Introduzione ad OPL IDE CPLEX

SAPIENZA
UNIVERSITÀ DI ROMA

Dipartimento di Scienze Statistiche
Lavinia Amorosi
E-mail: lavinia.amorosi@uniroma1.it

Cos'è OPL IDE CPLEX?

- CPLEX: solver per problemi di ottimizzazione (lineari, misti, quadratici)
- OPL: Optimization Programming Language
- IDE: Integrated Development Environment
- Altri linguaggi:
  -AMPL
  -GAMS
  …
- Altri solver:
  - Xpress-MP
  - GuRoBi
  …
- OPL CPLEX sarà utile per il corso e per chi deciderà di fare una tesi in ottimizzazione!

Potenza dell'ottimizzazione

Miglioramenti nel software e nell'hardware hanno accelerato la risoluzione di modelli di programmazione lineare intera mista di **220 miliardi di volte** rispetto a 15/20 anni fa.

Descrizione di un problema di ottimizzazione

OPL è un linguaggio creato per scrivere problemi di ottimizzazione

Cosa dobbiamo definire?

- 1 Le costanti del problema.
- 2 Le variabili del problema.
- 3 La funzione obiettivo.
- 4 I vincoli che definiscono la regione ammissibile del problema.

Rappresentare un problema

OPL distingue il problema da una sua istanza, un progetto in OPL è quindi composto da:

•Modello: file con estensione .mod, descrive la struttura del problema (variabili, funzione obiettivo e vincoli).

•Istanza: file con estensione .dat , descrive i dati del problema (costanti).

•Setting: file con estensione .ops, descrive diverse opzioni relative alla risoluzione del problema (metodo risolutivo, time limit, …)

•In OPL IDE, un .mod ed un .dat sono associati in una Run Configuration. L'inserimento del file di setting .ops è opzionale.

Programmamre in OPL

- Un programma in OPL consiste in una serie di istruzioni;

- ogni istruzione deve terminare con “;”

- è possibile inserire commenti: essi devono essere compresi fra “/*” e “*/”, oppure dopo “//” fino a fine linea;

- colori del testo:
  - blu: parole-chiave
  - verde: commenti
  - nero: altro testo

- c’è differenza fra carattere minuscolo e maiuscolo.

Definizione di costanti e variabili

- I nomi di tutte le variabili devono cominciare sempre con un carattere alfabetico o underscore (seguito da caratteri alfabetici o numerici, o underscore);
- una variabile decisionale viene dichiarata tale con **dvar**;
- Opl permette di considerare differenti tipi di costanti e variabili:

Costanti

- float
- int
- string

Variabili

- dvar float
- dvar float+
- dvar int
- dvar int+
- dvar boolean

Gli insiemi

- L’insieme è una collezione di elementi non indicizzati e non duplicabili;
- Se T è un tipo, allora **{T}**, o in alternativa **setof(T)**, denota un insieme di tipo T.
- Esempio:

```
{string} nomi = {”Antonio”, ”Marco”, ”Giovanni”};
setof(string) nomi = {”Antonio”, ”Marco”, ”Giovanni”};
```

Operazioni sugli insiemi

Esempi:

- {int} s1 = {1,2,3};
- {int} s2 = {1,4,5};
- {int} s3 = s1 inter s2;
- {int} s4 = {1,4,8,10} inter s2;
- {int} s5 = s1 union {5,7,9};
- {int} s6 = s1 diff s2;

Range

Un range è utile per:

- indicizzare un array;
- specificare il campo di valori (lower bound e upper bound) delle variabili decisionali;
- indicizzare i vincoli quando si usa l'istruzione forall.

La sintassi dell'istruzione range è: range nome_range = l..u.

Esempi:

- range R = 1..100; dvar int i in R;
- range R = 1..100; forall (i in R) vincolo;

Array monodimensionali e multidimensionali

Esempi:

- int a[1..4] = [10, 20, 30, 40];
- float f[1..4] = [1.2, 2.3, 3.4, 4.5];
- string d[1..2] = [”Antonio”, ”Giovanni”];
- range R = 1..100; int A[R];
- range R = 1..100; dvar float+ z[R];

Esempio:

• float cc[1..4][1..4] = [[0, 150, 400, 300], [150, 0, 250, 200], [400, 250, 0, 200], [300, 200, 200, 0]];

NOTA: un array può anche essere indicizzato da stringhe o anche da tuple.

Array generici indicizzati

Esempi:

- int a[i in 1..10] = i+1;
- int m[i in 0..10][j in 0..10] = 10*i + j;
- /* t = trasposta di m */
  int m[Dim1][Dim2] = ...;
  int t[j in Dim2][i in Dim1] = m[i][j];

sum

- Effettua una sommatoria;
- Sintassi: sum (qualificatori) espressione;

• Esempio:

```
int n = 10;
range Range = 0..n-1;
dvar int s[Range];
subject to
{
sum(i in Range) s[i] == n;
sum(i in Range) s[i]*i == n;
}
```

forall

- Consente di scrivere un gruppo di vincoli;
- Sintassi: forall (qualificatori) vincolo;

- Esempio:

```
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;
}
```

Definizione di costanti e variabili

Possiamo rappresentare i nostri dati come array definendoli nel .mod :

```
//definisco una costante di tipo int
int n = 4;
//definisco un range (in questo caso da 1 a 4)
range r = 1..n;
/*definisco un vettore b, le cui 4 componenti sono costanti di tipo float+ pari rispettivamente ad 1,2,3 e 4 */
int b[r] = [1, 2, 3, 4];
```

