`zimpler` - Programación entera, más fácil

Javier Martínez-Viademonte

Universidad Nacional de General Sarmiento
`javiermv@ungs.edu.ar`

Resumen *zimpl* es un lenguaje para modelar problemas de optimización como programas lineales enteros mixtos. Este trabajo presenta una nueva herramienta libre, `zimpler`, con dos contribuciones: 1) integración de datos nativos como Excel o JSON en modelos zimpl, y 2) automatización de la descarga e invocación de un solver para solucionar el modelo. Se describen los mecanismos provistos para agregar nuevos formatos de datos, cambiar el solver o adaptar el proceso de solución.

Keywords: programación lineal entera mixta, lenguaje de modelado, software libre, zimpl, optimización matemática

`zimpler` - Integer Programming, easier

Abstract. *zimpl* is a language to model optimization problems as mixed-integer linear programs. This work presents a new libre tool, `zimpler`, with two contributions: 1) integration of native data such as Excel or JSON into zimpl models, and 2) automation of download and invocation of a solver to solve the model. Provided mechanisms are described to add new data formats, change the solver or adapt the solution process.

Keywords: mixed-integer linear programming, modeling language, free software, zimpl, mathematical optimization

1. Introducción

Existe una variedad de lenguajes para modelar problemas de programación lineal entera mixta (Kallrath, 2004). Según sus características pueden agruparse en familias como los *Lenguajes de Modelado Algebraico* (LMA) o los *Lenguajes de Modelado de Instancias* (LMI), entre otras categorizaciones posibles. Los LMAs son prácticos para modelar ya que permiten describir problemas abstractos, usando operadores e identificadores genéricos para sus elementos; por ejemplo, permiten la expresión $\sum_{i \in I} w_i x_i$. Los LMI, en cambio, son simples de implementar ya que solo permiten usar valores y variables concretas; por ejemplo, la expresión anterior se podría reescribir como $2x_1 + x_3$.

A pesar de la variedad existente, no hay un lenguaje que sea, a la vez, soportado por los programas que solucionan estos problemas y práctico para modelar. Por motivos técnicos y comerciales, los desarrolladores de solvers se enfocan en LMIs con amplio soporte y, en algunos casos, en un LMA propio. Un inconveniente relacionado es que, habitualmente, los datos de los modelos matemáticos provienen de una variedad de fuentes (planillas de cálculo, bases de datos, ...) sin soporte en los lenguajes de modelado.

El lenguaje *zimpl* (Koch, 2004) es un LMA de código abierto que se interpreta y se puede traducir a otros lenguajes con la herramienta `zimpl`. Este trabajo extiende el lenguaje, permitiendo **declarar** los datos, **sin definirlos**. En contraposición, la herramienta existente requiere una definición estricta de los datos; a los conjuntos (`set`) y parámetros (`param`) se les debe asignar valores concretos. Separar las declaraciones de las definiciones facilita la integración de modelos zimpl con nuevos formatos de datos, como Excel o JSON, sin necesidad de realizar conversiones trabajosas. Al proceso de buscar y asignar valores para los **datos** declarados e indefinidos se lo denomina *resolución*; es un concepto diferente al de *solución*, que busca asignar valores a las **variables** del modelo.

Un proceso simplificado para solucionar programas matemáticos es:

1. generar un modelo,
2. invocar al solver,
3. leer la solución.

En la práctica, suele ser necesario atender a otros detalles, como cargar datos de fuentes heterogéneas, combinarlos con un modelo abstracto o considerar las diferencias entre solvers. A su vez, se deben proveer los programas que entiendan en cada caso; por ejemplo, para usar `zimpl` en el paso 1 y el solver `cbc` (Forrest y Lougee-Heimer, 2005) en el paso 2, ambos programas deben estar disponibles.

Este trabajo presenta a `zimpler`[1], una herramienta libre que ayuda en la solución de programas lineales enteros mixtos. En la Figura 1 se muestran las transformaciones que realiza la herramienta sobre un fragmento de modelo.

```
param w[I];
sum <i> in I: w[i]*x[i] <= C;
```
(a) `zimpler`

```
param w[I] := read "weights.csv" as "<1n> 2n";
sum <i> in I: w[i]*x[i] <= C;
```
(b) `zimpl`

```
2*x1 + 0*x2 + 1*x3 + 0*x4 <= 5
```
(c) `cbc`

Figura 1: Ejemplo de transformaciones sucesivas

En el paso (a) se tiene la declaración de una colección `w` de pesos (*weights*) asociados a los elementos de un conjunto `I`, y una restricción con una sumatoria;

