Framework de desarrollo C++para resolución de problemas de programación lineal

Manual de usuario

Versión 1.0

infozara.es

SOBRE INFOZARA

Infozara es una empresa que se constituyó en 2006 como spin off de la Universidad de Zaragoza a través del Grupo Nóesis, Grupo Consolidado de Investigación Aplicada del Gobierno de Aragón (España) dirigido por el Profesor Eladio Domínguez.

Infozara tiene una amplia experiencia en la realización de proyectos de I+D+I y en la prestación de servicios a clientes, que cuentan con un alto nivel de valor añadido derivado de las investigaciones industriales realizadas en dichos proyectos.

Todos los productos y desarrollos se han realizado para ser explotados a través de la Web, bajo la forma de lo que actualmente se llama servicios SaaS (Software as a Service).

PROPIEDAD Y CONFIDENCIALIDAD

ÍNDICE

Contents

1 Introducción

Existe una gran cantidad de problemas de optimización que se pueden modelar mediante programación lineal. Cuando se resuelve un problema de programación lineal, es necesario utilizar software de cálculo que aplica algoritmos específicos para resolver este tipo de problemas. Este software es conocido en general como *solver*. Existen varios solvers comerciales y también solvers de uso libre. Estos solvers ofrecen un API para definir modelos de programación lineal y acceder a los motores de cálculo. Cada solver presenta un API diferente, de forma que cambiar el solver de resolución de programación lineal, es prácticamente hacer un nuevo programa. Para poder utilizar cada uno de los solvers, el programador debe estudiar a fondo el manual de usuarios del solver y conocer al detalle el modelo de funcionamiento.

Lin2any es un framework de desarrollo en C++, que permite modelar problemas de programación lineal, y resolverlos con cualquier solver. Lin2any establece una capa de abstracción para modelar problemas de programación lineal, utilizando los conceptos que se utilizan en la programación lineal, y después permite utilizar un solver, sin que el programador sepa nada del solver que está utilizando. Lin2any permite cambiar el solver, con solo cambiar una línea de código fuente.

Este documento es el manual de usuario del framework Lin2any, y presenta toda la información necesaria para poder utilizarlo, con varios ejemplos. Este documento es también una guía de usuario que describe todas las clases y funciones del framework.

Este manual está dirigido a programadores que desarrollan aplicaciones de programación lineal, con conocimientos de modelos de programación lineal.

1.1 En este manual

Este manual está estructurado es los siguientes capítulos:

- Programación lineal

  En este capítulo se describe la programación lineal, ya que Lin2any es un framework que resuelve problemas de programación lineal. No es un manual completo de programación lineal, sino que muestra los conceptos generales, que serán útiles para un buen uso del framework Lin2any.

- Framework Lin2any

  Se ofrece una guía detallada para un correcto uso del framework Lin2any. Se comienza con el modelo conceptual del sistema, se describen los pasos a seguir para construir modelos de programación lineal, y presenta varios ejemplos de uso.

- Guía de usuario

  Se detallan todas las clases y funciones del framework Lin2any.

1.2 ¿Quiénes somos?

Infozara es una empresa que se constituyó en 2006 como spin off de la Universidad de Zaragoza a través del Grupo Nóesis, Grupo Consolidado de Investigación Aplicada del Gobierno de Aragón (España) dirigido por el Profesor Eladio Domínguez.

Infozara tiene una amplia experiencia en la realización de proyectos de I+D+I y servicios con un alto nivel de valor añadido, derivado de las investigaciones industriales realizadas en dichos proyectos.

Una característica común a todos los proyectos ha sido la construcción, en cada uno de ellos, de productos en estado precompetitivo y el desarrollo de una metodología de construcción industrial del software como parte integral de un servicio.

Desde su fundación, Infozara:

- Ha participado en diversos proyectos de Desarrollo e Investigación Industrial destacando los proyectos SPOCS (www.spocs.es), LISBB (www.lisbb.es), QRP (qrp.infozara.es), AMBÚ (ambu.infozara.es) y SMOTY (www.smoty.es) del Plan nacional de I+D+i.
- Ha desarrollado productos y componentes industriales en estado precompetitivo en el marco de los proyectos anteriores o como desarrollo posterior ante demanda del mercado.
- Ha construido y está construyendo servicios en la Cloud y en el ámbito del Internet de las Cosas (IoT).
- Tiene una cartera de clientes a los que se les está ofreciendo servicios de valor añadido.

Todos los productos y desarrollos se han realizado para ser explotados a través de la Web, bajo la forma de lo que actualmente se llama servicios SaaS (Software as a Service).

1.3 Conocimientos necesarios

Este manual asume que el lector tiene los conocimientos necesarios para plantear modelos de programación lineal. Como Lin2any es un framework escrito para programadores en C++, este manual asume que el lector tiene experiencia en el desarrollo de programas en C++, y tiene conocimientos de uso de algún entorno de desarrollo en C++.

1.4 Lo que no es Lin2any

Lin2any no es un solver de programación lineal, es decir, no resuelve directamente problemas de programación lineal.

Lin2any permite modelar y resolver problemas de programación lineal con un interface único, que abstrae del solver de programación lineal utilizado.

1.5 Lo que no incluye Lin2any

Lin2any no incluye licencias, ni ningún tipo de derecho sobre ningún solver de programación lineal. Antes de usar Lin2any, usted debe adquirir licencias del solver de programación lineal que desee utilizar.

1.6 Marcas registradas

Lin2any 2.1 ofrece interface a los solvers de programación lineal GLPK[(1)] y Gurobi [(2)].

(1) The GLPK package is part of the GNU Project released under the aegis of GNU.

Copyright c 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2013, 2014, 2015 Andrew Makhorin, Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia. All rights reserved.

Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.

(2) Gurobi Optimization, Inc., Gurobi, and the Gurobi logo and design are registered trademarks or trademarks of Gurobi Optimization, Inc. All other company and product names are registered trademarks or trademarks of their respective holders, and are hereby acknowledged as such by Gurobi Optimization, Inc.

Microsoft® y Windows® son marcas registradas de Microsoft Corporation en los EE.UU.

