UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DEPARTAMENTO DE ORGANIZACIÓN INDUSTRIAL

MODELOS MATEMÁTICOS DE OPTIMIZACIÓN

Pedro Linares
Andrés Ramos
Pedro Sánchez
Ángel Sarabia
Begoña Vitoriano

Octubre 2001

Alberto Aguilera 23 – E 28015 Madrid – Tel: 34 91 542 2800 – Fax: 34 91 541 1132 – www.doi.icai.upco.es

ÍNDICE

I.1 Optimización

I.1.1 Investigación operativa y optimización

> "In the last decade, new advances in algorithms have been as important as the impressive advances in computer technology" George L. Nemhauser (1994).

> "The technology improvements in algorithms, modeling languages, software, and hardware have made the methodology accessible, easy to use, and fast. So the Age of Optimization has arrived" George L. Nemhauser (1994).

Definir el término investigación operativa no es una tarea fácil ya que su evolución permanente hace que sea difícil dar con precisión una definición. La investigación operativa se puede definir como la aplicación de métodos científicos en la mejora de la efectividad en las operaciones, decisiones y gestión, ver [Robinson, 1999]. Otra definición más extensa es la siguiente: la investigación operativa es la aplicación, por grupos interdisciplinarios, del método científico a los problemas complejos producidos en la dirección y gestión de grandes sistemas de hombres, máquinas, etc. La principal característica consiste en construir un modelo científico del sistema del cual se pueden predecir y comparar los resultados de diversas estrategias, decisiones, incorporando medidas del azar y del riesgo. El objetivo es ayudar a los responsables a determinar su política y actuaciones en forma científica.

Los profesionales de la investigación operativa colaboran con los decisores en el diseño y mejora de las operaciones y decisiones, resuelven problemas y ayudan en las funciones de gestión, planificación o predicción, aportan conocimiento y ayuda en la toma de decisiones. Aplican las técnicas científicas más adecuadas seleccionadas de la matemática, ingeniería o cualquier ciencia social o de administración de empresas. Su trabajo normalmente consiste en recoger y analizar datos, desarrollar y probar modelos matemáticos, proponer soluciones o recomendaciones, interpretar la información y, en definitiva, ayudar a implantar acciones de mejora. Como resultado desarrollan e implantan aplicaciones informáticas, sistemas, servicios técnicos o productos.

La investigación operativa tiene sus orígenes en la Segunda Guerra Mundial, debido a la necesidad urgente de asignación de recursos escasos en las operaciones militares, en problemas tácticos y estratégicos. Estas mismas técnicas se han extendido con posterioridad a las empresas.

La optimización es una parte relevante dentro de la investigación operativa. Tuvo un progreso algorítmico inicial muy rápido. Muchas técnicas –programación lineal (*linear programming*) LP, programación dinámica (*dynamic programming*) DP– son anteriores a 1960. Por ejemplo, el método Simplex[1] de programación lineal debido a Dantzig es de 1947, el principio de optimalidad de Bellman base de la programación dinámica se formuló en 1957. En la última década se han producido avances significativos generados por el desarrollo en 1984 por parte de Karmarkar de un método de punto interior para programación lineal. Por ejemplo, en una nota técnica de ILOG se presenta que desde el optimizador CPLEX 3.0 en 1994 a CPLEX 7.0 en 2000 la reducción de tiempo de resolución ha sido de 28 veces en el método simplex dual para un problema concreto de LP. Para otro caso se observa una mejora global, de software y algorítmica, de 10000 veces entre la versión de CPLEX 1.0 de 1988 y la 7.0 del 2000. Como referencia, se estima que la mejora en el rendimiento del hardware ha sido del mismo orden de magnitud. Si tomamos conjuntamente ambas mejoras hoy se pueden resolver problemas en segundos que habrían tardado años en ser resueltos hace una docena de años. Estos avances han sido tan importantes como los realizados en el campo de la informática, según la opinión de George L. Nemhauser uno de los expertos actuales en programación entera, y se han producido acompasadamente con ellos. Hoy es posible resolver un problema LP de 150000 ecuaciones con 150000 variables y 1000000 de elementos no nulos en la matriz de restricciones en un PC con suficiente memoria principal.

El estilo de este documento es eminentemente aplicado, práctico, ingenieril, a caballo entre una visión matemática de los problemas y de los algoritmos y la visión económica o de gestión empresarial de algunas de sus aplicaciones. Este documento trata de explicar suficientemente los fundamentos matemáticos como para permitir desarrollar aplicaciones de optimización de manera rigurosa y precisa. Al mismo tiempo, se presentan algunas aplicaciones a problemas concretos de ingeniería.

Al final del capítulo se citan algunos libros generales o de referencia de investigación operativa que pueden servir de consulta o como texto para un nivel de pregrado y postgrado. Luego, en cada capítulo se indican además referencias específicas de los diferentes temas. Dentro de los libros generales, [Hillier y Lieberman, 1997] es un libro clásico de investigación operativa muy ampliamente utilizado que compendia numerosos temas y tiene una orientación ingenieril. [Taha, 1998] presenta los temas con una orientación más matemática

[1] En castellano la traducción de esta palabra es símplice pero no es habitual su uso para denominar este método de optimización lineal.

mientras que [Winston, 1994] los presenta con una perspectiva más de administración de empresas. [Sarabia, 1996] da una base teórica suficiente para poder resolver una colección de problemas relacionados con el temario de investigación operativa.

Entre las revistas principales que tratan sobre optimización se pueden incluir: *Interfaces*, *Operations Research*, *European Journal of Operational Research*, *Mathematics of Operations Research*, *OR/MS Today*, *Mathematical Programming*, *INFORMS Journal on Computing, Journal of the Operational Research Society, Journal of Optimization Theory and Applications*. Existe una enciclopedia de investigación operativa que puede servir como consulta inicial y referencia de un tema específico, ver [Gass, 2001].

La optimización consiste en la selección de una alternativa mejor, en algún sentido, que las demás alternativas posibles. Es un concepto inherente a toda la investigación operativa, sin embargo, determinadas técnicas propias de la investigación operativa se recogen bajo el nombre de optimización o programación matemática. Además de las técnicas de optimización, la investigación operativa incluye también otras como teoría de la decisión, de juegos, de grafos o de colas que no se presentan en este documento.

Los problemas de optimización se componen generalmente de estos tres ingredientes:

- *función objetivo*
  Es la medida cuantitativa del funcionamiento del sistema que se desea optimizar (maximizar o minimizar). Como ejemplo de funciones objetivo se pueden mencionar: la minimización de los costes variables de operación de un sistema eléctrico, la maximización de los beneficios netos de venta de ciertos productos, la minimización del cuadrado de las desviaciones con respecto a unos valores observados, la minimización del material utilizado en la fabricación de un producto, etc.

- *variables*
  Representan las decisiones que se pueden tomar para afectar el valor de la función objetivo. Desde un punto de vista funcional se pueden clasificar en variables *independientes* o *principales* o *de control* y variables *dependientes* o *auxiliares* o *de estado*, aunque matemáticamente todas son iguales. En el caso de un sistema eléctrico serán los valores de producción de los grupos de generación o los flujos por las líneas. En el caso de la venta, la cantidad de cada producto fabricado y vendido. En el caso de la fabricación de un producto, sus dimensiones físicas.

- *restricciones*
  Representan el conjunto de relaciones (expresadas mediante ecuaciones e inecuaciones) que ciertas variables están obligadas a satisfacer. Por ejemplo, las potencias máxima y mínima de operación de un grupo de generación, la capacidad de producción de la fábrica para los diferentes productos, las dimensiones del material bruto del producto, etc.

Resolver un problema de optimización consiste en encontrar el valor que deben tomar las variables para hacer óptima la función objetivo satisfaciendo el conjunto de restricciones.

Existen algunos tipos de problemas de optimización que alteran ligeramente este esquema:

- *sistemas de ecuaciones lineales – no lineales*
  No existe una función objetivo como tal. Únicamente interesa encontrar una solución factible a un problema con un conjunto de restricciones.

- *optimización sin restricciones*
  Se trata de encontrar el conjunto de valores de las variables que determinan el mínimo/máximo de una función. Algunas de las técnicas que se verán en programación no lineal son para optimización sin restricciones.

- *optimización multiobjetivo*
  Existe más de una función objetivo. El problema que se plantea es cómo tratar varias funciones objetivo a la vez, teniendo en cuenta que el óptimo para un objetivo no lo es para otro, son objetivos en conflicto entre sí.

Los métodos de optimización los podemos clasificar en: métodos *clásicos* (que son los que habitualmente se explican en los libros de optimización) y métodos *metaheurísticos* (que aparecieron ligados a lo que se denominó inteligencia artificial). Dentro de los primeros se encuentra la optimización lineal, lineal entera mixta, no lineal, estocástica, dinámica, etc. que se explican en el documento. En el segundo grupo se incluyen los algoritmos evolutivos (genéticos entre otros), el método del recocido simulado (*simulated annealing*) o las búsquedas heurísticas (método tabú, búsqueda aleatoria, etc.). De forma muy general y aproximada se puede decir que los métodos clásicos buscan y garantizan un óptimo local mientras que los métodos metaheurísticos tienen mecanismos específicos para alcanzar un óptimo global aunque no garantizan su alcance.

En la siguiente tabla se muestran las expresiones matemáticas generales de algunos tipos de problemas de optimización dentro de los métodos clásicos. Los problemas se distinguen por el carácter de las funciones que intervienen (lineales o no lineales) y de las variables (reales/continuas o enteras/discretas).

Programación lineal (*linear programming*) LP	$\min_x c^T x$ $Ax = b$ $x \geq 0$ $x \in \mathbb{R}^n, c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$
Programación lineal entera mixta (*mixed integer programming*) MIP	$\min_x c^T x + d^T y$ $Ax + By = b$ $x, y \geq 0$ $x \in \mathbb{Z}^n, y \in \mathbb{R}^l, c \in \mathbb{R}^n, d \in \mathbb{R}^l$ $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times l}, b \in \mathbb{R}^m$
Programación cuadrática (*quadratic programming*) QP	$\min_x c^T x + \frac{1}{2} x^T Q x$ $Ax = b$ $x \geq 0$ $x \in \mathbb{R}^n, c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}$ $Q \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^m$
Programación no lineal (*non linear programming*) NLP	$\min_x f(x)$ $g(x) = 0$ $h(x) \leq 0$ $l \leq x \leq u$ $f : \mathbb{R}^n \rightarrow \mathbb{R}$ $g, h : \mathbb{R}^n \rightarrow \mathbb{R}^m$
Programación multiobjetivo (*multiobjective programming*)	$\min_x (f_1(x), ..., f_k(x))$ $Ax = b$ $x \geq 0$ $x \in \mathbb{R}^n, c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$ $f_i(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

Existen decisiones que no pueden ser representadas de forma adecuada mediante variables continuas. Por ejemplo, las decisiones de inversión son variables discretas (por ejemplo, planificación de la expansión de la generación o de la red, adquisición de equipos singulares) o binarias (como localización de plantas o almacenes). Estos problemas se denominan, genéricamente, de programación lineal entera mixta, son problemas *lineales* donde algunas o todas las variables son enteras. Los problemas lineales con variables enteras se pueden clasificar en: PIP (*pure integer programming)* si todas las variables son enteras, BIP (*binary integer programming)* si todas son binarias o MIP (*mixed integer programming*) si algunas son enteras o binarias y el resto continuas.

Un caso particular, pero muy frecuente, de variables *enteras* son las variables *binarias* (0/1), ya que permiten modelar condiciones de asignación o condiciones lógicas. Por otra parte, toda variable entera x se puede expresar como suma de variables binarias y_i, donde $x = \sum_{i=0}^{N} 2^i y_i$ siendo u una cota superior de x, $0 \le x \le u$, y estando u comprendida en el intervalo $2^N \le u \le 2^{N+1}$.

La formulación matemática de algunos problemas de optimización especiales por no incluir alguno de los componentes se presenta en la siguiente tabla.

Optimización no lineal sin restricciones	$\min_x f(x)$ $f : \mathbb{R}^n \to \mathbb{R}$
Ajuste no lineal mínimo cuadrático	$\min_x f(x) = \sum_{i=1}^{N} F_i^2(x)$ $F_i : \mathbb{R}^n \to \mathbb{R}$
Problema mixto complementario (*mixed complementarity problem*) MCP	$xF(x) = 0$ $x \in \mathbb{R}^n$ $F : \mathbb{R}^n \to \mathbb{R}^n$

I.1.2 Referencias

Gass, S.L. and Harris, C.M. (eds.) (2001) Encyclopedia of Operations Research and Management Science. Centennial Edition. Kluwer Academic Publishers.

Hillier, F.S., Lieberman, G.J. (1997) Introducción a la Investigación de Operaciones. McGraw Hill.

Robinson R. (1999) "Welcome to OR Territory" OR/MS Today pp. 40-43 August.

Sarabia, A. (1996) La Investigación Operativa. UPCO.

Taha, H.A. (1998) Investigación de operaciones. Una introducción. Prentice Hall.

Winston, W.L. (1994) Investigación de Operaciones. Aplicaciones y Algoritmos. Grupo Editorial Iberoamericana.

