FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
Centro de Ciências Tecnológicas – CCT
Programa de Pós-Graduação em Informática Aplicada – PPGIA

Período: 2014.2 **Disciplina**: D279 – Otimização Combinatória e em Redes

Atividade computacional

Instruções:
1. A atividade deve ser realizada individualmente.
2. Implementações semelhantes estarão sujeitas a anulação imediata e definitiva.
3. A avaliação será progressiva através de acompanhamento pelo professor.

Parte I – Implementação de um modelo para *Fixed Charge Capacitated Network Design Problem*.

Formulação. Seja $G = (N, A)$ uma rede orientada, onde N é o conjunto de nós e A é o conjunto de arcos. Seja K o conjunto de demandas, cada uma delas caracterizada por uma origem s^k, um destino t^k, e uma quantidade d^k que deve ser transportada da origem para o destino. Seja f_{ij} o custo fixo de utilização do arco ij, c_{ij} o custo variável para transportar uma unidade de fluxo através do arco ij, e u_{ij} a capacidade do arco ij. Considere a variável x_{ij}^k que indica a quantidade de fluxo referente à demanda k no arco ij, e a variável binária y_{ij} que indica se o arco ij é utilizado ou não. O objetivo é minimizar os custos fixos e os custos variáveis.

$$\min \sum_{k\in K}\sum_{ij\in A} c_{ij}x_{ij}^k + \sum_{ij\in A} f_{ij}y_{ij}$$

$$\text{s.t.} \sum_{ij\in A} x_{ij}^k - \sum_{ji\in A} x_{ji}^k = \begin{cases} d^k, & \text{for } i = s^k \\ -d^k, & \text{for } i = t^k \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in N, \forall k \in K$$

$$\sum_{k\in K} x_{ij}^k \le u_{ij}y_{ij} \qquad \forall ij \in A$$

$$x_{ij}^k \ge 0 \qquad \forall ij \in A, \forall k \in K$$

$$y_{ij} \in \{0,1\} \qquad \forall ij \in A$$

Implementação. Implementar um modelo de programação matemática utilizando Java Concert para o problema. Uma instância deve ser lida de um arquivo de entrada (FCND.dat). Devem ser gerados 3 arquivos de saída: um arquivo com a saída padrão do CPLEX (FCND.log), um arquivo com a formulação matemática (FCND.lp), e um arquivo de solução (FCND.out) conforme especificado abaixo.

Entrada. A primeira linha contém um inteiro n, indicando a quantidade de nós (rotulados de 1 a n), e um inteiro m, indicando a quantidade de arcos. Cada uma das m linhas seguintes possui uma quintupla que caracteriza um arco: $(i, j, f_{ij}, c_{ij}, u_{ij})$. A linha seguinte contém um inteiro k indicando a quantidade de demandas. Cada uma das k linhas seguintes possui uma tripla que caracteriza uma demanda: (s^k, t^k, d^k).

Saída. O arquivo de solução deve apresentar o custo total, o tempo e os arcos efetivamente utilizados.

Exemplo de entrada:

```
 6              8
 1              2            100              1            100
 2              3             50              1             60
 2              4            100              1            100
 3              1            100              1            100
 4              6            100              1            100
 5              3            100              1            100
 5              4             50              1             60
 6              5            100              1            100
10
 1              3             10
 1              4             10
 2              1             10
 2              5             10
 3              6             10
 4              1             10
 4              5             10
 5              1             10
 5              6             10
 6              2             10
```

Exemplo de saída:

```
Objetive:              950,00
Lower bound:           950,00
Gap:                   0,0000
Status:               Optimal
Time:                    0,03
 1              2            100
 2              3             50
 2              4            100
 3              1            100
 4              6            100
 5              3            100
 6              5            100
```

Parte II – Implementação de uma relaxação lagrangeana para o problema.