1.7 Nomenclatura

A lo largo del presente documento se hará referencia a los siguientes términos:

LP	*Linear programming*, programación lineal
ILP	*Integer linear programming*, programación lineal entera.
um	Unidades monetarias.

Las direcciones web o direcciones de correo electrónico que se referencian en este documento, se muestran en color azul, como www.vetu.es/webvetu/lin2any.do.

1.8 Soporte

Si es usted ha adquirido licencias de Lin2any, puede obtener soporte técnico sobre el uso del framework, poniéndose en contacto con el soporte técnico de Infozara.

Si no es usted no ha adquirido licencias del framework de Infozara, y tiene cualquier duda o sugerencia, puede ponerse en contacto con el equipo de Infozara por los mecanismos descritos en el apartado 1.10.

1.9 Actualización de versiones

Lin2any es una pasarela a diferentes solvers de programación lineal. Si usted ha contratado el servicio de mantenimiento de Lin2any, recibirá actualizaciones a medida que los solvers vayan publicando nuevas versiones. Los proyectos desarrollados con Lin2any no se verán afectados por los cambios de versión, ya que el interface de Lin2any para modelar problemas de programación lineal no cambiará.

1.10 Más información

Si desea más información sobre cualquier aspecto del framework Lin2any, puede ponerse en contacto con Infozara, a través de los siguientes medios:

Teléfono: +34 976 25 43 76

Correo electrónico: informa@infozara.es

También puede consultar las siguientes direcciones web:

www.infozara.es

www.vetu.es/webvetu/lin2any.do

www.vetu.es

2 Manual de usuario

2.1 Introducción

Lin2any es un framework C++ para modelar problemas de programación lineal (LP) y programación lineal entera (ILP), y resolverlos mediante diversos solvers comerciales.

2.2 Contexto

La programación lineal permite modelar y resolver problemas de asignación de recursos. Para resolver un problema de programación lineal, es necesario utilizar un solver, que es un software especializado. Existen varios solvers comerciales de programación lineal, además de varios solvers de uso gratuito. Estos solvers ofrecen una biblioteca de programación para acceder a sus motores de resolución de problemas. Cambiar de solver implica cambiar el programa en su mayor parte, y además requiere de conocimientos específicos del solver.

Lin2any crea una capa de abstracción para modelar problemas de programación lineal, que después se pueden enviar a varios solver, cambiando una sola línea de código fuente.

Lin2any permite modelar el problema mediante conceptos generales de programación lineal como variables, restricciones o función objetivo, y no requiere conocimientos de ningún solver específico.

2.3 Programación lineal

Lin2any asume la siguiente formulación para un problema de programación lineal (LP):

Minimizar $Z = c_1 \cdot x_1 + c_2 \cdot x_2 + \ldots + c_n x_n$ (1)

Sujeto a:

$c_{11} \cdot x_1 + c_{12} \cdot x_2 + \ldots + c_{1n} x_n \leq b_1$

$c_{21} \cdot x_1 + c_{22} \cdot x_2 + \ldots + c_{2n} x_n \leq b_2$ (2)

…

$C_{m1} \cdot x_1 + c_{m2} \cdot x_2 + \ldots + c_{mn} x_n \leq b_m$

$x_1 \geq 0, x_2 \geq 0, \ldots, xn \geq 0$ (3)

El anterior modelo se conoce como la forma estándar de un modelo de programación lineal, donde $x_1, x_2, \ldots, x_n$ son las variables de decisión, Z es la función objetivo, $c_1, c_2, \ldots, c_n$ son los coeficientes de las variables en la función objetivo, las ecuaciones lineales (2) son las restricciones funcionales, $b_1, b_2, \ldots, b_n$ se conocen como el lado derecho de las restricciones, y las ecuaciones (3) son las restricciones de no negatividad, que definen los valores que pueden tomar las variables.

Además de la forma estándar, la programación lineal admite otras formas de modelo:

- Función objetivo a maximizar
- Restricciones funcionales de igualdad: $c_{i1} \cdot x_1 + c_{i2} \cdot x_2 + \ldots + c_{in} x_n = b_i$

- Restricciones funcionales de en sentido mayor o igual $c_{i1} \cdot x_1 + c_{i2} \cdot x_2 + \ldots + c_{in} x_n \geq b_i$
- Otros dominios de definición de la variables: $l_i \leq x_i \leq u_i$

Cualquier combinación de estas formas de modelo llevan a un modelo de programación, que se pude resolver mediante las técnicas de resolución conocidas para este tipo de problemas.

Resolver el problema de programación lineal es encontrar el valor de las variables de decisión que:

- satisfacen todas las restricciones (2),
- respetan el dominio de definición de esas variables (3), y
- proporcionan el menor (o mayor) valor de la función objetivo

2.4 Programación lineal entera

Un problema de programación lineal entera (ILP) es un modelo de programación lineal (LP), donde algunas variables añaden la condición de que su valor debe ser un número entero. Un modelo de problema ILP tiene la misma formulación que un modelo LP, con la restricción adicional de valor entero de algunas variables.

2.5 Contenido

Al adquirir una licencia del framework Lin2any, Infozara envía a cliente el framework de desarrollo Lin2any, con todo el código fuente para:

- Facilitar la integración del framework en el proyecto de desarrollo del cliente.
- Facilitar las tareas de extensión del framework de desarrollo, según las necesidades de cada proyecto.
- Facilitar la depuración del proyecto, pudiendo navegar por el interior del código fuente, y obtener así mejor información para la detección de errores.

El framework Lin2any está escrito en C++, y no está concebido para ningún entorno de desarrollo ni sistema operativo en especial. El lenguaje C++ se puede compilar y ejecutar en todos los sistemas operativos. Lin2any está escrito en C++ estándar, y solo el proceso de verfificación de licencias hace llamadas al sistema operativo.

2.6 Instalación

El framework Lin2any se distribuye como un fichero comprimido con el código fuente y la documentación, según la siguiente estructura:

\include	Contiene los ficheros de cabecera .h
\lib	Contiene le fichero de biblioteca Lib2any.lib
\solver	Contiene el código fuente (ficheros .h y .cpp) de los solver para GLPK y Gurobi
\doc	Contiene el manual de usuario

La instalación consiste en descomprimir el fichero para obtener la estructura anterior, y copiar los ficheros al directorio de proyecto de desarrollo.