[1] Disponible en `https://gitlab.com/zimpler-tool`

`zimpler` resuelve la declaración de `w` y quizás otras. En el paso (b) se tiene la definición completa del modelo y, para `I` $= \{1..4\}$, `w` $= (2, 0, 1, 0)$ y `C` $= 5$, `zimpl` puede traducirlo al dialecto CPLEX (IBM, 2024) del formato LP. Por último, en el paso (c) el solver `cbc` busca una solución para la instancia. Las invocaciones a `zimpl` y `cbc` también son realizadas por `zimpler`. Un tutorial (Martínez-Viademonte, 2024) presenta ejemplos completos y más detalles.

2. Resultados preliminares

El primer resultado de este trabajo es permitir modelar en zimpl, con las herramientas existentes, pero sin preocupación por el formato de los datos.

Para implementar un nuevo mecanismo de resolución se desarrollaron una gramática y un parser zimpl[2]. Al leer un modelo se analizan las declaraciones de datos; los elementos definidos permanecen sin cambios, los elementos indefinidos causan la invocación del mecanismo de resolución. Se integra al parser un conjunto de adaptadores. Al día de hoy hay implementados adaptadores para leer Excel, CSV, Properties y grafos en formato DOT. Cada adaptador define una estrategia de lectura; por ejemplo, en los archivos Excel, los nombres de las hojas y las columnas se consideran como posibles identificadores de datos. Cada adaptador es un programa independiente y su documentación presenta los detalles. Con el conjunto de adaptadores, se recorre el directorio actual de trabajo y se leen todos los archivos que los adaptadores son capaces de interpretar. Si una declaración no puede asociarse a una definición, como resultado se obtiene la declaración original. Si se encuentran múltiples definiciones para una declaración, se elije una cualquiera y se muestra una advertencia. Finalmente, se genera un nuevo modelo zimpl, con la definición de todas las declaraciones resueltas, listo para ser usado por cualquiera de las herramientas existentes.

El segundo resultado de este trabajo es automatizar, de manera genérica, el proceso de resolución, traducción y solución.

El mecanismo de resolución toma un modelo zimpl y genera otro, habiendo asociado las declaraciones con definiciones. A continuación se debe traducir el modelo generado a un formato que el solver sea capaz de interpretar. Esta tarea la realiza `zimpl`, generando otro modelo en algún LMI apropiado. Entonces, un solver puede tomar el modelo de instancia y buscarle una solución. Para automatizar este proceso, incluyendo la descarga de los programas necesarios, se definió una interfaz, *Optimizer*, que representa a una elección tecnológica, junto con *CliOptimizer*, una implementación que usa un script de línea de comandos. Se prevén otras implementaciones que usen las APIs propias de cada solver. De no existir previamente, CliOptimizer genera un script llamado `solve.sh` en Linux y Mac, `solve.bat` en Windows, capaz de descargar e invocar a `zimpl` y `cbc`. Finalmente, se le transfiere al script la responsabilidad de coordinar las invocaciones sucesivas. Para usar otro solver, otro formato de intercambio de datos u otro proceso de solución, alcanza con modificar el script.

[2] Disponibles en `https://gitlab.com/antlr-java-parser/zimpl`

3. Trabajo futuro

Se está trabajando en que las próximas versiones de `zimpler` interpreten los resultados del solver y generen un reporte amigable. Además, sería deseable aumentar la cantidad de adaptadores de datos y mejorar la integración con los distintos solvers, por medio de sus APIs.

Referencias

IBM. (2024). *CPLEX User's Manual*. IBM Corporation. https://www.ibm.com/docs/en/icos/22.1.2?topic=optimizers-users-manual-cplex

Kallrath, J. (2004). *Modeling languages in mathematical optimization* (Vol. 88). Springer. https://doi.org/10.1007/978-1-4613-0215-5

Forrest, J., & Lougee-Heimer, R. (2005). *CBC User's Guide*. Computational Infrastructure for Operations Research. https://www.coin-or.org/Cbc/

Koch, T. (2004). *Rapid Mathematical Prototyping* [Tesis doctoral, Technische Universität Berlin]. http://opus.kobv.de/zib/volltexte/2005/834/

Martínez-Viademonte, J. (2024). Zimpler: use native input data with zimpl. *IFORS Newsletter*, *19*(3), 5-6. https://ifors.org/newsletter/ifors-news-sept-2024.pdf