LEMON Lp/Mip interface

Library for Efficient Models and Optimization in Networks

Balázs Dezső

ELTE, Operációkutatási tanszék

Lineáris program(LP)

A lineáris programozási feladatban adott $A : I \times J \to \mathrm{R}$ mátrix, $b : I \to \mathrm{R}$ és $c : J \to \mathrm{R}$ vektor. Az $I^{\leq}$ és $I^{=}$ az I, és $J = J^{\geq}$ és $J^{=}$ a J diszjunkt felbontása.

$$\begin{aligned}
\sum_{j \in J} A_{ij} x_j &\leq b_i & i \in I^{\leq} \\
\sum_{j \in J} A_{ij} x_j &= b_i & i \in I^{=} \\
x_j &\geq 0 & j \in J^{\geq} \\
\max & \sum_{j \in J} c_j x_j
\end{aligned}$$

Teremtsük meg a feladatot

```
#include <lemon/lp.h>

Lp lp;
// LpGlpk lp;
// LpSoplex lp;
// LpCplex lp;
// MipGlpk mip;
// MipCplex mip;
```

glpk http://www.gnu.org/software/glpk/

soplex http://soplex.zib.de/

cplex http://www.ilog.com/products/cplex/

clp http://www.coin-or.org/Clp/

Teremtsük meg a változókat

```
std::vector<Lp::Col> x(m);
for (int j = 0; j < m; ++j) {
  x[j] = lp.addCol();
  std::ostringstream name;
  name << 'x' << j;
  lp.colName(x[j], name.str());
  if (j < mm) lp.colLowerBound(x[j], 0);
  // lp.colUpperBound(x[j], 1);
  // mip.colType(x[j], INT);
}
```

- Változó alsó korlátja
- Változó felsŽ korlátja
- Változó neve (IO-nál hasznos)

Teremtsük meg a feltételeket

```
for (int i = 0; i < n; ++i) {
  Lp::Expr e;
  for (int j = 0; j < m; ++j) {
    e += a[i][j] * x[j];
  }
  if (i < nn) lp.addRow(e <= b[i]);
  else lp.addRow(e == b[i]);
}
```

- $e \leq f$
- $e = f$
- $e \geq f$
- $l \leq e \leq u$

Teremtsük meg a célfüggvényt

```
Lp::Expr obj;
for (int j = 0; j < n; ++j) {
  obj += c[j] * x[j];
}
lp.obj(obj);
lp.max();
```

- min
- max

És pihenjünk, míg a solver dolgozik

```
lp.solve();

for (int j = 0; j < m; ++j) {
  std::cout << lp.colName(x[j]) << '='
            << lp.primal(x[j]) << std::endl;
```

További lehetőségek?

- Megoldhatóság?
- Bázis megoldás?
- Duális lekérdezés?
- Korlátosság?
- Nem korlátos sugár?
- Vágás(sor) generálás?
- Oszlop generálás?

Feladat

Oldjuk meg a következő feladatot Lemon Lp segítségével:

$$\begin{aligned} 15x_1 + 20x_2 &\leq 1440 \\ 3x_1 + 2x_2 &\leq 240 \\ x_2 &\leq 50 \\ x_1, x_2 &\geq 0 \\ \max 600x_1 &+ 500x_2 \end{aligned}$$

Feladat

A feladat módosult!!!

Alíz és Béla a következő játékot játsza, mindketten bedobnak egy kalapba 1 és n forint között valamennyit:

- Ha a két érték között kettőnél kisebb a különbség, akkor Béla nyer.
- Ellenkező esetben pedig Alíz.
- A szereplők hogyan maximalizálhatják a várható nyereményüket vagy hogyan minimalizálhatják a várható veszteségüket?

Megoldás

Zéróösszegű játék nyeremény mátrixa Béla szerint:

$$M_{ij} = \begin{cases} i, & \text{ha } |i-j| \leq 1 \\ -j, & \text{ha } |i-j| > 1 \end{cases}$$

Béla nyereményének a maximalizálása:

$$\begin{aligned} \sum M_{ij} x_j &\geq z \\ \sum x_j &= 1 \\ &\max x_j \end{aligned}$$

Megoldás C++-ban

```
Lp lp;

Lp::Col z = lp.addCol();

std::vector<Lp::Col> x(n);
Lp::Expr sum;
for (int j = 0; j < n; ++j) {
    x[j] = lp.addCol();
    lp.colLowerBound(x[j], 0.0);
    sum += x[j];
}
lp.addRow(sum == 1);
```

Megoldás C++-ban

```
for (int i = 0; i < n; ++i) {
    Lp::Expr e;
    for (int j = 0; j < n; ++j) {
        e += x[j] * (i - j >= -1 && i - j <= 1
                     ? (i + 1) : - (j + 1));
    }
    lp.addRow(e >= z);
}

lp.obj(z);
lp.max();

lp.solve();
```

Megoldás C++-ban

```
// debugging to standard output
LemonWriter lw(std::cout);
LpWriter lpw(lw, lp);
lw.run();

for (int j = 0; j < n; ++j) {
    std::cout << j + 1 << " -> "
              << lp.primal(x[j]) << std::endl;
}

std::cout << lp.primalValue() << std::endl;
```