2.7 Licencias

Lin2any se licencia por cada ordenador donde se utiliza el framework. Las licencias van asociadas a la mac address de cada ordenador.

El proceso de validación de licencias es el siguiente:

- Localizar la MAC ADDRESS del ordenador donde se va a instalar la licencia. Para obtener la MACC ADDRESS en un ordenador son sistema operativo Windows, se debe teclear ipconfig /all en la línea de comandos, y obervar el valor 'Physical Address'.
- Enviar la MAC ADDRESS al servicio técnico de Inforzara.
- El servicio técnico de Inforzar enviará al cliente un fichero de licencia (.lic).
- Colocar el fichero de licencia en el directorio donde esté cada ejecutable que incluye al framework Lin2any.

2.8 Configuración

Para poder hacer uso del código fuente de Lin2any en un proyecto C++:

- Se debe añadir las directivas #include con los ficheros .h correspondientes a las clases de Lin2any.
- Se debe añadir al proyecto la biblioteca Lin2any.lib, y asegura que el proyecto enlaza con ella.
- Se Añade la clase solver que se dessse utilizar. Con Lin2any se entrega el códig fuente esta clases, porque es responsabilidad del programador el realizar la configuración necesari apra la correcta complicación y enlazado es estas clase, con el solver que ha adquirido.

2.9 Modelo conceptual

La Figura 1 muestra un diagrama de clases con el modelo conceptual de Lin2any. El modelo está divido en dos partes muy diferenciadas: la primera parte corresponde a la construcción del modelo matemático de programación lineal que representa a un problema, y la segunda corresponde a la resolución del problema.

El modelo se construye según la terminología de la programación lineal y no tiene relación con ningún solver. El modelo (LPModel) se construye creando variables (LPVariable), añadiendo restricciones (LPConstraint), y fijando los coeficientes (LPCoefficient) de la función objetivo.

La resolución del problema se hace mediante clases específicas para cada solver (LP_GLPKSolver para el solver GLPK y LP_GurobiSolver para el solver Gurobi), que derivan de la clase abstracta LPSolver, que define los métodos para que LPModel traslade la información del modelo al solver genérico LPModel.

La unión de la representación del modelo de programación lineal y la resolución del problema viene dada por la relación entre LPModel y LPSolver. Como LPSolver es una clase abstracta, LPModel nunca sabrá cuál es el solver que va a resolver el problema. Para añadir un nuevo solver en un programa, se debe crear una nueva clase que derive de LPSolver y que sobrescriba todos sus métodos abstractos. Desde ese momento, se podrá utilizar el nuevo solver sin que se haya modificado para nada la representación del problema. Otra opción interesante que admite este modelo conceptual es que se pueden crear clases especializadas

de LP_GLPKSolver o de LP_GurobiSolver, que permitan ajustar el valor de parámetros específicos del solver particular, para sacar mayor rendimiento de ejecución para un tipo de problema concreto. Este trabajo requiere un cierto conocimiento específico de cada solver.

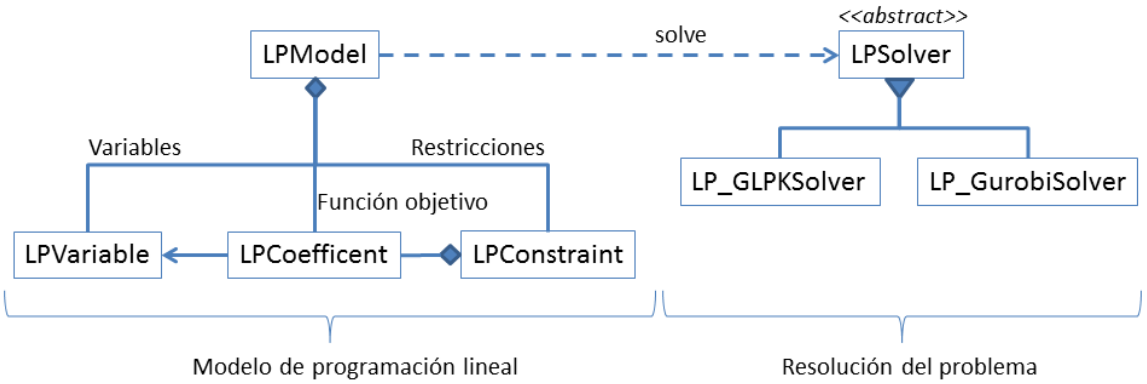

Figura 2-1. Modelo conceptual de Lin2any

El modelo de Lin2any es una variante de bien conocido patrón de diseño Bridge[1], cuya idea es desacoplar una abstracción de su implementación, para que ambas puedan variar independientemente. En el caso de Lin2any, la abstracción es el modelo de programación lineal que representa a un problema, y la implementación corresponde al solver que resuelve el problema. Con este modelo, se puede cambiar el solver sin tocar el modelo, o incluso se puede resolver el problema con varios solvers dentro de un mismo programa.

2.10 Ejemplo sencillo

Se va a mostrar el uso del framework Lin2any a través de un sencillo ejemplo. Sea el siguiente problema:

> 'En una fábrica se elaboran tres tipos de herramientas: A, B y C. En la fábrica trabajan 3 obreros durante 8 horas diarias y un revisor, para comprobar las herramientas una vez construidas, que trabaja 1 hora diaria. Para la construcción de A se emplean 3 horas diarias de mano de obra y precisa de 6 minutos de revisión, para la construcción de B se emplean igualmente 3 horas de mano de obra y 4 minutos para su revisión, y para C es necesaria 1 hora diaria de mano de obra y 3 minutos para su revisión. Por problemas de producción en la fábrica no se pueden fabricar más de 12 herramientas diarias y el precio de cada herramienta A, B y C es de 4000, 3000 y 2000 um respectivamente. Hallar cuántas unidades se deben elaborar cada día de cada una de ellas para obtener un beneficio máximo.'