Relaxação. Relaxar a restrição de capacidade de cada arco *ij* e associar um multiplicador lagrangeano μ_{ij}, conforme modelo abaixo.

$$\min_{y} \sum_{k\in K}\sum_{ij\in A} c_{ij}x_{ij}^{k} + \sum_{ij\in A} f_{ij}y_{ij} + \sum_{ij\in A}\mu_{ij}\left(\sum_{k\in K} x_{ij}^{k} - u_{ij}y_{ij}\right)$$

$$\text{s.t.} \quad \sum_{ij\in A} x_{ij}^{k} - \sum_{ji\in A} x_{ji}^{k} = \begin{cases} d^{k}, & \text{for } i = s^{k} \\ -d^{k}, & \text{for } i = t^{k} \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in N, \forall k \in K$$

$$x_{ij}^{k} \geq 0 \qquad \forall ij \in A, \forall k \in K$$

$$y_{ij} \in \{0,1\} \qquad \forall ij \in A$$

Implementação. Implementar o algoritmo do subgradiente conforme ilustrado abaixo. UB deve ser obtido pela resolução do modelo original onde todos os links são utilizados ($y_{ij} = 1, \forall ij \in A$). Em cada iteração, o subproblema lagrangeano deve ser resolvido, os limites inferiores e superiores devem ser adequadamente atualizados. Considere os parâmetros $K = 1000, \beta = 10, \varepsilon = 0.000001$. Note que a cada iteração apenas a função objetivo deve ser modificada de acordo com os multiplicadores lagrangeanos. Deve ser gerado um arquivo de saída FCND.lgr conforme especificado abaixo.

Algorithm 1 Subgradient algorithm

{Input}
An upper bound UB
{Initialization}
$\mu^0 = 0$
$\lambda_0 = 2$
{Subgradient iterations}
$k = 0$
while $k \leq K$ **do** {stopping criterion}
  $\gamma^k = Ax^k - b$ {gradient of $L(\mu^k)$}
  $\theta_k = \lambda_k(UB - L(\mu^k))/||\gamma^k||^2$ {step size} $\{||\gamma|| = (\sum_j \gamma_j^2)^{1/2}\}$
  $\mu^{k+1} = \max\{0, \mu^k + \theta_k\gamma^k\}$
  if $||\mu^{k+1} - \mu^k|| < \varepsilon$ **then**
    Stop
  end if
  if no progress in more than β iterations **then**
    $\lambda_{k+1} = \lambda_k/2$
  else
    $\lambda_{k+1} = \lambda_k$
  end if
  $k = k + 1$
end while

Saída. Para cada iteração do algoritmo, o arquivo de saída deve apresentar o limite superior, o limite inferior, o gap e o tempo decorrido. Ao final, deve ser também apresentado o limite inferior obtido pela relaxação linear do modelo original.

Parte III – Implementação de uma decomposição de Benders para o problema.

Reformulação. Note que podemos reformular o problema conforme abaixo.

$$\min_{y\in\{0,1\}^{|A|}} \sum_{ij\in A} f_{ij}\, y_{ij} +$$

$$\begin{aligned}
\min_{x}\quad & \sum_{k\in K}\sum_{ij\in A} c_{ij} x_{ij}^k \\
\text{s.t.}\quad & \sum_{ij\in A} x_{ij}^k - \sum_{ji\in A} x_{ji}^k = \begin{cases} d^k, & \text{for } i = s^k \\ -d^k, & \text{for } i = t^k \\ 0, & \text{otherwise} \end{cases} && \forall\, i\in N, \forall\, k\in K \\
& \sum_{k\in K} x_{ij}^k \le u_{ij}\, y_{ij} && \forall\, ij\in A \\
& x_{ij}^k \ge 0 && \forall\, ij\in A, \forall\, k\in K
\end{aligned}$$

Fixação de *y*. Para um dado valor de $y=\overline{y}$, obtemos o seguinte programa linear e seu dual.

