && !_cl.IsProject)
          return 0;

      Trace("initial");
      OplRunConfiguration rc;
```

```
OplErrorHandler errHandler = oplF.CreateOplErrorHandler();
if (_cl.IsProject)
   {
    OplProject prj = oplF.CreateOplProject(_cl.getProjectPath());
    rc = prj.MakeRunConfiguration(_cl.getRunConfigurationName());
   }
else
   {
     if (_cl.DataFileNames.Length == 0)
         rc = oplF.CreateOplRunConfiguration(_cl.ModelFileName);
     else
         rc = oplF.CreateOplRunConfiguration(_cl.ModelFileName, _cl.DataFileNames);
   }
rc.ErrorHandler = errHandler;
OplModel opl = rc.OplModel;
OplSettings settings = opl.Settings;
settings.IsWithLocations = true;
settings.IsWithNames = true;
settings.IsForceElementUsage = _cl.IsForceElementUsage;

      status = 9;
      if (opl.ModelDefinition.hasMain())
      {
          status = opl.Main();
          Console.Out.WriteLine("main returns " + status);
          Trace("main");
      }
      else if (errHandler.Ok)
      {
          opl.Generate();
          Trace("generate model");
          if (opl.HasCplex)
          {
              if (_cl.ExportName != null)
              {
                  opl.Cplex.ExportModel(_cl.ExportName);
                  Trace("export model " + _cl.ExportName);
              }
              if (_cl.IsRelaxation)
              {
              Console.Out.WriteLine("RELAXATIONS to obtain a feasible problem: ");
                  opl.PrintRelaxation(Console.Out);
```

```
            Console.Out.WriteLine("RELAXATIONS done.");
        }
        if (_cl.IsConflict)
        {
          Console.Out.WriteLine("CONFLICT in the infeasible problem: ");
            opl.PrintConflict(Console.Out);
            Console.Out.WriteLine("CONFLICT done.");
        }
        if (!_cl.IsRelaxation && !_cl.IsConflict)
        {
            bool result = false;
            try
            {
                result = opl.Cplex.Solve();
            }
            catch (IloException ex)
            {
            Console.Out.WriteLine("### ENGINE exception: " + ex.Message);
            }
            if (result)
            {
                Trace("solve");
                Console.Out.WriteLine();
                Console.Out.WriteLine();
                Console.Out.WriteLine("OBJECTIVE: " + opl.Cplex.ObjValue.ToString("F"));
                opl.PostProcess();
                Trace("post process");
                if (_cl.IsVerbose)
                    opl.PrintSolution(Console.Out);
                status = 0;
            }
            else
            {
                Trace("no solution");
                status = 1;
            }
        }
    }
    else
    {
        bool result = false;
        try
```

```
            {
                result = opl.CP.Solve();
            }
            catch (IloException ex)
            {
                Console.Out.WriteLine("### Engine exception: " + ex.Message);
            }
            if (result)
            {
                Trace("solve");
                if (opl.CP.HasObjective())
                {
                    Console.Out.WriteLine();
                    Console.Out.WriteLine();
                    Console.Out.WriteLine("OBJECTIVE: " + opl.CP.ObjValue.ToString("F"));
                }
                else
                {
                    Console.Out.WriteLine();
                    Console.Out.WriteLine();
                    Console.Out.WriteLine("OBJECTIVE: no objective");
                }
                opl.PostProcess();
                Trace("post process");
                if (_cl.IsVerbose)
                    opl.PrintSolution(Console.Out);
                status = 0;
            }
            else
            {
                Trace("no solution");
                status = 1;
            }
        }
    }
    if (_cl.ExternalDataName != null)
    {
        StreamWriter ofs = new StreamWriter(_cl.ExternalDataName, false);
        opl.PrintExternalData(ofs);
        ofs.Close();
        Trace("write external data " + _cl.ExternalDataName);
    }
```

```
            if (_cl.InternalDataName != null)
            {
                StreamWriter ofs = new StreamWriter(_cl.InternalDataName, false);
                opl.PrintInternalData(ofs);
                ofs.Close();
                Trace("write internal data " + _cl.InternalDataName);
            }

            Trace("done");

            PostProcessingDataHandler.ProcessData(isFirstMod, opl);

            return status;
        }
```

El último método de este archivo es el *Trace* que es el método invocado por el método *Run* para escribir en la consola el progreso de la resolución del modelo.

```
void Trace(string title, string info)
    {
      Console.Out.WriteLine();
      Console.Out.Write("<<< " + title);
      if (info != null)
          Console.Out.Write(": " + info);
      if (_cl.IsVerbose)
       {
         Console.Out.Write(", at " + _timer.getAbsoluteTime() + "s, took " + _timer.getTime() + "s");
        _timer.restart();
       } Console.Out.WriteLine();
     }
void Trace(string title)
    {
      Trace(title, null);
    }
```

GeneralInfo.cs

Este archivo contiene la información general, como nombres de archivos, nombres de índices, etc., que necesita el modelo para poder ser ejecutado. Las clases usadas fueron:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

A continuación se muestra el diagrama de bloques correspondiente a esta clase (*GeneralInfo*) y se explica el código desarrollado:

Figura 2.2: Diagrama de bloques de la clase GeneralInfo

```
*/
Las variables marcadas como "const" son constantes, al resto se les asignan
valores durante el transcurso del programa.
/*

static class GeneralInfo
    {
        public const string configFileName = "KCC_Solution_Config.txt";
        public const string datFilePath = "mod1Data.dat";
        public const string secondDatFilePath = "mod2Data.dat";
        public static string csvFolderPath = "";
        public static string modFilePath = "";
        public static string secondModFilePath = "";

        public const string tilimFlag = "//TIME LIMIT FLAG";

*/
En este punto se define la duración total de procesamiento del programa. Los
organizadores del concurso limitaron el tiempo total de ejecución en 10 minutos,
por lo tanto se decidió prefijar el tiempo de procesamiento en 9 minutos y medio
```

```
y dejar 30 segundos para tareas que no son contempladas por el código pero que se realizan cuando se ejecuta el archivo (sobran algunos segundos que se dejan intencionalmente por un tema de seguridad).
/*

        public const double _globalTimeLimit = 9.5 * 60;

        public static List<float> epochCount;
        public static List<float> epochInitValues;
        public static List<float> epochValues;
        public const string epochFile = "PPS_DEMAND.csv";
        public const int epochColumn = 1;
        public const int epochValueColumn = 2;
        public static float horizon;

        public const int epochDurationFixValue = 30;

        public const int machinesEpochColumn = 2;
        public const string machinesFiles = "PPS_MACHINES.csv";
        public const int durationColumn = 3;

        public const string horizonFile = "PPS_PARAMETERS.csv";
        public const string horizonColValue = "Horizon";
        public const int horizonColIndex = 1;

        public const string dea3Flag = "//DEMANDAENARBOL3 FLAG";