El modelo de programación lineal que representa al problema anterior es el siguiente:

$$\text{Max} \quad Z = 4000\,x_1 + 3000\,x_2 + 2000\,x_3 \qquad [1]$$

$$\text{Sujeto a:} \quad 3\,x_1 + 3\,x_2 + x_3 \leq 24 \qquad [2]$$

$$6\,x_1 + 4\,x_2 + 3\,x_3 \leq 60 \qquad [3]$$

$$x_1 + x_2 + x_3 \leq 12 \qquad [4]$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0 \qquad [5]$$

[1] Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Addison-Wesley Professional Computing Series. ISBN: 0-201-63361-2.

donde:

x_1 : número de unidades diarias del tipo A

x_2 : número de unidades diarias del tipo B

x_3 : número de unidades diarias del tipo C

El programa que resuelve este problema con el framework Lin2any es el siguiente (se muestra el número de las líneas de código para facilitar la posterior explicación):

```
1     // Crea el modelo
2     LPModel oModel;
3
4     // Definición de variables
5     LPVariable* x1 = oModel.addVariable( Decimal, 0.0, 999999.0, "x1" );
6     LPVariable* x2 = oModel.addVariable( Decimal, 0.0, 999999.0, "x2" );
7     LPVariable* x3 = oModel.addVariable( Decimal, 0.0, 999999.0, "x3" );
8
9     // Definición de restricciones
10    LPConstraint* pC1 = new LPConstraint( MinorEqual, 24.0 );
11    pC1->add( new LPCoefficient( x1, 3 ) );
12    pC1->add( new LPCoefficient( x2, 3 ) );
13    pC1->add( new LPCoefficient( x3, 1 ) );
14    oModel.addConstraint( pC1 );

15    LPConstraint* pC2 = new LPConstraint( MinorEqual, 60.0 );
16    pC2->add( new LPCoefficient( x1, 6 ) );
17    pC2->add( new LPCoefficient( x2, 4 ) );
18    pC2->add( new LPCoefficient( x3, 3 ) );
19    oModel.addConstraint( pC2 );
20
21    LPConstraint* pC3 = new LPConstraint( MinorEqual, 12.0 );
22    pC3->add( new LPCoefficient( x1, 1 ) );
23    pC3->add( new LPCoefficient( x2, 1 ) );
24    pC3->add( new LPCoefficient( x3, 1 ) );
25    oModel.addConstraint( pC3 );
26
27    // Sentido y función objetivo
28    oModel.setDirection( Max );
29    oModel.addObjCoefficient( x1, 4000 );
30    oModel.addObjCoefficient( x2, 3000 );
31    oModel.addObjCoefficient( x3, 2000 );
32
33    // Resuelve con el solver GLPK
34    LP_GLPKSolver oSolver;
35    double dZ = 0;
36    try
37    {
38       dZ = oModel.solve( &oSolver );
39    }
40    catch( LPException* ex )
41    {
42       std::cout << ex->message();
43       delete ex;
44    }
45
46    // Obtiene la solución
47    double dX1 = x1->getValue();
48    double dX2 = x2->getValue();
```

```
49      double dX3 = x3->getValue();
```

Usualmente, los pasos para resolver un problema de programación lineal con Lin2any son: crear el modelo, crear las variables, añadir las restricciones, fijar los coeficientes de la función objetivo, resolver el problema y obtener el resultado. En el ejemplo anterio se sigue estos pasos, pero Lin2any es flexible en ese orden y a veces, especialmente en problemas grandes, para ahorrar tiempo de proceso, se pueden ir añadiendo restricciones y fijando los coeficientes de la función objetivo, a medida que se crean las variables de programación lineal.

En la línea 1 del código fuente del ejemplo anterior, se crea una instancia de modelo. Como se crrea por valor, al terminar el programa, se libera´la memoria de todo el modelo (LPModel libera la memoria de las variables, las restricciones y la función objetivo). En las líneas 5, 6 y 7, se insta al modelo para que cree tres variables decimales (o contínuas), que cuyo valor puede estar entre 0 y 999999, y que se llaman x1, x2, y x3, respectivamente. Al definir las variables, estamos modelando las restricciones de no negatividad del modelo matemático (ecuaciones [5] del modelo matemático). El programa guarda un punto a cada una de estas variables, para poder utilizarlas en la definición de las restricciones y la función objetivo. En los siguientes bloques de código fuente, se crean las restricciones. Los pasos para crear restricciones son: crear la restricción, añadir los coeficientes a la restricción, y añadir la restricción al modelo. En la línea 10 se crea una restricción de sentido 'menor o igual que 24', que corresponde a la ecuación [2] del modelo matemático. Una vez creada la restricción, en la línea 11 se ñade el sumando $3 \cdot x_1$ correspondiente a la ecuación [2] del modelo, en la línea 12 se añade el sumando $3 \cdot x_2$ de la ecuación [2] del modelo, y por último, en la línea 13 se añade el sumando $3 \cdot x_1$ de la ecuación. Una vez añadidos los coeficientes de la restricción, en la línea 14 se añade la restricción al modelo. De la misma forma, entre las líneas 15 y 19 del código fuente, se crea, define y añade la ecuación [3] del modelo matemático, y entre las líneas 21 y 25 se crea, define y añade la ecuación [4] del modelo matemático. En la línea 28 de fija la dirección del problema como un problema de maximización. En la línea 29 se fija el sumando 4.000·x1 de la función objetivo (ecuación [1] del modelo matemático), en la línea 30 se fija el sumanto 3.000·x2, y en la línea 31 se fija el sumando 2.000·x3, con lo que se tiene definida la función objetivo del modelo. El siguiente paso consiste en resolver el problema, que se hace en las líneas 34 y 38. El modelo necesita un solver para resolver el problema. En este caso, en la línea 34 se crea el solver correspondiente a la biblioteca GLPK, y se pasa como argumento a la función solve del modelo LPModel, en la línea 38. Cuando se ejecuta la instrucción de la línea 38, se resulve el problema. El último paso es obtener la solución, que se hace accediendo directamente al valor que toma cada una de las variables de decisión (líneas 47, 48 y 49).