I.2 Modelos de optimización

I.2.1 Modelo y modelado

> **Modelo**. Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja (por ejemplo, la evolución económica de un país), que se elabora para facilitar su comprensión y el estudio de su comportamiento. DICCIONARIO DE LA LENGUA ESPAÑOLA. REAL ACADEMIA ESPAÑOLA.

Un modelo es una representación matemática simplificada de una realidad compleja. Modelar es la acción de construir un modelo, de encorsetar la realidad. Implica la relación entre dos figuras (no necesariamente encarnadas por personas únicas sino por equipos): el *modelador* (encargado de la especificación y desarrollo del modelo) y el *experto* sobre la realidad (conocedor del problema real). La mayoría de las veces, el desarrollo de un modelo puede involucrar a un equipo multidisciplinar compuesto por matemáticos, estadísticos, ingenieros, economistas, psicólogos, etc. que aportan diferentes perspectivas y conocimiento en la representación de la realidad. Un modelo debe equilibrar la necesidad de contemplar todos los detalles con la factibilidad de encontrar técnicas de solución adecuadas.

Un modelo es, en definitiva, una herramienta de ayuda a la toma de decisiones. Por esta razón, sus resultados deben ser inteligibles y útiles. Modelar se puede entender simultáneamente como *ciencia* y como *arte*. Es una ciencia pues se basa en un conjunto de procesos estructurados: análisis y detección de las relaciones entre los datos, establecimiento de suposiciones y aproximaciones en la representación de los problemas, desarrollo o uso de algoritmos específicos de solución. Es un arte porque materializa una visión o interpretación de la realidad no siempre de manera unívoca. Cada persona imprime su estilo en el modelo mismo y en la especificación, en el desarrollo y en la documentación. Características tales como elegancia o simplicidad pueden atribuirse a un modelo. El desarrollo de un modelo es una creación hecha con ayuda de ciencias básicas o herramientas de apoyo.

Entre los beneficios explícitos o implícitos, tanto para el modelador como para el experto, derivados del proceso de modelado además del modelo en sí mismo, se pueden mencionar:

- Ayuda a establecer un diálogo con intercambio de información entre el modelador y el experto

- Organiza los datos, la información disponible sobre el sistema
- Organiza, estructura y mejora la comprensión del sistema
- Internaliza la estructura organizativa de la empresa
- Permite compartir supuestos y resultados entre el modelador y el experto
- Proporciona un entorno ágil para el análisis y la sensibilidad
- Indica la dirección de mejora en las decisiones

En este capítulo se tratará exclusivamente de modelos de optimización, es decir, aquéllos donde existe un conjunto de *variables* de decisión que deben maximizar/minimizar una *función objetivo* sometidas a un conjunto de *restricciones*. Los modelos de programación lineal son más utilizados que todos los otros tipos de optimización juntos y abarcan cualquier tipo de actividad humana como micro y macroeconomía, finanzas, marketing, economía de la energía, organización de la producción, planificación de la operación, selección de procesos, asignación de tareas, ingeniería química, forestal, agrónoma, comercio internacional, desarrollo económico, etc. Como referencias generales de modelado de problemas de optimización que se pueden utilizar en la enseñanza de pregrado o postgrado cabe citar a [Schrage, 1997] y [Williams, 1999].

I.2.2 Etapas en el desarrollo de un modelo

Las etapas que componen el *ciclo de vida* de un modelo son las siguientes:

IDENTIFICACIÓN DEL PROBLEMA

Consiste en la recolección y análisis de la información relevante para el problema, en el intercambio de información entre el modelador y el experto, en establecer una relación simbiótica y una estrecha coordinación entre ambos.

Los problemas reales suelen estar definidos en términos vagos e imprecisos. Se debe hacer la tarea de traducción o interpretación en frases precisas, convertibles en ecuaciones matemáticas. En esta etapa se establecen y documentan los supuestos realizados que en etapas posteriores deberán ser validados.

Esta etapa es fundamental para que las soluciones proporcionadas, las conclusiones obtenidas sean útiles, las decisiones adoptadas sean correctas. Los datos suelen ser vitales para conseguir un realismo o aplicabilidad en las soluciones. A menudo representan el cuello de botella del proceso de modelado.

ESPECIFICACIÓN MATEMÁTICA Y FORMULACIÓN

Escritura matemática del problema de optimización, definiendo sus variables, sus ecuaciones, su función objetivo, sus parámetros. En esta etapa se analiza el

tamaño del problema, la estructura de la matriz de restricciones, su tipo (LP, MIP, NLP). Es una etapa de creación donde se debe prestar especial atención a la precisión en la formulación y a la escritura de las ecuaciones que describen el problema. Hay que tener en cuenta, además, que existen diversas alternativas de modelado (especialmente en programación entera) que afectan de manera fundamental en la resolución del mismo, existiendo un desarrollo cada vez mayor en la reformulación de problemas.

La caracterización de un problema LP según su tamaño resulta difícil y ha sufrido un gran cambio desde los recientes desarrollos de algoritmos simplex mejorados y, sobre todo, desde la aparición de los métodos de punto interior. En la tabla 1.1 se propone una clasificación de tipos de problemas LP según su tamaño. Esta clasificación debe ser tomada como guía o referencia relativa actual pero téngase en cuenta que los tamaños relativos de los problemas cambiarán conforme evolucionen los códigos de optimización. Actualmente se puede afirmar que los códigos de optimización lineal implantan algoritmos muy eficientes, son fiables y numéricamente robustos y están ampliamente disponibles.

	Restricciones	Variables
Caso ejemplo	100	100
Tamaño medio	10000	10000
Gran tamaño	100000	100000
Muy gran tamaño	$>$ 100000	$>$ 100000

Tabla 1.1 Tipos de problemas LP según su tamaño.

En lo referente a MIP o NLP ni siquiera se pueden dar criterios generales de tamaño ya que la dificultad de resolución no tiene por qué estar ligada al tamaño del problema, siendo incluso preferible reformular un problema aunque aumenten las dimensiones, para lograr una resolución más eficiente.

RESOLUCIÓN

Se trata de implantar un algoritmo de obtención de la solución numérica (muy próxima a la matemática) óptima o cuasióptima. El algoritmo puede ser de propósito general (método simplex) o específico. Puede haber diferentes métodos de solución de un problema o diferentes implantaciones de un mismo método. El tiempo de resolución de un problema también puede depender drásticamente de cómo esté formulado.

La solución óptima debe ser suficientemente satisfactoria, debe ser una guía de actuación para el experto.

VERIFICACIÓN, VALIDACIÓN Y REFINAMIENTO

Esta etapa conlleva la eliminación de los errores en la codificación, es decir, conseguir que el modelo haga lo que se desea (depurar y verificar). Es necesario comprobar la validez de las simplificaciones realizadas a través de los resultados obtenidos, incluso contrastando éstos con situaciones reales ya transcurridas (validar).

Esta etapa de verificación, validación, comprobación da lugar a nuevas necesidades de modelado para mejorar la capacidad de representación de la realidad, a nuevos refinamientos indicados por el usuario.

INTERPRETACIÓN Y ANÁLISIS DE LOS RESULTADOS

Esta etapa consiste en proponer soluciones. Permite conocer en detalle el comportamiento del modelo al hacer un análisis de sensibilidad en los parámetros de entrada, estudiar diferentes escenarios plausibles de los parámetros, detectar soluciones alternativas cuasióptimas pero suficientemente atractivas, comprobar la robustez de la solución óptima.

IMPLANTACIÓN, DOCUMENTACIÓN Y MANTENIMIENTO

Ésta es una etapa fundamental del desarrollo de un modelo para garantizar su amplia difusión. La documentación ha de ser clara, precisa y completa. El manual de usuario debe incluir la especificación técnica funcional, matemática e informática. El propio código debe incluir una buena documentación para facilitar la tarea del mantenimiento. Piénsese que la mayor parte del ciclo de vida de un modelo no está en el desarrollo sino en la fase de uso y mantenimiento.

En esta etapa se incluye también la tarea de formación para los usuarios del modelo.

I.2.3 Referencias

Schrage, L. (1997) Optimization Modeling with LINDO. Duxbury Press.

Williams, H.P. (1999) Model Building in Mathematical Programming. 4th Edition. John Wiley and Sons.

I.3 Codificación de problemas de optimización

I.3.1 Lenguajes de modelado

I.3.1.1 Lenguajes de modelado

Las principales alternativas actuales para el desarrollo de modelos de optimización suelen ser, Sharda (1995):

- *Lenguajes de programación de propósito general* (C, C++, Java, Visual Basic, FORTRAN 90) que llaman a una biblioteca de optimización
  Tienen sentido cuando el tiempo de solución es crítico o el modelo es ejecutado con mucha frecuencia o cuando se necesitan interfaces a medida para la entrada de datos o salida de resultados o cuando el modelo tiene que ser integrado en otra aplicación o se necesitan algoritmos de optimización específicos. Además permiten la implantación del modelo en un entorno software o hardware especial. Como contrapartida requiere un tiempo de desarrollo muy elevado y, sobre todo, presenta una gran dificultad y consumo de recursos para el mantenimiento del código.

- *Lenguajes o entornos de cálculo numérico o simbólico* (hojas de cálculo, lenguajes para cálculo numérico intensivo, como MATLAB, o para cálculo simbólico, como Maple o Mathematica, etc.)
  Los optimizadores de las hojas de cálculo, por ser aplicaciones muy comunes y conocidas, pueden ser un vehículo eficaz de difusión de un modelo entre cierto tipo de usuarios y facilitan el manejo de datos que se encuentren ya en dicho formato [Ragsdale, 1998]. Como ventajas específicas se pueden mencionar: su facilidad de uso, su integración total con la hoja de cálculo, la familiaridad con el entorno que facilita la explicación del modelo y de sus resultados, así como la facilidad de presentación de resultados en gráficos. Sin embargo, no inducen una buena práctica de programación, presentan la dificultad de su desarrollo, verificación, validación, actualización, documentación y, en general, el mantenimiento del modelo y no permiten modelar problemas complejos o de gran tamaño [Gass, 1995].
  Los lenguajes de cálculo numérico o simbólico no son específicos de problemas de optimización pero facilitan la manipulación numérica o simbólica de matrices y vectores. También disponen de funciones de optimización.

Todas estas alternativas pueden ser utilizadas para desarrollo rápido de un prototipo o una demostración ya que presentan capacidades de presentación gráfica que pueden ser aprovechadas. Son difícilmente utilizables cuando se plantean problemas de optimización de tamaño medio o superior.

- *Lenguajes algebraicos de modelado*
  Son las alternativas más complejas y potentes por su capacidad de indexación de las variables y ecuaciones, permiten cambiar sin dificultad las dimensiones del modelo, de forma natural separan datos de resultados. Desde el punto de vista del modelador permiten la detección de errores de consistencia en la definición y verificación del modelo. Desde el punto de vista del usuario simplifican drásticamente su mantenimiento. Entre los lenguajes de modelado más conocidos se pueden mencionar: GAMS (www.gams.com) y AMPL (www.ampl.com) de origen estadounidense y AIMMS (www.aimms.com) y XPRESS-MP (www.dash.co.uk) de origen europeo, por citar algunos.
  Existe una herramienta integrada denominada OPLStudio (www.ilog.com), en la que se dispone de un lenguaje de modelado (OPL) y varios optimizadores dependiendo del modelo propuesto. Está especialmente desarrollada para problemas de programación (*scheduling*) y planificación, aunque admite también cualquier modelo de optimización lineal y lineal entera mixta. Es una herramienta integrada ya que además del lenguaje de modelado, incluye sus propios optimizadores, Scheduler, Planner, Solver, CPLEX, estando los tres primeros basados en la programación de restricciones[2] y el último en programación matemática.
  GAMS es el lenguaje más ampliamente difundido comercialmente con su propia lista de discusión de usuarios (gams-l@listserv.gmd.de) mientras que AMPL se está potenciando mucho en las universidades estadounidenses. Existe un proyecto denominado NEOS (www.neos.mcs.anl.gov) para el cálculo distribuido que permite el envío de problemas de optimización escritos en AMPL o GAMS a través de internet y éstos son resueltos en servidores de la red devolviendo los resultados de la optimización.
  Existen libros específicos que describen sus características y que sirven como guías de usuario tanto para el lenguaje GAMS [Brooke, 1998], [McCarl, 1998], para AMPL, [Fourer, 2000], o para OPL [Van Hentenryck, 1999]. Incluso en España se ha publicado un libro de optimización que se apoya en GAMS para la presentación de ejemplos [Mocholí, 1996]. Los campos de

[2] Se denomina programación de restricciones a un tipo de programación lógica donde el dominio de las variables viene definido por relaciones lógicas y por restricciones.

aplicación de estos lenguajes son tan amplios como los de la optimización propiamente dicha. Abarcan desde la micro y macroeconomía, a la economía de la energía, a la planificación energética o eléctrica, a la ingeniería química o forestal, a la planificación del desarrollo económico o del comercio internacional o a la cobertura de riesgos financieros. En el caso de la programación de restricciones ésta aparece especialmente en problemas combinatorios para modelar restricciones lógicas.

