UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Ricerca Operativa

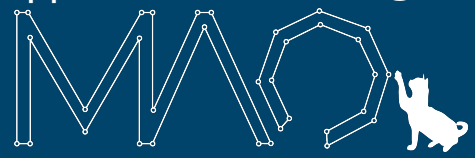
Introduzione a Gurobi

Corso di laurea in Ingegneria informatica

Anno accademico 2019/2020

Prof.ssa Renata Mansini

Gruppo di Ricerca MAO@DII



Modelli e Algoritmi di Ottimizzazione

Dipartimento di Ingegneria dell'Informazione

Sito web: `http://or-dii.unibs.it`

Cos'è GUROBI?

Gurobi Optimizer è un software commerciale (scritto in C) dedicato alla risoluzione di problemi di programmazione matematica (Programmazione Lineare, Programmazione Lineare Mista Intera,...)

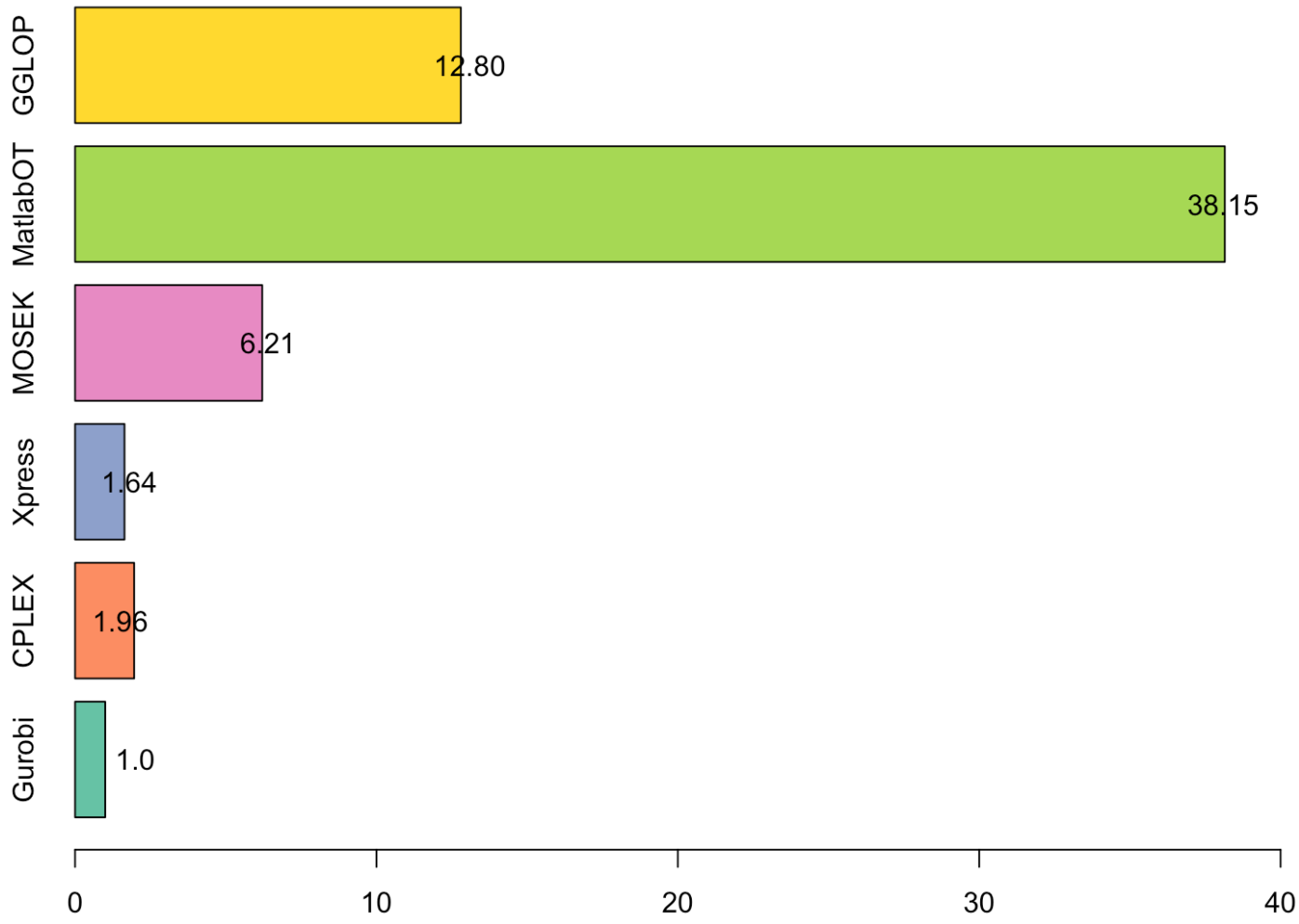
Altri risolutori di problemi di programmazione matematica:

- CPLEX
- Xpress
- MOSEK
- MATLAB Optimization Toolbox
- Google Glop
- ...