$$\begin{aligned}
\sum_{ij\in A} f_{ij}\,\overline{y}_{ij} + \min_{x}\quad & \sum_{k\in K}\sum_{ij\in A} c_{ij} x_{ij}^k \\
\text{s.t.}\quad & \sum_{ij\in A} x_{ij}^k - \sum_{ji\in A} x_{ji}^k = \begin{cases} d^k, & \text{for } i = s^k \\ -d^k, & \text{for } i = t^k \\ 0, & \text{otherwise} \end{cases} && \forall\, i\in N, \forall\, k\in K \\
& -\sum_{k\in K} x_{ij}^k \ge -u_{ij}\,\overline{y}_{ij} && \forall\, ij\in A \\
& x_{ij}^k \ge 0 && \forall\, ij\in A, \forall\, k\in K
\end{aligned}$$

$$\Updownarrow$$

$$\begin{aligned}
\sum_{ij\in A} f_{ij}\,\overline{y}_{ij} + \max_{\alpha,\beta}\quad & \sum_{i\in N}\sum_{k\in K} g(i,k)\,\alpha_i^k - \sum_{ij\in A} (u_{ij}\,\overline{y}_{ij})\,\beta_{ij} \\
\text{s.t.}\quad & \sum_{v\in N} h(ij,v)\,\alpha_v^k - \beta_{ij} \le c_{ij} && \forall\, ij\in A, \forall\, k\in K \\
& \alpha_i^k \in \mathbb{R} && \forall\, i\in N, \forall\, k\in K \\
& \beta_{ij} \ge 0 && \forall\, ij\in A
\end{aligned}$$

where

$$g(i,k) = \begin{cases} d^k, & \text{for } i = s^k \\ -d^k, & \text{for } i = t^k \\ 0, & \text{otherwise} \end{cases} \qquad \text{and} \qquad h(ij,v) = \begin{cases} 1, & \text{for } i = v \\ -1, & \text{for } j = v \\ 0, & \text{otherwise} \end{cases}$$

Região viável. Note que a região viável do problema dual não depende do valor de *y*. Se o problema dual é inviável, ~~ou o problema original é ilimitado ou~~ o problema original é inviável. Do contrário, podemos em tese enumerar todos os pontos extremos e direções extremas da região viável do problema dual. Dado um valor de $y=\overline{y}$, o problema dual pode ser resolvido:

(1) checando-se se para algum raio extremo $\sum_{i\in N}\sum_{k\in K} g(i,k)\,\alpha_i^k - \sum_{ij\in A} (u_{ij}\,\overline{y}_{ij})\,\beta_{ij} > 0$. Neste caso, temos que o problema dual é ilimitado e o problema primal é inviável;

(2) encontrando-se um ponto extremo que maximiza o valor da função objetivo do problema dual $\sum_{ij\in A} f_{ij}\,\overline{y}_{ij} + \sum_{i\in N}\sum_{k\in K} g(i,k)\,\alpha_i^k - \sum_{ij\in A} (u_{ij}\,\overline{y}_{ij})\,\beta_{ij}$. Neste caso, tanto o problema primal e o problema dual apresentam soluções ótimas finitas.

Problema mestre.

$$\min_{y,z} \quad z$$

$$\text{s.t.} \quad z \geq \sum_{ij \in A} f_{ij} y_{ij} + \sum_{i \in N} \sum_{k \in K} g(i,k) \overline{\alpha}_i^k - \sum_{ij \in A} (u_{ij} \overline{\beta}_{ij}) y_{ij} \qquad \forall \text{ extreme points}$$

$$\sum_{i \in N} \sum_{k \in K} g(i,k) \overline{\alpha}_i^k - \sum_{ij \in A} (u_{ij} \overline{\beta}_{ij}) y_{ij} \leq 0 \qquad \forall \text{ extreme rays}$$

$$z \in \mathbb{R}$$

$$y_{ij} \in \{0,1\} \qquad \forall ij \in A$$

Subproblema.