I.3.1.2 Lenguajes algebraicos de modelado

Los lenguajes algebraicos son lenguajes de alto nivel que han sido diseñados específicamente para el desarrollo e implantación de modelos de optimización de forma más directa para los programadores y más inteligible para los usuarios. En consecuencia, el campo de actuación y utilidad de los modelos de optimización se ha ampliado tremendamente al utilizar estos lenguajes. Entre sus *características* y *ventajas* principales destacan las siguientes:

- Proporcionan una formulación sencilla de modelos grandes y complejos
- Facilitan sobremanera el desarrollo de prototipos
- Mejoran sustancialmente la productividad de los modeladores al permitir dedicar más tiempo al diseño, ejecución del modelo y análisis de los resultados y menos a la codificación del mismo
- Estructuran los buenos hábitos de modelado al exigir una representación concisa y exacta de los parámetros/variables y sus relaciones
- Recogen simultáneamente la estructura del modelo y su documentación
- Separan de manera natural los datos de la estructura del modelo y ésta de los algoritmos de solución
- La formulación del problema es independiente del tamaño. Permiten el uso de la estructura del modelo para diferentes casos[3]
- Los optimizadores pueden ser intercambiados sin dificultad, se pueden probar nuevos optimizadores, nuevos métodos o nuevas versiones
- Por ejemplo, en el lenguaje GAMS se encuentran entre otros disponibles los optimizadores CPLEX, OSL, XA y XPRESS para problemas LP y MIP, MINOS y CONOPT para problemas NLP, DICOPT para problemas MINLP y MILES y PATH para problemas MCP.
- Permiten la realización de cambios en el modelo de manera sencilla y segura, es decir, se puede afrontar un refinamiento continuo en la formulación del problema

[3] Una manera habitual de desarrollar es utilizar una maqueta (caso ejemplo) para la depuración y verificación del modelo y una vez comprobada su validez utilizar el caso real a ser resuelto.

- Cualquier tipo de problemas de programación lineal, no lineal, flujos en redes o mixta complementaria resulta muy fácil implantar su formulación
- Permiten la implantación de algoritmos avanzados, que incluyan varias llamadas al optimizador o procedimientos específicos para el problema (como por ejemplo los métodos de descomposición)
- Permiten la portabilidad de los modelos entre plataformas y sistemas operativos

Como *desventajas* principales se pueden mencionar las siguientes:

- No son adecuados para la resolución de problemas de pequeño tamaño por parte de usuarios esporádicos por la barrera de entrada que supone el aprendizaje de un nuevo lenguaje
- No pueden utilizarse para la resolución directa de problemas gigantescos cuya formulación completa incluso no se puede realizar (por ejemplo, a partir de 1 millón de restricciones y/o variables)
- En la ejecución se incluye un tiempo de creación del modelo y de interfaz con el optimizador que ralentiza la obtención de la solución, por lo tanto no es recomendable cuando el tiempo de ejecución es un factor crítico.

Las *tendencias* o características más actuales en el desarrollo de lenguajes algebraicos se mueven hacia:

- Interfaces de entrada y salida de datos más estrechamente relacionadas con bases de datos u hojas de cálculo
- El desarrollo de interfaces gráficas que faciliten al usuario la formulación visual y el entendimiento de problemas de optimización
- Interfaz con lenguajes de propósito general para la incorporación de funciones externas definidas por el usuario dentro de la optimización
- El avance en las capacidades de resolución directa de problemas estocásticos (con adición de características específicas en el propio lenguaje y uso de algoritmos de descomposición en el optimizador) o problemas no lineales complejos
- La posibilidad de ocultar el código fuente produciendo versiones ejecutables para usuarios finales
- La selección automática del método y optimizador

La experiencia personal por el uso de estos lenguajes de modelado ha sido tremendamente positiva. En el Instituto de Investigación Tecnológica (IIT) de la Escuela Técnica Superior de Ingeniería se pasó a partir del año 1991 de

utilizar lenguajes de propósito general (como FORTRAN) para el desarrollo de modelos de optimización a utilizar exclusivamente lenguajes de modelado (como GAMS). Esto ha representado un salto importante en cuanto a la productividad de los modeladores. Aplicaciones que antes requerían decenas de miles de líneas de código (en FORTRAN) ahora se desarrollan con una décima parte de la longitud original y con un esfuerzo muy inferior en tiempo (menos de la cuarta parte). De hecho, el IIT ha desarrollado desde el año 1993 numerosos modelos de optimización para el sector eléctrico y ha sido el precursor en España del uso de lenguajes algebraicos en el campo de la planificación, operación y economía del sector eléctrico. En este sector, cuya regulación ha cambiado recientemente, ha sido de vital importancia la capacidad de mantener o modificar de manera muy sencilla los modelos de planificación debido al uso de estos lenguajes.

I.3.1.3 Referencias

Brooke, A., Kendrick, D. and Meeraus, A. (1998) *GAMS: A User's Guide*. GAMS Development Co.

Fourer, R., Gay, D.M. and Kernighan, B.W. (2000) *AMPL: A Modeling Language for Mathematical Programming.* The Scientific Press. 2nd ed.

Ragsdale, C. T. (1998) *Spreadsheet modeling and decision analysis: a practical introduction to management science*. South-Western College. 2nd ed.

Gass, S.I., Hirshfeld, D.S. and Wasil, E.A. (1995) "Model World: The Spreadsheeting of OR/MS" *Interfaces* pp. 72-81. September-October.

McCarl. B. A. and Spreen. Th. H. (1998) Applied Mathematical Programming using Algebraic Systems. Technical Report.

Mocholí, M. y Sala, R. (1996) *Decisiones de optimización* Tirant lo Blanch. Valencia.

Sharda, R. and Rampal, G. (1995) "Algebraic Modeling Languages on PCs" *OR/MS Today* pp. 58-63. June.

Van Hentenryck, P. (1999) *The OPL Optimization Programming Language*. The MIT Press.

I.3.2 Modelado en GAMS

En este apartado se presentan varios ejemplos sencillos que permiten mostrar algunas de las características del lenguaje GAMS. Sin embargo, el manual de usuario contiene un capítulo tutorial y la referencia de todas las características del lenguaje.

I.3.2.1 Ejemplo de transporte

Veamos a continuación un caso típico de un problema de optimización lineal clásico y cómo este problema se codifica en el lenguaje GAMS. En el apartado de modelado en programación lineal entera mixta se presenta formalmente este problema y sus características. Sean i fábricas de envasado y j mercados de consumo. Cada fábrica tiene una capacidad máxima de producción de a_i cajas y cada mercado demanda una cantidad b_j de cajas (se supone que la capacidad de producción total de las fábricas es superior a la demanda total para que el problema sea factible). El coste de transporte entre cada fábrica i y cada mercado j por cada caja es c_{ij}. Se desea satisfacer la demanda de cada mercado al mínimo coste. Las variables de decisión del problema serán las cajas transportadas entre cada fábrica i y cada mercado j, x_{ij}. Las ecuaciones que deben satisfacerse son:

Límite de capacidad máxima de producción de cada fábrica

$$\sum_j x_{ij} \leq a_i \text{ para cada fábrica } i$$

Satisfacción de la demanda de cada mercado

$$\sum_i x_{ij} \geq b_j \text{ para cada mercado } j$$

La función objetivo será la minimización de los costes totales de transporte

$$\sum_i \sum_j c_{ij} x_{ij}$$

Ésta es la forma algebraica de representación de este problema de optimización. La codificación en lenguaje GAMS aparece a continuación.

```
$TITLE MODELO DE TRANSPORTE

SETS
   I fábricas de envasado / VIGO, ALGECIRAS /
   J mercados de consumo  / MADRID, BARCELONA, VALENCIA /

PARAMETERS
   A(i) capacidad de producción de la fábrica i [cajas]
        / VIGO      350
          ALGECIRAS 700 /

   B(j) demanda del mercado j [cajas]
        / MADRID    400
          BARCELONA 450
          VALENCIA  150 /

TABLE C(i,j) coste unitario transporte entre i y j [miles de pesetas por caja]
           MADRID BARCELONA VALENCIA
VIGO        0.06     0.12     0.09
ALGECIRAS   0.05     0.15     0.11

VARIABLES
   X(i,j) cajas transportadas entre fábrica i y mercado j [cajas]
   CT     coste de transporte [miles de pesetas]

POSITIVE VARIABLE X
```

```
EQUATIONS
   COSTE        coste total de transporte   [miles de pesetas]
   CAPACIDAD(i) capacidad máxima de cada fábrica i     [cajas]
   DEMANDA(j)   satisfacción demanda de cada mercado j [cajas] ;

COSTE ..        CT =E= SUM[(i,j), C(i,j) * X(i,j)] ;

CAPACIDAD(i) .. SUM[j, X(i,j)] =L= A(i) ;

DEMANDA(j) ..   SUM[i, X(i,j)] =G= B(j) ;

MODEL TRANSPORTE / COSTE, CAPACIDAD, DEMANDA /

SOLVE TRANSPORTE USING LP MINIMIZING CT
```

Se ha puesto especial cuidado en presentar de forma clara, concisa y limpia el código. El contenido de este código resulta prácticamente autoexplicativo. El resultado de la ejecución del modelo de transporte se presenta a continuación.

```
C o m p i l a t i o n

COMPILATION TIME     =        0.000 SECONDS    0.7 Mb      WIN-19-115

Equation Listing    SOLVE TRANSPORTE USING LP FROM LINE 39

---- COSTE  =E=  coste total de transporte   [miles de pesetas]

COSTE..  - 0.06*X(VIGO,MADRID) - 0.12*X(VIGO,BARCELONA) - 0.09*X(VIGO,VALENCIA)

      - 0.05*X(ALGECIRAS,MADRID) - 0.15*X(ALGECIRAS,BARCELONA)

      - 0.11*X(ALGECIRAS,VALENCIA) + CT =E= 0 ; (LHS = 0)

---- CAPACIDAD  =L=  capacidad máxima de cada fábrica i     [cajas]

CAPACIDAD(VIGO)..  X(VIGO,MADRID) + X(VIGO,BARCELONA) + X(VIGO,VALENCIA) =L= 350 ;

      (LHS = 0)

CAPACIDAD(ALGECIRAS)..  X(ALGECIRAS,MADRID) + X(ALGECIRAS,BARCELONA)

      + X(ALGECIRAS,VALENCIA) =L= 700 ; (LHS = 0)

---- DEMANDA  =G=  satisfacción demanda de cada mercado j [cajas]

DEMANDA(MADRID)..  X(VIGO,MADRID) + X(ALGECIRAS,MADRID) =G= 400 ;

      (LHS = 0, INFES = 400 ***)

DEMANDA(BARCELONA)..  X(VIGO,BARCELONA) + X(ALGECIRAS,BARCELONA) =G= 450 ;

      (LHS = 0, INFES = 450 ***)

DEMANDA(VALENCIA)..  X(VIGO,VALENCIA) + X(ALGECIRAS,VALENCIA) =G= 150 ;

      (LHS = 0, INFES = 150 ***)

Column Listing      SOLVE TRANSPORTE USING LP FROM LINE 39

---- X  cajas transportadas entre fábrica i y mercado j [cajas]

X(VIGO,MADRID)
                (.LO, .L, .UP = 0, 0, +INF)
       -0.06    COSTE
        1       CAPACIDAD(VIGO)
        1       DEMANDA(MADRID)

X(VIGO,BARCELONA)
                (.LO, .L, .UP = 0, 0, +INF)
       -0.12    COSTE
        1       CAPACIDAD(VIGO)
        1       DEMANDA(BARCELONA)

X(VIGO,VALENCIA)
                (.LO, .L, .UP = 0, 0, +INF)
       -0.09    COSTE
        1       CAPACIDAD(VIGO)
        1       DEMANDA(VALENCIA)

REMAINING 3 ENTRIES SKIPPED
```

```
---- CT  coste de transporte [miles de pesetas]

CT
                (.LO, .L, .UP = -INF, 0, +INF)
        1       COSTE

Model Statistics    SOLVE TRANSPORTE USING LP FROM LINE 39

MODEL STATISTICS

BLOCKS OF EQUATIONS       3     SINGLE EQUATIONS        6
BLOCKS OF VARIABLES       2     SINGLE VARIABLES        7
NON ZERO ELEMENTS        19

GENERATION TIME      =        0.140 SECONDS    1.4 Mb      WIN-19-115

EXECUTION TIME       =        0.140 SECONDS    1.4 Mb      WIN-19-115

               S O L V E      S U M M A R Y

     MODEL   TRANSPORTE          OBJECTIVE  CT
     TYPE    LP                  DIRECTION  MINIMIZE
     SOLVER  CPLEX               FROM LINE  39

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE               93.5000

 RESOURCE USAGE, LIMIT          0.401     1000.000
 ITERATION COUNT, LIMIT         5        10000

GAMS/Cplex    May 18, 2000 WIN.CP.CP 19.3 016.014.038.WAT For Cplex 6.6
Cplex 6.6.1, GAMS Link 16, Using a GAMS/Cplex demo license installed at runtime.

Optimal solution found.

Objective :          93.500000

                        LOWER     LEVEL     UPPER    MARGINAL

---- EQU COSTE            .         .         .        1.000

  COSTE  coste total de transporte   [miles de pesetas]

---- EQU CAPACIDAD  capacidad máxima de cada fábrica i     [cajas]

             LOWER     LEVEL     UPPER    MARGINAL

VIGO          -INF    350.000   350.000    -0.030
ALGECIRAS     -INF    650.000   700.000      .

---- EQU DEMANDA  satisfacción demanda de cada mercado j [cajas]

             LOWER     LEVEL     UPPER    MARGINAL

MADRID      400.000   400.000     +INF      0.050
BARCELONA   450.000   450.000     +INF      0.150
VALENCIA    150.000   150.000     +INF      0.110

---- VAR X  cajas transportadas entre fábrica i y mercado j [cajas]

                        LOWER     LEVEL     UPPER    MARGINAL

VIGO     .MADRID          .         .        +INF      0.040
VIGO     .BARCELONA       .      350.000     +INF       .
VIGO     .VALENCIA        .         .        +INF      0.010
ALGECIRAS.MADRID          .      400.000     +INF       .
ALGECIRAS.BARCELONA       .      100.000     +INF       .
ALGECIRAS.VALENCIA        .      150.000     +INF       .

                        LOWER     LEVEL     UPPER    MARGINAL

---- VAR CT              -INF     93.500     +INF       .

  CT  coste de transporte [miles de pesetas]

**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED

EXECUTION TIME       =        0.030 SECONDS    0.7 Mb      WIN-19-115

**** FILE SUMMARY

INPUT      D:\TR.GMS
OUTPUT     D:\TR.LST
```

