

Integrando o Gurobi em aplicações C++

André L. Maravilha

Universidade Federal de Minas Gerais (UFMG)



Belo Horizonte, 10 de junho de 2013

Sumário

- 1 Introdução
- 2 Instalação do Gurobi
 - Windows
 - Linux
 - Licença de uso
- 3 Integrando o Gurobi em aplicações C++
 - Utilizando a API para C++
 - Compilando o código
- 4 Configurando parâmetros do Gurobi

Sumário

1 Introdução

2 Instalação do Gurobi

- Windows
- Linux
- Licença de uso

3 Integrando o Gurobi em aplicações C++

- Utilizando a API para C++
- Compilando o código

4 Configurando parâmetros do Gurobi

Apresentação do Gurobi

- O Gurobi é uma ferramenta para resolver problemas de programação matemática:
 - problemas de programação linear;
 - problemas de programação quadrática;
 - problemas de programação quadrática com restrições;
 - problemas de programação linear inteira mista;
 - problemas de programação quadrática inteira mista;
 - problemas de programação quadrática inteira mista com restrições.
- Projetado para explorar os recursos oferecidos por arquiteturas de multi-processamento.
- Faz uso de implementações avançadas dos algoritmos utilizados.

Integração com outros aplicativos

- O Gurobi oferece APIs para integração com diversas linguagens de programação:
 - C
 - C++
 - Java
 - C#
 - Python
 - MATLAB
 - R
- As APIs oferecem uma interface de fácil utilização.
- Possibilita controlar diversos aspectos do processo de otimização.

Sumário

1 Introdução

2 Instalação do Gurobi

- Windows
- Linux
- Licença de uso

3 Integrando o Gurobi em aplicações C++

- Utilizando a API para C++
- Compilando o código

4 Configurando parâmetros do Gurobi

Instalação do Gurobi

- Acesse <http://www.gurobi.com/>
- Registre-se gratuitamente.
- Faça o download do Gurobi (disponível para Windows, Linux e Mac OS).
- Obtenha a licença de uso:
 - a licença é gratuita para estudantes;
 - deve ser obtida a partir da rede da universidade.
- O processo de instalação varia de acordo com o sistema utilizado.

Instalando no Windows

- Execute o arquivo de instalação obtido a partir do site.
- Após a instalação, é necessário obter e instalar a licença de uso.
- Será criado um atalho para o Gurobi:
 - este atalho abre um terminal para interagir com o Gurobi;
 - a interação é realizada por comandos digitados diretamente no terminal.

Instalando no Linux

- Descompacte o arquivo obtido em **/opt**. Com isso será criado o diretório **/opt/gurobi550/linux64/**. Este caminho pode alterar dependendo da versão do Gurobi e do sistema utilizado.
- Adicione as seguintes linhas no arquivo **.bashrc**:

```
export GUROBI_HOME="/opt/gurobi550/linux64"  
export PATH="${PATH}:${GUROBI_HOME}/bin"  
export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:${GUROBI_HOME}/lib"
```

- Algumas distribuições Linux utilizam outra forma de definir os caminhos para as bibliotecas ao invés da variável do sistema **LD_LIBRARY_PATH**.
- Após a instalação, é necessário obter a licença de uso.
- Para abrir o terminal para interação com o Gurobi, utilize o comando **gurobi.sh**.

Obtendo e instalando a licença de uso

- O Gurobi possui diversos tipos de licença. Aqui será descrito como obter a licença acadêmica.
- Primeiro é necessário se registrar no site do Gurobi.
- Em seguida, se autenticar utilizando a conta cadastrada.
- Acesse a opção **Download > Licenses**.
- Escolha **Free academic**.
- Aceite os termos de uso e clique em **Request License**.
- Será exibido algo como:

grbgetkey xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx

- Execute este comando no Iniciar/Executar (Windows) ou em um terminal (Linux).
- Acesse o terminal de interação com o Gurobi, se não ocorrer nenhum erro, então a licença foi corretamente instalada.

Sumário

- 1 Introdução
- 2 Instalação do Gurobi
 - Windows
 - Linux
 - Licença de uso
- 3 Integrando o Gurobi em aplicações C++
 - Utilizando a API para C++
 - Compilando o código
- 4 Configurando parâmetros do Gurobi

Integrando o Gurobi em aplicações C++

- A ideia geral para modelar e resolver um problema de programação matemática utilizando as APIs do Gurobi é:
 - obter os dados do problema;
 - criar um novo modelo;
 - criar as variáveis;
 - definir a função objetivo;
 - definir as restrições;
 - resolver o modelo.
- Para exemplificar o uso da API do Gurobi para a linguagem de programação C++, será utilizado o problema da mochila binário.