Perché GUROBI?

- È uno dei solver tra i più utilizzati sia in ambito aziendale che in ambito accademico;
- Offre elevate prestazioni e possibilità di personalizzazione;
- È strutturato per sfruttare nativamente processori multi-core e calcolo distribuito su rete;
- API disponibili per una vasta gamma di linguaggi e ambienti di programmazione (C, C++, **Java**, .NET, Python, MATLAB, R).

Perché GUROBI?



Confronto tra i solver più utilizzati (tempi risolutivi su benchmark MIPLIB).

Installare Gurobi

- Andare all'indirizzo `www.gurobi.com` e creare un account;
- Scaricare e installare la versione 9.0.1 di Gurobi Optimizer;
- Andare alla pagina *Download & Licenses* → *Academic license* e seguire le istruzioni per la richiesta e attivazione della licenza.
  N.B. Al momento dell'attivazione della licenza è necessario essere connessi alle rete universitaria.
- Per poter utilizzare Gurobi, includere la libreria gurobi.jar nel proprio progetto. Tale libreria sarà localizzata nel percorso scelto al momento dell'installazione del software.

Installare Gurobi (senza connessione alla rete universitaria)

- Andare all'indirizzo `www.gurobi.com` e creare un account;
- Scaricare e installare la versione 9.0.1 di Gurobi Optimizer;
- Andare alla pagina *Download & Licenses* → *Online Course License* e seguire le istruzioni per la richiesta e attivazione della licenza.

  N.B. La licenza è attivabile da qualsiasi rete, ma è limitata a 2000 variabili e 2000 vincoli (più che sufficienti per le prossime settimane di corso) e deve essere rinnovata ogni mese.
- Per poter utilizzare Gurobi, includere la libreria gurobi.jar nel proprio progetto. Tale libreria sarà localizzata nel percorso scelto al momento dell'installazione del software.

L'esempio

Useremo un semplice modello di esempio per presentare le principali caratteristiche dell'interfaccia Java di Gurobi.
In questa lezione vedremo come:

- costruire un modello matematico;
- ottimizzare un modello;
- ricavare informazioni relative alla soluzione ottenuta.

L'esempio

Useremo un semplice modello di esempio per presentare le principali caratteristiche dell'interfaccia Java di Gurobi.
In questa lezione vedremo come:

- costruire un modello matematico;
- ottimizzare un modello;
- ricavare informazioni relative alla soluzione ottenuta.

Ottimizzeremo il seguente modello:

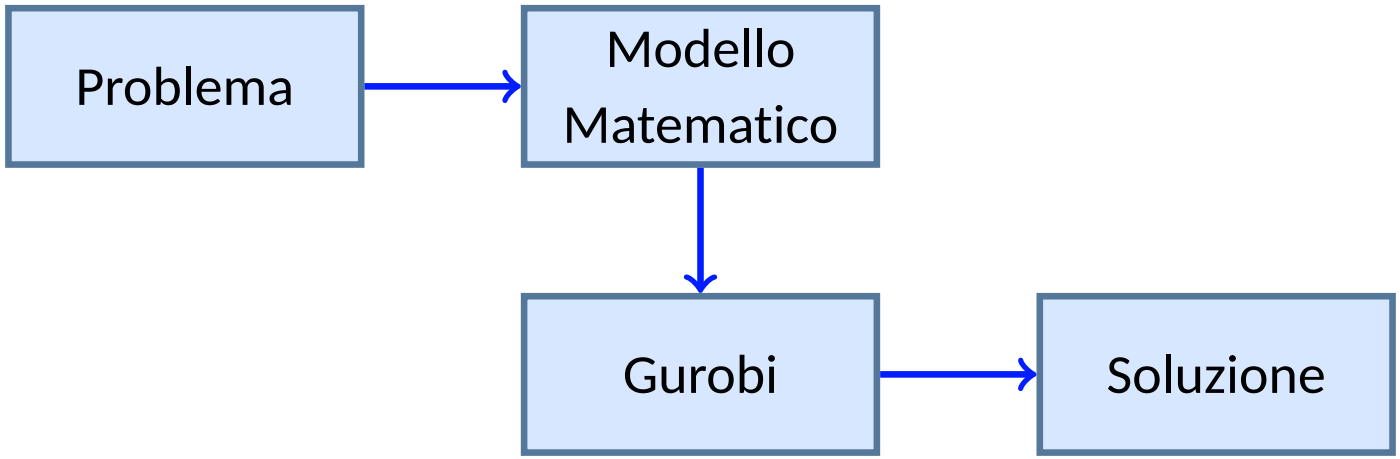
$$
\begin{aligned}
\textbf{Max} \quad & x + y + 2z \\
\textbf{s.a.} \quad & x + 2y + 3z \leq 4 \\
& x + y \leq 1 \\
& x, y, z \geq 0
\end{aligned}
$$