I.3.2.2 Ejemplo de planificación de la producción

A continuación se presenta un ejemplo de planificación de la producción de una fábrica de papel. Se dispone de varias máquinas para producir diferentes tipos de papel. Se trata de determinar cuáles son las cantidades óptimas a producir de cada tipo de papel en cada máquina para maximizar el beneficio. La demanda de cada tipo de papel se considera fija y conocida y existen limitaciones en el tiempo de producción disponible en cada máquina.

```
$TITLE Planificación de la producción de una papelera

* la papelera puede producir cuatro tipos diferentes de papel en tres máquinas
* distintas. Dada una demanda fija se trata de determinar la producción que
* maximiza los beneficios mensuales

SETS
   M máquinas       / maquina1 * maquina3 /
   P tipos de papel / prensa, folio, imprenta, reciclado /

TABLE TASAPROD(p,m) tasa de producción (tm por h)
                  maquina1  maquina2  maquina3
      prensa         53        52        49
      folio          51        49        44
      imprenta       52        45        47
      reciclado      42        44        40

TABLE COSTEPROD(p,m) coste de producción (€ por tm)
                   maquina1  maquina2  maquina3
      prensa         76        75        73
      folio          82        80        78
      imprenta       96        95        92
      reciclado      72        71        70

TABLE DATDEM(p,*) demanda y precio de venta
                    demanda   precio
*               (tm por mes)(€ por tm)
      prensa         30000      77
      folio          20000      81
      imprenta       12000      99
      reciclado       8000     105

PARAMETER TIEMPOMAQ(m) tiempo disponible al mes de cada máquina (h)
                     / maquina1   672
                       maquina2   600
                       maquina3   480 /

VARIABLES
   PRODUCC(p,m) producción de cada tipo papel en cada máquina (tm por mes)
   BENEFICIO    beneficio  (€ por mes)

POSITIVE VARIABLE PRODUCC

EQUATIONS
   CAPACMAQ(m) capacidad de cada máquina     (h  por mes)
   DEMANDAP(p) demanda de cada tipo de papel (tm por mes)
   BENEF       beneficio                     (€  por mes) ;

CAPACMAQ(m) .. SUM[p, PRODUCC(p,m)/TASAPROD(p,m)] =L= TIEMPOMAQ(m) ;

DEMANDAP(p) .. SUM[m, PRODUCC(p,m)] =E= DATDEM(p,'demanda') ;

BENEF       .. BENEFICIO =E= SUM[p, DATDEM(p,'demanda')*DATDEM(p,'precio')]
                           - SUM[(p,m), COSTEPROD(p,m)*PRODUCC(p,m)] ;

MODEL PAPELERA / ALL / ;

SOLVE PAPELERA USING LP MAXIMIZING BENEFICIO ;
```

I.3.2.3 Ejemplo de secuenciación de órdenes de trabajo

Dada una máquina y 5 trabajos que hay que realizar en ella, en cualquier orden, se dispone del tiempo de ejecución de cada trabajo

TR1	TR2	TR3	TR4	TR5
15	13	12	14	16

y del tiempo de ajuste de la máquina para pasar de ejecutar el trabajo i (fila) a ejecutar el trabajo j (columna)

	TR1	TR2	TR3	TR4	TR5
TR1		2	5	1	6
TR2	3		4	2	5
TR3	4	2		3	4
TR4	5	3	6		5
TR5	4	4	4	3	

Resolver el problema de determinar cuál es el menor tiempo posible para completar los 5 trabajos y cómo hacerlo. Se considera un ciclo de trabajo cerrado, que se repite y vuelve a comenzar.

```
* La segunda formulación evita subciclos de parejas de trabajos

SETS
   I trabajos que se van a ejecutar / TR1 * TR5 /

ALIAS (i,j)

TABLE C(i,j) tiempo de ajuste para pasar del trabajo i al trabajo j
        TR1    TR2    TR3    TR4    TR5
TR1             2      5      1      6
TR2      3             4      2      5
TR3      4      2             3      4
TR4      5      3      6             5
TR5      4      4      4      3

VARIABLES
   X(i,j) paso del trabajo i al trabajo j
   TT     tiempo total en completar los trabajos

BINARY VARIABLE X

EQUATIONS
   TIEMPO       tiempo total de trabajo
   ANTERIOR(i)  de cada trabajo se parte una vez
   POSTERIOR(j) a  cada trabajo se llega una vez
   PAREJAS(i,j) suma de los trabajos por parejas ;

TIEMPO       .. TT =E= SUM[(i,j) $(NOT SAMEAS(i,j)), C(i,j)*X(i,j)] ;
ANTERIOR(i)  .. SUM[j $(NOT SAMEAS(i,j)), X(i,j)] =E= 1 ;
POSTERIOR(j) .. SUM[i $(NOT SAMEAS(i,j)), X(i,j)] =E= 1 ;
PAREJAS(i,j) $(ORD(i) < ORD(j)) .. X(i,j) + X(j,i) =L= 1 ;

MODEL AJUSTE1 / TIEMPO, ANTERIOR, POSTERIOR          / ;
MODEL AJUSTE2 / TIEMPO, ANTERIOR, POSTERIOR, PAREJAS / ;

SOLVE AJUSTE1 USING MIP MINIMIZING TT ;

* número de restricciones de parejas son C(5,2)=10

SOLVE AJUSTE2 USING MIP MINIMIZING TT ;
```

I.3.2.4 Ejemplo de asignación de grupos térmicos

El problema de la asignación de grupos térmicos de producción de electricidad consiste en la decisión de qué grupos térmicos hay que acoplar en cada hora del día (o semana) de manera que:

- Se minimicen los costes variables de generación (incluyendo costes de combustible y costes de arranque y parada)
- Se suministre la demanda en cada hora
- Se mantenga un cierto nivel de reserva rodante
- Se respeten los parámetros de funcionamiento de los grupos térmicos (mínimos técnicos, rampas de subida y bajada)

Datos

D_h demanda térmica en la hora h [MW]
R coeficiente de reserva rodante con respecto a la demanda [p.u.]

a_t término lineal del coste de combustible del grupo térmico t [pta/MWh]
b_t término fijo del coste de combustible del grupo térmico t [pta/h]
ca_t coste de arranque del grupo térmico t [pta]
cp_t coste de parada del grupo térmico t [pta]
$\overline{P}_t$ potencia máxima del grupo térmico t [MW]
$\underline{P}_t$ potencia mínima del grupo térmico t [MW]
rs_t rampa de subida del grupo térmico t [MW/h]
rb_t rampa de bajada del grupo térmico t [MW/h]

Variables

P_{ht} potencia producida por el grupo térmico t en la hora h [MW]
A_{ht} acoplamiento del grupo térmico t en la hora h [0,1]
AR_{ht} arranque del grupo térmico t en la hora h [0,1]
PR_{ht} parada del grupo térmico t en la hora h [0,1]

$$\min \sum_{h=1}^{H} \sum_{t=1}^{T} \left(a_t P_{ht} + b_t A_{ht} + ca_t AR_{ht} + cp_t PR_{ht} \right)$$

$$\sum_{t=1}^{T} P_{ht} = D_h \qquad H$$

$$\sum_{t=1}^{T} (\bar{P}_t A_{ht} - P_{ht}) = RD_h \qquad H$$

$$\underline{P}_t A_{ht} \le P_{ht} \le \bar{P}_t A_{ht} \qquad 2HT$$

$$A_{ht} - A_{h-1t} = AR_{ht} - PR_{ht} \qquad (H-1)T$$

$$P_{ht} - P_{h-1t} \le rs_t \qquad (H-1)T$$

$$P_{h-1t} - P_{ht} \le rb_t \qquad (H-1)T$$

$$P_{ht} \ge 0 \qquad A_{ht}, AR_{ht}, PR_{ht} \in \{0,1\}$$

```
$TITLE ASIGNACIÓN HORARIA DE GRUPOS TÉRMICOS

SETS
T grupos térmicos /GALICIA, CATALUNA, MADRID, VALENCIA,
                   EXTREMAD, ANDALUCI, CASTLEON/
H hora h /h1 * h5/

SCALAR
r porcentaje de reserva rodante sobre la demanda [p.u.] /0.2/

PARAMETERS
d(h) demanda cada hora [MW]
/h1 1000 , h2 1400 , h3 2400 , h4 2000 , h5 1000/
pmax(t) pot máxima de cada térmico [MW]
/GALICIA 400, CATALUNA 500, MADRID 700, VALENCIA 400,
 EXTREMAD 300, ANDALUCI 800, CASTLEON 800/
pmin(t) pot mínima de cada térmico [MW]
/GALICIA 100, CATALUNA 150, MADRID 150, VALENCIA 50,
 EXTREMAD 50, ANDALUCI 400, CASTLEON 200 /
rs(t) rampa de subida [MW por hora]
/GALICIA 200, CATALUNA 300, MADRID 500, VALENCIA 300,
 EXTREMAD 100, ANDALUCI 500, CASTLEON 400/
rb(t) rampa de bajada [MW por hora]
/GALICIA 300, CATALUNA 300, MADRID 200, VALENCIA 100,
 EXTREMAD 100, ANDALUCI 500, CASTLEON 400/
c(t)  coste lineal de producción [pta por MWh]
/GALICIA 4, CATALUNA 4, MADRID 4, VALENCIA 4,
 EXTREMAD 3, ANDALUCI 2, CASTLEON 7/
b(t)  coste fijo de producción [pta]
/GALICIA 50, CATALUNA 30, MADRID 30, VALENCIA 25,
 EXTREMAD 30, ANDALUCI 80, CASTLEON 70/
ca(t) coste de arranque
/GALICIA 10, CATALUNA 20, MADRID 10, VALENCIA 15,
 EXTREMAD 20, ANDALUCI 10, CASTLEON 15/
cp(t) coste de parada
/GALICIA 5, CATALUNA 10, MADRID 5, VALENCIA 10,
 EXTREMAD 5, ANDALUCI 15, CASTLEON 10/

VARIABLES
CT      coste variable total del sistema [Mpta]
A(t,h)  acoplamiento del grupo t a las h horas [0-1]
AR(t,h) arranque     del grupo t a las h horas [0-1]
PR(t,h) parada       del grupo t a las h horas [0-1]
P(t,h)  generación producida por el grupo t a las h horas [MW]

BINARY VARIABLE A,AR,PR
POSITIVE VARIABLE P

EQUATIONS
COSTE         costes variables de generación-función objetivo [pta]
DEMANDA(h)    abastecimiento de la demanda [MW]
RESERVA(h)    reserva rodante del sistema [MW]
COTASUP(t,h)  cota superior de producción del grupo t [MW]
COTAINF(t,h)  cota inferior de producción del grupo t [MW]
RAMPASUB(t,h) limitación de rampa de subida del grupo t [MW]
RAMPABAJ(t,h) limitación de rampa de bajada del grupo t [MW]
LOGICA(t,h)   relación lógica entre variables de acoplamiento arranque y parada ;

COSTE .. CT =E= SUM[(T,H), c(t)*P(t,h)+b(t)*A(t,h)+ca(t)*AR(t,h)+cp(t)*PR(t,h)] ;
```

```
DEMANDA(h) .. SUM[T, P(t,h)] =E= d(h) ;
RESERVA(h) .. SUM[T, A(t,h)*pmax(t)-P(t,h)] =G= d(h)*r;
COTASUP(t,h) .. P(t,h) =L= pmax(t)*A(t,h);
COTAINF(t,h) .. P(t,h) =G= pmin(t)*A(t,h);
RAMPASUB(t,h) .. P(t,h)-P(t,h-1) =L= rs(t);
RAMPABAJ(t,h) .. P(t,h-1)-P(t,h) =L= rb(t);
LOGICA(t,h) .. A(t,h)-A(t,h-1) =E= AR(t,h)-PR(t,h);
MODEL ASIGNA /COSTE,DEMANDA,RESERVA,COTASUP,COTAINF,RAMPASUB,RAMPABAJ,LOGICA/ ;
P.UP(t,h) = pmax(t) ;
SOLVE ASIGNA USING MIP MINIMIZING CT;
```

I.3.2.5 Ejemplo de flujo de cargas óptimo

Veamos a continuación la formulación de un ejemplo de sistemas de energía eléctrica. En particular, se trata de un *flujo de cargas óptimo en corriente continua con y sin pérdidas óhmicas* que se formula como un problema de optimización lineal y no lineal dependiendo de la consideración o no de las pérdidas. Este ejemplo ilustra alguna característica muy interesante del GAMS como son los conjuntos dinámicos y su uso para restringir las variables y ecuaciones del problema. Aunque no se entienda el significado físico del problema por carecer de los conocimientos adecuados en sistemas de energía eléctrica su formulación en GAMS se puede seguir sin dificultad.