        public const string solutionCsvPath = "Solutions";
        public const string sequenceRankFileName = "PPS_SEQUENCE_RANK.csv";
        public const string machineResultsFileName = "PPS_MACHINE_RESULTS.csv";
        public const string costResultsFileName = "PPS_COST_RESULTS.csv";
    }
```

CsvHandler.cs

Este archivo lee los archivos .csv de entrada y les modifica algunos valores a las columnas (como se describió en las tareas a desarrollar por el ingeniero de software), guardando la información en variables. También crea los archivos .csv de salida en la carpeta de salida. A continuación se muestra el diagrama de bloques y el código desarrollado:

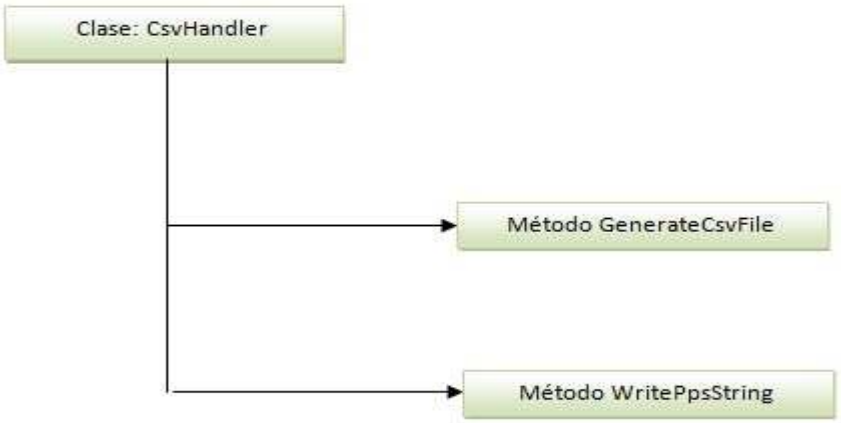


Figura 2.3: Diagrama de bloques de la clase CsvHandler

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Globalization;

static class CsvHandler
    {

*/
El siguiente método (GenerateCsvFile) crea los archivos .csv de salida. Recibe
como argumentos: las líneas del archivo y el nombre del mismo y lo genera dentro
de la carpeta de soluciones (definida en GeneralInfo.solutionsCsvPath)
/*

        public static void GenerateCsvFile(string[] lines, string fileName)
        {
            if (!Directory.Exists(GeneralInfo.solutionCsvPath))
                Directory.CreateDirectory(GeneralInfo.solutionCsvPath);

            TextWriter                writer                =                new
            StreamWriter(Path.Combine(GeneralInfo.solutionCsvPath, fileName));
            try
            {
                foreach (string line in lines)
                {
                    writer.WriteLine(line);
                    writer.Flush();
```

```
                }
            }
            finally
            {
                writer.Close();
            }
        }

*/
En los archivos .dat hay por cada archivo .csv una cierta cantidad de líneas. Este método es llamado por cada archivo .csv, y genera el texto que tiene que agregarle al archivo .dat
Maneja todos los cambios de valores necesarios (sobrescribir ciertas columnas, redondear columnas, dividir valores de distintas columnas) como también, siempre que sea necesario, corrige las comillas (que deben aparecer si es un valor de texto, y no deben estar si es un valor numérico).
/*

public static void WritePpsString(TextWriter writer, string csvFilePath, List<string[]> columnsToOverride, List<string[]> columnsToRound, List<string[]> columnsToCross, bool changeMachinesValues)
        {
            TextReader reader = new StreamReader(csvFilePath);
            try

             {
                bool hasColsToOverride = false;
                if (columnsToOverride != null)
                    hasColsToOverride = true;

                bool hasColsToRound = false;
                if (columnsToRound != null)
                    hasColsToRound = true;

                bool hasColsToCross = false;
                if (columnsToCross != null)
                    hasColsToCross = true;
                float crossTempValue = 0;

                bool isEpochReader = false;
                if (csvFilePath.Contains(GeneralInfo.epochFile))
                    isEpochReader = true;
```

```
                bool isMachinesFile = false;
                if (csvFilePath.Contains(GeneralInfo.machinesFiles))
                    isMachinesFile = true;

                bool isHorizonFile = false;
                if (csvFilePath.Contains(GeneralInfo.horizonFile))
                    isHorizonFile = true;

                bool writeHorizonValue = false;

                int currentEpoch = -1;

                string line;

    if (reader.ReadLine() != null)
       {
        while ((line = reader.ReadLine()) != null)
        {
              string[]    items    =    line.Split(new    string[]    {    ","    },
              StringSplitOptions.RemoveEmptyEntries);

              string outLine = "<";

              for (int i = 0; i < items.Length; i++)
                {
                  string item = "";
                  try
                   {
                    if (isEpochReader && GeneralInfo.epochValueColumn == i)

                     GeneralInfo.epochInitValues.Add(float.Parse(items[i].Trim().Repl
                     ace("\"", ""), CultureInfo.InvariantCulture.NumberFormat));

*/
A partir de este lugar explica como sobrescribir alguna columna de ser necesario:
/*

                    if (hasColsToOverride)
                     {
                      foreach (string[] col in columnsToOverride)
                      {
                      //col[0]: File Name
```

```
            //col[1]: Column Index
            //col[2]: Value
            if (Convert.ToInt32(col[1]) == i)
             {
               items[i] = col[2];
               break;
              }
            }
          }
item = items[i].Trim().Replace("\"", "");

    if (writeHorizonValue && i == GeneralInfo.horizonColIndex)
      {
        item = GeneralInfo.horizon.ToString();
         writeHorizonValue = false;
      }

     if (isHorizonFile && item.Contains(GeneralInfo.horizonColValue))
         writeHorizonValue = true;

         float f = float.Parse(item, CultureInfo.InvariantCulture.NumberFormat);

if (changeMachinesValues && isMachinesFile && GeneralInfo.machinesEpochColumn == i)
      {
        currentEpoch = (int)f;
      }
else if (changeMachinesValues && isMachinesFile && GeneralInfo.durationColumn == i)
      {
          item = (f * GeneralInfo.epochDurationFixValue /
          GeneralInfo.epochInitValues[currentEpoch -
          1]).ToString(CultureInfo.InvariantCulture.NumberFormat);
        }

    if (isEpochReader && GeneralInfo.epochColumn == i)
       {
        if (!GeneralInfo.epochCount.Contains(f))
            GeneralInfo.epochCount.Add(f);
        else
            isEpochReader = false;
        }
   else if (isEpochReader && GeneralInfo.epochValueColumn == i)
         GeneralInfo.horizon += f;
```

```
// A partir de este lugar explica como redondear alguna columna de ser necesario:

      if (hasColsToRound)
        {
         foreach (string[] col in columnsToRound)
           {
              //col[0]: File Name
              //col[1]: Column Index
              //col[2]: Up or Down
            if (Convert.ToInt32(col[1]) == i && item.Contains("."))
              {
               f = float.Parse(item.Remove(item.IndexOf(".")));
               item      =      (col[2]      ==      "Up")      ?      ((float)(f      +
               1)).ToString(CultureInfo.InvariantCulture.NumberFormat)                    :
               f.ToString(CultureInfo.InvariantCulture.NumberFormat);
               break;
              }
           }
         }

*/
A partir de este lugar explica cómo realizar operaciones entre columnas de ser
necesario:
/*

 if (hasColsToCross)
   {
    foreach (string[] col in columnsToCross)
      {
       //col[0]: File Name
       //col[1]: 1st Column Index
       //col[2]: 2nd Column Index
       //col[3]: Divide Multiply Add Substract
       if (Convert.ToInt32(col[2]) == i)
         {
          crossTempValue = f;
          break;
         }
       else if (Convert.ToInt32(col[1]) == i)
        {
         switch (col[3])
          {
           case "Divide":
```

```
                if (crossTempValue != 0)
                    item = (f / crossTempValue).ToString(CultureInfo.InvariantCulture.NumberFormat);
                break;
                case "Multiply":
                    item = (f * crossTempValue).ToString(CultureInfo.InvariantCulture.NumberFormat);
                break;
                case "Add":
                    item = (f + crossTempValue).ToString(CultureInfo.InvariantCulture.NumberFormat);
                break;
                case "Substract":
                    item = (f - crossTempValue).ToString(CultureInfo.InvariantCulture.NumberFormat);
                break;
              }
            break;
           }
        }
    }
    catch (Exception)
      {
       item = "\"" + item + "\"";
      }
    finally
      {
        outLine += item + " ";
      }
    }
  outLine += "> ";
  writer.WriteLine(outLine);
  writer.Flush();
 }
}

*/
Si se genera alguna excepción de tipo FileNotFoundException, se atrapa y se muestra un mensaje que dice que el archivo .csv no fue encontrado en la carpeta que se indicó.
/*

catch (FileNotFoundException)
  {
   throw new Exception("Archivo csv no encontrado: " + csvFilePath);
  }
finally
```

```
{
 reader.Close();
}
```

DatHandler.cs

En este archivo se define la clase *DatHandler*, cuyo objetivo es generar los dos archivos .dat (mod1Data.dat y mod2Data.dat) partiendo de los archivos .csv con ayuda de la clase *CsvHandler*. Estos serán el input de los dos modelos desarrollados en el optimizador (Initial.mod y Solution.mod). A continuación se muestra el diagrama de bloque de dicha clase y el correspondiente código con sus explicaciones:

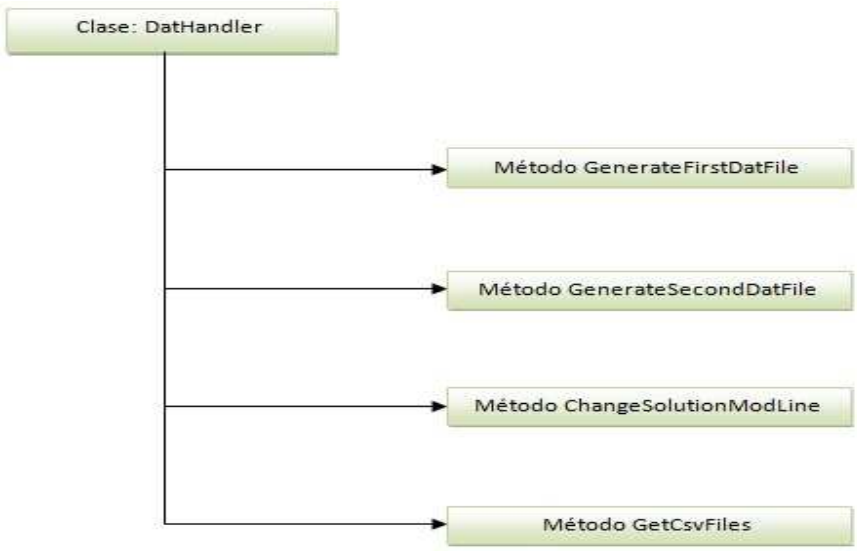


Figura 2.4 Diagrama de bloques de la clase DatHandler

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Globalization;

*/
Cuando se comenzó a programar esta clase surgió la idea de crear un método
utilizando la técnica de "overload". Esto es, crear un método que se puede
invocar con distinto número de argumentos. En este caso, se crea el método
GenerateFirstDatFile, al que se le van asignando distintos argumentos acorde a lo
que uno necesita en cada momento:
/*
```