L’esempio

Potete trovare l’esempio completo nella comunità didattica:

esempioGurobi.java

Gurobi come risolutore black-box



Soluzione: Assegnamento di un valore a ciascuna variabile del problema (se un assegnamento ammissibile esiste).

Il risolutore potrebbe non produrre in output la soluzione ottima:

- Non esiste alcuna soluzione ammissibile;
- Gurobi richiede troppo tempo e\o troppa memoria.

L’environment

Si inizia importando le necessarie classi di Gurobi

```
import gurobi.*;
```

L'environment

Si inizia importando le necessarie classi di Gurobi

```
import gurobi.*;
```

Si istanzia poi un *GRBEnv*, l'environment

```
GRBEnv env = new GRBEnv("esempioGurobi.log");
```

Al costruttore si può specificare il nome/percorso del file di log.

Per creare e ottimizzare un modello ci servirà sempre un environment.

L'environment

Si inizia importando le necessarie classi di Gurobi

```
import gurobi.*;
```

Si istanzia poi un *GRBEnv*, l'environment

```
GRBEnv env = new GRBEnv("esempioGurobi.log");
```

Al costruttore si può specificare il nome/percorso del file di log.

Per creare e ottimizzare un modello ci servirà sempre un environment.

È possibile cambiare decine di parametri dell'environment:

```
env.set(GRB.IntParam.Threads, 4);
env.set(GRB.IntParam.Presolve, 2);
env.set(GRB.DoubleParam.TimeLimit, 600);
```

Parametri dell'Environment

È possibile cambiare decine di parametri dell'environment:

```
env.set(GRB.IntParam.Threads, 4);
```

Il parametro *Threads* determina quanti threads (e di conseguenza core) Gurobi userà durante l'ottimizzazione del modello.

Utile quando si vogliono risolvere più modelli in parallelo o nel caso si voglia mantenere potenza computazionale a disposizione per altri task.

Il valore di default è 0 (*Threads* = numero di core del processore utilizzato).

Parametri dell'Environment

È possibile cambiare decine di parametri dell'environment:

```
env.set(GRB.IntParam.Presolve, 2);
```

Il parametro *Presolve* controlla lo sforzo computazionale nella fase di presolve. -1 corrisponde a un settaggio automatico. Altre opzioni sono: nessun presolve (0), conservativo (1) e aggressivo (2).

Una fase di presolve più aggressiva richiede più tempo, ma può talvolta portare a modelli più semplici e veloci da risolvere.

-1 è il valore di default.

Parametri dell'Environment

È possibile cambiare decine di parametri dell'environment:

```
env.set(GRB.DoubleParam.TimeLimit, 600);
```

Il parametro *TimeLimit* determina il tempo massimo che Gurobi può dedicare alla risoluzione del modello.

Viene espresso in secondi.

Il valore di default è ∞

Il modello

Ore che abbiamo un environment, possiamo creare un modello.

```
GRBModel model = new GRBModel(env);
```

Un *GRBModel* contiene un singolo problema di ottimizzazione. Consiste di un set di variabili, un set di vincoli e di una funzione obiettivo, oltre a tutti i loro attributi.

Con la precedente linea di codice viene creato un modello vuoto, a cui aggiungeremo poi le parti necessarie.