La *estructura* general de un modelo de optimización escrito en GAMS se presenta en la tabla 1.2. Esta estructura es la que se ha empleado en este caso ejemplo.

Índices y parámetros	Todos los índices y parámetros del modelo se declaran al comienzo del mismo. Se inicializarán a sus valores por omisión aquellos que sea necesario.
Variables	Definición de las variables según sean positivas, libres, binarias, etc.
Ecuaciones	Declaración y definición de las restricciones. Se controlará con cuidado las condiciones de validez u ocurrencia de las mismas.
Modelo	Declaración de las ecuaciones que componen el modelo.
Inclusión y manipulación de datos de entrada	Los datos de entrada se introducen desde ficheros independientes, después se realizan los cálculos de parámetros auxiliares dependientes de los datos de entrada.
Acotamiento e	Acotamiento de las variables a sus cotas físicas e

inicialización de variables	inicialización cuando tenga sentido.
Resolución del problema de optimización	
Presentación de resultados	Presentación de los resultados elaborados a partir de la solución del problema de optimización.

Tabla 1.2 Estructura de un modelo de optimización en GAMS.

Se trata de minimizar los costes variables de operación en un intervalo horario unitario compuestos por los costes variables de los grupos térmicos, los costes de oportunidad de los grupos hidráulicos cuando producen por encima de su potencia programada y el coste variable de la potencia no suministrada.

$$\sum_{t=1}^{T} v_t GTR_t + \sum_{h=1}^{H} v_h GHE_h + \sum_{n=1}^{N} v_n PNS_n$$

siendo datos v_t el coste variable unitario de generación del grupo térmico t, v_h el coste de oportunidad unitario de la hidráulica de emergencia h y v_n el coste variable unitario de la potencia no suministrada en el nudo n. Las variables son: GTR_t potencia producida por el grupo térmico t, GHE_h potencia hidráulica de emergencia del grupo hidráulico h y PNS_n potencia no suministrada en el nudo n. T, H y N son todos los grupos térmicos, hidráulicos y nudos del sistema respectivamente.

Las restricciones que condicionan este problema se muestran a continuación.

La primera ley de Kirchhoff que establece el balance entre generación y demanda en cada nudo:

$$\sum_{t \in n} GTR_t + \sum_{h \in n} (GHP_h + GHE_h) + PNS_n + \sum_{i=1}^{I} F_{i \to n} - \sum_{j=1}^{J} F_{n \to j} = D_n$$

siendo $t \in n$, $h \in n$ los grupos térmicos e hidráulicos localizados en el nudo n, GHP_h es la variable potencia hidráulica programada del grupo hidráulico h, $F_{i \to j}$ es la variable flujo de potencia que va del nudo i al nudo j y D_n corresponde al dato de la demanda de potencia en el nudo n.

La segunda ley de Kirchhoff nos dice que el flujo por una línea es proporcional a la diferencia de los ángulos de tensión de sus nudos extremos:

$$\frac{X_{i \to j}}{S_B} F_{i \to j} = \theta_i - \theta_j$$

donde se conocen $X_{i\to j}$ es la reactancia de la línea que une los nudos i y j y S_B es la potencia base del sistema y θ_i es la variable de ángulo de tensión del nudo i.

Algunas variables del problema están acotadas entre ciertos valores. La potencia térmica producida de cada grupo t se encuentra entre su valor mínimo $\underline{GTR_t}$ y máximo $\overline{GTR_t}$.

$$\underline{GTR_t} \le GTR_t \le \overline{GTR_t}$$

La potencia hidráulica programada de cada grupo h puede tomar como valor máximo $\overline{GHP_h}$, valor dado por un programa de coordinación hidrotérmica de jerarquía superior.

$$0 \le GHP_h \le \overline{GHP_h}$$

La potencia hidráulica de emergencia de cada grupo h puede alcanzar como máximo el valor $\left(\overline{GHM_h} - \overline{GHP_h}\right)$, es decir, la potencia máxima del grupo menos su potencia programada.

$$0 \le GHE_h \le \left(\overline{GHM_h} - \overline{GHP_h}\right)$$

La potencia no suministrada como mucho será la demanda del nudo.

$$0 \le PNS_n \le D_n$$

El flujo por la línea está acotado en valor absoluto por $\overline{F_{i\to j}}$.

$$-\overline{F_{i\to j}} \le F_{i\to j} \le \overline{F_{i\to j}}$$

Además de la formulación anterior, se presenta otra donde se eliminan las variables de flujo y se sustituyen por la expresión de la segunda ley de Kirchhoff en función de los ángulos. La ecuación de balance entre generación y demanda tiene ahora esta formulación

$$\sum_{t\in n} GTR_t + \sum_{h\in n}(GHP_h + GHE_h) + PNS_n +$$
$$\sum_{i=1}^{I}(\theta_i - \theta_n)S_B / X_{i\to n} - \sum_{j=1}^{J}(\theta_n - \theta_j)S_B / X_{n\to j} = D_n$$

y las cotas de las variables de flujo se transforman ahora en restricciones

$$\theta_i - \theta_j \le \overline{F_{i\to j}}\,\frac{X_{i\to j}}{S_B}$$

$$\theta_i - \theta_j \geq -\overline{F_{i\to j}} \frac{X_{i\to j}}{S_B}$$

En la primera formulación se tienen más variables pero menos restricciones[4] que en la segunda y el número total de elementos no nulos de la matriz de restricciones será menor.

En el código escrito en GAMS se añade, además, una formulación del *flujo de cargas óptimo en corriente continua con pérdidas óhmicas*. Las pérdidas óhmicas de una línea se modelan con una expresión *no lineal* en función del coseno de la diferencia angular. Esto convierte el problema de optimización en no lineal.

$$L_{i\to j} = 2S_B \frac{r_{i\to j}}{r_{i\to j}^2 + X_{i\to j}^2}\left[1 - \cos(\theta_i - \theta_j)\right]$$

siendo $r_{i\to j}$ la resistencia de la línea que une los nudos i y j.

Éstas se incluyen como dos cargas adicionales iguales en los extremos de la línea. La primera ley de Kirchhoff tiene ahora esta expresión

$$\sum_{t\in n} GTR_t + \sum_{h\in n}(GHP_h + GHE_h) + PNS_n + \sum_{i=1}^{I} F_{i\to n} - \sum_{j=1}^{J} F_{n\to j} = D_n + L_n$$

siendo L_n las pérdidas en el nudo n

$$L_n = \left(\sum_{i=1}^{I} L_{i\to n} + \sum_{j=1}^{J} L_{n\to j}\right)/2$$

```
$TITLE Flujo de cargas en corriente continua con y sin pérdidas

SETS
   ND          nudos
   GR          generadores
   TR(gr)      generadores térmicos
   HD(gr)      generadores hidráulicos
   NDGR(nd,gr) localización de generadores en nudos
   LN(nd,nd)   líneas

   CN características nudos       / dem, cpns /
   CG características generadores / coste, pmin, pmax, cshd, hdrpro, hdrmax /
   CL características líneas      / r, x, flmax /

ALIAS (nd, ni, nf) ;

SCALARS
   SBASE  potencia base [GW]                              / 0.1 /
   OPCPRD opción de modelado de las pérdidas (no 0 si 1) / 0 /

* definición de la estructura de datos sin incluir explícitamente éstos

PARAMETERS
   DATNUD(nd,cn)    datos de los nudos
   DATGEN(gr,cg)    datos de los generadores
   DATLIN(nd,nd,cl) datos de las líneas

* planteamiento matemático del problema

VARIABLES
```

[4] Las cotas en las variables no cuentan como restricciones desde el punto de vista del tiempo de cálculo, ya que los algoritmos de optimización las tratan de forma específica.

```
   COSTE     función objetivo                              [Mpta]
   TT(nd)    ángulo de tensión en el nudo                  [rad]
   FL(ni,nf) flujo de potencia                             [GW]

POSITIVE VARIABLES
   GTR(gr)   generación térmica                            [GW]
   GHP(gr)   generación hidráulica programada              [GW]
   GHE(gr)   generación hidráulica de emergencia           [GW]
   PNS(nd)   potencia no suministrada                      [GW]
   PRDAS(nd) pérdidas de las líneas conectadas al nudo     [GW]

EQUATIONS
   FO          costes de generación y de indisponibilidad [Mpta]
   KR1F(nd)    primera ley de Kirchhoff para cada nudo en función de flujos
   KR1A(nd)    primera ley de Kirchhoff para cada nudo en función de ángulos
   FLJ(ni,nf)  flujo en función de ángulos de tensión
   FLJP(ni,nf) diferencia angular máxima en cada línea en un   sentido
   FLJN(ni,nf) diferencia angular máxima en cada línea en otro sentido
   EPRDAS(nd)  pérdidas de las líneas conectadas al nudo ;

FO      .. COSTE =E= SUM[tr, DATGEN(tr,'coste') * GTR(tr)]
                   +  SUM[hd, DATGEN(hd,'cshd')  * GHE(hd)]
                   +  SUM[nd, DATNUD(nd,'cpns')  * PNS(nd)] ;

KR1F(nd) ..
   SUM[NDGR(nd,tr), GTR(tr)] + SUM[NDGR(nd,hd), GHP(hd) + GHE(hd)]
 + SUM[LN(ni,nd), FL(ni,nd)] - SUM[LN(nd,nf), FL(nd,nf)]
 + PNS(nd) =E= DATNUD(nd,'dem') + PRDAS(nd) $OPCPRD ;

KR1A(nd) ..
   SUM[NDGR(nd,tr), GTR(tr)]
 + SUM[NDGR(nd,hd), GHP(hd) + GHE(hd)]
 + SUM[LN(ni,nd), (TT(ni) - TT(nd)) / DATLIN(ni,nd,'x')] * SBASE
 - SUM[LN(nd,nf), (TT(nd) - TT(nf)) / DATLIN(nd,nf,'x')] * SBASE
 + PNS(nd) =E= DATNUD(nd,'dem') + PRDAS(nd) $OPCPRD ;

FLJ(LN(ni,nf)) .. FL(ni,nf) * DATLIN(ni,nf,'x') / SBASE =E= TT(ni) - TT(nf) ;

FLJP(LN(ni,nf)) ..
   TT(ni) - TT(nf) =L=   DATLIN(ni,nf,'flmax') * DATLIN(ni,nf,'x') / SBASE ;

FLJN(LN(ni,nf)) ..
   TT(ni) - TT(nf) =G= - DATLIN(ni,nf,'flmax') * DATLIN(ni,nf,'x') / SBASE ;

EPRDAS(nd) .. PRDAS(nd) =E=
           SBASE * SUM[LN(ni,nd), (1-cos(TT(ni) - TT(nd)))*
           DATLIN(ni,nd,'r')/(DATLIN(ni,nd,'r')**2+DATLIN(ni,nd,'x')**2)]
         + SBASE * SUM[LN(nd,nf), (1-cos(TT(nd) - TT(nf)))*
           DATLIN(nd,nf,'r')/(DATLIN(nd,nf,'r')**2+DATLIN(nd,nf,'x')**2)] ;

MODEL FC  / FO, KR1F, FLJ         / ;
MODEL FCA / FO, KR1A, FLJP, FLJN  / ;
MODEL FCP / FO, KR1F, FLJ, EPRDAS / ;

* caso de estudio
*** esta parte iría en ficheros independientes y se introduciría con $include

SETS
   ND     nudos                  / nudo-1 * nudo-9 /
   GR     generadores            / genr-1 * genr-9, genh-1 * genh-4 /
   NDGR(nd,gr) localización de generadores en nudos
  /
   nudo-1 . genr-1
   nudo-1 . genr-2
   nudo-1 . genr-3
   nudo-2 . genr-4
   nudo-2 . genr-5
   nudo-2 . genr-6
   nudo-3 . genr-7
   nudo-3 . genr-8
   nudo-3 . genr-9
   nudo-1 . genh-1
   nudo-3 . genh-2
   nudo-6 . genh-3
   nudo-8 . genh-4
  / ;

TABLE DATNUD(nd,cn) datos de los nudos
             dem   cpns
*             MW pta/kWh
   nudo-1      1   150
   nudo-2    240   150
   nudo-3     40   150
   nudo-4    160   150
   nudo-5    240   150
   nudo-6     80   150
   nudo-7    100   150
   nudo-8     15   150
```

```
   nudo-9    100   150 ;

TABLE DATGEN(gr,cg) datos de los generadores
            coste   pmin  pmax   cshd  hdrpro hdrmax
*          pta/kWh   MW    MW  pta/kWh   MW     MW
   genr-1     6.5      0    75
   genr-2     7.0      0   125
   genr-3     7.5      0   100
   genr-4     5.9      0   100
   genr-5     6.7      0    50
   genr-6     7.4      0    50
   genr-7     6.1      0   100
   genr-8     7.6      0    50
   genr-9     8.0      0    50
   genh-1                          1.0   300    300
   genh-2                          1.0   160    160
   genh-3                          1.0   150    150
   genh-4                          1.0   100    100 ;

TABLE DATLIN(ni,nf,cl) datos de las líneas
                     r       x      flmax
*                    p.u.    p.u.     MW
   nudo-1 . nudo-2  0.0777  0.2913  500
   nudo-1 . nudo-4  0.0544  0.2041  500
   nudo-2 . nudo-3  0.0424  0.1695  500
   nudo-2 . nudo-4  0.1     0.4     500
   nudo-2 . nudo-5  0.05    0.2     500
   nudo-2 . nudo-6  0.1     0.4     500
   nudo-3 . nudo-5  0.0248  0.099   500
   nudo-3 . nudo-8  0.1     0.4     500
   nudo-4 . nudo-6  0.15    0.6     500
   nudo-5 . nudo-6  0.05    0.2     500
   nudo-5 . nudo-8  0.1     0.4     500
   nudo-6 . nudo-7  0.15    0.6     500
   nudo-6 . nudo-9  0.05    0.2     500
   nudo-7 . nudo-9  0.05    0.2     500 ;

*** hasta aquí son ficheros independientes

* activación de generadores térmicos hidráulicos y líneas

TR(gr)    $DATGEN(gr,'pmax')   = YES ;
HD(gr)    $DATGEN(gr,'hdrpro') = YES ;
LN(ni,nf) $DATLIN(ni,nf,'x')   = YES ;

* escalación de datos de potencia a GW

DATNUD(nd,'dem')    = DATNUD(nd,'dem')    / 1e3 ;
DATGEN(tr,'pmin')   = DATGEN(tr,'pmin')   / 1e3 ;
DATGEN(tr,'pmax')   = DATGEN(tr,'pmax')   / 1e3 ;
DATGEN(hd,'hdrpro') = DATGEN(hd,'hdrpro') / 1e3 ;
DATGEN(hd,'hdrmax') = DATGEN(hd,'hdrmax') / 1e3 ;
DATLIN(ln,'flmax')  = DATLIN(ln,'flmax')  / 1e3 ;

* acotamiento de las variables (cotas físicas)

GTR.LO(tr) = DATGEN(tr,'pmin') ;
GTR.UP(tr) = DATGEN(tr,'pmax') ;

GHP.UP(hd) = DATGEN(hd,'hdrpro') ;
GHE.UP(hd) = DATGEN(hd,'hdrmax') - DATGEN(hd,'hdrpro') ;

PNS.UP(nd) = DATNUD(nd,'dem') ;

FL.LO(ln)  = - DATLIN(ln,'flmax') ;
FL.UP(ln)  =   DATLIN(ln,'flmax') ;

* cotas algorítmicas de los ángulos

TT.LO(nd) = - 1.5 ;
TT.UP(nd) =   1.5 ;

* nudo de referencia

TT.FX(nd) $(ORD(nd) EQ 1) = 0 ;

* opción sin pérdidas

OPCPRD = 0 ;

* flujo de cargas con variables de flujo

SOLVE FC  USING  LP MINIMIZING COSTE ;

* control sobre aprovechamiento de base previa

OPTION BRATIO = 1 ;
```

```
* flujo de cargas con variables de ángulos de tensión
SOLVE FCA USING  LP MINIMIZING COSTE ;
* opción con pérdidas
OPCPRD = 1 ;
* flujo de cargas con variables de flujo
SOLVE FCP USING NLP MINIMIZING COSTE ;
```

