优化软件与应用

主讲人:　　雒兴刚
东北大学系统工程研究所
Email：luoxinggang@ise.neu.edu.cn
Tel: 83682292

信息科学与工程学院
COLLEGE OF INFORMATION SCIENCE AND ENGINEERING

第九章 ILOG OPL 建模语言

OPL 数据类型

1、整型：范围 $-2^{31} + 1$ to $2^{31} - 1$

例：

```
int i = 25;

int n = 3;
int size = n*n; //注意这种初始化很特别
```

2、浮点型：双精度，IEEE 754 standard

```
float f = 3.2;
```

第九章 ILOG OPL 建模语言

OPL 数据类型

3、字符串型

例如{string} Tasks = {"masonry","carpentry","plumbing","ceiling", "roofing","painting","windows","facade", "garden","moving"};
定义字符串一个集合 。

字符串中的特殊字符：

\b	backspace
\t	tab
\n	newline
\f	form feed
\r	carriage return
\"	double quote
\\	backslash
\ooo	octal character ooo
\xXX	hexadecimal character XX

字符串换行：

```
"first line \
second line"
```

第九章 ILOG OPL 建模语言

OPL 数据结构

1、Range：给定最小和最大值。

```
range Rows = 1..10;

int n = 8; range Rows = n+1..2*n+1;
```

用途1：数组定义

```
range R = 1..100;
int A[R]; // A is an array of 100 integers
```

用途2：循环

```
range R = 1..100;
forall(i in R) { //element of a loop ... }
```

用途3：变量定义

```
dvar int i in R;
```

第九章 ILOG OPL 建模语言

OPL 数据结构

2、数组

一维数组：

int a[1..4] = [10, 20, 30, 40];
float f[1..4] = [1.2, 2.3, 3.4, 4.5];
string d[1..2] = ["Monday", "Wednesday"];

int a[Days] = [10, 20, 30, 40, 50, 60, 70];
即元素下标可以是字符串，如**a["Monday"],...,a["Sunday"].**

tuple Edges { int orig; int dest; }
{Edge} Edges = {<1,2>, <1,4>, <1,5>};
int a[Edges] = [10,20,30];
即下标也可以是**Tuple**，**a[<1,2>], a[<1,4>], and a[<1,5>]**

第九章 ILOG OPL 建模语言

OPL 数据结构

2、数组

多维数组：

```
int a[1..2][1..3] = ...;

int a[Days][1..3] = ...; //混合下标

{string} Warehouses = ...;
{string} Customers = ...;
tuple Route { string w; string c; }
{Route} routes = ...;
int transp[routes] = ...   //实际上transp是二维数组

{string} Warehouses ...;
{string} Customers ...;
tuple Route { Warehouses w; Customers c; }
{Route} routes = ...;
int transp[routes] = ...
```

可能是稀疏矩阵

两种哪个好些？

第九章 ILOG OPL 建模语言

OPL 数据结构

3、Tuple：结构体

```
tuple Point { int x; int y; };
Point point[i in 1..3] = <i, i+1>;

Point p = <2,3>;
Point point[i in 1..3] = <i, i+1>; //Tuple数组
{Point} points = {<1,2>, <2,3>}; //Tuple集合
tuple Rectangle { Point ll; Point ur; } //Tuple的Tuple

Point p = <2,3>;
int x = p.x; //取Tuple的成员
```

但是，Tuple的定义里不能出现Tuple集合和Tuple数组！

第九章 ILOG OPL 建模语言

OPL 数据结构

4、集合：可以写成{T}, 或者 setof(T)

```
{int} setInt = ...;
setof(Precedence) precedences = ...;
```

集合初始化：

```
tuple Precedence { int before; int after; }
{Precedence} precedences = {<1,2>, <1,3>, <3,4>};
```

第九章 ILOG OPL 建模语言

OPL 决策变量和约束

OPL决策变量使用关键字dvar

```
dvar int transp[Orig][Dest] in 0..100; //二维数组变量；限制决策变量范围

tuple Route { City orig; City dest }
{Route} routes = ...:
dvar int transp[routes] in 0..100; //以有限tuple集routes 为索引

range Capacity = 0..limitCapacity;
dvar int transp[Orig][Dest] in Capacity; //in 后面是range

dvar int averageDelay in 0..maxDelay; //in 后面接变量
```