Aggiungere variabili

Possiamo iniziare ad aggiungere variabili al modello.

```
GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "x");
GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "y");
GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "z");
```

Le variabili vengono aggiunte tramite il metodo *addVar* dell'oggetto modello. Una variabile è sempre associata a un singolo oggetto modello.

Aggiungere variabili

Possiamo iniziare ad aggiungere variabili al modello.

```
GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "x");
GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "y");
GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "z");
```

Il primo parametro è il lower bound della variabile.

Aggiungere variabili

Possiamo iniziare ad aggiungere variabili al modello.

```
GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "x");
GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "y");
GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "z");
```

Il primo parametro è il lower bound della variabile.

Il secondo parametro è l'upper bound della variabile.

Aggiungere variabili

Possiamo iniziare ad aggiungere variabili al modello.

```
GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "x");
GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "y");
GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "z");
```

Il primo parametro è il lower bound della variabile.

Il secondo parametro è l'upper bound della variabile.

Il terzo parametro è il coefficiente assunto dalla variabile nella funzione obiettivo. In questo caso è zero per che costruiremo in seguito la funzione obiettivo.

Aggiungere variabili

Possiamo iniziare ad aggiungere variabili al modello.

```
GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0,
        GRB.CONTINUOUS, "x");
GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0,
        GRB.CONTINUOUS, "y");
GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0,
        GRB.CONTINUOUS, "z");
```

Il quarto parametro è il tipo della variabile. Ci sono tre tipologie principali:

- *GRB.BINARY*
- *GRB.INTEGER*
- *GRB.CONTINUOUS*

Aggiungere variabili

Possiamo iniziare ad aggiungere variabili al modello.

```
GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "x");
GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "y");
GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0,
    GRB.CONTINUOUS, "z");
```

Il quarto parametro è il tipo della variabile. Ci sono tre tipologie principali:

- *GRB.BINARY*
- *GRB.INTEGER*
- *GRB.CONTINUOUS*

L'ultimo parametro è il nome della variabile.

Funzione obiettivo

Il prossimo step è la costruizione della funzione obiettivo:

```
//Aggiunge la funzione obiettivo: max x + y + 2 z

GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x);
expr.addTerm(1.0, y);
expr.addTerm(2.0, z);
model.setObjective(expr, GRB.MAXIMIZE);
```

Funzione obiettivo

Il prossimo step è la costruizione della funzione obiettivo:

```
//Aggiunge la funzione obiettivo: max x + y + 2 z

GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x);
expr.addTerm(1.0, y);
expr.addTerm(2.0, z);
model.setObjective(expr, GRB.MAXIMIZE);
```

Per prima cosa si crea una espressione lineare vuota.

Si aggiungono poi i tre termini necessari all'espressione, usando il metodo *addTerm*. Il primo parametro è il coefficiente che moltiplica la variabile, che è il secondo parametro.

Funzione obiettivo

Il prossimo step è la costruizione della funzione obiettivo:

```
//Aggiunge la funzione obiettivo: max x + y + 2 z

GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x);
expr.addTerm(1.0, y);
expr.addTerm(2.0, z);
model.setObjective(expr, GRB.MAXIMIZE);
```

Infine, si indica che questa espressione è la funzione obiettivo del nostro modello tramite il metodo *setObjective*.

Il secondo parametro indica il senso dell'ottimizzazione (max vs min).

I vincoli

Aggiungiamo ora i vincoli al modello.

```
// Aggiunge il vincolo: x + 2 y + 3 z <= 4
expr = new GRBLinExpr();
expr.addTerm(1.0, x);
expr.addTerm(2.0, y);
expr.addTerm(3.0, z);
GRBConstr c0 = model.addConstr(expr,
                               GRB.LESS_EQUAL, 4.0, "c0");
```

I vincoli

Aggiungiamo ora i vincoli al modello.

```
// Aggiunge il vincolo: x + 2 y + 3 z <= 4
expr = new GRBLinExpr();
expr.addTerm(1.0, x);
expr.addTerm(2.0, y);
expr.addTerm(3.0, z);
GRBConstr c0 = model.addConstr(expr,
                               GRB.LESS_EQUAL, 4.0, "c0");
```

Come per le variabili, anche i vincoli devono essere associati a un modello. I vincoli vengono aggiunti tramite il metodo *addConstr*.