I.3.3 Elementos de estilo de programación

"En los últimos años se ha reconocido la programación de computadores como una disciplina cuyo dominio es básico y crucial para el éxito de muchos proyectos de ingeniería" Niklaus Wirth (1976).

I.3.3.1 Generales

La programación no es un castigo divino para los humanos, es *ciencia* y *arte*. Es ciencia en la medida que se pueden implantar modelos matemáticos complejos y que el pensamiento, la disciplina, la rigurosidad y la experimentación acompañan este desarrollo. El resultado es arte por la belleza, elegancia, sensación que puede transmitir un modelo y la profesionalidad de su creador.

Una forma de aprender a escribir con estilo y estructura ordenada es mediante la *lectura* de ejemplos ilustrativos o de código ajeno. Una manera de programar es por *refinamiento gradual* de los detalles. Es importante recordar que en el desarrollo de una aplicación el diablo se esconde en los detalles.

La potencia y concisión de los modelos escritos en un lenguaje de modelado hacen que el propio código forme parte de la documentación. De hecho la reutilización de modelos fue una de las causas que dieron origen a los lenguajes de modelado. La etapa de diseño del modelo cobra gran importancia para permitir posteriores ampliaciones. Por esta razón es importante el estilo en la programación, que incide en la *calidad* y *mantenibilidad*[5] del código desarrollado. Piénsese que el tiempo dedicado a mantenimiento y ampliación de un modelo es muy superior al inicial de desarrollo. El desarrollo y la depuración del modelo se debe hacer con una maqueta (caso ejemplo sencillo) para finalmente utilizar un problema real.

He aquí algunas *recomendaciones* para la escritura de un modelo que inciden en la *calidad* del desarrollo:

MODULARIDAD

[5] Se entiende por mantenibilidad la reutilización, reparación o modificación de un modelo.

Estructurar el modelo en diversos módulos con diferentes propósitos. Por ejemplo, la inclusión de los datos[6] y la escritura de resultados deben separarse en diferentes ficheros que son convenientemente insertados mediante la instrucción `$include` en el módulo principal, que contiene la formulación del problema de optimización.

Utilización de las entidades (parámetros, escalares, etc.) con el mismo propósito y significado en las diferentes partes del código. Es decir, mantener la definición y uso de cada parámetro y escalar en todo el código para evitar la confusión del lector.

Comprobación de la pertenencia de un subconjunto a un conjunto de forma explícita en su definición, es decir, evitar el uso de índices comodín en vectores y matrices. Esta es una manera de validar y evitar errores en la introducción de los datos.

ESCRIBIR CÓDIGO PARA FACILITAR SU LECTURA

Estas otras *recomendaciones* están orientadas al cuidado exquisito de la estética. Es el primer paso en el desarrollo profesional de un modelo. Son fundamentales, aunque aparentemente carecen de importancia para el desarrollador, pero se hacen imprescindibles para su *mantenimiento* y ampliación. El código debe ser limpio y claro para que pueda ser mantenido.

Mantener una coherencia en las reglas de escritura, de manera que se observe una norma sistemática en todo el código. Por ejemplo, endentación en las instrucciones repetitivas, sangría de tres espacios cada vez que se realiza una instrucción tipo `LOOP`, `IF`. Las palabras reservadas del lenguaje van en mayúsculas (`LOOP`, `IF`, `THEN`, `ELSE`, `SET`, `SCALAR`, `PARAMETER`, `TABLE`, etc.). La coma del final de instrucción va separada por un blanco. El signo de igualdad en las asignaciones se separa por espacios en blanco a ambos lados.

Establecer paralelismos o réplicas entre instrucciones consecutivas semejantes.

Las líneas de código deben tener una longitud aproximada de 100 columnas, no sobrepasando nunca las 110. Romper la instrucción en cuantas líneas sea necesario para cumplir esta recomendación.

Los comentarios deben ser suficientemente ilustrativos del contenido y estar bien localizados. Deben ayudar a documentar la naturaleza y origen de los datos

Se deben utilizar nombres largos y descriptivos para las entidades del modelo.

[6] Se recomienda la introducción de los datos tal como son recogidos y entendidos por el usuario y se hacen en el modelo los cálculos auxiliares que sean necesarios.

Los nombres y los índices de los parámetros, variables y ecuaciones han de ser acrónimos que representen su significado. Se recomienda una longitud de hasta 10 caracteres para los primeros y de hasta 2 para los segundos. Los comentarios explicativos pueden hacerse de hasta 80 caracteres.

Las definiciones de las entidades del modelo deben llevar las dimensiones físicas del problema.

Hacer un uso sistemático de mayúsculas y minúsculas con algún criterio predefinido, que debe ser coherente y mantenerse a lo largo de todo el programa. Por ejemplo, los nombres de los parámetros, variables y ecuaciones van en mayúsculas. Los nombres de sus índices van en minúsculas.

REFORMULACIÓN MANUAL DEL PROBLEMA

Un primer estadio en la formulación de un problema está en la elección de la propia formulación. A veces se pueden utilizar formulaciones semejantes con coste computacional muy diferente. Por ejemplo, para la representación de las pérdidas en un circuito eléctrico se puede utilizar una función no lineal o una poligonal aproximada. Habitualmente la formulación poligonal convexa requiere mucho menos tiempo.

Diferentes formulaciones matemáticamente equivalentes de un mismo problema de optimización pueden requerir tiempos de optimización muy distintos. Esta afirmación es especialmente relevante en problemas de programación lineal entera mixta y programación no lineal. Por esta razón, siempre es conveniente un ejercicio continuo de experimentación y reformulación de los problemas.

Veamos estas tres formulaciones de un problema no lineal

$$\min \sum_{i=1}^{n} \sum_{j=i+1}^{n} q_{ij} x_i x_j$$

$$\sum_{j=1}^{n} x_j = 1$$

$$\sum_{j=1}^{n} r_j x_j = r_0$$

$$\min \sum_{i=1}^{n} x_i \sum_{j=i+1}^{n} q_{ij} x_j$$

$$\sum_{j=1}^{n} x_j = 1$$

$$\sum_{j=1}^{n} r_j x_j = r_0$$

$$\min \sum_{i=1}^{n} x_i w_i$$

$$w_i = \sum_{j=i+1}^{n} q_{ij} x_j$$

$$\sum_{j=1}^{n} x_j = 1$$

$$\sum_{j=1}^{n} r_j x_j = r_0$$

La ventaja de la formulación segunda con respecto a la primera es inmediata. La formulación 1 requiere para evaluar la función objetivo aproximadamente $2n\,n/2$ multiplicaciones. En la formulación 2 se necesitan $n + n\,n/2$ aproximadamente. La tercera formulación tiene esencialmente las mismas multiplicaciones pero aparecen en restricciones lineales. El número de restricciones aumenta sustancialmente pero todas son lineales y los métodos de manipulación de restricciones lineales son extremadamente eficientes. La formulación 3 resulta ser la más eficiente.

De hecho, algunos optimizadores realizan una etapa previa de preproceso[7] del problema antes de su resolución (parámetro de control `presolve` en el CPLEX). Como ejemplo, el impacto en el tamaño de dos problemas LP debido al preproceso realizado por el optimizador CPLEX 6.0 se muestra en la tabla 1.3.

	Caso 1			**Caso 2**		
	Restricc.	Variables	Elementos	Restricc.	Variables	Elementos
Sin `presolve`	19047	27262	81215	48971	63935	187059
Con `presolve`	15744	21982	51079	40794	56133	135361
Decremento	17,3%	19,4%	37,1%	16,7%	12,2%	27,6%

Tabla 1.3 Reducción de tamaños con la opción de preproceso.

En la formulación del problema la opción `profile` muestra el tiempo y memoria consumidos y el número de asignaciones realizadas o restricciones creadas.

Entre las principales consideraciones para mejorar la formulación de un problema de optimización se pueden citar:

[7] El desarrollo de las técnicas de preproceso y reformulación han originado avances muy importantes en la resolución de problemas MIP.

- Cálculo analítico del número de restricciones y variables
  Éste es una ayuda para ser consciente del tamaño esperable del problema y ver su dependencia en función de los elementos básicos que lo componen. El número real de restricciones para un caso concreto se muestra con la opción `profile`. Puede ser utilizado para detectar errores en la formulación. Por ejemplo, por excesivo número de ecuaciones al haber puesto dimensiones superfluas no controladas convenientemente con conjuntos dinámicos.
  Es conveniente también conocer la estructura de la matriz de restricciones, es decir, los bloques que la componen. Existe alguna utilidad, citada en el siguiente apartado, que lo permite hacer.