En este sencillo ejemplo, se ponen de manifiesto las ventajas que tiene utilizar Lin2any, frente a programar directamente sobre el solver específico:

- Lin2any utiliza los conceptos fundamentales de la programación lineal (modelo, variable, restricción, función objetivo), con lo que un programador con conocimientos básicos de C++, y con fundamentos en programación lineal, puede resolver modelos de programación lineal en cuestión de minutos.
- Se puede cambiar el solver de resolución del problerm, cambiando una línea de código: si se quiere utilizar el solver de Gurobi, se debe cambiar la línea 34 por la siguiente:

  ```
  34 // Resuelve con el solver de Gurobi
  35 LP_GurobiSolver oSolver;
  ```

  Esto es así, porque Lin2any separa la representación del modelo del problema, de su proceso de resolución.

- No es necesario conocer el API de ningún solver de programación lineal.

La solución del modelo anterior es:

x_1 = 6 herramientas diarias del tipo A

x_2 = 0, no se fabrican herramientas del tipo B

x_3 = 6 herramientas diarias del tipo C

Z = 36.000 um de beneficio

2.11 Modelo de programación

En el ejemplo del apartado anterior, se ha visto que construir el modelo de un problema de programación lineal con Lin2any, casi consiste en transcribir el modelo de programación llineal al dominio de clases que define Lin2any. Esta simlitud se contrasta porque es un ejemplo my sencillo. En problema smás complicados, está frma de trabajar puede no ser la mejor. Lin2any propone un modelo de progrmación donde se separan el dominio del problema, la representación matemática del modelo, y la resolución, según muestra la Figura 2-2. Por una parte, está el dominio del problema, que estará representado por una serie de clases que representan al problema. Por ejemplo en un probema de planificación de personal pueden aparecer las clases empleado, contrato o carga de trabajo, o en problema de rutas de vehículos puede aparecer las clases cmión y entrega. El dominio del problema construye el modelo matemático, creando la clase LPModel, a añadiendo las variables, restricciones y función objetivo. Es usual que las clases del dominio del problema contengan referencias a las clases del modelo matemático, sobretodo a las variables de programación lineal. Usualmente, las restricciones se construirán a partir la información de varias clases del dominio del problema. Una vez construido el problema, desde LPModel se resuelve el problema con un solver. La solución la recoge el dominio del problema a través de la clase LPVariable del modelo matemático. Para mostrar el modelo de programación que propone Lyn2any, en el siguiente apartado se muestra un ejemplo de un problema más avanzado que el del apartado anterior.

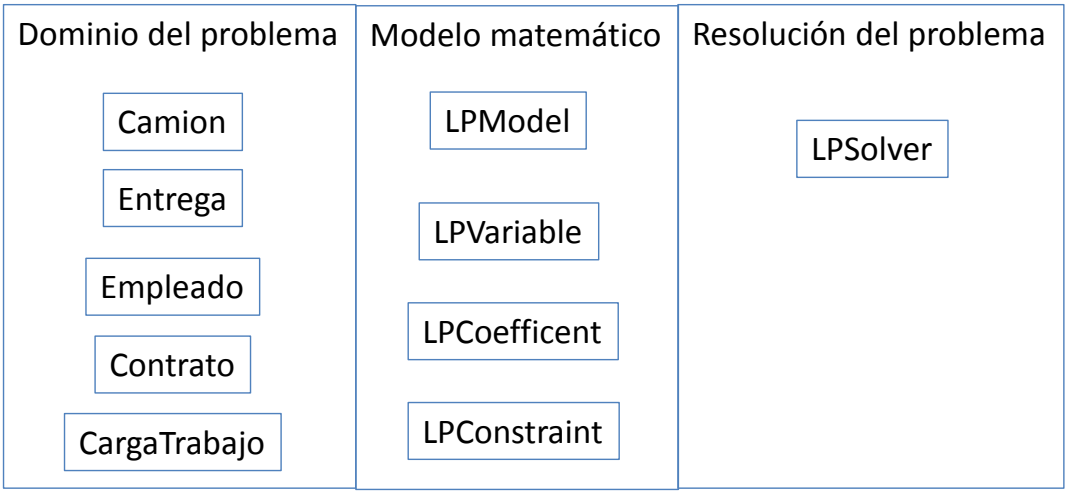

Figura 2-2. Modelo de programación

2.12 Extensión de un solver

Lin2any proporciona las clases ILP_GLPKSolver e ILP_GurobiSolver, que encapsulan la ejecución del modelo de progrmación lineal que contiene LPModel, sobre los solvers GLPK y Gurobi, respectivamente. Las implementaciones des estas clases, trasladan el modelo de programcacion lineal definido en LPModel, al dominio de programación que establece cada uno de los solver. El paso del modelo incluye las operaciones de crear las variables, las resttriciones, y la función objetivo. Pero los solvers ofrecen una serie de opciones de configuración, que no están recogidas en los solver que implementa Lin2any. Esto no significa que el pogramador deba renunciar a la características de cada solver por utilizar Lin2any. El programador puede beneficiarse de las ventajas de Lin2any sin perder la opción del uso de las características particulares que aporta cada solver. Para poder utilizar las características particulares de un solver desde Lin2any, sin romper el modelo de programación de Lin2any ni la compatiblidad de versiones de Lin2any, se deben extender las clases LP_GLPKSolver o LP_GurobiSolver, y pasar el puntero a esas clases derivadas a la función solve de LPModel. Como ejemplo, Gurobi ofrec la posibilidad de fijar el número de hilos de ejecución.Esta opción no la ofrece el solver genérico ILPSolver de Lin2any, ni el solver particular LP_GurobiSolver. Se debe crear la clase derivada de LP_GurobiSolver:

```
class CustomGurobiSolver : public LP_GurobiSolver
{
public:

	CustomGurobiSolver (void) {}
	virtual ~ CustomGurobiSolver (void) {}

	// Fija el número de hilos
	void setThreads( int nThreads )
	{
	    moModel.Params.Threads = nThreads;
	}
};
```

Para utilizar esta clase en un programa basado en Lin2any, por ejemplo para el problema TSP del apartafdo anterior, se debe hacer programar de la siguiente forma:

```
159	// Resuelve el problema
160	CustomGurobiSolver oSolver;
161	oSolver.setThreads( 4 );
162	double dDistancia = oModel.solve( &oSolver );
```

La extensión de las clases solver de Lin2any supone un ncierto nivel de conocimiento del modelo de programación del solver.