```
static class DatHandler
    {

//Genera el primer archivo .dat

       public static void GenerateFirstDatFile(string csvFolderPath, string
       datFilePath, bool changeMachinesValues)
        {
              GenerateDatFile(csvFolderPath, datFilePath, null, null, null, null,
              changeMachinesValues);
        }

//Genera los archivos .dat con algunos valores sobrescritos

       public static void GenerateFirstDatFile(string csvFolderPath, string
       datFilePath, List<string[]> overrideColumnValues, bool
       changeMachinesValues)
        {
              GenerateDatFile(csvFolderPath, datFilePath, overrideColumnValues,
              null, null, null, changeMachinesValues);
        }

//Genera los archivos .dat con algunas columnas redondeadas y otras sobrescritas

       public static void GenerateFirstDatFile(string csvFolderPath, string
       datFilePath, List<string[]> overrideColumnValues, List<string[]>
       roundColumns, bool changeMachinesValues)
        {
              GenerateDatFile(csvFolderPath, datFilePath, overrideColumnValues,
              roundColumns, null, null, changeMachinesValues);
        }

*/
Genera los archivos .dat con algunas columnas redondeadas, otras sobrescritas e
ignora ciertos archivos .csv
/*

       public static void GenerateFirstDatFile(string csvFolderPath, string
       datFilePath, List<string[]> overrideColumnValues, List<string[]>
       roundColumns, List<string> csvFilesToIgnore, bool changeMachinesValues)
        {
```

```
            GenerateDatFile(csvFolderPath, datFilePath, overrideColumnValues,
            roundColumns, csvFilesToIgnore, null, changeMachinesValues);
        }

*/
```

Finalmente el método queda definido con los siguientes argumentos, pudiéndoselo ejecutar sólo con alguno de ellos:

a- la carpeta de donde leer los archivos .csv (información que se encuentra en la clase *GeneralInfo*)
b- la ubicación y el nombre del archivo .dat a generar
c- una lista de las columnas cuyos valores hay que sobrescribir o modificar
d- una lista de las columnas que hay que redondear (en este caso “null”, ya que ninguna necesita ser redondeada)
e- una lista de archivos .csv que hay que ignorar (en este caso "PPS_SETUP.csv")
f- un “bool” (verdadero o falso) para indicar si hay que cambiar los valores de “Machines” (en este caso verdadero)

```
/*

        public static void GenerateFirstDatFile(string csvFolderPath, string
        datFilePath, List<string[]> overrideColumnValues, List<string[]>
        roundColumns, List<string> csvFilesToIgnore, List<string[]> crossColumns,
        bool changeMachinesValues)
        {
            TextWriter writer = new StreamWriter(datFilePath);

            GeneralInfo.epochCount = new List<float>();
            GeneralInfo.epochValues = new List<float>();
            GeneralInfo.epochInitValues = new List<float>();
            GeneralInfo.horizon = 0;

            try
            {
                FileInfo[] csvFiles = GetCsvFiles(csvFolderPath);

                for (int i = 0; i < csvFiles.Length; i++)
                {
                    FileInfo file = csvFiles[i];
                    List<string[]> columnsToOverride = new List<string[]>();
                    List<string[]> columnsToRound = new List<string[]>();
                    List<string[]> columnsToCross = new List<string[]>();
```

```
                if (csvFilesToIgnore == null || !csvFilesToIgnore.Contains(file.Name))
                {
                    writer.WriteLine(file.Name.Replace(file.Extension, "") + " = {");

                    if (overrideColumnValues != null)
                        foreach (string[] column in overrideColumnValues)
                            if (column[0] == file.Name)
                                columnsToOverride.Add(column);

                    if (roundColumns != null)
                        foreach (string[] column in roundColumns)
                            if (column[0] == file.Name)
                                columnsToRound.Add(column);

                    if (crossColumns != null)
                        foreach (string[] column in crossColumns)
                            if (column[0] == file.Name)
                                columnsToCross.Add(column);

                    if (columnsToOverride.Count < 1)
                        columnsToOverride = null;
                    if (columnsToRound.Count < 1)
                        columnsToRound = null;
                    if (columnsToCross.Count < 1)
                        columnsToCross = null;

                    CsvHandler.WritePpsString(writer, file.FullName,
                    columnsToOverride, columnsToRound, columnsToCross,
                    changeMachinesValues);

                    writer.WriteLine("};");
                }
            }
        }

*/
La función “catch” atrapa alguna excepción (si existe) y muestra un mensaje diciendo que no se pudo encontrar el directorio de archivos .csv en el lugar que se le indicó.
/*
```

```
            catch (System.Exception)
            {
              throw new Exception("Directorio de archivos csv no encontrado: " +
              csvFolderPath);
            }
            finally
            {
                writer.Close();
            }
        }

*/
El siguiente método genera el segundo archivo .dat, llamando al método GenerateFirstDatFile. Además reemplaza la línea del segundo archivo .mod donde se define demandaenarbol3 (método ChangeSolutionModLine), por los datos obtenidos de la solución del primer archivo .mod (estos datos se pasan en el argumento float[][] ingresos).
/*

       public static void GenerateSecondDatFile(float[][] ingresos, string
       csvFolderPath, string secondDatFilePath, string secondModFilePath)
        {
            string dea3Def = "float demandaenarbol3[PPS_MATERIALS][epochs] = [";

            for (int i = 0; i < ingresos.Length; i++)
            {
                string line = "[";
                foreach (float f in ingresos[i])
                    line += f.ToString(CultureInfo.InvariantCulture.NumberFormat) + ",";

                line = line.Remove(line.LastIndexOf(",")) + "]";
                if (i < ingresos.Length - 1)
                    line += ",";
                dea3Def += line;
            }

            dea3Def += "];";

            ChangeSolutionModLine(secondModFilePath, GeneralInfo.dea3Flag,
            dea3Def);

            List<string[]> crossColumns = new List<string[]>();
```

```
            crossColumns.Add(new string[] { "PPS_SETUP.csv", "4", "3", "Divide" });

            List<string[]> roundColumns = new List<string[]>();
            roundColumns.Add(new string[] { "PPS_DEMAND.csv", "2", "Down" });
            roundColumns.Add(new string[] { "PPS_LEADTIMES.csv", "2", "Up" });
            roundColumns.Add(new string[] { "PPS_SETUP.csv", "3", "Up" });

            GenerateDatFile(csvFolderPath, secondDatFilePath, null, roundColumns,
            null, crossColumns, false);
        }

*/
El siguiente método modifica el segundo archivo .mod (recibe cierta línea y
cierto lugar donde pegarla y realiza el cambio)
/*

        public static void ChangeSolutionModLine(string modFilePath, string
        changeFlag, string newLine)
        {
            TextReader reader = new StreamReader(modFilePath);

            string newModFile = "";
            string line;
            bool changeNextLine = false;

            try
            {
                while ((line = reader.ReadLine()) != null)
                {
                    if (changeNextLine)
                    {
                        line = newLine;
                        changeNextLine = false;
                    }
                    else if (line.Contains(changeFlag))
                        changeNextLine = true;

                    newModFile += line + "\r\n";
                }
            }
            finally
            { reader.Close(); }
```

```
            TextWriter writer = new StreamWriter(modFilePath);

            try
            { writer.Write(newModFile); }
            finally
            { writer.Close(); }
        }