- No crear variables ni ecuaciones superfluas.
  Hay que tener cuidado con lo que se entiende por superfluas porque algunas condiciones redundantes pueden realmente llevar a obtener un modelo más fuerte en el contexto de programación entera. Sin embargo, el conocimiento de la naturaleza del problema permite introducir condiciones lógicas (mediante el uso del operador `$`) que eliminan algunas de ellas en la escritura de las ecuaciones o de las variables. Por ejemplo, en el caso de una red se suprimen variables o ecuaciones asociadas a líneas entre nudos no conectados entre sí. Aunque los optimizadores pueden detectar algunas de estas ecuaciones/variables superfluas, es más eficiente evitarlo mediante condiciones expresas.
  La opción `solprint=on` o la utilidad `gamschk` puede ayudar en la detección de las variables o ecuaciones superfluas (porque toman valor 0 o conocido bajo toda circunstancia en la solución).

- Reducción del número de restricciones y/o elementos de la matriz aun a costa de aumentar el número de variables.
  Una manera de reducir el número de ecuaciones o de variables es introduciendo expresamente el conocimiento que se tiene del problema real (casos particulares que pueden aparecer y sus implicaciones). Es más conveniente hacerlo manualmente a dejar que lo intente el preproceso del optimizador.
  Se puede hacer mediante sustitución, definición de nuevas variables, reformulación en general se debe intentar reducir el número de restricciones y/o de elementos de la matriz.
  Como norma general para la formulación de problemas lineales es conveniente saber que el tiempo necesario para su solución por el método simplex depende aproximadamente del cubo del número de restricciones, no siendo demasiado influyente el número de variables. En el método de punto

interior el tiempo de ejecución depende principalmente del número de elementos (densidad) de la matriz de restricciones.

- Escalación tanto de variables como de coeficientes y valores de restricciones a números alrededor de 1
  Esto mejora el comportamiento numérico en la resolución del problema y reduce el tiempo de ejecución. La escalación resulta muy conveniente en problemas LP de gran tamaño pero es imprescindible en problemas NLP. Implícitamente los valores por omisión de los parámetros de control de los optimizadores están fijados suponiendo que el problema está bien escalado alrededor de 1. Con una escalación razonable puede haber como mucho 6 órdenes de magnitud de diferencia (por ejemplo, coeficientes de las variables entre 0.001 y 1000). La utilidad `gamschk` es una herramienta muy útil para observar los intervalos de variación de los coeficientes de las variables en las restricciones y de las costas de éstas para detectar potenciales problemas de escalado.
  La escalación se puede hacer *manualmente* –expresando las variables, parámetros y ecuaciones en unidades naturales con sentido físico para el problema– o *automáticamente* mediante las opciones disponibles en el lenguaje (`nombre_modelo.scaleopt=1`) o en el optimizador (`scale`). La escalación manual requiere más cuidado y control pero es preferible porque conserva la naturaleza física del problema dentro del código y es igual de efectiva que la automática.
  En cualquier caso hay que tener cuidado al realizar el escalado, especialmente en problemas no lineales, donde pueden existir efectos no lineales que invaliden la nueva formulación.

- Acotamiento de las variables.
  Las cotas en las variables no cuentan como restricciones desde el punto de vista del tiempo de cálculo, ya que los algoritmos de optimización las tratan de forma específica. Las cotas pueden tener sentido *físico* (y, por tanto, forman parte de la naturaleza del problema) o ser *algorítmicas* (es decir, cotas superfluas que nunca deben ser activas en la solución óptima pero que reducen el tiempo de optimización).
  El preproceso generalmente incluye procedimientos para el fortalecimiento de las cotas de las variables (reducción de las cotas superiores y aumento de las inferiores).

TRATAMIENTO EXPLÍCITO DE CONJUNTOS ORDENADOS SOSN

Los conjuntos ordenados (*Special Ordered Sets SOS*) son conjuntos de variables que cumplen las siguientes condiciones:

Como mucho n elementos del conjunto toman valores diferentes de 0. El resto de elementos ha de ser 0

Si hay n elementos que son diferentes de 0 deben ser contiguos

Los conjuntos ordenados tienen un tratamiento especial en la optimización, por lo que su definición puede mejorar mucho el tiempo requerido para la resolución.

SELECCIÓN DEL OPTIMIZADOR Y TIPO ALGORITMO DE OPTIMIZACIÓN

Un lenguaje de modelado permite utilizar diferentes optimizadores para la resolución de un mismo problema de optimización. Esta característica representa una gran ventaja por la flexibilidad que aporta en la selección del optimizador más adecuado a las características del problema.

En Internet (www-c.mcs.anl.gov/otc/guide/faq/linear-programming.html) y en la revista *OR/MS Today*, Fourer (1999), se pueden encontrar opiniones y revisiones del software disponible para la resolución de problemas de optimización de todo tipo. Entre los optimizadores a los que se ha tenido acceso destacan CPLEX y OSL para LP, MINOS y CONOPT para NLP y MILES y PATH para MCP.

El mejor método para un problema concreto depende de las características del problema, de los detalles de implantación del método simplex o del punto interior y del ordenador utilizado. Por esta razón los paquetes comerciales de LP importantes incluyen métodos de punto interior (habitualmente primal-dual predictivo-correctivo), métodos simplex (en su versión primal y dual) y de resolución de flujos de redes (simplex de red). Se debe utilizar el método de optimización (punto interior o barrera, simplex primal o simplex dual) más adecuado al tipo o tamaño del problema.

Como recomendación general, para problemas de tamaño medio (hasta aproximadamente de 10000 x 10000) el método más adecuado es el simplex y para problemas de gran tamaño (desde 10000 x 10000 hasta 100000 x 100000) el mejor método es el de punto interior (especialmente en problemas degenerados). Para problemas de tamaño superior se requiere el uso de técnicas de optimización específicas (como, por ejemplo, las de descomposición entre otras [Ramos, 1996]). La selección de un método u otro se debe realizar principalmente en función del tamaño del problema. [Bixby, 2000] es un artículo práctico reciente donde se presentan algunas comparaciones entre métodos de solución del optimizador CPLEX tanto para problemas lineales como enteros mixtos.

El método simplex también resulta adecuado en la realización de análisis de sensibilidad, es decir, cuando se trata de resolver problemas similares disponiendo de una solución próxima y una base previa, como sucede en el método de ramificación y acotamiento para resolver problemas lineales enteros.

A continuación se presenta la tabla 1.4 de comparación entre varios optimizadores y métodos de optimización. En la tabla 1.5 se muestra la diferencia de funcionamiento entre las opciones de preproceso de dos optimizadores.

		Caso 1			**Caso 2**		
		Tiempo	Índice	Iter.	Tiempo	Índice	Iter.
CPLEX 6.0	Punto interior	41.8	1.0	32	237.3	1.0	35
	Simplex dual	99.8	1.4	12692	1812.6	6.6	48695
	Simplex primal	156.2	3.7	21622	1217.5	5.1	50280
MINOS 5.3	Simplex primal	1863.6	44.6	23927	–	–	–
OSL 2.1	Punto interior	163.9	3.9	10798	774.4	3.3	19524
	Simplex primal	530.9	12.7	12685	7426.6	31.3	62019

Tabla 1.4 Comparación entre diferentes optimizadores en problemas LP.

	Caso 1			**Caso 2**		
	Restricc.	Variables	Elementos	Restricc.	Variables	Elementos
Sin prep	19047	27847	82295	49715	64679	189477
Prep CPLEX	–14,8%	–19,3%	–36,2%	–17,9%	–13,2%	–28,6%
Prep OSL	–4,9%	0,0%	–2,4%	–15,6%	0,0%	–9,1%

Tabla 1.5 Comparación entre diferentes preprocesos.

Las diferencias en tiempo de resolución que pueden encontrarse entre métodos de optimización o entre implantaciones de un mismo método llegan a ser significativas (de hasta 45 veces para una comparación entre CPLEX 6.0 utilizando un método de punto interior y MINOS 5.3 utilizando el método simplex para un problema de 19000 restricciones, 28000 variables y 82000 elementos no nulos). Para un mismo método de optimización se han encontrado diferencias de hasta 3 veces entre implantaciones.

UTILIZACIÓN DE ÚLTIMAS VERSIONES

En general, las últimas versiones aportan mejoras de tiempo o funcionalidad con respecto a versiones previas.

En particular, una característica muy atractiva de los lenguajes de modelado es la posibilidad de actualizar la versión del optimizador o cambiar de optimizador sin necesidad de realizar modificaciones en el código del modelo. Ser consciente de ello y aprovecharlo forma parte de un uso avanzado del lenguaje.

AJUSTE DE PARÁMETROS DE CONTROL DEL OPTIMIZADOR

Habitualmente los parámetros de control de un optimizador toman unos valores por omisión generalmente adecuados para un problema estándar de optimización. Sin embargo, cuando se trata de problemas difíciles, como pueden ser los LP de muy gran tamaño o los NLP o MIP, son convenientes pruebas específicas de ajuste con algunos parámetros. En particular, algunos relacionados con la eficiencia y estabilidad numérica del algoritmo.

Los parámetros son propios de cada optimizador y también pueden serlo de cada método de optimización. Por mencionar algunos que pueden ser importantes en MINOS (`linesearch tolerance`, `penalty`, `major iterations`, `minor iterations`, `factorization frequency`) y en CPLEX (`epopt`, `eprhs`, `epmrk`).

Como consejo para evitar errores o confusiones es conveniente la creación de los ficheros de parámetros de control del optimizador dentro del código en lugar de editarlos manualmente.

USO DE SOLUCIONES INICIALES Y/O BASES PREVIAS

El uso de puntos iniciales es particularmente importante en el caso de problemas no lineales, donde se debe ejecutar un problema lineal cuya solución resulte cercana a la previsible solución del problema no lineal.

Cuando se trata de ejecuciones sucesivas es conveniente, desde el punto de vista de cálculo, aprovechar en el algoritmo del simplex las bases de soluciones previas del mismo problema u otro similar que haya sido resuelto previamente. La base contiene la información relativa a las variables primales y duales del problema. El aprovechamiento se controla con la opción `bratio` que marca un criterio de aceptación o rechazo de la misma.

Como ejemplo del impacto en el tiempo de optimización del aprovechamiento de la base, un problema LP de 8000 restricciones, 10000 variables y 30000 elementos requiere 10.3, 4.4, 4.7 y 2.6 segundos en sucesivas resoluciones.

A pesar de ello esta ventaja puede no ser suficiente para ciertos tamaños como para superar al método de punto interior. Por ejemplo, aproximadamente a partir de 10000 restricciones por 10000 variables el método de punto interior resulta más competitivo que el simplex aun comenzando éste con una base previa de un problema anterior.

DETECCIÓN DE INFACTIBILIDADES

Un método muy sencillo aunque laborioso y que puede producir problemas de consistencia y de dimensiones, es introducir variables de holgura en cada restricción y penalizarlas en la función objetivo.

Alternativamente algunos optimizadores tienen un parámetro que detecta el núcleo menor de restricciones infactibles de un problema (parámetro *Irreducible Infeasible Subsets* `iis`) y, por consiguiente, ayudan a localizar su posible causa. Una vez conocidas el desarrollador debe modificar o eliminar alguna del conjunto para que el problema se haga factible.

ANÁLISIS DE SENSIBILIDAD

Proporciona información adicional sobre la solución de un problema de optimización lineal.

Algunos optimizadores permiten realizar directamente un análisis de sensibilidad a cambios en los coeficientes de la función objetivo que no producen una alteración de la base óptima o a cotas de las restricciones que no producen pérdida de factibilidad (parámetros `objrng, rhsrng`).

I.3.3.2 Específicos de GAMS

Existe un informe técnico que incluye un conjunto general de recomendaciones para el desarrollo y la reparación de modelos escritos en GAMS, [McCarl, 1998].

USO AVANZADO PARA OPTIMIZACIÓN

En este apartado se desarrollan algunas consideraciones que permiten la implantación avanzada de modelos escritos en GAMS. Estas recomendaciones recogen la experiencia práctica adquirida en años de uso del lenguaje GAMS, principalmente en modelos de optimización lineal y estocástica, de ahí el valor que tienen a la hora de implantar problemas de optimización de gran tamaño.

Una implantación ingenua (de novato, "no profesional") de un problema de optimización puede llevar aparejado un consumo excesivo de recursos computacionales. Los más relevantes son el *tiempo de ejecución* y/o la *memoria*. El tiempo de ejecución de un modelo es especialmente crítico en aplicaciones de muy gran tamaño (e.g., a partir de 100000 restricciones por 100000 variables en el caso lineal) o, sobre todo, en el caso de resolución iterativa de numerosos (e.g., más de 100) problemas de optimización de mediano tamaño (como sucede en los métodos de descomposición o en la simulación de Monte Carlo). Los requisitos de memoria pueden ser limitativos en el caso de problemas de muy

gran tamaño. Algunas de las acciones que permiten reducir tiempo también disminuyen los requerimientos de memoria.

Los métodos de descomposición no son más que técnicas matemáticas que permiten resolver problemas gigantescos (por ejemplo, de más de 1 millón de restricciones y variables) con una estructura especial, que ni siquiera se pueden formular explícitamente, mediante la solución iterativa de problemas de menor tamaño. Como ejemplo se puede mencionar el caso de un problema de coordinación hidrotérmica en un sistema eléctrico cuya resolución se efectúa mediante descomposición anidada estocástica, ver Jacobs (1995).

