

# Relatório Parcial de Iniciação Científica

**Aluno:** Tiago Madeira

**Orientador:** Prof. Dr. Carlos Eduardo Ferreira

28 de março de 2010

## Sumário

1	Introdução	2
2	Metodologia	3
3	Solução com programação linear inteira	4
4	Implementações da formulação linear	6
5	Testes e análise dos resultados	8
6	Conclusão	9

# 1 Introdução

Sejam  $T_I = (V_I, E_I)$  e  $T_M = (V_M, E_M)$  árvores com  $n$  e  $m$  vértices, respectivamente. Sejam  $A_I := \{(i, j) \mid i < j \text{ e } ij \in E_I\}$  e  $A_M := \{(i, j) \mid ij \in E_M \text{ ou } k = l \in V_M\}$ . Um mapeamento de  $T_I$  para  $T_M$  é uma função  $\alpha : V_I \rightarrow V_M$ . Fixados  $k \in V_M$  e um mapeamento  $\alpha$  de  $T_I$  para  $T_M$ , definimos  $\lambda(k) := \{i \in V_I \mid \alpha(i) = k\}$  e  $T_I[\lambda(k)]$  a subárvore de  $T_I$  induzida por  $\lambda(k)$ . Dadas duas funções de custo  $c : V_I \times V_M \rightarrow \mathbb{R}_+$  e  $w : A_I \times A_M \rightarrow \mathbb{R}_+$ , definimos o custo de um mapeamento  $\alpha$  como:

$$C(\alpha) := \sum_{i \in V_I} c(i, \alpha(i)) + \sum_{(i,j) \in A_I} w((i,j), (\alpha(i), \alpha(j)))$$

Solucionar a variação que estudamos do *problema de correspondência inexacta de grafos (PCIG)* é encontrar um mapeamento de  $T_I$  para  $T_M$  de custo mínimo, satisfazendo as seguintes restrições:

**Conectividade:**  $T_I[\lambda(k)]$  é uma árvore para todo  $k \in V_M$ .

**Cobertura de arestas:**  $(\alpha(i), \alpha(j)) \in E_M$  ou  $\alpha(i) = \alpha(j)$  para todo  $ij \in E_I$ .

Por simplicidade, usaremos  $c_{ik}$  e  $w_{kl}^{ij}$  no lugar de, respectivamente,  $c(i, k)$  e  $w((i, j), (k, l))$ .

## 2 Metodologia

O trabalho consistiu em aprendizagem sobre o *PCIG* e suas variantes, programação linear em geral e geração de árvores aleatórias. Foram experimentadas duas bibliotecas que resolvem problemas lineares[3][4] e foi estudada, implementada e testada uma formulação de programação linear inteira apresentada na dissertação de mestrado de Alexandre da Silva Freire[1] para resolver a variação do *PCIG* apresentada na Introdução deste trabalho.

Também foi estudado um algoritmo de programação dinâmica descrito em [1], mas este não chegou a ser implementado.

### 3 Solução com programação linear inteira

Para todos  $i \in V_I$  e  $k \in V_M$ , definimos:

$$x_{ik} := \begin{cases} 1 & \text{se } \alpha(i) = k \\ 0 & \text{caso contrário} \end{cases}$$

Para todos  $(i, j) \in A_I$  e  $(k, l) \in A_M$ , definimos:

$$y_{kl}^{ij} := \begin{cases} 1 & \text{se } \alpha(i) = k \text{ e } \alpha(j) = l \\ 0 & \text{caso contrário} \end{cases}$$

Então podemos definir a função objetivo como:

$$\min z' = \sum_{i=1}^n \sum_{k=1}^m c_{ik} \cdot x_{ik} + \sum_{(i,j) \in A_I} \sum_{(k,l) \in A_M} w_{kl}^{ij} \cdot y_{kl}^{ij} \quad (1)$$