如果不同决策变量的范围不同，可以这样定义

```
int capacity[route] = ...;
dvar int transp[r in routes] in 0..capacity[r];
```

第九章 ILOG OPL 建模语言

OPL 决策变量和约束

也可以用＋关键字限制决策变量只能为正：

dvar int+ x; // non negative integer decision variable
dvar float+ y; // non-negative decision variable
dvar boolean z; // boolean decision variable

上述定义等价于：
dvar int x in 0..maxint;
dvar float y in 0..infinity;
dvar int z in 0..1;

其中**maxint**、 **infinity**为**OPL**关键字。

二维决策变量数组也可以逐个元素给定范围：
dvar float transp[o in Orig][d in Dest] in 0..cap[o][d];

约束可以单个定义，也可以定义成数组形式，如：
constraint capCstr[Machines];

第九章 ILOG OPL 建模语言

OPL 数据初始化

总的来说，**OPL**数据初始化可以分为2种，一种是在**mod**文件完成，另一种是在**dat**文件完成。

1、数组初始化

初始化多维数组：

```
/* .mod file */
int a[1..2][1..3] = ...;
/* .dat file */
a = [ [10, 20, 30], [40, 50, 60] ];
```

按照(index, value)的方式初始化数组：
但注意要用#[…]#方式
元素次序无关。参见下页例子：

```
/* .mod file */
int a[Days] = ...;

/* .dat file */
a = #[ "Monday": 1,
"Tuesday": 2,
"Wednesday": 3,
"Thursday": 4,
"Friday": 5,
"Saturday": 6,
“Sunday”: 7 ]#;
```

第九章 ILOG OPL 建模语言

OPL 数据初始化

前面的整型索引数组的初始化也可以写成：

```
/* .mod file */
int a[1..2][1..3] = ...;
/* .dat file */
a = #[ 2: [40, 50, 60], 1: [10, 20, 30] ]#;
```

行下标，注意这里故意颠倒了次序，但结果相同

数组初始化也可以用ILOG脚本实现，如：

```
range R = 1..10;
 int a[R];
execute {
for(var i in R)
{
        a[i] = i + 1;
}
}
```

第九章 ILOG OPL 建模语言

OPL 数据初始化

也可以用表达式方式初始化，例如上面的例子也可写为：

```
int a[i in 1..10] = i+1;
```

多维数组也可以用这种方式，如：

```
int m[i in 0..10][j in 0..10] = 10*i + j;
```

也可以用一个已知数组初始化，如：

```
int m[Dim1][Dim2] = ...;
int t[j in Dim2][i in Dim1] = m[i][j];
```

也可以用**index： item** 方式初始化，如：

```
int a[1..10] = [ i-1 : i | i in 2..11 ]; //效果同前面的2个例子
int m[0..10][0..10] = [ i : [ j : 10*i+j ] | i,j in 0..10 ];
```

第九章 ILOG OPL 建模语言

OPL 数据初始化

再如，下面的**ILOG脚本初始化**：

```
GasType gas[Gasolines];
execute {
      for(var g in gasData)
      {
            gas[g.name] = g;
      }
}
```

用**index： item** 方式可以写成：

```
GasType gas[Gasolines] = [ g.name : g | g in gasData ];
```

第九章 ILOG OPL 建模语言

OPL 数据初始化

2、Tuple初始化

单个变量初始化直接用<…> 给出成员即可，如：
Point p = <3,2> ;

Tuple中含有数组的初始化：
tuple Rectangle { int id; Point p[2]; }
Rectangle r = <1, [<0,0>, <10,10>]>;

Tuple中含有集合的初始化：
{string} Task ...;
tuple Precedence { Task name; {string} after; }
Precedence p = <a1, {a2, a3, a4, a5}>;

第九章 ILOG OPL 建模语言

OPL 数据初始化

3、集合初始化

结合采用一对大括号进行初始化，如：

```
tuple Precedence { int before; int after; }
{Precedence} precedences = ...;
precedences = {<1,2>, <1,3>, <3,4>};
```