//Este método devuelve todos los archivos .csv que se encuentren en una carpeta

        private static FileInfo[] GetCsvFiles(string folderPath)
        {
            DirectoryInfo dir = new DirectoryInfo(folderPath);
            return dir.GetFiles("*.csv");
        }
    }
}
```

PostProcessingDataHandler.cs

En el quinto archivo se crea una clase que permite leer las variables de decisión que da cada uno de los archivos .mod que son necesarias para cumplir las tareas a desarrollar. El diagrama de bloques y el código desarrollado se pueden ver a continuación:

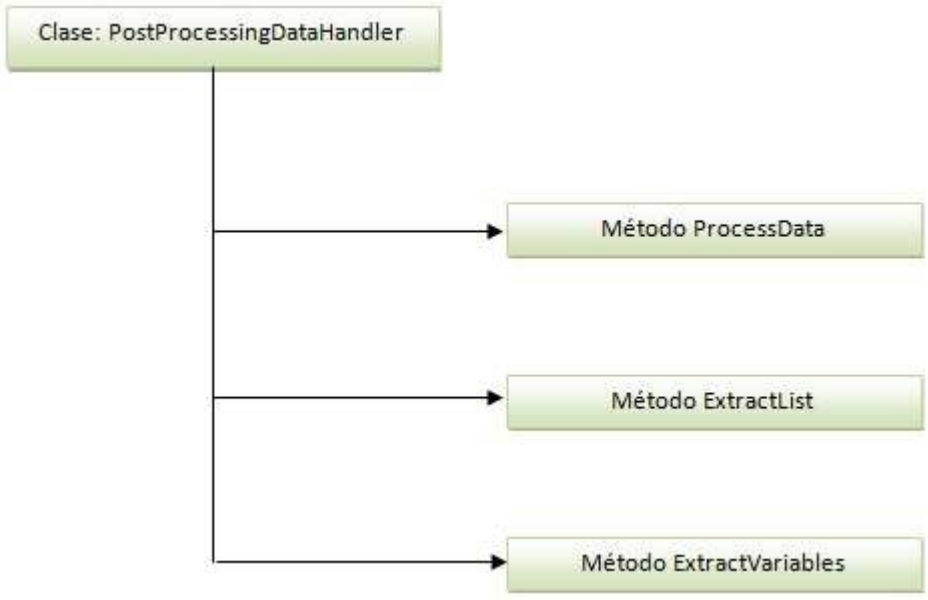


Figura 2.5: Diagrama de bloques de la clase PostProcessingDataHandler

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ILOG.Concert;
using System.IO;
using ILOG.OPL;
using System.Globalization;

static class PostProcessingDataHandler
    {

*/
Método llamado cada vez que se soluciona un modelo (ocurre dos veces cada vez que se ejecuta el programa) y procesa los datos de salida, realizando las tareas definidas (según si es el primer archivo .mod o el segundo)
/*

        public static void ProcessData(bool isFirstMod, OplModel opl)
        {
            StringBuilder builder = new StringBuilder();
            TextWriter writer = new StringWriter(builder);
            opl.PrintInternalData(writer);
            string internalData = builder.ToString();

*/
Si isFirstMod es verdadero (se acaba de resolver Initial.mod), arma el vector "demandaenarbol3" y llama a GenerateSecondDatFile para crear el segundo archivo .dat y ejecutar el segundo archivo .mod.
/*

if (isFirstMod)
     {
       string dea3Str = internalData.Substring(internalData.ToLower().IndexOf("demandaenarbol3"));
       dea3Str = dea3Str.Remove(dea3Str.IndexOf(";")).Split(new string[] { "=" },
       StringSplitOptions.None)[1];
       dea3Str = dea3Str.Substring(dea3Str.IndexOf("[") + 1);

       string[] dea3StrArr = dea3Str.Split(new string[] { "[" },
       StringSplitOptions.RemoveEmptyEntries);
       float[][] demandaEnArbol3 = new float[dea3StrArr.Length][];
```

```
        for (int i = 0; i < dea3StrArr.Length; i++)
           {
             string ingresos = dea3StrArr[i];

              if (ingresos.Contains("]"))
                  ingresos = ingresos.Remove(ingresos.IndexOf("]"));
              string[] iStr = ingresos.Split(new string[] { " " },
              StringSplitOptions.RemoveEmptyEntries);
              demandaEnArbol3[i] = new float[iStr.Length];
              for (int j = 0; j < iStr.Length; j++)
                  demandaEnArbol3[i][j] = float.Parse(iStr[j].Trim());
          }
DatHandler.GenerateSecondDatFile(demandaEnArbol3,      GeneralInfo.csvFolderPath,
GeneralInfo.secondDatFilePath, GeneralInfo.secondModFilePath
}

*/
Si es falso (se acaba de resolver Solution.mod), prepara los datos finales y
llama a GenerateCsvFile para escribir los archivos .csv de salida.
/*

 else
            {
              string        sRankStr        =        ExtractVariable(internalData,
              "pps_sequence_rank");
              sRankStr = sRankStr.Remove(sRankStr.IndexOf(";")).Split(new string[]
              { "=" }, StringSplitOptions.None)[1];
              sRankStr = sRankStr.Substring(sRankStr.IndexOf("[") + 1);

              string       mResultsStr       =       ExtractVariable(internalData,
              "pps_machine_results");
              mResultsStr = mResultsStr.Remove(mResultsStr.IndexOf(";")).Split(new
              string[] { "=" }, StringSplitOptions.None)[1];
              mResultsStr = mResultsStr.Substring(mResultsStr.IndexOf("[") + 1);

              string cResults = ExtractVariable(internalData, "pps_cost_results");
              cResults = cResults.Replace("<", "").Replace(">", "").Replace("[",
              "").Replace("]", "").Trim();
              string[]        cResultsValues        =        cResults.Replace(";",
              "").Substring(cResults.IndexOf("=") + 1).Trim().Split(new string[] {
              " " }, StringSplitOptions.RemoveEmptyEntries);
              for (int i = 0; i < cResultsValues.Length; i++)
```

```
            {
              if (cResultsValues[i].ToLower().Contains("e+"))
                {
                   string        val        =        cResultsValues[i].Replace(",",
                   CultureInfo.InvariantCulture.NumberFormat.NumberDecimalSeparator
                   ).Replace(".",
                   CultureInfo.InvariantCulture.NumberFormat.NumberDecimalSeparator
                   );
                   val                      =                    float.Parse(val,
                   CultureInfo.InvariantCulture.NumberFormat).ToString("f99").Tr
                   imEnd('0');
                   if (val.EndsWith(",") || val.EndsWith("."))
                       val = val.Replace(",", "").Replace(".", "");

                   cResultsValues[i] = val;
                }
             }

          string[] costResults = new string[2];
          costResults[0] = "TOTAL_PROFIT,SERVICE_LEVEL,PROD_COST,INV_COST,BO_COST,TARGET_COST";
          costResults[1] = "";
          foreach (string s in cResultsValues)
          costResults[1] += s + ",";
          costResults[1] = costResults[1].Remove(costResults[1].Length - 1);

          List<string> sequenceRank = ExtractList(sRankStr);
          List<string> machineResults = ExtractList(mResultsStr);

          sequenceRank.Insert(0, "MACHINE,MATERIAL,RANK");
          machineResults.Insert(0, "MACHINE,MATERIAL,START_TIME,LOT_SIZE,DURATION");

          CsvHandler.GenerateCsvFile(sequenceRank.ToArray(), GeneralInfo.sequenceRankFileName);
          CsvHandler.GenerateCsvFile(machineResults.ToArray(), GeneralInfo.machineResultsFileName);
          CsvHandler.GenerateCsvFile(costResults, GeneralInfo.costResultsFileName);
             }
        }
```

*/

El siguiente método transforma un vector de su representación textual a una lista para poder pegarlo en el archivo que corresponda.

/*

```
        private static List<string> ExtractList(string stringValue)
        {
            List<string> result = new List<string>();

            string[] arr = stringValue.Split(new string[] { "<" },
            StringSplitOptions.RemoveEmptyEntries);

            for (int i = 0; i < stringValue.Length; i++)
            {
                string line = arr[i].Trim();
                if (line.Contains("\"\""))
                    break;
                result.Add(line.Remove(line.IndexOf(">"), 1).Replace(" ",
              ",").Replace("\"", ""));
            }

            return result;
        }

*/
El siguiente método extrae el valor de una variable del texto de salida que
devuelve el Cplex como solución de la ejecución de un modelo.
/*

        Private static string ExtractVariable(string data, string variableName)
        {
           return data.Substring(data.ToLower().IndexOf(variableName.ToLower()));
        }
    }
}
```

CommandLine.cs

La tarea de esta clase es, mediante la ayuda del constructor "public CommandLine(string modFile, string datFile)", prefijar la información que se utilizará para ejecutar el modelo. Dicha información es leída con la ayuda de los métodos *getProjectPath* (archivo .mod) y *getRunConfigurationName* (archivo .dat). Las líneas de este código también se extrajeron del tutorial de *Cplex* y se le realizaron las modificaciones necesarias para que funcione correctamente en este caso. A continuación se puede observar el diagrama de bloques y el código de dicha clase:

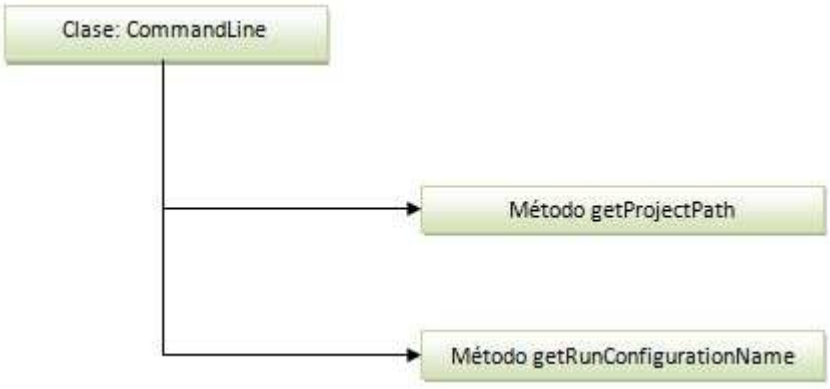


Figura 2.6: Diagrama de bloques de la clase CommandLine

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

class CommandLine
    {
        public string ModelFileName;
        public string[] DataFileNames;
        public string ExportName;
        public string CompileName;
        public string ExternalDataName;
        public string InternalDataName;
        public Boolean IsVerbose = false;
        public Boolean IsForceElementUsage = false;
        public Boolean IsRelaxation = false;
        public Boolean IsConflict = false;
        public Boolean IsProject = false;

        public string getProjectPath()
        {
            if (IsProject)
                return ModelFileName;
            else
                return null;
        }

        public string getRunConfigurationName()
        {
            if (IsProject && DataFileNames.Length == 1)
```