Oppure rappresentare i nostri dati come array leggendoli dal .dat:

```
int n = …;
range r = 1..n;
int b[r] = ...;
/*Dove i tre punti seguiti dal ; stanno ad indicare che i valori di quei parametri verranno letti dal .dat */
```

Definizione di costanti e variabili

Possiamo definire singole variabili (nel .mod):

```
dvar float+ x1;
dvar float+ x2;
dvar float+ x3;
dvar float+ x4;
```

Oppure definire un array di variabili (nel .mod):

```
int n= 4;
range var = 1..n;
dvar float+ x[var];
```

Definizione della funzione obiettivo

Consideriamo la seguente funzione obiettivo di esempio:

$$\max 6x1 + 8x2 + 5x3 + 9x4$$

Possiamo scrivere tale funzione nei due modi equivalenti che seguono:

1°Modo (forma estesa)

```
dvar float+ x1;
dvar float+ x2;
dvar float+ x3;
dvar float+ x4;
maximize 6*x1 + 8*x2 + 5*x3 + 9*x4;
```

Definizione della funzione obiettivo

Consideriamo la seguente funzione obiettivo di esempio:

$$\max 6x1 + 8x2 + 5x3 + 9x4$$

Possiamo scrivere tale funzione nei due modi equivalenti che seguono:

2° Modo (forma compatta)

```
int n= 4;
range cols = 1..n;
//definisco il vettore dei coefficienti c
float c[cols] = [6, 8, 5, 9];
//definisco il vettore delle variabili x
dvar float+ x[cols];
//definisco la f.ne obiettivo come il prodotto scalare dei vettori c ed x
maximize sum(i in cols) c[i] * x[i];
```

Definizione della regione ammissibile

Supponiamo che la regione ammissibile sia definita dalle seguenti disuguaglianze:

$$2x1 + x2 + x3 + 3x4 \leq 5$$

$$x1 + 3x2 + x3 + 2x4 \leq 3$$

In OPL possiamo scrivere tale regione nel modo seguente (nel .mod):

```
range rows=…;
range cols=…;
float A[rows][cols] = …;
float b[rows] = …;
dvar float+ x[cols];
//dichiariamo i vincoli che definiscono la regione ammissibile del problema
subject to {
forall (j in rows) {
sum(i in cols) ( A[j][i] * x[i] ) <= b[j]; }}
```

Il modello completo

Possiamo quindi scrivere il .mod completo per l'esempio considerato:

```
range rows=…;
range cols=…;
float A[rows][cols] = …;
float b[rows] = …;
float c[cols] = …;
dvar float+ x[cols];
maximize sum(i in cols) c[i] * x[i];
subject to {
forall (j in rows) {
sum(i in cols) ( A[j][i] * x[i] ) <= b[j]; }}
```

Struttura di un progetto in OPL

Riassumendo, un progetto in OPL si compone di:

- Un file. mod contenente:

1 Definizione delle costanti (float b = 3.0;)

2 Definizione delle variabili decisionali (dvar float+ x;)

3 Definizione della funzione obiettivo (maximize ...)

4 Vincoli (subject to {... })

- Un file .dat contenente i dati definiti con = ...; nel file .mod.
- Facoltativamente, un file .ops contentente altre opzioni relative alle procedure di ottimizzazione

Scrittura dell'output su un file .txt

Per poter scrivere su file di testo i valori della soluzione, nel file .mod deve essere dichiarata una funzione. Dopo la chiusura del «subject to» deve essere definita:

- Una struttura dati che conterrà i valori delle variabili decisionali.
- Una funzione che va a riempire questa struttura con i valori della soluzione.
- Una funzione che permette la stampa su file di tale struttura dati.

Esempio in OPL : gas.mod

```
float Solution[Products];

execute {
for (var i in Products)
Solution[i]=Production[i];
}


execute Output_Script
{
var ofile = new IloOplOutputFile("Risultati.txt")
ofile.writeln("Objective=", cplex.getObjValue());
for (var i in Products)
ofile.writeln("x",i,"=", Solution[i]);
ofile.close();


}
```

Strutture dati: le tuple

In opl oltre agli array ed alle matrici sono presenti delle strutture dati definite «tuple» che consentono di rappresentare oggetti più complessi utili ad implementare alcuni modelli di ottimizzazione.

Esempio nel caso di un modello su rete:

```
//Dichiariamo un oggetto chiamato arc
tuple arc{
int fromNodo;
int toNodo;
float cap;
}
{arc} Arcs={<1,2,10>,<1,3,5>}; //la dichiarazione dell'insieme è direttamente nel .mod
{arc} Arcs =…; //la dichiarazione dell'insieme viene fatta nel .dat
```

Iscrizione a moodle

Collegarsi al link seguente e compilare il form:

https://goo.gl/Z3rfUy

Tutto il materiale didattico sarà caricato nella pagina del corso di **Ricerca Operativa**

Esercitazioni di laboratorio

- Potete svolgere le esercitazioni a gruppi di 2/3 persone;

- Ogni esercitazione dovrà essere consegnata caricando il report finale su moodle;

- Il report finale (in Word) dovrà contenere:
  - descrizione del problema;
  - parametri e variabili decisionali;
  - funzione obiettivo e vincoli;
  - riepilogo della formulazione complessiva con descrizione del tipo di modello ottenuto;
  - codice Opl (relativo al .mod ed al .dat);
  - soluzione ottenuta e commento ai risultati.