Los valores numéricos que se aportan para contrastar el impacto de algunas recomendaciones deben tomarse como indicaciones relativas de las mejoras esperables nunca como seguros. Piénsese que cualquier mejora está asociada a un tipo de problemas, no necesariamente es generalizable para todos.

El tiempo de ejecución de un modelo escrito en GAMS se puede descomponer en estos tres tipos principales[8]:

- tiempo de *creación*
  formulación del problema de optimización específico, es decir, creación de las variables y de las restricciones.

- tiempo de *interfaz*
  escritura del problema de optimización en disco para su lectura por el optimizador y viceversa.

- tiempo de optimización
  resolución del problema de optimización por parte del optimizador.

Además de éstos hay que añadir el tiempo de compilación del modelo. Sin embargo, este tiempo se da únicamente una vez al comienzo y habitualmente es despreciable frente al resto.

El valor e importancia de cada uno de estos tiempos se puede conocer con las opciones `stepsum`, que resume el consumo de tiempo entre llamadas al optimizador, y `profile`, que informa sobre el consumo de tiempo y memoria en cada instrucción del código. Antes de iniciar las acciones de mejora es necesario realizar un análisis de los consumos de tiempo del modelo y de cómo se reparten.

[8] Esta clasificación del tiempo de ejecución de un modelo en tres componentes es relevante para los modelos escritos en GAMS. Quizá con otros lenguajes de modelado alguno de estos tiempos puede ser despreciable.

La relación entre ellos depende de las diversas características del problema: tamaño y estructura de la matriz de restricciones, número de optimizaciones, variación de los parámetros en sucesivas optimizaciones, como más importantes. Las direcciones de mejora que se presentan a continuación tienen una orientación o bien informática o bien matemática, aunque indudablemente en el tiempo de ejecución resultante influyen ambas. Las primeras están basadas en el uso del lenguaje GAMS. Las segundas modifican el problema o su resolución. La efectividad de cada mejora dependerá de las características del problema de optimización. Se sugieren algunos criterios heurísticos que permiten utilizarlas adaptándose al caso concreto tal como se menciona posteriormente. Éstos han de tomarse con cautela. En ningún momento se les quiere dar a estos criterios más que un valor indicativo, ajeno a cualquier tipo de generalización.

Para dar resultados numéricos en algunas de las mejoras estudiadas se han utilizado dos casos de estudio de optimización lineal. El primer caso es un problema de optimización de 12 etapas con un tamaño aproximado de 20000 x 25000 con 80000 elementos en la matriz de restricciones. El segundo caso tiene 6 etapas y un tamaño de 50000 x 65000 con 200000 elementos. Las referencias de tiempo se han tomado utilizando un ordenador personal con procesador Pentium II a 233 MHz con suficiente memoria RAM (96 MB y cuyo tiempo de acceso es de 60 ns) y con disco SCSI (tiempo de acceso xx ns). En un PC con otra frecuencia del procesador los tiempos serían aproximadamente proporcionales.

USO DE UN DISCO VIRTUAL

El tiempo de interfaz se debe a la escritura de los ficheros[9] de comunicación con el optimizador. El tiempo del proceso de escritura depende del hardware del equipo utilizado, en particular, del manejo y tamaño de la memoria caché[10] y del tiempo de escritura en el disco. A su vez, el disco puede ser local (localizado en la máquina que ejecuta el modelo) o remoto (conectado a través de una red de área local).

La minimización del tiempo de interfaz exige un conocimiento detallado de los recursos de hardware utilizables. Por ejemplo, uso de discos locales en lugar de remotos, discos rápidos (SCSI) en lugar de lentos (IDE), tamaños elevados memorias caché, etc. En cualquier caso, una solución sencilla en un PC es la creación y uso de discos virtuales localizados en memoria RAM (utilidad RAMDISK), siempre más rápida que los discos magnéticos. Un tamaño de 16 MB de disco RAM es suficiente para estos casos de estudio.

[9] En GAMS la comunicación entre el lenguaje y los optimizadores se hace mediante ficheros. En otros lenguajes esta relación se establece a través de variables localizadas en la memoria principal.

[10] La memoria caché mantiene una copia de la última información leída o escrita en disco, de manera que pueden evitarse accesos a disco cuyo tiempo de acceso es superior.

La reducción en tiempo esperable depende de la relación entre el tiempo de acceso al disco frente al acceso a la RAM. Por ejemplo, para el anterior PC se ha obtenido una reducción en tiempo de un 20 % del tiempo de interfaz.

CAMBIOS EN INSTRUCCIONES DE ASIGNACIÓN

GAMS es un lenguaje "peligroso" desde el punto de vista de consumo de tiempo por su naturaleza intrínseca, ser muy compacto y de alto nivel. Es relativamente fácil escribir instrucciones sencillas que involucren entidades con múltiples dimensiones que consuman un tiempo y/o memoria elevada. La opción `profile` permite conocer el consumo de tiempo y memoria y el número de asignaciones realizadas en cada instrucción.

Como recomendaciones específicas para no desperdiciar tiempo:

- El orden de colocación de los índices/dimensiones debe ser consistente para todos los parámetros, ecuaciones y variables.
- Se debe pensar desde el punto de vista de una ordenación natural de todos los índices para el conjunto del problema. Esta ordenación influye también en la formulación de las ecuaciones.
- El orden de colocación de los índices en las instrucciones reiterativas (sumatorios, productorios, bucles) debe ser el mismo que en los parámetros, variables o ecuaciones que se estén manipulando.
- Se debe hacer un uso extensivo de la exclusión mediante condiciones en asignaciones (uso del operador `$`) controladas preferentemente mediante conjuntos dinámicos (mejor que con valores de parámetros), es decir, activar sólo las variables y restricciones necesarias. En el ejemplo previo del flujo de cargas se observa que se consideran sólo las líneas eléctricas que existen y no cualquier posible conexión entre dos nudos.

TAMAÑOS MÁXIMOS ALCANZADOS

Como referencia final una indicación sobre el tamaño de los problemas que se están resolviendo y a los que se ha llegado aplicando estas recomendaciones. Se han podido resolver sin dificultad problemas de 117000 restricciones por 225000 variables con 655000 elementos no nulos en la matriz de restricciones en 240 segundos.

ALGUNAS INSTRUCCIONES ADICIONALES

A continuación se presenta una miscelánea de instrucciones de GAMS que no están suficientemente documentadas en el manual de usuario o se han utilizado con otra perspectiva.

Máxima supresión de información de salida

La supresión de la información de salida en el `nombre_fichero.lst` se consigue con las siguientes opciones.

```
$OFFSYMLIST, OFFSYMXREF, OFFUELLIST, OFFUELXREF
OPTION LIMROW=0, LIMCOL=0, SOLPRINT=OFF, SYSOUT=OFF
nombre_modelo.SOLPRINT=2 ;
```

y escribiendo en la invocación de GAMS

```
gams nombre_modelo.gms suppress 1
```

Además, también se puede suprimir la información en pantalla que produce el optimizador con los consiguientes parámetros (por ejemplo, para CPLEX `simdisplay 0 bardisplay 0` y `mipdisplay 0`).

```
gams nombre_modelo.gms ll 0 lo 0
```

Presentación de información por pantalla

Las siguientes instrucciones permiten la definición de la pantalla para posteriormente escribir información en ella. En el uso real hay que tener en cuenta posibles buffers que hacen que esta información no se muestre inmediatamente.

```
$SET CONSOLA
$IF %system.filesys% == UNIX $SET CONSOLA /dev/tty
$IF %system.filesys% == MS95 $SET CONSOLA CON
$IF %system.filesys% == MSNT $SET CONSOLA CON
$IF "%consola%." == "." ABORT "Fichero no reconocido" ;
FILE PANTALLA / '%consola%' / ;
```

Opciones `SAVE` y `RESTART`.

Permiten la segregación de una parte de código para su depuración evitando su ejecución completa.

También permiten la creación y distribución de una versión ejecutable, es decir, aquélla donde el usuario final no tiene acceso a la definición del problema de optimización. Para ello el código se separa en dos partes. La primera contiene las declaraciones y definiciones de variables y ecuaciones y la segunda la inclusión de ficheros de datos y resolución del problema.

Esta opción también puede utilizarse para paralelizar bucles[11]. La parte común se genera con la instrucción `SAVE` en el procesador principal. Después, la parte paralelizada (cada ciclo del bucle) se ejecuta con un `RESTART` en cada procesador independiente y asíncronamente. Una vez terminadas todas las ejecuciones se integran los resultados obtenidos.

Recorrido inverso de un índice

[11] El IIT ha desarrollado una utilidad que permite la ejecución asíncrona de scripts de UNIX que pueden ser utilizados para la paralelización de aplicaciones en GAMS.

La siguiente instrucción permite recorrer el parámetro `PP` en sentido inverso

```
PP(i+[card(i)-2*ord(i)+1])
```

empezando por `card(i), card(i)-1, .... 2, 1`.

Eliminación de las variables fijas

Se trata de aquellas variables cuyas cotas inferior y superior coinciden y el mismo lenguaje las convierte en parámetros, de manera que no son consideradas como tal por el optimizador. Por consiguiente, no se puede obtener información dual sobre ellas.

```
nombre_modelo.HOLDFIXED = 1 ;
```

Otras características no documentadas en el manual

`SAMEAS(elemento_de_set1.elemento_de_set2)`

Función que devuelve verdadero si las cadenas de caracteres de los nombres de los elementos de set son iguales o falso en caso contrario.

`DIAG(elemento_de_set1.elemento_de_set2)`

Devuelve 1 en el caso de igualdad y 0 en caso contrario.

`$CALL`

Llamada externa que se ejecuta en el momento de la compilación.

`$EXECUTE`

Llamada externa a una aplicación que devuelve el control a GAMS cuando ésta finaliza.

```
option SOLSLACK = 1
```

Presenta el valor de las variables de holgura de las restricciones en lugar del valor de la restricción como tal.

Utilidades complementarias

Existen algunas aplicaciones conectadas con GAMS que añaden funcionalidad, facilitan la interfaz con el lenguaje o la presentación de resultados. Las nueve primeras se apoyan en la instrucción `$libinclude`. Entre ellas cabe citar:

Aplicaciones de análisis, depuración y mejora de modelos

`gams-f`

Permite definir funciones que posteriormente son sustituidas mediante un preproceso en las definiciones de parámetros o ecuaciones.

`gamschk`

Aplicación que permite examinar empíricamente modelos escritos en GAMS para detectar posibles errores.

`gamsbas`

Aplicación que permite guardar la información relativa a la base que posteriormente puede ser utilizada para modelos subsiguientes.

Exportación del modelo a otros sistemas

GAMS permite la exportación del problema de optimización en formato MPS o LP que pueden ser leídos por numerosos optimizadores. En el fichero MPS se define la matriz de restricciones del problema, vista por columnas, y las cotas de las variables. Los nombres de las restricciones y de las variables están limitados a 8 caracteres y el formato de los datos es fijo por columnas. En el fichero LP se define el problema de forma más natural al poner directamente las expresiones de las ecuaciones pero utilizando nombres de las variables no indexados.

Interfaz con una hoja de cálculo

`ssimport.gms`
Lee datos de una hoja de cálculo durante la compilación.
`ssdump.gms`
Escribe datos y etiquetas en una hoja de cálculo. Admite tamaños dinámicos.
`ssexport.gms`
Escribe datos en una hoja de cálculo. Los intervalos de escritura son fijos.

Interfaz de presentación de resultados

Algunas de estas utilidades se pueden usar para realizar interfaces más sencillas con bases de datos.

`gams2tbl.gms`
Facilita la escritura automática de informes en forma de tablas.
`gams2txt.gms`
Escribe en un fichero los valores de parámetros, variables, ecuaciones o sets.
`gams2prm.gms`
Escribe en un fichero la declaración y valores de parámetros, variables. ecuaciones o sets.
`gams2zip.gms`
Escribe en un fichero comprimido la declaración y valores de parámetros, variables, ecuaciones o sets.
`zip2gams.gms`
Recupera de un fichero comprimido la declaración y valores de parámetros, variables, ecuaciones o sets.
`gnuplot.gms`
Permite la creación de gráficos con GNUPLOT que representen valores de parámetros.

Interfaz con MATLAB

`matout.gms` y `gams.dll`

Permite el uso desde MATLAB de las capacidades de optimización que proporciona GAMS y la facilidad de visualización de MATLAB de los resultados de una optimización.

I.3.3.3 Referencias

Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E. and Wunderling, R. (2000) *MIP: Theory and Practice - Closing the Gap*. Technical Report.

Fourer. R. (1999) "Linear Programming" *OR/MS Today* pp. 64-71. August.

McCarl. B. A. (1998) *So Your GAMS Model Didn't Work Right. A Guide to Model Repair*. Technical Report.

Jacobs. J., Freeman. G., Grygier. J., Morton. D., Schultz. G., Staschus. K. and Stedinger. J. (1995) "SOCRATES: A system for scheduling hydroelectric generation under uncertainty" *Annals of Operations Research* 59. pp. 99-133.

Ramos. A. et al. (1996) "Computational Experience with Optimization for a Bulk Production Cost Model" *12th PSCC*. Dresden, Germany.