```
                return DataFileNames[0];
            else
                return null;
        }

        public CommandLine(string modFile, string datFile)
        {
            ModelFileName = modFile;
            DataFileNames = new String[1] { datFile };
        }
    }
```

Timer.cs

En el último archivo se puede encontrar la definición de la clase *Timer* que permite que el modelo sea ejecutado satisfactoriamente, en el tiempo correcto. Dicha clase es usada por el método *Run* (Program.cs) para mostrar el tiempo transcurrido por cada operación en la pantalla (utiliza los métodos *getTime* o *getAbsoluteTime*).

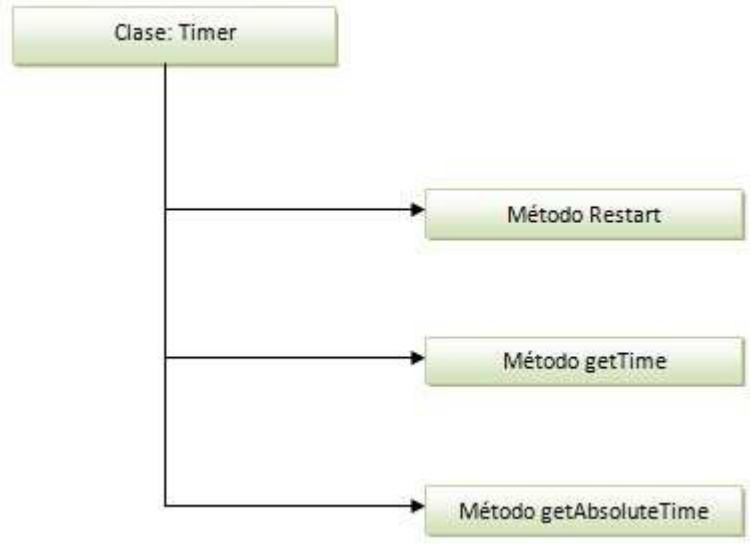


Figura 2.7: Diagrama de bloques de la clase Timer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

class Timer
    {
        public int _time = System.Environment.TickCount;
```

```
        int _startTime = System.Environment.TickCount;
        public Timer()
        {
        }
        public void restart()

// Este método reinicia el Timer

        {
            _time = System.Environment.TickCount;
        }
        public double getTime()

// Este método devuelve el tiempo entre el último reinicio y el presente

        {
            return (System.Environment.TickCount - _time) / 1000;
        }
        public double getAbsoluteTime()

*/
Este método devuelve el tiempo transcurrido desde el comienzo del programa
(ignorando los reinicios)
/*

        {
            return (System.Environment.TickCount - _startTime) / 1000;
        }
    }
```

Luego de desarrollar el código existe una opción en el C# que lo compila en un archivo ejecutable (.exe) creando de esta forma una aplicación de consola final para el usuario. Sin embargo, dicha compilación no puede ser ejecutada desde ninguna plataforma de Windows si uno no tiene instalado el “framework” correspondiente. Cuando uno ejecuta un programa compilado en C# es distinto que hacerlo en C++, pues en este último caso la compilación podría ejecutarse desde cualquier plataforma de Windows. C# utiliza un lenguaje intermedio, a diferencia de C++ que utiliza un lenguaje básico. La traducción de ese lenguaje intermedio lo realiza el “framework”. Como las reglas del concurso indican que los “runtimes” que estarán disponibles son los de .Net 2008, y esto equivale utilizar la versión 3.5 de “framework”, el programa fue compilado para dicha versión.

CAPÍTULO 3: ANÁLISIS DEL CÓDIGO DESARROLLADO

INCONVENIENTES SURGIDOS Y SUS SOLUCIONES

A lo largo del desarrollo de la solución existieron inconvenientes con el cumplimiento de las tareas, tanto por el lado del experto en optimización, como por el lado del ingeniero de software. Este apartado explica cuales fueron los problemas más relevantes en las tareas que realizó el ingeniero y como pudo solucionarlos.

El primer gran inconveniente está relacionado con la herramienta que programa la interfaz del algoritmo. Como ya se explicó, primero se eligió utilizar Java, pero al ver que surgían complicaciones cuya solución no era fácil de visualizar, se eligió cambiar de programa por Microsoft Visual C# 2010 Express. El primer inconveniente que surge es que no se sabía cómo ejecutar una solución creada en *Cplex* desde el C#. Programar la interfaz entre dos programas no es fácil y los conocimientos en la materia deben ser profundos. Como la resolución a través de ese camino no era factible por el poco tiempo que existía, se buscaron caminos alternativos. La solución fue buscar códigos ya programados en C#, en ejemplos del *Cplex*. Buscando en tutoriales se encontró un código que ayudaba a ejecutar soluciones del optimizador desde C#. Sin embargo, el problema no se pudo solucionar, ya que al ejecutarlo, surgía un error con la compatibilidad de las licencias de ambos software (este error puede surgir cuando alguna licencia no es original o cuando las versiones de los programas no pueden trabajar en conjunto). Pidiendo ayuda a IBM se consiguieron las licencias académicas correspondientes para utilizar el motor del optimizador sin inconvenientes. Por otro lado, se aprovechó el acuerdo que existe entre Microsoft y el ITBA para poder utilizar una versión original del Microsoft Visual C# 2010 Express. De esta forma se consiguió solucionar el primer inconveniente.

El segundo surge cuando se busca leer las variables de decisión, que genera el *Cplex*, desde C#. Dicho problema se debe en parte por falta de conocimientos y en parte a la falta de documentación que permita al alumno aprender a realizar la tarea. Se tenía la certeza de que existía la solución al problema, ya que en los tutoriales se mencionaba que hay clases de las bibliotecas del *Cplex* que contienen métodos que pueden realizar dicha función (sin aclarar cuáles). Para encontrar una solución simplemente se recurrió al método de prueba y error: se probó hasta llegar a una solución. Primero se identificó que con el método “GetElement” de la clase “OplModel” se podían leer las variables de

decisión, pero no se podía extraer la información. Entonces se probó con todos los métodos de dicha clase hasta encontrar el método “PrintInternalData”, con el cual se pudo extraer la información para guardarla en variables en el C#.

Este inconveniente surge cuando el experto en optimización requiere que con información de distintas variables de salida del primer archivo .mod (Initial.mod), el ingeniero cree un vector que debe ser incorporado al segundo archivo .mod (Solution.mod) para su correcto funcionamiento. Una vez que se identifican las variables, se realizan las operaciones correspondientes para calcular dicho vector. A continuación se muestra el último código desarrollado para poder cumplir con el objetivo:

```
for (int i = 0; i < etapas; i++)
                {
                 for (int j = 0; j < demandaEnArbol3.Length; j++)
                   {
                        float val = demandaEnArbol3[j][i];
                        if (val > etapasMax[i])
                           etapasMax[i] = val;
                   }
                }
                float[][] dea3Result = new float[demandaEnArbol3.Length][];
                for (int i = 0; i < demandaEnArbol3.Length; i++)
                {
                    dea3Result[i] = new float[etapas];
                    for (int j = 0; j < etapas; j++)
                        if (demandaEnArbol3[i][j] > etapasMax[j] * .1)
                            dea3Result[i][j] = 1;
                        else
                            dea3Result[i][j] = 0;
                }
```

La solución de este inconveniente se trabajó en conjunto con el experto en optimización, llegando a la conclusión de que la mejor opción es que éste defina la variable y la calcule dentro de su algoritmo y luego el ingeniero en software la extraiga utilizando el método correspondiente para poder manipularla desde C# (de esta forma, sólo trabaja con 1 variable en vez de varias como se planteó originalmente).

El tercer problema surge cuando se trata de lograr que el segundo archivo .mod lea del segundo archivo .dat el vector que se extrajo de la solución del primer archivo .mod. El

problema yacía en no poder identificar la sintaxis correcta para escribir dicho vector (un "array" bidimensional) en el archivo .dat, de forma que permita ser interpretado desde el archivo .mod. La solución que se pudo encontrar fue reemplazar directamente en la línea indicada en el segundo archivo .mod el vector extraído, ya que la sintaxis en dicho caso era conocida. Para lograrlo, se definió el método *ChangeSolutionModFile* en la clase *DatHandler.cs* que coloca el vector en la línea correcta del segundo archivo .mod para que éste se ejecute de forma correcta.

El último inconveniente consistió en cómo limitar la duración total del proceso a 10 minutos. El experto en optimización desarrolló dos archivos .mod con el objetivo de que el primero encuentre una pre-solución que facilite el trabajo al segundo archivo. La tarea del ingeniero en software consistió en medir el tiempo que le tomaba al primer archivo .mod llegar a una solución y definirle el tiempo máximo que tiene el segundo archivo .mod para encontrar una solución (definiendo una instancia de la clase *Timer*). Una vez calculado el tiempo, mediante el método *ChangeSolutionModFile* se reemplaza el valor calculado en la línea correspondiente del segundo archivo .mod, y de esta forma se limita la duración de su ejecución para encontrar una solución.

VERIFICACIÓN DE LA ROBUSTEZ DEL CÓDIGO

Una vez que se desarrolló el código es muy importante evaluar su robustez, ya que el producto final no será evaluado en el mismo entorno en el que fue creado. Es por esta razón que, una vez que se lo finalizó, se identificaron cuáles son las variables que, al someterlas a condiciones extremas, podrían causar que el software no funcionara. Los resultados fueron los siguientes:

- La cantidad de información que recibe como entrada desde los archivos .csv.
- Funcionalidad en distintos entornos (plataformas de Windows, velocidad de procesadores, memoria disponible, etc.).