2.13 Añadir nuevo solver

Lin2any propone un modelo de representación de problemas de progmación lineal, independiente de la resolución del problema, mediante llamada a solvers independientes a Lin2any. Si el usuario de Lin2any dispone de otro solver diferente al que integra Lin2any, puede utilizar ese solver desde Lin2any, sin perder acceso a la ventaja que proporciona este framework. Para incorporar un nuevo solver al modelo de Lin2any, se debe extender la clase abstracta LPSolver, y encapsular la llamada al nuevo solver, sobreescribiendo todas las funciones de LPSolver. Se pueden tomar como ejemplo las implementaciones de las clases LP_GLPKSolver y LP_GurobiSolver.

3 Manual de referencia

Este capítulo hace referencia a todas las clases del framework Lin2any 2.1.

3.1 LPModel

Categoría Clase de desarrollo.

Descripción

LPModel contiene el modelo de programación lineal. Las variables de programación lineal las crea LPModel. Las restricciones y la función objetivo se añaden a LPModel.

Fichero include

```
#include <fstream>
#include <vector>
using namespace std;

#include "LPConstraint.h"

#include  "LPSolver.h"


class LPModel
{
	// Vector of variables
	vector<LPVariable*> m_vpVariables;

	// Constraints
	vector<LPConstraint*> m_vpConstraints;

	// Objetive function
	vector<LPCoefficient*> m_vpObjCoefficients;


	int	m_eDirection;
	int	m_eKind;

	// Maximum allowed time for the execution of the algorithm
	// (in seconds)
	int m_nTimeLimit;

public:

	LPModel(void);
	virtual ~LPModel(void);

	LPVariable* variable( int nIndex );
	int variablesCount(void) const;
	int constraintsCount(void) const;

	LPVariable*addVariable(	int eType, double dLowerBound,
				double dUpperBound,
				string strName = "" );

	void addConstraint( LPConstraint* pConstraint );

	LPCoefficient* addObjCoefficient( LPVariable* pVariable,
					double dCoefficient );

	void setDirection( int eDirection );

	void setKind( int eKind );

	void setTimeLimit( int nSeconds );

	virtual double solve( LPSolver* pSolver );
```

```
        void printModel( ofstream& os );
        void printSolution( ofstream& os );
};
```

Constructores

```
LPModel(void);
```

El constructor no recibe ningún parámetro.

Funciones miembro

```
LPVariable* variable( int nIndex );
```

Devuelve el puntero a una variable de programación lineal, dada por el índice de la variable.

```
int variablesCount(void) const;
```

Devuelve el número de variables de programación lineal que contiene el modelo.

```
int constraintsCount(void) const;
```

Devuelve el número de restricciones que contiene el modelo de programación línea

|

```
LPVariable* addVariable(    int eType, double dLowerBound,
                            double dUpperBound,
                            string strName = "" );
```

Añade una variable de programación lineal al modelo. eType indica el tipo de variable de programación lineal (Integer es una variable entera –toma valores enteros-, Decimal es una variable continua –toma números reales-, y Bynary es una variabe binaria –toma valores 0 o 1-). dLowerBound indica el mínimo valor que puede tomar la variable, y dUpperBound indica el máximo valor que puede tomar la variable. strName es el nombre opcional de la variable. Mientras se está depurando la construcción del modelo matemático, es muy útil dar un nombre a las variables, pero se debe vigilar especialmente que dos variables no tengan el mismo nombre.

La función devuelve un puntero a la variable que ha creado.

```
void addConstraint( LPConstraint* pConstraint );
```

Añade una restricción al modelo matemático. La restricción la debe crear la clase cliente de LPModel. pConstraint es la restricción que se añade al modelo.

```
LPCoefficient* addObjCoefficient( LPVariable* pVariable,
                                  double dCoefficient );
```

Añade la contribución de una variable a la funcion objetivo del modelo de programación lineal. pVariable es la variable que se añade la función objetivo, y dCoefficient es el coeficiente de esa variable en la función objetivo.

```
void setDirection( int eDirection );
```

Indica el sentido de optimización del problema. Si eDirection vale Max es un problema de maximizar, y si eDirection vale Min, es un problema de

minimizar.

```
void setKind( int eKind );
```

Indica si el modelo es un problema de progamción lineal continua, o un problema de problem de programación lineal entera. Si eKind vale CLP es un problema de programación lineal continua, y si eKind vale ILP, es un modelo de programación lineal entera.

```
        void setTimeLimit( int nSeconds );

        virtual double solve( LPSolver* pSolver );

        void printModel( ofstream& os );
        void printSolution( ofstream& os );
};
```

3.2 LPVariable

Categoría Clase de desarrollo.

Descripción LPVariable encapsula el comportamiento de una variable de programación lineal.

Estas variables no se pueden crear directamente, llamando al constructor de la clase, sino que debe crearlas la clase LPModel.

Fichero include

```
#include <string>
using namespace std;

#define Integer     0
#define Decimal     1
#define Binary      2

class LPModel;

class LPVariable
{
       friend class LPModel;

       int m_nIndex;
       string m_strName;
       int m_eType;
       double m_dLowerBound;
       double m_dUpperBound;

       double m_dValue;

protected:

       LPVariable(   int nIndex, int eType, double dLowerBound,
                     double dUpperBound, string strName = "" );

public:

       virtual ~LPVariable(void);

       int index(void) const;
       string name(void) const;
       bool isInteger(void) const;
       bool isBinary(void) const;
       double lowerBound(void) const;
       double upperBound(void) const;

       void setValue( double dValue );
       double getValue(void) const;
};
```

Constructores

```
LPVariable(   int nIndex, int eType, double dLowerBound,
              double dUpperBound, string strName = "" );
```

Crea una instancia de la clase LPVariable, que representa una vaaiable de programación lineal.

Index es el identificador numérico único de la variable, y lo fija LPModel. eType indica el tipo de variable de programación lineal (Integer es una variable entera –toma valores enteros-, Decimal es una variable continua –toma números reales-, y Bynary es una variabe binaria –toma valores 0 o 1-). dLowerBound indica el mínimo valor que puede tomar la variable, y dUpperBound indica el máximo valor que puede tomar la variable. strName es el nombre opcional de la variable. Mientras se está depurando la construcción del modelo matemático, es muy útil dar un nombre a las variables, pero se debe vigilar especialmente que dos variables no tengan el mismo nombre.