可以在初始化时使用集合运算符，如：

```
{int} s1 = {1,2,3};
{int} s2 = {1,4,5};
{int} i = s1 inter s2;
{int} j = {1,4,8,10} inter s2;
{int} u = s1 union {5,7,9};
{int} d = s1 diff s2;
```

结果是：

```
i = {1},
u = {1,2,3,5,7,9},
d ={2,3},
sd = {2,3,4,5}.
```

注意inter 等是集合运算符

第九章 ILOG OPL 建模语言

OPL 数据初始化

可以利用**range**初始化集合，如：

{int} s = asSet(1..10) //初始化 s为 {1,2,..,10}

asSet是内置函数，功能是将**range**转换为集合

也可以用表达式方式初始化，格式是**p in S : condition**，如：

{int} s = {i | i in 1..10: i mod 3 == 1}; //结果是{1,4,7,10}.

也可以定义集合数组（数组元素为一个集合），如：

{int} a[i in 3..4] = {e | e in 1..10: e mod i == 0};

初始化**a[3]** 为 **{3,6,9}**， **a[4]** 为 **{4,8}**

第九章 ILOG OPL 建模语言

OPL 数据初始化

集合很多时候可以用来表示稀疏矩阵，如：

```
{string} Nodes ...;
int edges[Nodes][Nodes] = ...;
tuple Edge { Nodes o; Nodes d; }
{Edge} setEdges = {<o,d> | o,d in Nodes : edges[o][d]==1};
```

关键字，强制i<j

另一个稍复杂一些的例子：

```
{string} Resources ...;
{string} Tasks ...;
Tasks res[Resources] = ...;
 tuple Disjunction { {string} first; {string} second; }

{Disjunction} disj = {<i,j> | r in Resources, ordered i,j in res[r] };
```

关键字，强制i<j

第九章 ILOG OPL 建模语言

OPL 数据一致性

为了保证输入数据的正确性，可以通过**assert**语句来判定数据的一致性。这样在程序运行前，可以通过编译系统提前发现问题。例如，原是的需求和供应数据具有关联性（总和相等）：

```
int demand[Customers] = ...;
int supply[Suppliers] = ...;
assert sum(s in Suppliers) supply[s] == sum(c in Customers) demand[c];
```

再如，如果是多产品的情况：

```
int demand[Customers] [Products] = ...;
int supply[Suppliers] [Products] = ...;
assert forall(p in Products)
        sum(s in Suppliers)
                supply[s][p] == sum(c in Customers) demand[c][p];
```

第九章 ILOG OPL 建模语言

OPL 数据前处理

OPL前处理使用ILOG脚本，可以进行CPLEX参数设定，可以改变变量的值、决策变量的范围，但range和约束类型不能改变。

下面是一个前处理的例子：

```
int n = ...;
range R = 1..n;
int A[R] = ...
execute {
        for(r in R)
        {
                if ( A[r]<0 ) { A[r] = 0; }
        }
}
```

OPL数据处理的次序如下：

1、各种数据源（如mod文件，dat文件，excel文件，ODBC等）；

2、execute块；

3、assert块；

第九章 ILOG OPL 建模语言

表达式和运算符

整型表达式：

可以使用+, -, *, div（整除）, mod (or %)等运算符；

可以使用abs等系统函数； 整型常量maxint ；

浮点表达式：

可以使用+, -, /, *等运算符；浮点常量infinity ；

条件表达式和C语言类似：
语法：(condition)?thenExpr : elseExpr

例：

```
int value = ...; int signValue = ( value>0 ) ? 1 : ( value<0 ) ? -1 : 0;
int absValue = ( value>=0 ) ? value : -value;
```

第九章 ILOG OPL 建模语言

表达式和运算符

聚合表达式：

可以利用聚合运算符计算和 (sum), 连乘 (prod), 最小 (min), 最大 (max) 等。

例如：
int capacity[Routes] = ...;
int minCap = min(r in Routes) capacity[r];