Il primo parametro di *addConstr()* è il membro sinistro del vincolo, che viene creato esattamente come la funzione obiettivo.

I vincoli

Aggiungiamo ora i vincoli al modello.

```
// Aggiunge il vincolo: x + 2 y + 3 z <= 4
expr = new GRBLinExpr();
expr.addTerm(1.0, x);
expr.addTerm(2.0, y);
expr.addTerm(3.0, z);
GRBConstr c0 = model.addConstr(expr,
                               GRB.LESS_EQUAL, 4.0, "c0");
```

Il secondo parametro è il verso del vincolo (*GRB.LESS_EQUAL*, *GRB.GREATER_EQUAL* o *GRB.EQUAL*)

Il terzo parametro è il termine destro (una costante nel nostro esempio).

L'ultimo parametro è il nome del vincolo.

I vincoli

Il secondo vincolo è creato allo stesso modo

```
// Aggiunge il vincolo: x + y <= 1
expr = new GRBLinExpr();
expr.addTerm(1.0, x);
expr.addTerm(1.0, y);
GRBConstr c1 = model.addConstr(expr,
                    GRB.LESS_EQUAL, 1.0, "c1");
```

Altri metodi

Altri metodi utili

```
expr.addConstant(5);
```

Il metodo *addConstant* aggiunge un valore costante all'espressione.

Altri metodi

Altri metodi utili

```
expr.addConstant(5);
```

Il metodo *addConstant* aggiunge un valore costante all'espressione.

```
model.getVarByName("x");
```

Il metodo *getVarByName* ritorna la variabile del modello con il nome specificato.

Ottimizzazione

Il nostro modello è pronto per essere ottimizzato:

```
model.optimize();
```

Questo metodo esegue l'ottimizzazione del modello e popola diversi attributi del modello, delle variabili e dei vincoli.

L'attributo *Status* del modello ci indica la conclusione a cui è giunto Gurobi al termine dell'ottimizzazione.
Alcuni dei valori più comuni sono:

- *GRB.OPTIMAL*
- *GRB.INFEASIBLE*
- *GRB.TIME_LIMIT*

Analisi dei risultati 1/3

Ora che l'ottimizzazione è completata, possiamo analizzare i risultati ottenuti.

```
x.get(GRB.StringAttr.VarName);
x.get(GRB.DoubleAttr.X);
model.get(GRB.DoubleAttr.ObjVal);
```

L'attributo *VarName* contiene il nome della variabile.

L'attributo *X* contiene il valore della variabile nella soluzione corrente.

L'attributo del modello *ObjVal* contiene il valore della funzione obiettivo della soluzione corrente.

Analisi dei risultati 2/3

Altri attributi delle variabili che useremo

```
x.get(GRB.DoubleAttr.RC);
x.get(GRB.DoubleAttr.SAObjLow);
x.get(GRB.DoubleAttr.SAObjUp);
```

L'attributo *RC* contiene il valore del coefficiente di costo ridotto associato alla variabile.

Gli attributi *SAObjLow* e *SAObjUp* contengono informazioni di sensitività dei coefficienti in funzione obiettivo (cfr. lezione su analisi di sensitività)

Analisi dei risultati 3/3

Alcuni attributi dei vincoli che useremo

```
c0.get(GRB.DoubleAttr.Slack);
c0.get(GRB.DoubleAttr.Pi);
c0.get(GRB.DoubleAttr.SARHSLow);
c0.get(GRB.DoubleAttr.SARHSUp);
```

L'attributo *Slack* contiene il valore della variabile di slack (surplus) associata al vincolo.

L'attributo *Pi* contiene il valore del prezzo ombra associato al vincolo (cfr. lezioni su dualità)

Gli attributi *SARHSLow* e *SARHSUp* contengono informazioni di sensitività dei termini noti (cfr. lezione su analisi di sensitività)

Rilassamento continuo

Quando si lavora con un modello di Programmazione Lineare Intera, è spesso utile risolvere il suo *rilassamento continuo*:

```
GRBModel rilassamento = model.relax();
rilassamento.optimize();
```

Rilascio della memoria

Quando non è più necessario, è necessario liberarsi dell'environment e del modello:

```
model.dispose();
env.dispose();
```

Questi metodi liberano le risorse associate al modello e all'environment. Il Garbage collector recupererebbe eventualmente queste risorse, ma, avendo a che fare con un software esterno, il rilascio non è immediato.