En el caso de una variable binaria (eType = Bynary), el sistema fija dLowerBound = 0 y dUpperBound = 1.

Funciones miembro

```
int index(void) const;
```

Devuelve el índice único de la variable.

```
string name(void) const;
```

Devuelve el nombre de la variable.

```
bool isInteger(void) const;
```

Indica si la vraiable se ha definido como entera.

```
bool isBinary(void) const;
```

Indica si la variable se ha definido como binaria.

```
double lowerBound(void) const;
```

Devuelve el menor valor que puede tomar la variable.

```
double upperBound(void) const;
```

Devuelve el mayor valor que pede tomar la variable.

```
void setValue( double dValue );
```

Fija el resultado que toma la variable, después de resolver el problema. Este valor lo fija la clase LPModel.

```
double getValue(void) const;
```

Devuelve el valor que toma la variable en la solucion del problema.

3.3 LPCoefficient

Categoría Clase de utilidad.

Descripción Representa un sumando de una ecuación lineal (constante multiplicada por variable), utilizada para las definiciones de la función objetivo y de las restricciones del modelo de programación lineal.

Fichero include

```
#include "LPVariable.h"

class LPCoefficient
{
	LPVariable* m_pVariable;
	double m_dCoefficient;

public:

	LPCoefficient( LPVariable* pVariable, double dCoefficient );

	virtual ~LPCoefficient(void);

	LPVariable* variable(void) const;
	double coefficient(void) const;
};
```

Constructores

`LPCoefficient( LPVariable* pVariable, double dCoefficient )`

Recibe la variable de programación lineal y el coeficiente constante que representan el sumando de una ecuación lineal. pVariable es la variable de programación lineal y dCoefficient es el coeficiente constante.

Funciones miembro

`LPVariable* variable(void) const`

Devuelve la variable de programación lineal que almacena esta clase.

`double coefficient(void) const`

Devuelve el coeficiente constante que almacena esta clase.

3.4 LPConstraint

Categoría Clase de desarrollo.

Descripción Representa una restricción de un modelo de propgracmación lineal.

Fichero include

```
#include <vector>
using namespace std;

#include "LPCoefficient.h"

#define Fixed       0
#define MinorEqual  1
#define MajorEqual  2

class LPConstraint
{
	vector<LPCoefficient*> m_vCoefficients;
	int m_eCriterion;
	double m_dRigthValue;
	string m_strName;

public:
	LPConstraint( int eCriterion, double dRigthValue, string
strName = "" );

	virtual ~LPConstraint(void);

	string name(void) const;

	void add( LPCoefficient* pCoefficient );
	void setRigthValue( double dValue);
	int coefficientsCount(void);

	vector<LPCoefficient*>* coefficients(void);
	int criterion(void) const;
	double rigthValue(void) const;
};
```

Constructores

```
LPConstraint( int eCriterion, double dRigthValue, string strName =
"" );
```

Recibe el crierio de la restricción y el valor del lado drecho de la restricción. eCriterion es el criterio de la restricción, y puede valer Fixed si la restricción es de igualdad, MinorEqual si la restrcción es de menor o igual, o MajorEqual si la restricción es de mayor o igual. dRightValue es el valor del término derecho de la restricción. strName es el nombre opcional para la restricción.

Funciones miembro

```
string name(void) const;
```

Devuelve el nombre de la restricción.

```
void add( LPCoefficient* pCoefficient );
```

Añade un coeficiente a la restricción. pCoefficient es el coeficiente.

```
void setRigthValue( double dValue );
```

Cambia el valor del lado drecho de la restricción.

```
int coefficientsCount(void);
```

Devuelve el número de coeficientes que tiene la restricción.

```
vector<LPCoefficient*>* coefficients(void);
```

Devuelve el vector de coeficientes de la restricción.

```
int criterion(void) const;
```

Devuelve el criterio de la restricción.Si la restricción es de igualda, devuelve Fixed si la restricción es de igualdad, MinorEqual si la restrcción es de menor o igual, o MajorEqual si la restricción es de mayor o igual.

```
double rigthValue(void) const;
```

Devuelve el valor del lado derecho de la restricción.

3.5 LPSolver

Categoría Clase abstracta.

Descripción Representa un solver genérico. Los solver que soporta Lin2any derivan de esta clase, y deben sobreescribir todos sus métodos abstractos.

El problema de programación lineal se representa en la clase LPModel, y lo resuelve un solver, que está encapsulado en una clase derivada de LPSolver.

Fichero include

```
#include "LPVariable.h"
#include "LPConstraint.h"
```

```
#define Max 0
#define Min 1

#define CLP 0
#define ILP 1

#define Optimal     0
#define NoSolution  1


class LPSolver
{
protected:

	int m_nTimeLimit;

public:
	LPSolver(void);
	virtual ~LPSolver(void);

	void setTimeLimit( int nSeconds );

	virtual void createProblem(void) = 0;
	virtual void setDirection( int eDirection ) = 0;
	virtual void setKind( int eKind ) = 0;
	virtual void addVariables( vector<LPVariable*>* pvVariables
) = 0;
	virtual void addObjCoefficients( vector<LPCoefficient*>*
pvObjCoefficients ) = 0;
	virtual void addConstraints( vector<LPConstraint*>*
pvConstraints ) = 0;
	virtual double solve( vector<LPVariable*>* pvVariables ) =
0;
	virtual void finish(void) = 0;
};
```

Constructores `LPSolver(void);`

Constructor vacío de la clase abstracta.

