Manual AMPL

1 Instalação

A instalação do pacote gratuito do AMPL (versão para estudantes) pode ser realizada através do link: `http://ampl.com/try-ampl/download-a-free-demo/`.

Essa é uma versão grátis, a qual permite a resolução de problemas com até 500 variáveis, com 500 restrições lineares ou com 300 restrições não-lineares. Ainda, essa versão acompanha os principais resolvedores disponíveis para AMPL.

Os principais necessários para o uso do ambiente AMPL são o executável `ampl` e a sua licensa `ampl.lic`, os quais podem ser movidos diretamente para o diretório onde se encontra o resolvedor a ser utilizado.

Após, para que seja possível incluir o nosso resolvedor em C++ ao AMPL, é preciso baixar o diretório com todos os arquivos disponíveis em:

`http://ampl.com/netlib/ampl/solvers/index.html`

Esse diretório contém a fonte para a biblioteca de rotinas que auxiliam resolvedores a funcionarem com AMPL. Essa biblioteca é chamada de `amplsolver.a`, para o caso de sistemas Unix. Os serviços providos por essa biblioteca incluem a leitura de arquivos de saída `.nl` do AMPL e escrita de arquivos de solução `.sol`. Para compilar o `amplsolver.a`, é preciso, via terminal, executar os seguintes comandos:

1. No diretório baixado, executar

   ```
   ./configure
   ```

   para criar o diretório `sys.'uname -m'.'uname -s'`, específico para o sistema operacional utilizado.

2. Execute:

   ```
   cd sys.'uname -m'.'uname -s'
   make
   ```

   e a biblioteca `amplsolver.a` será criada nesse diretório específico para o sistema.

2 Resolvendo Problemas em AMPL

Para códigos em C++, é necessário incluir o arquivo `asl.h`, o qual facilita a leitura e acesso às informações dos problema a ser resolvido, como número de variáveis, restrições e objetivos.

Esse arquivo `asl.h` é encontrado na pasta de *includes* necessários para a conexão do nosso resolvedor e o AMPL. No caso, essa pasta contém todos os arquivos `*.h` e `*.c` contidos do subdiretório `sys.'uname -m'.'uname -s'`.

A partir do arquivo *asl.h* é necessário declarar um ponteiro do tipo `ASL`, o qual será utilizado diretamente para ler e acessar o modelo do problema. A função `ASL_alloc`(*ASLtype*) aloca um objeto do tipo `ASL` e o parâmetro *ASLtype* determina quantas não-linearidades são tratadas. No caso, como parâmetro foi utilizado `ASL_read_fg`, o qual auxilia na leitura e tratamento de problemas não-lineares. Mais argumentos e suas características podem ser encontrados em **[1]**.

Inicialmente, o nome do problema no formato *.mod* deve ser informado como parâmetro para o código principal do seu resolvedor. O nome da instância é armazenado na variável `stub` e é utilizada para acessar o arquivo através da função `jac0dim(char *stub, fint stub_len)`.

A partir do arquivo `.nl` retornado por `jac0dim(char *stub, fint stub_len)`, é possível extrair todas as informações do problema informado através da função `fg_read_ASL(ASL *asl, FILE *nl, int flag)`, a qual auxilia na leitura e tratamento de problemas não-lineares.

Através da função `objval(int nobj, double *x, fint *nerror)` é possível obter o valor do objetivo `nobj` para um ponto `*x` escolhido. Se o problema possui um objetivo, então estamos interessados em obter o valor do objetivo `nobj` $= 0$. Ainda, o argumento `fint *nerror` controla o que acontece se ocorre algum erro na função de avaliação.

Para facilitar a utilização dessa função de avaliação em qualquer parte do código do resolvedor, declara-se uma função na estrutura principal do resolvedor que, no caso mono-objetivo, recebe um valor no tipo `double` e retorna o valor da função de avaliação `objval`. Dessa forma, essa nova função pode ser passada por parâmetro para outras rotinas do nosso resolvedor e a função de avaliação estará sempre acessível sem ser necessário todo o processo de leitura do modelo.

No caso do problema possuir limites sobre as variáveis do problema, essas informações podem ser acessadas através dos vetores `LUv` e `Uvx`, limites inferiores e superiores, respectivamente. Ainda, quando `Uvx` é nulo, é possível acessar tanto os limites inferiores e superiores através do vetor `LUv`, onde os índices pares armazenam os limites inferiores e os índices ímpares armazenam limites superiores.

Ainda, no caso do problema possuir restrições, essas informações podem ser acessadas através dos vetores `LUrhs` e `Urhsx`, as restrições inferiores e superiores, respectivamente. Da mesma forma que no caso de limites sobre as variáveis, se `Urhsx` for nulo, é possível acessar ambas restrições através do vetor `LUrhs`.

Uma vez executada a leitura do problema e armazenado todas as informações necessárias, é possível usar essas informações como parâmetros para outras rotinas do nosso resolvedor, assim como a função de avaliação. Na seção 4 é possível encontrar um exemplo de como acessar informações de um problema em AMPL não-linear e com limites nas suas variáveis.