$$\max_{\alpha,\beta} \quad \sum_{ij \in A} f_{ij} \overline{y}_{ij} + \sum_{i \in N} \sum_{k \in K} g(i,k) \alpha_i^k - \sum_{ij \in A} (u_{ij} \overline{y}_{ij}) \beta_{ij}$$

$$\text{s.t.} \quad \sum_{v \in N} h(ij,v) \alpha_v^k - \beta_{ij} \leq c_{ij} \qquad \forall ij \in A, \forall k \in K$$

$$\alpha_i^k \in \mathbb{R} \qquad \forall i \in N, \forall k \in K$$

$$\beta_{ij} \geq 0 \qquad \forall ij \in A$$

Implementação. Implementar o algoritmo de Benders conforme ilustrado abaixo. Considere solução inicial onde todos os links são utilizados ($y_{ij}=1, \forall ij \in A$). Em cada iteração, o subproblema e o problema mestre devem ser resolvidos, e os limites inferiores e superiores devem ser adequadamente atualizados. Considere o parâmetro $\varepsilon = 0.000001$. Note que a cada iteração apenas a função objetivo do subproblema deve ser modificada e um novo corte adicionado ao problema mestre. Deve ser gerado um arquivo de saída FCND.ben conforme especificado abaixo.

```
{initialization}
y := initial feasible integer solution
LB := −∞
UB := ∞
while UB − LB > ε do
  {solve subproblem}
  max_u{f^T ȳ + (b − Bȳ)^T u | A^T u ≤ c, u ≥ 0}
  if Unbounded then
    Get unbounded ray ū
    Add cut (b − By)^T ū ≤ 0 to master problem
  else
    Get extreme point ū
    Add cut z ≥ f^T y + (b − By)^T ū to master problem
    UB := min{UB, f^T ȳ + (b − Bȳ)^T ū}
  end if
  {solve master problem}
  min_y{z | cuts, y ∈ Y}
  LB := z̄
end while
```

Saída. Para cada iteração do algoritmo, o arquivo de saída deve apresentar o limite superior, o limite inferior, o gap e o tempo decorrido.

Detalhes de implementação.

(1) O parâmetro *presolver* deve estar desabilitado no subproblema:

```
cplex.setParam(IloCplex.BooleanParam.PreInd, false);
```

(2) Nome das variáveis duais do subproblema:

```
alpha = new IloNumVar[data.nNodes][data.nDemands];
for (int n = 0; n < data.nNodes; n++) {
	for (int k = 0; k < data.nDemands; k++) {
		String varName = "alpha_" + n + "_" + k;
		alpha[n][k] = cplex.numVar(-Double.MAX_VALUE, Double.MAX_VALUE, varName);
	}
}
beta = new IloNumVar[data.nLinks];
for (int m = 0; m < data.nLinks; m++) {
	String varName = "beta_" + m;
	beta[m] = cplex.numVar(0, Double.MAX_VALUE, varName);
}
```

(3) Teste de status do subproblema:

```
cplex.solve();
if (cplex.getStatus() == IloCplex.Status.Optimal) {
	recoverPoint();
	return 1;
} else if (cplex.getStatus() == IloCplex.Status.Unbounded) {
	recoverRay();
	return 0;
} else {
	return -1;
}
```

(4) Recuperação de raio extremo do subproblema:

```
private void recoverRay() throws IloException {
	alphaValue = new double[data.nNodes][data.nDemands];
	betaValue = new double[data.nLinks];
	Pattern patternAlpha = Pattern.compile("(alpha_)(\\d+)(_)(\\d+)");
	Pattern patternBeta = Pattern.compile("(beta_)(\\d+)");
	IloLinearNumExpr rayExpression = cplex.getRay();
	IloLinearNumExprIterator iterator = rayExpression.linearIterator();
	Matcher matcher;
	while (iterator.hasNext()) {
		IloNumVar variable = iterator.nextNumVar();
		String name = variable.getName();
		matcher = patternAlpha.matcher(name);
		if (matcher.find()) {
			int n = Integer.parseInt(matcher.group(2));
			int k = Integer.parseInt(matcher.group(4));
			alphaValue[n][k] = iterator.getValue();
		} else {
			matcher = patternBeta.matcher(name);
			if (matcher.find()) {
				int m = Integer.parseInt(matcher.group(2));
				betaValue[m] = iterator.getValue();
			}
		}
	}
}
```

(5) Adicionar corte no problema mestre:

```
cplex.addCut(cplex.ge(IloNumExpr arg0, double arg1));
```