No entanto, podemos nos livrar das variáveis  $x$ . Seja  $h_{kl}^{ij} := \frac{c_{ik}}{d(i)} + \frac{c_{jl}}{d(j)} + w_{kl}^{ij}$  o custo de mapear o par  $(i, j)$  para o par  $(k, l)$ , onde  $d(u)$  é o grau de  $u$ . Então, é fácil ver que (1) é equivalente a:

$$\min z = \sum_{(i,j) \in A_I} \sum_{(k,l) \in A_M} h_{kl}^{ij} \cdot y_{kl}^{ij} \quad (2)$$

Denotamos por  $\delta_T(u)$  os vizinhos de  $u$  em  $T$ . Dados vértices  $i, j \in V_I$  e  $k, l \in V_M$ , definimos  $S_{i-j} := \delta_{T_I}(i) \setminus \{j\}$ ,  $S_{ij} := S_{i-j} \cup S_{j-i}$  e  $D_{k-l} := \{k\} \cup \delta_{T_M}(k) \setminus \{l\}$ .

O problema linear pode ser definido agora da seguinte maneira:

$$\min \sum_{(i,j) \in A_I} \sum_{(k,l) \in A_M} h_{kl}^{ij} \cdot y_{kl}^{ij}$$

s. a.

$$\sum_{(k,l) \in A_M} y_{kl}^{ij} = 1 \quad \forall (i,j) \in A_I \quad (3)$$

$$\sum_{u \in S_{i-j}} \sum_{v \in D_{k-l}} y_{kv}^{iu} \geq |S_{i-j}| \cdot y_{kl}^{ij} \quad \forall (i,j) \in A_I, (k,l) \in A_M : k \neq l \quad (4)$$

$$\sum_{u \in S_{j-i}} \sum_{v \in D_{l-k}} y_{vl}^{uj} \geq |S_{j-i}| \cdot y_{kl}^{ij} \quad \forall (i,j) \in A_I, (k,l) \in A_M : k \neq l \quad (5)$$

$$\sum_{u \in S_{ij}} \sum_{v \in \delta_{T_M}(k) \cup \{k\}} y_{kv}^{iu} \geq |S_{ij}| \cdot y_{kk}^{ij} \quad \forall (i,j) \in A_I \text{ e } k \in V_M \quad (6)$$

$$\sum_{(i,j) \in A_I} (y_{kl}^{ij} + y_{lk}^{ij}) \leq 1 \quad \forall (k,l) \in A_M \text{ tais que } k < l \quad (7)$$

$$y_{kl}^{ij} \in \{0, 1\} \quad \forall (i,j) \in A_I \text{ e } (k,l) \in A_M \quad (8)$$

As restrições (3) garantem que cada aresta de  $T_I$  é mapeada pra exatamente uma aresta de  $T_M$  ou exatamente um vértice de  $T_M$ .

As restrições (4), (5) e (6) garantem a **cobertura de arestas**: se  $i$  é mapeado para  $k$  e  $j$  é mapeado pra  $l$ , então cada vértice  $u$  adjacente a  $i$  precisa estar mapeado para  $k$  ou para um vértice adjacente a  $k$  e, analogamente, cada vértice  $v$  adjacente a  $j$  precisa estar mapeado a  $l$  ou a um vértice adjacente a  $l$ .

As restrições (7) garantem a **conectividade** e as restrições (8) caracterizam o problema de programação linear inteira.

Em [1] é provado que esta é uma formulação válida para resolver nossa variante do PCIG.