3 Utilizando o ambiente AMPL

Para sistema operacional Linux, no pacote do AMPL existe o arquivo executável `ampl` e é possível utilizar o ambiente AMPL em terminal executando esse arquivo através de `./ampl`.

Uma vez dentro do ambiente AMPL em terminal Linux é possível resolver problemas em arquivos *.mod* utilizando qualquer resolvedor. Para isso, inicialmente é necessário incluir o problema ao ambiente através de:

```
ampl: include ack.mod;
ampl: solve;
```

Quando o comando `solve` é executado, então o problema incluído é resolvido pelo resolvedor padrão que acompanha o AMPL, o MINOS. Para utilizar o nosso resolvedor, é necessário que ele tenha sido compilado a priori e gerado um arquivo executável. Suponha que após compilar no nosso resolver tenha sido gerado o executável `solver`. Então, modificamos o resolvedor do AMPL através de:

```
ampl: include ack.mod;
ampl: options solver './solver';
ampl: solve;
```

Além de permitir que o resolvedor seja modificado, o comando `options` permite modificar outros parâmetros para o ambiente AMPL.

Outra alternativa para resolver problemas em AMPL é escrever um *script* de execução em um arquivo *.run*. Uma vez possuindo um `script.run` não é necessário entrar no ambiente `ampl` para incluir os problemas e modificar o resolvedor, basta apenas ter tais comandos no arquivo `script.run` e executá-lo através de:

```
./ampl script.run
```

Uma questão importante a ser citada é como compilar o resolvedor de forma a utilizá-lo no ambiente AMPL. Para o C++, pode-se proceder da seguinte forma:

```
g++ -c -g -Iinclude/ *.cpp -D Param1=$a -D Param2=$b
g++ -o solver main.o Estrategias.o libs/funcadd0.o libs/amplsolver.a -w
```

Nesse caso, além do `main.o`, podem ser colocados os nomes de todos os arquivos `*.cpp` que estão incluídos no `main.cpp`. Além, a biblioteca `amplsolver.a` e o arquivo `funcadd0.o` devem também informados no processo de compilação. O arquivo `funcadd0.o` é necessário para quando o modelo *.mod* utiliza funções que são definidas pelo usuário e o resolvedor precisa avaliar essas funções.

Note que, se o resolvedor necessita de parâmetros (além do nome do problema, pois esse é informado ao resolvedor de forma automática no ambiente AMPL), é mais fácil informá-los nesse momento, através de `-D Param1=$a -D Param2=$b`, onde `Param1` e `Param2` são os nomes parâmetros necessários no resolvedor e `$a` e `$b` são seus valores.

4 Exemplo de Utilização do AMPL

(desenvolvido a partir de exemplos encontrados em `http://ampl.com/resources/the-ampl-book/example-files/`)

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "asl.h"
#include "getstub.h"
#include "Estrategias.h"
#include "string.h"
#include <iomanip>

#define INF 1E+21
#define asl cur_ASL

using namespace std;

static fint NERROR = -1;

double objfun(double *x){
    return objval(0, x, &NERROR);
}

int main(int argc, char** argv){
    if(argc > 2){
        NERROR = -1;
        ASL *asl;
        FILE *nl;
        char *stub;
```

```
        asl = ASL_alloc(ASL_read_fg);
        stub = argv[1];
        nl = jac0dim(stub, (fint)strlen(stub));

        fg_read_ASL(asl, nl, 0);

        double *lb = NULL;
        double *ub = NULL;

        if(LUv[0] > -Infinity && LUv[1] < Infinity){
            lb = new double[n_var];
            ub = new double[n_var];
            for(int i=0; i<n_var; i++){
                lb[i] = LUv[2*i];
                ub[i] = LUv[2*i+1];
            }
        }
        cout<<";"<<n_var; // numero de variáveis do problema

        double mean = 0;
        for(int seed=1; seed<=SD; seed++){
            mean += PSO(&objfun, n_var, seed, lb, ub);
        }
        mean = mean/(double)SD;

        cout<<";"<<setprecision(10)<<mean;

    }
    else{

    }
    return 0;
}
```

5 Exemplo de Script para AMPL

```
rm resultados.csv
d=31
echo "FUNCAO;DIMENSAO;MEDIA" >> resultados.csv

for f in Instances/*
do
  echo $f
```

```
  echo -ne -e $f ";">> resultados.csv

  rm problem.run
  g++ -c -g -Iinclude/ *.cpp -D SD=$d
  g++ -o solver main.o Estrategias.o Solution.o libs/funcadd0.o libs/amplsolver.a -w

  echo "include" $f";" >> problem.run
  echo "options solver './solver';" >> problem.run
  echo "solve;" >> problem.run

  ./ampl problem.run >> resultados.csv
  echo >>resultados.csv

done
```

6 Referências

[1] David M. Gay. Hooking Your Solver to AMPL, 1997.