En el primer caso se pretendía observar el comportamiento del código si los archivos de entrada eran muy grandes. Para ello, se programó un método en C# que permite agregarle materiales al conjunto de datos, de forma que, el resultado final sea el mismo que si dichos materiales no estuviesen pero, independientemente de ello, el programa invertirá memoria procesándolos. A continuación se muestra y se explica dicho código:

```
private static void AGREGARMATERIALES()
        {

*/
La primer parte del método busca el archivo PPS_MATERIAL.csv y lo abre. Como la cantidad de filas del archivo se encuentra directamente relacionada con la cantidad de materiales que tiene la muestra, le agrega 2000 materiales (filas) nuevos llamados XX-1, XX-2, etc., donde el resto de los argumento son 50 (Price), 0(Target), 0 (Holdingcost), 0 (Backordercost). De esta forma se garantiza que los nuevos materiales no afectarán el resultado.
/*

            int materialCount = 2000;

            TextReader reader = new StreamReader(@"C:\Users\Maxi\Desktop\KCC -
            Contest\Set de Datos con nombres cambiados\SetA\PPS_MATERIALS.csv");
            string s = "";

            try
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    if (line.Contains("XX"))
                        break;
                    s += line + "\r\n";
                }
            }
            finally
            {
                reader.Close();
            }

            TextWriter writer = new StreamWriter(@"C:\Users\Maxi\Desktop\KCC -
            Contest\Set de Datos con nombres cambiados\SetA\PPS_MATERIALS.csv");

            try
            {
                writer.Write(s);
                writer.Flush();

                for (int i = 1; i < materialCount; i++)
```

```
            {
                writer.Write("XX-" + i + ",50,0,0,0\r\n");
            }

        }
        finally
        {
            writer.Close();
        }
```

*/

La segunda parte del método busca el archivo PPS_MATERIALPROD.csv (el otro archivo .csv donde se encuentra el argumento *MATERIALS*), lo abre y le agrega la misma cantidad de materiales que al archivo anterior, asignándole los siguientes valores fijos: S1MAQ1 (*Machine*), 30 (*Prodrate*), 10 (*Prodcost*).

/*

```
        reader = new StreamReader(@"C:\Users\Maxi\Desktop\KCC - Contest\Set
        de Datos con nombres cambiados\SetA\PPS_MATERIALPROD.csv");
        s = "";

        try
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                if (line.Contains("XX"))
                    break;
                s += line + "\r\n";
            }
        }
        finally
        {
            reader.Close();
        }

        writer = new StreamWriter(@"C:\Users\Maxi\Desktop\KCC - Contest\Set
        de Datos con nombres cambiados\SetA\PPS_MATERIALPROD.csv");

        try
        {
            writer.Write(s);
```

```
                writer.Flush();

                for (int i = 1; i < materialCount; i++)
                {
                    writer.Write("S1MAQ1,\"XX-\"" + i + ",30,10\r\n");
                }

            }
            finally
            {
                writer.Close();
            }
        }
}
```

Los resultados fueron muy satisfactorios, ya que al agregarle más información a los archivos de entrada se comprueba que, independientemente del tamaño del conjunto de datos con el cual los organizadores evaluaren el código, éste no iba a generar errores en la ejecución del programa. Sin embargo, se pudo observar algo que era esperable: el tiempo de carga aumenta de forma considerable (de segundos a 1 minuto).

Para poder probarlo en distintos entornos, una vez compilado el código y generado el archivo ejecutable, se lo copió en cinco computadoras con características totalmente distintas. Según lo indicado por las reglas del concurso, el archivo va a ser ejecutado en una computadora con Windows Vista, 2GB de memoria RAM y con un procesador Intel Core 2 CPU 1.83GHz. Lamentablemente, no fue posible conseguir una máquina con las mismas características, es por esto que se lo probó en máquinas que tengan Windows XP, Windows 7 o Windows Vista, con dicho u otros procesadores, y con la misma cantidad de memoria RAM o más. Las conclusiones fueron las siguientes:

- Si cada máquina tenía instalada la versión de *ILOG Cplex* 12.2 y el "framework" 3.5, no existían problemas a la hora de ejecutar la solución.
- El tiempo programado para encontrar una solución (menor de 10 minutos) se cumplió en cada prueba realizada.
- Lo interesante fue observar que no siempre se llegaba a la misma solución a partir del mismo conjunto de datos originales. Investigando y haciendo pruebas, se llegó a la conclusión que las mejores soluciones se lograban cuando el procesador tenía mayor cantidad de "threads" (distintos caminos para resolver el

algoritmo en paralelo). Es por ello que al algoritmo se le impuso que utilice la máxima cantidad de “threads” disponible en la computadora donde se lo ejecute.

Una vez realizadas las pruebas y en vista a los resultados obtenidos, se concluyó que el código es lo suficientemente robusto como para que, llevando las variables investigadas a condiciones extremas, pueda ejecutarse y trabajar de forma correcta.

FORMATO DE ENTREGA

Una vez que se probó la robustez de la solución, el ingeniero en software debe encargarse de la compilación de toda la información que será entregada a los organizadores del concurso, en un archivo .zip, para su evaluación. Además de entregar un archivo ejecutable que se encargará de realizar todas las tareas sin intervención humana (leerá los archivos de entrada, buscará una solución y creará los archivos .csv de salida), los organizadores también requirieron que se entreguen los siguientes documentos:

1- Un resumen ejecutivo de dos hojas que describirá la solución lograda y comentará los resultados obtenidos. Es importante que este archivo se encuentre escrito de forma tal que personas, que no se encuentren familiarizadas con el proyecto, comprendan su contenido. Además como apéndice del mismo debe haber un gráfico de Gannt que muestre la solución conseguida en un ejemplo cargado por los organizadores del concurso.

2- Una descripción detallada (tres a ocho hojas) del modelo y de las técnicas que se utilizaron para resolver el problema.

El encargado de realizar ambos documentos fue el experto en optimización, ya que el contenido se encuentra directamente relacionado con su trabajo. Sin embargo, para ayudarlo, el ingeniero en software realizó el diagrama de Gannt correspondiente a un conjunto de datos de prueba, que se puede observar en el Anexo V.

En conclusión, los documentos que se encontraban dentro del archivo .zip fueron:

- *CSV Folder*: El programa lee la información de entrada desde esta carpeta. (aquí deben colocarse los nuevos conjuntos de datos).
- *Documents*: Dentro de esta carpeta se encontraban los archivos que los organizadores pidieron (resumen ejecutivo, Diagrama de Gannt y la descripción del modelo).
- *Solution*: Dentro de esta carpeta se crean los archivos .csv de salida con los nombres que la organización definió en las reglas del concurso.

- *Source Codes*: Dentro de esta carpeta se colocaron los archivos .cs, cada uno con una descripción breve de su contenido.
- *KCC_Solution.exe*: Este es el archivo compilado y el que ejecutará el programa.
- *Initial.mod* y *Solution.mod*: Son los dos archivos creados por el experto en optimización.
- *KCC_Solution_Config.txt:* En este archivo se le indica al código dónde se encuentran los archivos que necesita para trabajar (archivos .csv, *Initial.mod* y *Solution.mod*).
- Además se colocaron las bibliotecas del *Cplex* que necesitaba el código para funcionar correctamente.

CONCLUSIONES Y PROPUESTAS DE MEJORAS FUTURAS

Siempre existen formas de mejorar lo que se hizo, aunque el producto final haya funcionado y cumplido con los objetivos establecidos por los requisitos del concurso. A continuación se hará un análisis de cuáles son los procesos que se pueden mejorar según el ingeniero de software.

En primer lugar, al análisis de selección de la herramienta de optimización se le pueden realizar algunas mejoras, como realizar pruebas en los cuatro optimizadores posibles y a partir de los resultados decidir cuál es el mejor para buscar una solución al problema propuesto. Las razones por las cuales no se realizó lo dicho anteriormente fueron el tiempo que tenía el equipo para desarrollar y presentar una solución y los pocos conocimientos que el grupo poseía de cada uno de los optimizadores. Comenzar a crear algoritmos complejos que verdaderamente den una idea de cuán poderosa es cada herramienta conlleva una gran inversión de tiempo, que en ese momento no se podía realizar pero que, sin duda, ayudará a optimizar el producto final.

Por otro lado, relacionado al archivo que se entregó, puede comentarse lo siguiente:

1- Del lado operativo y de interfaz del producto final entregado a los organizadores, no hay que realizar modificaciones ya que se pudo cumplir satisfactoriamente con los objetivos requeridos por el concurso.
2- Ahora bien, si se observa la solución desde el punto de vista de su *performance*, hay que realizar las siguientes aclaraciones:
    a- Desde el punto de vista de las tareas desarrolladas por el ingeniero de software, estás pueden ser complicadas de programar y por esa razón se debe invertir mucho tiempo, pero el tiempo que le lleva al programa realizarlas es muy corto (del orden de los segundos). Es por esto que modificar lo que se realizó para mejorar la *performance* podría llevar muchísimo tiempo y los resultados no afectarán de forma considerable la *performance* final del producto.
    b- Desde el punto de vista de las tareas desarrolladas por el experto en optimización, la búsqueda de la solución al problema sí impacta directamente en la *performance* final, pero la solución a este tema excede los límites de este trabajo.