Funciones miembro

```
void setTimeLimit( int nSeconds ) = 0
```

Función abstracta. Fija el tiempo máximo de resolución del problema de programación lineal.

```
virtual void createProblem(void) = 0
```

Función abstracta. Realiza las tareas de inicialización del solver que va a resolver el problema.

```
virtual void setDirection( int eDirection )
```

Copiar de LPModel

```
virtual void setKind( int eKind ) = 0;
```

Función abstracta. Copiar de LPModel

```
virtual void addVariables( vector<LPVariable*>* pvVariables ) = 0
```

Función abstracta. Añade las variables del modelo de programación lineal al solver que va a resolver el problema.

```
virtual void addObjCoefficients( vector<LPCoefficient*>* pvObjCoefficients ) = 0;
```

Función abstracta. Añade la función objetivo del modelo de programación lineal, al solver que va a resolver el problema.

```
virtual void addConstraints( vector<LPConstraint*>* pvConstraints ) = 0;
```

Función abstracta. Añade las restricciones del modelo de programación lineal, al solver que va a resolver el problema.

```
virtual double solve( vector<LPVariable*>* pvVariables ) = 0;
```

Función abstracta. Resuelve el problema de programación lineal y devuelve las variables con la solución.

```
virtual void finish(void) = 0;
```

Función abstracta. Se ejecuta después de terminar la resolución del problema. Se suele utilizar para que el solver que ha resuelto el problema libere recursos (memoria fundamentalmente).

3.6 LP_GLPKSolver

Categoría

Clase de desarrollo. Encapsula al solver GLPK.

Descripción

Conecta el modelo de programación lineal definido con las clases de Lin2any con el solver GLPK, para resolver el problema de programación lineal.

LP_GLPKSolver encapsula el objeto LPX, que representa un problema de la biblioteca GLPK.

Las funciones sobre-escritas de esta clase, trasladan el modelo de programación lineal de LPModel a la estructura LPX.

Fichero include

```
include "LPSolver.h"

extern "C" {
	#include "glpk.h"
	#include "glpspx.h"
}

class LP_GLPKSolver : public LPSolver
{
	LPX* lp;
	int m_eKind;

public:
	LP_GLPKSolver(void);
	virtual ~LP_GLPKSolver(void);

	virtual void createProblem(void);
	virtual void setDirection( int eDirection );
	virtual void setKind( int eKind );
	virtual void addVariables( vector<LPVariable*>* pvVariables
);
	virtual void addObjCoefficients( vector<LPCoefficient*>*
pvObjCoefficients );
	virtual void addConstraints( vector<LPConstraint*>*
pvConstraints );
	virtual double solve( vector<LPVariable*>* pvVariables );
	virtual void finish(void);
};
```

Constructores

`GLPKSolver (void);`

Constructor vacío de la clase.

.

Funciones miembro

`void createProblem(void);`

Crea e inicializa la estructura LPX de GLPK.

`void setDirection( int eDirection );`

Fija el sentido de optimziación de LPModel a la estructura LPX de GLPK.

```
void setKind( int eKind );
```

Fija el tipo de problema de programación lineal de LPModel en la estructura LPX de GLPK.

```
void addVariables( vector<LPVariable*>* pvVariables );
```

Crea las variables de LPModel en la estructura LPX de GLPK.

```
void addObjCoefficients( vector<LPCoefficient*>* pvObjCoefficients
);
```

Crea la función objetivo de LPModel en la estructura LPX de GLPK.

```
void addConstraints( vector<LPConstraint*>* pvConstraints );
```

Crea las restricciones de LPModel en la estructura LPX de GLPK.

```
double solve( vector<LPVariable*>* pvVariables );
```

Resuelve el modelo de programación lineal de LPModel en GLPK.

```
void finish(void);
```

Libera la memoria de la estructura LPX de GLPK.

3.7 LP_GurobiSolver

Categoría Clase de desarrollo. Encapsula al solver Gurobi.

Descripción Conecta el modelo de programación lineal definido con las clases de Lin2any con el solver Gurobi, para resolver el problema de programación lineal.

LP_GurobiSolver encapsula el objeto GRBModel, que representa un problema de la biblioteca Gurobi.

Las funciones sobre-escritas de esta clase, trasladan el modelo de programación lineal de LPModel a la estructura GRBModel.

Fichero include

```
#include "LPSolver.h"

#include "gurobi_c++.h"
using namespace std;


class LP_GurobiSolver : public LPSolver
{
	GRBEnv moEnvironment;
	GRBModel moModel;

	int m_eKind;
	int m_eDirection;

public:
	LP_GurobiSolver(void);
	virtual ~LP_GurobiSolver(void);

	virtual void createProblem(void);
	virtual void setDirection( int eDirection );
	virtual void setKind( int eKind );

	virtual void addVariables( vector<LPVariable*>* pvVariables );

	virtual void addObjCoefficients( vector<LPCoefficient*>*
pvObjCoefficients );

	virtual void addConstraints( vector<LPConstraint*>*
pvConstraints );

	virtual double solve( vector<LPVariable*>* pvVariables );

	virtual void finish(void);
};
```

Constructores

```
GLPKSolver (void);
```

Constructor vacío de la clase.

.

Funciones miembro

```
void createProblem(void);
```

Crea e inicializa la estructura GRBModel de GLPK.

```
void setDirection( int eDirection );
```

Fija el sentido de optimziación de LPModel a la estructura GRBModel de Gurobi.

```
void setKind( int eKind );
```

Fija el tipo de problema de programación lineal de LPModel en la estructura GRBModel de Gurobi.

```
void addVariables( vector<LPVariable*>* pvVariables );
```

Crea las variables de LPModel en la estructura GRBModel de Gurobi.

```
void addObjCoefficients( vector<LPCoefficient*>* pvObjCoefficients
);
```

Crea la función objetivo de LPModel en la estructura GRBModel de Gurobi.

```
void addConstraints( vector<LPConstraint*>* pvConstraints );
```

Crea las restricciones de LPModel en la estructura GRBModel de Gurobi.

```
double solve( vector<LPVariable*>* pvVariables );
```

Resuelve el modelo de programación lineal de LPModel en Gurobi.

```
void finish(void);
```

Libera la memoria de la estructura GRBModel de Gurobi.

3.8 LPException

Categoría Clase de utilidad.

Descripción Representa las excepciones que se gestiona en Lin2any.

Fichero include

```
#include <string>
using namespace std;


class LPException
{
	string mstrMessage;

public:

	LPException(string strMessage)
		: mstrMessage( strMessage )
	{
	}

	virtual ~LPException(void)
	{
	}

	string message(void) const
	{
		return mstrMessage;
	}
};
```

Constructores

`LPException(string strMessage)`

Constructor que recibe el mensaje que produce la excepción.

Funciones miembro

`string message(void) const`

Devuelve el mensaje registrado en la excepción.