集合表达式：

可以利用**union, inter, diff**等集合运算符，也可以利用集合函数。
例如，现有

S 是一个集合**{ 3, 6, 7, 9 }**
item 是**S** 的一个集合成员
n 是一个整型数；

第九章 ILOG OPL 建模语言

表达式和运算符

S 是一个集合{ 3, 6, 7, 9 }

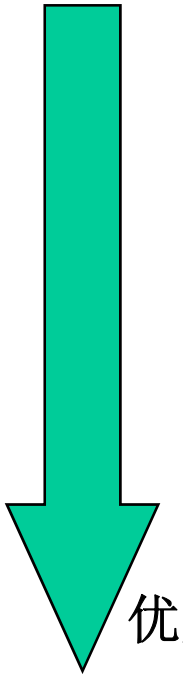
Function	Description
card	card(S) 返回集合个数.
ord	ord(S,item) 返回item在集合的位置（下标从0开始）。 例如 ord(S,6) =1，ord(S,9)= 3.
first	first(S)返回集合里的第一个成员
item	item(S,n) 返回第n个成员，如item(S,1) = 6
last	last(S) 返回最后一个成员
next	next(S,item) 返回item后的成员，如next(S,3) = 6
nextc	假象把集合元素连成环，调用 next. 例如nextc(S,9) = 3
prev	prev(S,item) 返回item前的成员，如prev(S,6) = 3
prevc	假象把集合元素连成环，调用 prev. 例如 prev(S,3) = 9

第九章 ILOG OPL 建模语言

表达式和运算符

各类运算符总结和优先级：

类型	操作符
逻辑	||
	&&
	!
	?:（条件运送符）
关系	==, <=, >=, <, >, !=
	not in
集合	+, -, union, diff
数学	+, -
聚合	sum, max, min
数学	*, /, div, %, mod, inter
聚合	prod, inter
	in



优先级高

第九章 ILOG OPL 建模语言

ILOG 约束

约束的位置：目标函数必须位于约束之前；

约束命名：虽然约束可以不命名，但不利于程序调试。约束也可以用数组方式，如：

```
constraint capacityCons[Resources];
constraint demandCons[Products];

minimize
  sum(p in Products) (insideCost[p]*inside[p] + outsideCost[p]*outside[p]);
subject to {
  forall(r in Resources)
    capacityCons[r]= sum(p in Products) consumption[p,r] * inside[p] <=
capacity[r];
  forall(p in Products)
    demandCons[p]= inside[p] + outside[p] >= demand[p];
}
```

第九章 ILOG OPL 建模语言

ILOG 遍历参数

有多种形式和用途：

1、*p in S*

```
int n=6;
int s == sum(i in 1..n) i*i;//用于range

{string} Products ={"car","truck"};
float cost[Products] =[12000,10000];
float maxCost = max(p in Products) cost[p]; //用于string set

{string} Cities = { “Paris”, “London”, “Berlin” }; //用于tuple set
tuple Connection { string orig;string dest; }
{Connection} connections = { <"Paris","Berlin">,<"Paris","London">};
float cost[connections] = [ 1000, 2000 ];
float maxCost= max(r in connections) cost[r];
```

第九章 ILOG OPL 建模语言

ILOG 遍历参数

2、*p in S : filtering condition*

```
int n=8;
dvar int a[1..n][1..n];
subject to {
        forall(i in 1..8)
                forall(j in 1..8: i < j)
                        a[i][j] >= 0; }

int s = sum(i,j in 1..n: i < j) i*j;
int s = sum(i in 1..n) sum(j in 1..n: i < j) i*j; //这2个等价

forall(i,j in 1..n : i < j) a[i][j] >= 0;
forall(i in 1..n, j in 1..n : i<j) a[i][j] >= 0;
forall(ordered i,j in 1..n) a[i][j] >= 0; //这3个等价
```

第九章 ILOG OPL 建模语言

ILOG 遍历参数

3、*filtering tuple*

考虑下面的例子

```
forall(c in connections)
        sum(<p,co> in routes: c == co) trans[<p,c>] <= limit;
```

对于一个给定的路线，所有货物的运输总量不超过limit

OPL也支持简写为：

```
forall(c in connections)
        sum(<p,c> in routes) trans[<p,c>] <= limit;
```

这种用法称为Implicit Slicing