Lo que sí puede desarrollar el ingeniero en software es un producto que funcione como "solver" genérico de cualquier modelo de *Cplex*. Para poder lograrlo se deberá invertir tiempo, pero el producto al ser genérico y no estar limitado por restricciones puede ser utilizado en cualquier caso, lo que lo hace una herramienta mucho más poderosa. A continuación se explican las modificaciones que se le debe realizar al producto desarrollado para lograr uno genérico:

- Aplicación de Windows y no de consola.
- Flexibilizar el código:
    a- Crear una interfaz para leer archivos (desde .csv, base de datos, etc) y crear archivos .dat
    b- Crear una interfaz para cargar archivos .mod
    c- Crear una interfaz para parametrizar la ejecución del modelo con las opciones que ofrece el *Cplex*.
    d- Crear una interfaz para encadenar soluciones de varios modelos.
    e- Crear una interfaz para generar archivos de salida (a base de datos, .csv, crear un Diagrama de Gannt, etc.).

A modo de conclusión se realizó, por un lado, una autocrítica de lo que se debió haber hecho para estar absolutamente seguro de haber tomado las mejores decisiones (selección de la herramienta de optimización) y, por otro lado, se explicó cómo se puede lograr de este producto personalizado uno genérico para la industria. La primera es una mejora que se le puede realizar al software, que puede o no afectar la performance total, ya que al realizar todas las pruebas, el resultado podría haber sido la elección de *Cplex*. Sin embargo, haber realizado el análisis, indudablemente hubiese agregado valor al trabajo. La segunda es un cambio que se le realizaría al software para que cualquier usuario pueda utilizarlo. Sin embargo, es importante destacar que la *performance* del programa no mejorará con dicho cambio pero que, al utilizarlo más gente, pueden surgir nuevas ideas que puedan llevar a su mejora.

BIBLIOGRAFÍA

- http://es.wikipedia.org/wiki/Wikipedia:Portada. Consultado el 26 de mayo de 2011
- http://www.fico.com/en/Products/DMTools/xpress-overview/Pages/Xpress-Optimizer.aspx. Consultado el 26 de mayo de 2011
- http://www.lionhrtpub.com/orms/surveys/LP/LP-surveymain.html. Consultado el 26 de mayo de 2011
- http://www.maximal-usa.com/mplman/. Consultado el 26 de mayo de 2011
- http://www.opttek.com/Products/Documentation.html. Consultado el 26 de mayo de 2011
- Microsoft Developer Network: C# Tutorials. http://en.csharp-online.net/CSharp_Resources_-_MSDN. Consultado el 24 de mayo de 2011
- Optimization with ILOG OPL Development Studio, Student Workbook, ILOG, 2007.

ANEXOS

Anexo I: Reglas del Concurso

1. Cada equipo debe estar integrado como máximo por 3 estudiantes (graduados o no graduados) y un tutor de la misma universidad. Los alumnos deben ser alumnos “full time” en universidades de Latinoamérica en el momento que se registren y no podrán ser menores de edad ni empleados de KCC. Tanto los estudiantes como el tutor deben registrarse utilizando un mail de la universidad. De no ser posible, el alumno debe enviar un certificado de alumno regular a: Contest.Register@kcc.com.
2. Kimberly-Clark Corporation (KCC) tendrá los derechos exclusivos de todos los productos que sean entregados (sean ganadores o no). KCC podrá utilizar la solución entregada sin pagar ningún derecho de autor. También tendrá la libertad de publicar cualquier solución entregada. Los autores deberán ser autorizados por KCC para publicar temas relacionados con la herramienta.
3. El idioma oficial del concurso es ingles.
4. Las siguientes compañías proveerán el software, que se describe a continuación, con licencias académicas o de prueba:
    a) IBM proveerá a cada equipo participante con una licencia académica de *Cplex* 12.x, *CP Optimizer* 2.x license, *ODM 3.x OPL Development Studio* 6.x.
    b) Fair Isaac Corporation proveerá a cada equipo participante con una licencia académica o de prueba de *FICO Xpress Optimization* Suite 7.
    c) OptTek proveerá a cada equipo participante con una licencia académica o de prueba de *OptQuest.*
    d) Maximal Software proveerá a cada equipo participante con una licencia académica o de prueba de *MPL*.
5. El uso de cualquier otro “solver” de optimización sólo está permitido si es de uso libre o no comercial.

6. Los participantes pueden asumir que los “runtimes” de Java Virtual Machine versión 1.6 y .Net 2008 se encuentran disponibles.

7. Los participantes deben enviar su solución en un archivo .zip. Este archivo debe contener un archivo ejecutable o “batch”, que se encargará de ejecutar todos los programas necesarios sin intervención humana.

8. El archivo ejecutable será ejecutado desde una computadora con sistema operativo Windows *Vista* con un procesador Intel Core 2 CPU 1.83GHz, *2 GB* de memoria *RAM*, con nuevos conjuntos de datos (desconocidos para los participantes). Si la tarea no se completa en menos de 10 minutos, la entrega será descalificada.

9. El ganador será elegido por un jurado de ALIO y KCC LAO entre aquellas 10 entregas que puedan resolver cada instancia del problema con el mayor beneficio, basado en los resultados y la documentación entregada.

10. El equipo ganador recibirá US$ 3000 (menos tasas aplicables) y será invitado a la graduación de pasantes de KCC en Roswell, GA (USA) a presentar la solución en el mes de Agosto del año 2011. KCC pagará los pasajes de los alumnos (pero no del profesor) desde cualquier lugar de Latinoamérica a Atlanta.

11. Los ganadores son responsables de adquirir la visa de Estados Unidos por su cuenta y sus propios gastos.

12. Algunos participantes podrán ser invitados a formar parte del equipo SORT luego de su graduación.

Anexo II: Glosario [2]

Algoritmo

Es importante aclarar que cuando se utiliza la palabra algoritmo se hace referencia al producto que surge de la herramienta para optimización.

"Array"

Un vector, "array", arreglo o alineación es un conjunto o agrupación de variables del mismo tipo cuyo acceso se realiza por índices. Los vectores o "arrays" de dos o más dimensiones se denominan matrices, que pueden tener tantas dimensiones como se desee.

Biblioteca

En informática, una biblioteca es una colección o conjunto de subprogramas usados para desarrollar software. En general, las bibliotecas no son ejecutables, pero sí pueden ser usadas por ejecutables que las necesitan para poder funcionar correctamente. La mayoría de los sistemas operativos proveen bibliotecas que implementan la mayoría de los servicios del sistema. Dichas bibliotecas contienen comodidades que las aplicaciones modernas esperan que un sistema operativo provea.

Clases

En la programación orientada a objetos, una clase es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo. El modelo describe el estado y el comportamiento que todos los objetos de la clase comparten. Un objeto de una determinada clase se denomina una instancia de la clase. La clase que contiene (y se utilizó para crear) esa instancia se puede considerar como del tipo de ese objeto, por ejemplo, una instancia del objeto de la clase "Persona" sería del tipo "Persona".

Código

A diferencia del algoritmo, cuando se habla de código se hace referencia al producto de la herramienta para generar el software.

Entorno

Un entorno es un espacio o escenario informático en donde operan determinados comandos, funciones o características comunes.

"Framework"

El término "framework", para Microsoft .NET, se refiere a toda tecnología o modelo de programación que contiene máquinas virtuales, compiladores, bibliotecas de administración de recursos en tiempo de ejecución y especificaciones de lenguajes.

Interfaz

Conexión e interacción entre hardware, software y el usuario. El diseño y construcción de interfaces constituye una parte principal del trabajo de los ingenieros, programadores y consultores. Los usuarios "conversan" con el software. El software "conversa" con el hardware y otro software. El hardware "conversa" con otro hardware. Todo este "diálogo" no es más que el uso de interfaces. Las interfaces deben diseñarse, desarrollarse, probarse y rediseñarse; y con cada encarnación nace una nueva especificación que puede convertirse en un estándar más, de hecho o regulado.

Método

En la programación orientada a objetos, un método es una subrutina asociada exclusivamente a una clase (llamados métodos de clase o métodos estáticos) o a un objeto (llamados métodos de instancia). Un método consiste generalmente de una serie de sentencias para llevar a cabo una acción, un juego de parámetros de entrada que regularán dicha acción y, posiblemente, un valor de salida (o valor de retorno) de algún tipo.

"Namespace" (espacio de nombres)

Un "Namespace" es una forma de agrupar clases que se encuentran relacionadas de alguna forma.

"Runtime"

La definición en castellano del término es tiempo de corrida. Sin embargo, cuando se refiere a los "runtimes" de un software se refiere a que el software pueda ser ejecutado.

"Threads" (hilo de ejecución)

Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea. En sistemas operativos, un hilo de ejecución o subproceso es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.