O problema da mochila binário

Dado um conjunto de n objetos, cada objeto $i \in [0, n - 1]$ possui um lucro p_i e um peso w_i . Considerando uma mochila de capacidade máxima C , deve-se definir um subconjunto de objetos que serão inseridos na mochila de forma que a soma do lucro destes objetos seja maximizada e, além disso, a soma dos pesos destes objetos inseridos na mochila não devem ultrapassar a capacidade máxima C . Para cada objeto pode-se associar uma variável binária x_i . Se esta variável for igual a 1, então o objeto foi inserido na mochila, 0 (zero) caso contrário.

$$\max \sum_{i=0}^{n-1} p_i x_i$$

sujeito a:

$$\sum_{i=0}^{n-1} w_i x_i \leq C$$

$$x_i \in \{0, 1\}, \quad \forall i = 0, 1, \dots, n - 1$$

Definir os dados do problema

```
// Creates a problem
int n = 100;
double C = 10000;
deque<double> profit;
deque<double> weight;

for (int i = 0; i < n; ++i) {
    profit.push_back(rand() % 1000);
    weight.push_back(rand() % 1000);
}
```

Criar um modelo

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

Criar as variáveis

```
// Create variables
deque<GRBVar> vars;
for (int i = 0; i < n; ++i) {
    GRBVar x = model.addVar(
        0.0,          // lower bound
        1.0,          // upper bound
        0.0,          // coefficient in the objective function
        GRB_BINARY,   // domain
        "x_" + i      // name (optional)
    );
    vars.push_back(x);
}

// Integrate new variables
model.update();
```


Definir a função objetivo

```
// Set objective
GRBLinExpr objective;
for (int i = 0; i < n; ++i) {
    objective += profit[i] * vars[i];
}

model.setObjective(objective, GRB_MAXIMIZE);
```

Definir as restrições

```
// Add the constraint
GRBLinExpr constraint;
for (int i = 0; i < n; ++i) {
    constraint += weight[i] * vars[i];
}

model.addConstr(constraint <= C);
```

Resolver o modelo

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

Obtendo e exibindo o resultado encontrado

```
for (int i = 0; i < n; ++i) {  
    cout << vars[i].get(GRB_StringAttr_VarName) << " = "  
        << vars[i].get(GRB_DoubleAttr_X) << endl;  
}  
  
cout << "Profit:  " << model.get(GRB_DoubleAttr_ObjVal) << endl;  
cout << "Elapsed time:  " << model.get(GRB_DoubleAttr_Runtime);
```

Compilando o código

- No processo de compilação é necessário informar ao compilador onde se encontram os arquivos de cabeçalho (arquivos .h).
- Os arquivos de cabeçalho se encontram no diretório **include**, localizado dentro do diretório onde o Gurobi foi instalado.
- No g++ isto é feito informando o seguinte parâmetro:
`-I/opt/gurobi550/linux64/include`
- Após a compilação, é feita a link-edição para gerar o arquivo executável. Neste processo é necessário informar ao link-editor onde se encontram as bibliotecas do Gurobi e quais devem ser utilizadas.
- No g++ isso é feito informando os seguintes parâmetros:
`-L/opt/gurobi550/linux64/lib -lgurobi_c++ -lgurobi55`
- Esses passos podem variar de acordo com o compilador utilizado.

Sumário

- 1 Introdução
- 2 Instalação do Gurobi
 - Windows
 - Linux
 - Licença de uso
- 3 Integrando o Gurobi em aplicações C++
 - Utilizando a API para C++
 - Compilando o código
- 4 Configurando parâmetros do Gurobi

Configurando parâmetros do Gurobi

- Não escrever dados da execução na saída padrão:
`model.getEnv().set(GRB_IntParam_OutputFlag, 0);`
- Limitar o tempo de execução em 3600 segundos (1 hora):
`model.getEnv().set(GRB_DoubleParam_TimeLimit, 3600.0);`
- Definir valor de *cutoff*:
`model.getEnv().set(GRB_DoubleParam_Cutoff, 1000);`
- Definir uma solução inicial:
`vars[i].set(GRB_DoubleAttr_Start, 1.0);`
- Existem diversos outros parâmetros. Para uma lista completa e detalhada consulte o manual de referência do Gurobi.

Referências

- <http://www.gurobi.com/>
- <http://groups.google.com/group/gurobi>