## 4 Implementações da formulação linear

Inicialmente foi utilizado o *software livre* GLPK[3] para a implementação da solução com programação linear inteira e depois o *software proprietário* Gurobi[4]. Pela forma encapsulada como foi construído o projeto, não foi difícil alterar a biblioteca; bastou fazer modificações como:

### Função para adicionar linha no GLPK

```
void ip_add_row(double *val, char s, double v)
{
    int col;
    int row = glp_add_rows(ip, 1) - 1;
    nrows = row + 1;
    glp_set_row_bnds(ip, row + 1, (int) s, v, v);
    IP = realloc(IP, sizeof(double *) * nrows);
    alloc(IP[row], sizeof(double) * ncols);
    for (col = 0; col < ncols; col++) {
        IP[row][col] = val[col];
    }
}
```

### Função para adicionar linha no Gurobi

```
void ip_add_row(double *val, char s, double v)
{
    int i, k = 0;
    double *V;
    int *ind;
    alloc(V, sizeof(double)*ncols);
    alloc(ind, sizeof(int)*ncols);
    for (i = 0; i < ncols; i++) {
        if (val[i] != 0) {
            ind[k] = i;
            V[k++] = val[i];
        }
    }
}
```

```
        }  
    }  
    GRBaddconstr(model, k, ind, V, s, v, NULL);  
    free(V);  
    free(ind);  
}
```

A mudança de biblioteca no meio do projeto foi interessante para permitir uma comparação entre os dois resolvidores lineares e checar se a implementação retorna o resultado esperado.

Não convém detalhar os códigos, mas as implementações completas e comentadas podem ser encontradas em [5] e a diferença de tempo de execução entre as duas implementações pode ser vista nos testes na próxima seção.

## 5 Testes e análise dos resultados

Foi estudado o *Código de Prüfer* para geração de árvores aleatórias[2] e implementado um código que gera árvores e funções de custos como entradas para o programa que resolve o PCIG[5].

Foram geradas árvores aleatórias com diferentes números de vértices e cronometrados os tempos de execução<sup>1</sup> para os dois programas encontrarem seu mapeamento de custo mínimo, sendo obtidos os dados da tabela abaixo:

Teste	$ V_M $	$ V_I $	Tempo (GLPK[3])	Tempo (Gurobi[4])
1	10	10	0.023s	0.023s
2	10	20	2.644s	0.095s
3	20	20	2m18.637s	1.762s
4	10	30	3m6.941s	3.741s
5	20	30	3h5m13.090s	6.646s
6	20	50	—	22.514s
7	20	100	—	3m18.480s
8	20	200	—	1h22m6.584s

Tabela 1: Tempo de execução das implementações para árvores aleatórias

A partir do *Teste 6* não foi mais utilizado o GLPK, pois o Gurobi mostrou-se muito mais eficiente.

Foi iniciado um teste com  $V_M = 50$  e  $V_I = 500$  (que resultou numa matriz de programação linear com 123302 linhas e 73852 colunas) num computador de oito processadores 2.83GHz e 32 GB RAM do Instituto de Matemática e Estatística da Universidade de São Paulo, mas este foi cancelado após mais de 48 horas sem resultado.

---

<sup>1</sup>Todos os testes foram feitos num Intel Core 2 Duo 2.0GHz rodando Linux 2.6.32 i686.

## **6 Conclusão**

O trabalho não atingiu nenhum novo resultado, mas foi importante para proporcionar ao aluno uma experiência com programação linear, com a utilização de resolvidores de problemas lineares, contato com um problema real de otimização combinatória e com pesquisa nessa área.

## Referências

- [1] A. S. Freire: *Correspondência inexata entre grafos*. Dissertação de mestrado, Instituto de Matemática e Estatística da Universidade de São Paulo, 2008.
- [2] L. Markenzon, O. Vernet, P. R. Pereira: *Codificação de Árvores: Conceitos e Algoritmos*. Artigo científico, 2005.
- [3] A. Makhorin: *GLPK (GNU Linear Programming Kit)*. Moscow Aviation Institute, 2003. <http://www.gnu.org/software/glpk/>
- [4] Z. Gu, E. Rothberg, R. Bixby: *Gurobi*. <http://www.gurobi.com/>
- [5] T. Madeira: *Implementação de soluções lineares para o PCIG produzidas na IC*. Março/2010. <http://www.ime.usp.br/~madeira/2009/ic/>