[2] www.wikipedia.org

Anexo III: Reglas para obtener licencias de los productos

Este año existen cuatro sponsors que proveen programas de optimización. Las instrucciones para obtener cualquier de las cuatro licencias se describen a continuación:

1. **IBM - ILOG**

   IBM presenta una iniciativa académica donde miembros de la universidad e investigadores académicos profesionales pueden tener acceso a una versión completa del software. Por favor lea las instrucciones y realice el proceso de registración en:
   http://www-01.ibm.com/software/websphere/products/optimization/academic-initiative.
   Este proceso debe realizarse con ayuda del tutor. Por favor, incluya en su pedido que Ud. está participando del KCC/ALIO Optimization Contest.

2. **FICO - Xpress**

   FICO proveerá una licencia para usuario y una contraseña por grupo. Si su grupo se encuentra interesado en usar este software, un representante del mismo debe enviar un mail a: contest.register@kcc.com. Instrucciones y una licencia se le otorgarán luego de su pedido.

3. **OptTek - OptQuest**

   OptTek FICO proveerá una licencia para usuario y una contraseña por grupo. Si su grupo se encuentra interesado en usar este software, un representante del mismo debe enviar un mail a: contest.register@kcc.com. Instrucciones y una licencia se le otorgarán luego de su pedido.

4. **Maximal Software - MPL**

   Maximal Software presenta una iniciativa académica donde alumnos y profesores pueden tener acceso a una versión complete del software. Por favor lea las instrucciones y realice el proceso de registración en: http://www.maximal-usa.com/academic/ . Por favor, incluya en su pedido que Ud. está participando del KCC/ALIO Optimization Contest.

Anexo IV: Código programado en Java

```
//Define el nombre del archivo: pruebajava
package pruebaJava;

//Importa las bibliotecas necesarias con las clases para poder desarrollar el código
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.io.File;
import java.io.IOException;
import com.csvreader.CsvReader;

//Estas clases sirven para poder medir el tiempo de ejecución

import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

//Se define una clase nueva: DatGenerator

public class DatGenerator {

//Método privado que devuelve el tiempo actual (con una precisión en milisegundos)

        private String GetDateTime()
        {
           DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss.SSS");
           Date date = new Date();
                return dateFormat.format(date);
        }

//Método que dice qué archivos hay en una carpeta y los devuelve en formato “array”

        private File[] GetFilesFromFolder(String folderPath)
        {
                File dir = new File(folderPath);
                return dir.listFiles();
        }

//Método que toma un archivo .csv y lo prepara para ponerlo en algún lugar (.dat)

        private void PrintPpsString(PrintWriter out, String filePath)
```

```
{
        try {
                BufferedReader in = new BufferedReader(new
                FileReader(filePath));

                boolean CAMBIARCOSA = false;
                if(filePath.contains("ppsSetup"))
                        CAMBIARCOSA = true;

                String str;

                if(in.readLine() != null)
              {
```

*/

Toma cada fila del archivo .csv con el comando “while” y crea una lista de “strings” y, cada vez que hay una coma, asigna dato entre comas a un “array”

/*

```
                        while ((str = in.readLine()) != null) {
                                String[] items = str.split(",");

                                String currentLine = "";

                                currentLine += "<";


                                for(int i = 0; i < items.length; i++)
                                {
                                        String item = "";
```

*/

Primero se sacan todas las comillas (saca de formato “string”) para pasar todo a formato numérico y si algo no es un número y es un “string”, genera una excepción que atrapa la función “catch” y le agrega las comillas

/*

```
try
        {
        item = items[i].trim().replace("\"","");
        float f = Float.valueOf(item).floatValue();
        if(CAMBIARCOSA)
                {
                if(item.contains("."))
                        {
                        item = item.substring(0, item.indexOf("."));
                            int num = Integer.valueOf(item).intValue() + 1;
                                item = Integer.toString(num);
                        }
                }
          }
```

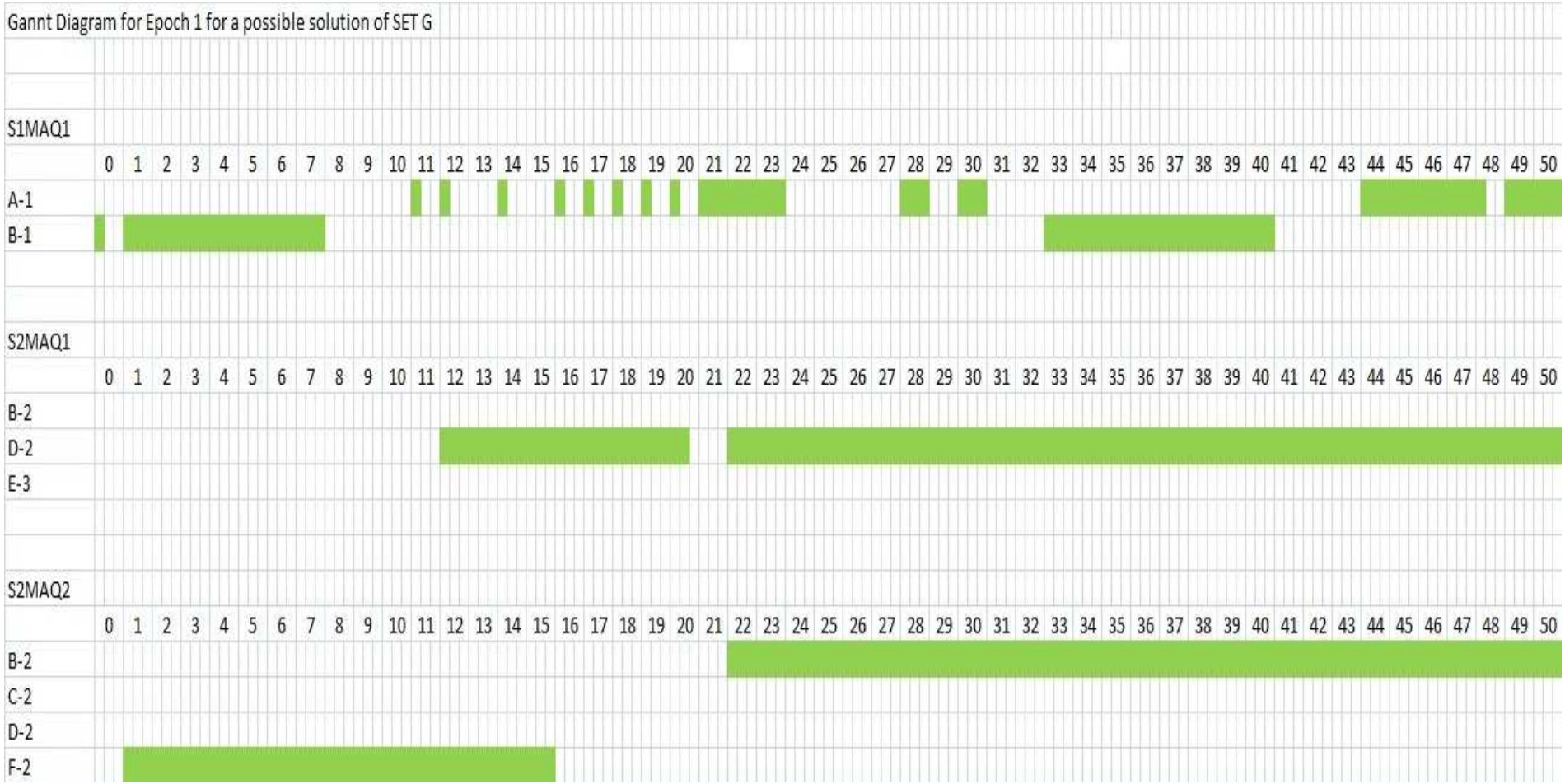
```
catch (NumberFormatException nfe)
   {
        item = "\"" + item + "\"";
   }
finally
  {
        currentLine += item + " ";
  }

*/
```

Si no encuentra el archivo .csv dará un error y lanzará un mensaje de “Archivo no encontrado”

```
/*

                catch (FileNotFoundException e)
                {
                        out.println("***** Archivo no encontrado ******");
                } catch (IOException e)
                {
                        out.println("Error:" + e.getMessage());
                }


//Crea el archivo .dat y calcula el tiempo total del proceso

        public String GenerateDatFile(String datFolderPath, String ppsFilesPath)
        {
                try {

                        String startTime = GetDateTime();

                        File[] ppsFiles = GetFilesFromFolder(ppsFilesPath);

                        FileWriter outFile = new FileWriter(datFolderPath);
                        PrintWriter out = new PrintWriter(outFile);

                        for (int i = 0; i < ppsFiles.length; i++)
                        {
                            out.println(ppsFiles[i].getName().replace(".csv","") + " = {");
                                PrintPpsString(out, ppsFiles[i].getAbsolutePath());

                                out.println("};");
                        }

                        out.flush();
                        out.close();
                        outFile.close();

                        return startTime + " -" + GetDateTime();
```

```
                } catch (IOException e){
                        e.printStackTrace();
                }

                return "";
        }
}
```

Anexo V: Diagrama de Gannt



Nota: Sólo se muestran los primeros 50 días (de los 215 totales) como ejemplo.